scalecast-examples

Release 0.0.1

Michael Keith

Sep 20, 2023

NOTEBOOKS:

Welcome! Here we overview scalecast models and features. If you notice a typo, think something could be stated more clearly, or have any other suggestion or bug you want addressed, contact mikekeith52@gmail.com or open an issue.

Back to scalecast: https://scalecast.readthedocs.io/en/latest/

CHAPTER

ONE

INTRODUCTORY EXAMPLE

This demonstrates an example using scalecast `0.18.0`. Several features explored are not available in earlier versions, so if anything is not working, try upgrading:

```
pip install --upgrade scalecast
```

If things are still not working, you spot a typo, or you have some other suggestion to improve functionality or document readability, open an issue or email mikekeith52@gmail.com.

Link to dataset used in this example: https://www.kaggle.com/datasets/neuromusic/avocado-prices.

Latest official documentation.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# read in data
data = pd.read_csv('avocado.csv',parse_dates = ['Date'])
data = data.sort_values(['region','type','Date'])
```


1.1 Univariate Forecasting

1.1.1 Load the `Forecaster` Object

- This is an object that can store data, run forecasts, store results, and plot. It's a UI, procedure, and set of models all-in-one.
- Forecasts in scalecast are run with a dynamic recursive approach by default, as opposed to a direct or other approach.
- Forecaster Object Documentation.

[3]:
```
from scalecast.Forecaster import Forecaster
```

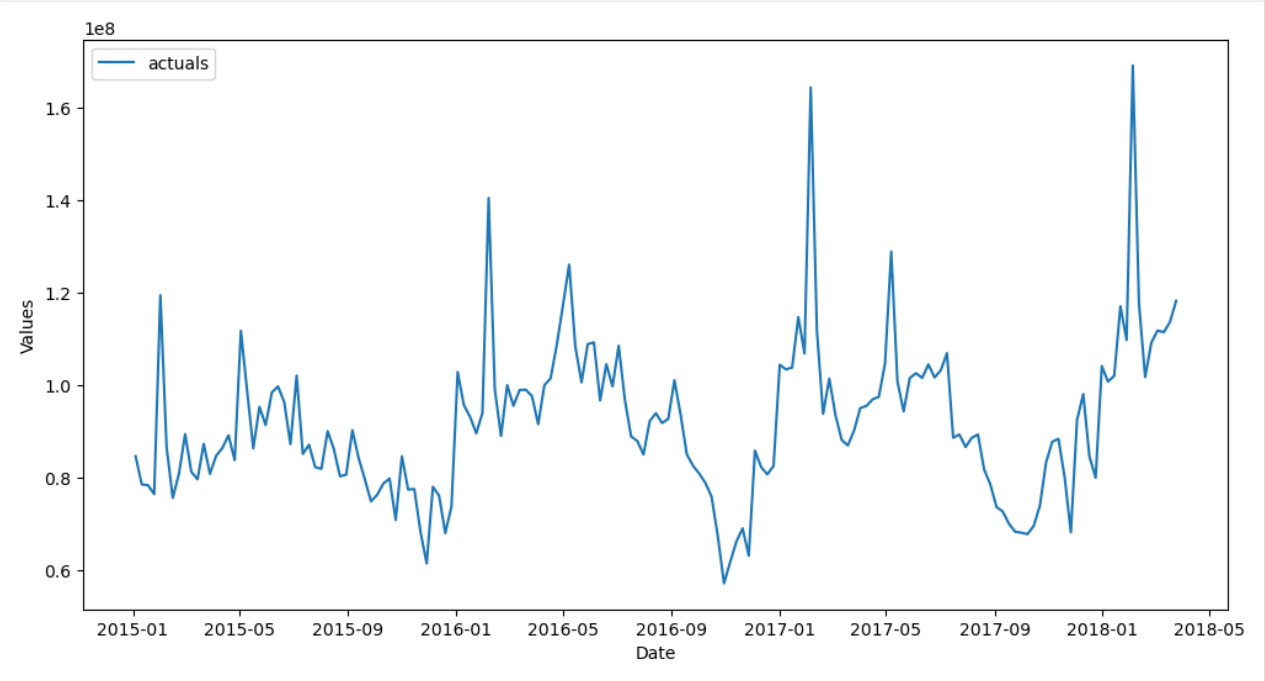
[4]:
```
volume = data.groupby('Date')['Total Volume'].sum()
```

[5]:
```
f = Forecaster(
    y = volume,
    current_dates = volume.index,
    future_dates = 13,
)

f
```

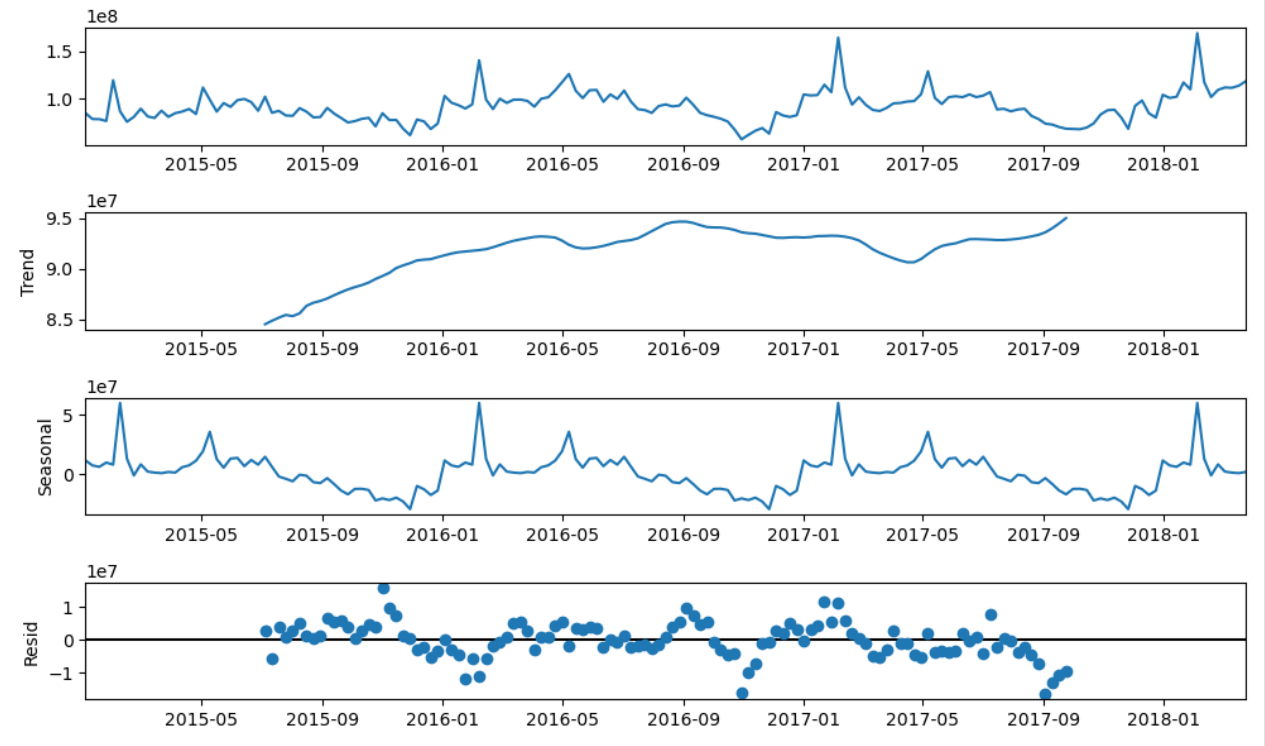
[5]:
```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=13
    Xvars=[]
    TestLength=0
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

1.1.2 Exploratory Data Analysis

[6]:
```
f.plot()
plt.show()
```



[7]:
```
plt.rc("figure",figsize=(10,6))
f.seasonal_decompose().plot()
plt.show()
```

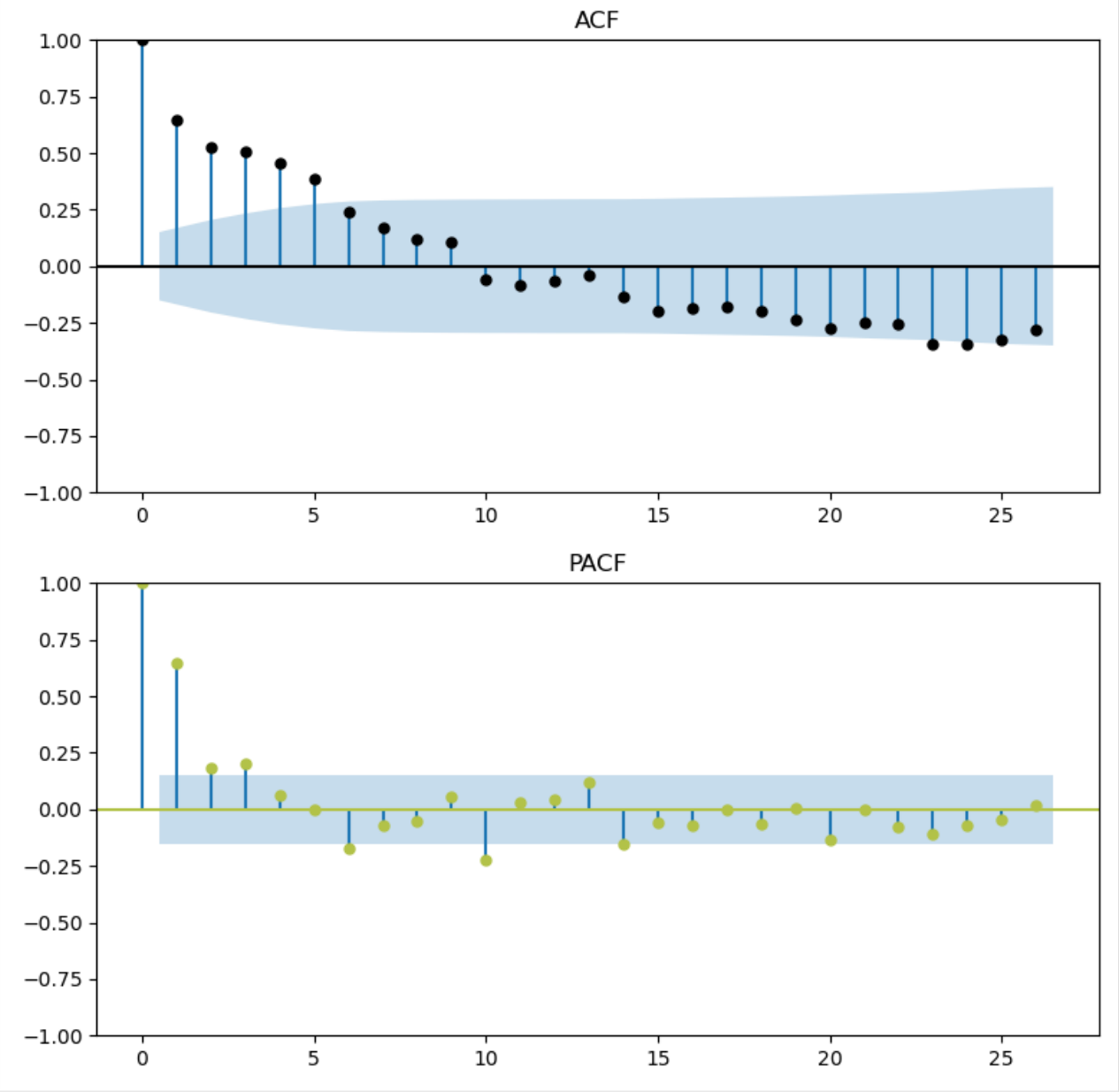


[8]:
```
figs, axs = plt.subplots(2, 1,figsize=(9,9))
f.plot_acf(ax=axs[0],title='ACF',lags=26,color='black')
```

(continues on next page)

(continued from previous page)

```
f.plot_pacf(ax=axs[1],title='PACF',lags=26,color='#B2C248',method='ywm')
plt.show()
```



1.1.3 Parameterize the `Forecaster` Object

1.1.3.1 Set Test Length

- Starting in scalecast version 0.16.0, you can skip model testing by setting a test length of 0.
- In this example, all models will be tested on the last 15% of the observed values in the dataset.

[9]:
```
f.set_test_length(.15)
```

1.1.3.2 Tell the Object to Evaluate Confidence Intervals

- This only works if there is a test set specified and it is of a sufficient size.
- See the documentation.
- See the example.

[10]:
```
# default args below
f.eval_cis(
    mode = True, # tell the object to evaluate intervals
    cilevel = .95, # 95% confidence level
)
```

1.1.3.3 Specify Model Inputs

Trend

[11]:
```
f.add_time_trend()
```

Seasonality

[12]:
```
f.add_seasonal_regressors('week',raw=False,sincos=True)
```

Autoregressive Terms / Series Lags

[13]:
```
f.add_ar_terms(13)
```

[14]:
```
f
```

[14]:
```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=13
    Xvars=['t', 'weeksin', 'weekcos', 'AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7',
↪'AR8', 'AR9', 'AR10', 'AR11', 'AR12', 'AR13']
```

(continues on next page)

(continued from previous page)

```
    TestLength=25
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

1.1.4 Run Models

- See the available models.
- See the blog post.
- The `dynamic_testing` argument for all of these will be 13 – test-set results will then be in terms of rolling averages of 13-step forecasts, which is also our forecast length.
- The resulting forecasts from this process are not well-fit. Better forecasts are obtained once more optimization is performed in later sections.

1.1.4.1 Linear Scikit-Learn Models

[15]:
```
f.set_estimator('mlr')
f.manual_forecast(dynamic_testing=13)
```

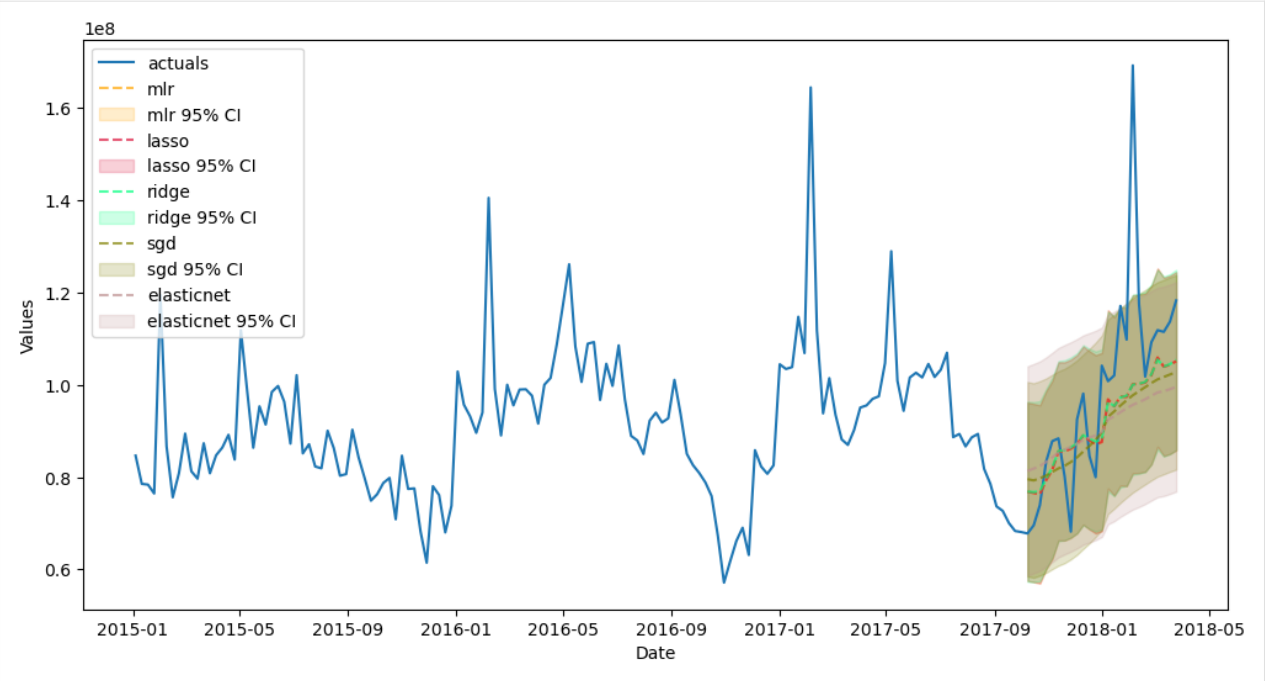
[16]:
```
f.set_estimator('lasso')
f.manual_forecast(alpha=0.2,dynamic_testing=13)
```

[17]:
```
f.set_estimator('ridge')
f.manual_forecast(alpha=0.2,dynamic_testing=13)
```

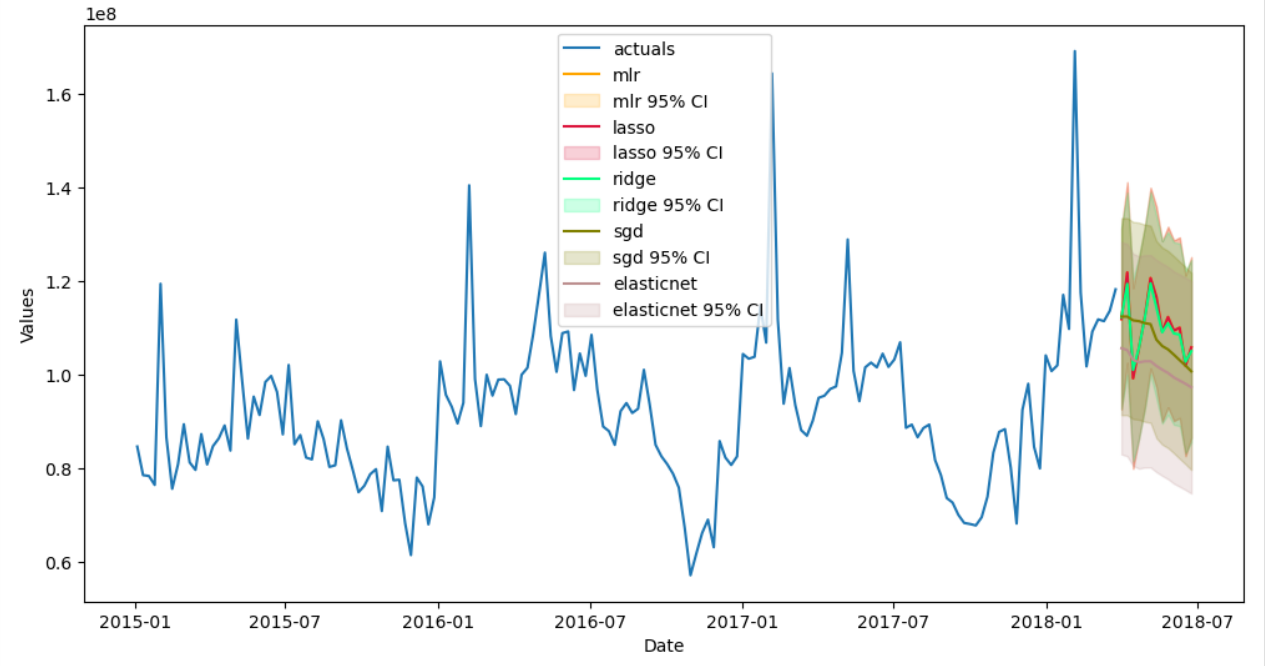
[18]:
```
f.set_estimator('elasticnet')
f.manual_forecast(alpha=0.2,l1_ratio=0.5,dynamic_testing=13)
```

[19]:
```
f.set_estimator('sgd')
f.manual_forecast(alpha=0.2,l1_ratio=0.5,dynamic_testing=13)
```

[20]:
```
f.plot_test_set(ci=True,models=['mlr','lasso','ridge','elasticnet','sgd'],order_by=
↪'TestSetRMSE')
plt.show()
```



[21]:
```
f.plot(ci=True,models=['mlr','lasso','ridge','elasticnet','sgd'],order_by='TestSetRMSE')
plt.show()
```



1.1.4.2 Non-linear Scikit-Learn Models

[22]:
```
f.set_estimator('rf')
f.manual_forecast(max_depth=2,dynamic_testing=13)
```

[23]:
```
f.set_estimator('gbt')
f.manual_forecast(max_depth=2,dynamic_testing=13)
```

[24]:
```
f.set_estimator('xgboost')
f.manual_forecast(gamma=1,dynamic_testing=13)
```

[25]:
```
f.set_estimator('lightgbm')
f.manual_forecast(max_depth=2,dynamic_testing=13)
```

[26]:
```
f.set_estimator('catboost')
f.manual_forecast(depth=4,verbose=False,dynamic_testing=13)
```

```
Finished loading model, total used 100 iterations
```

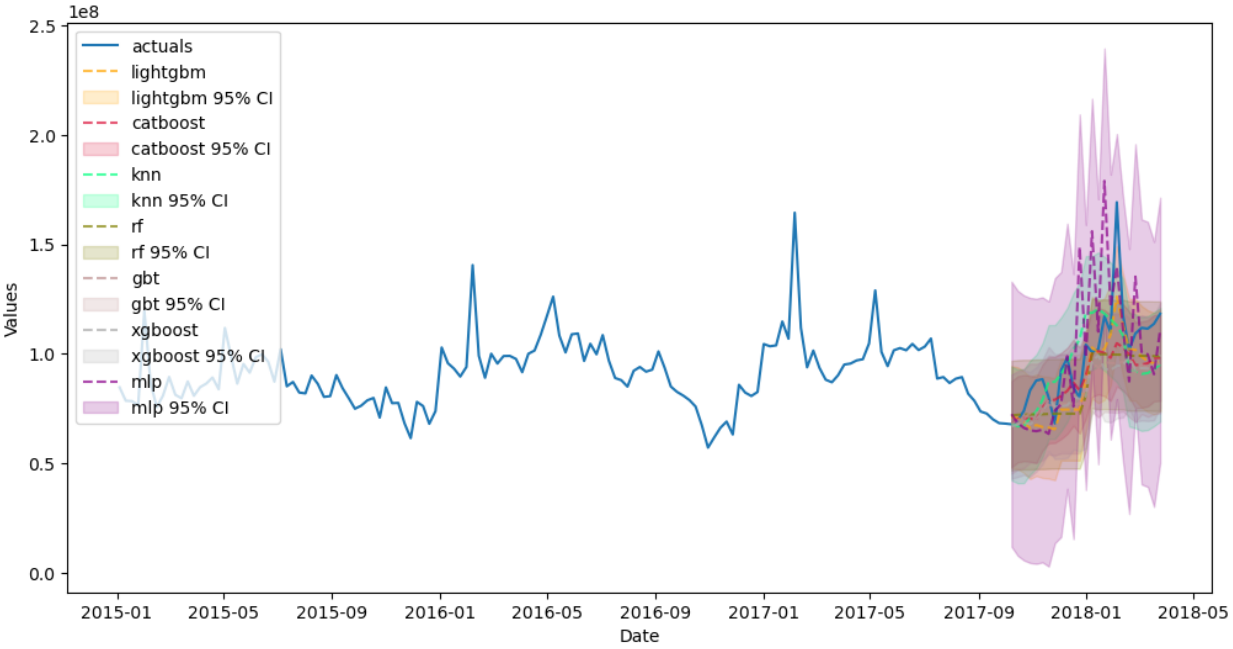
[27]:
```
f.set_estimator('knn')
f.manual_forecast(n_neighbors=5,dynamic_testing=13)
```

```
Finished loading model, total used 100 iterations
```

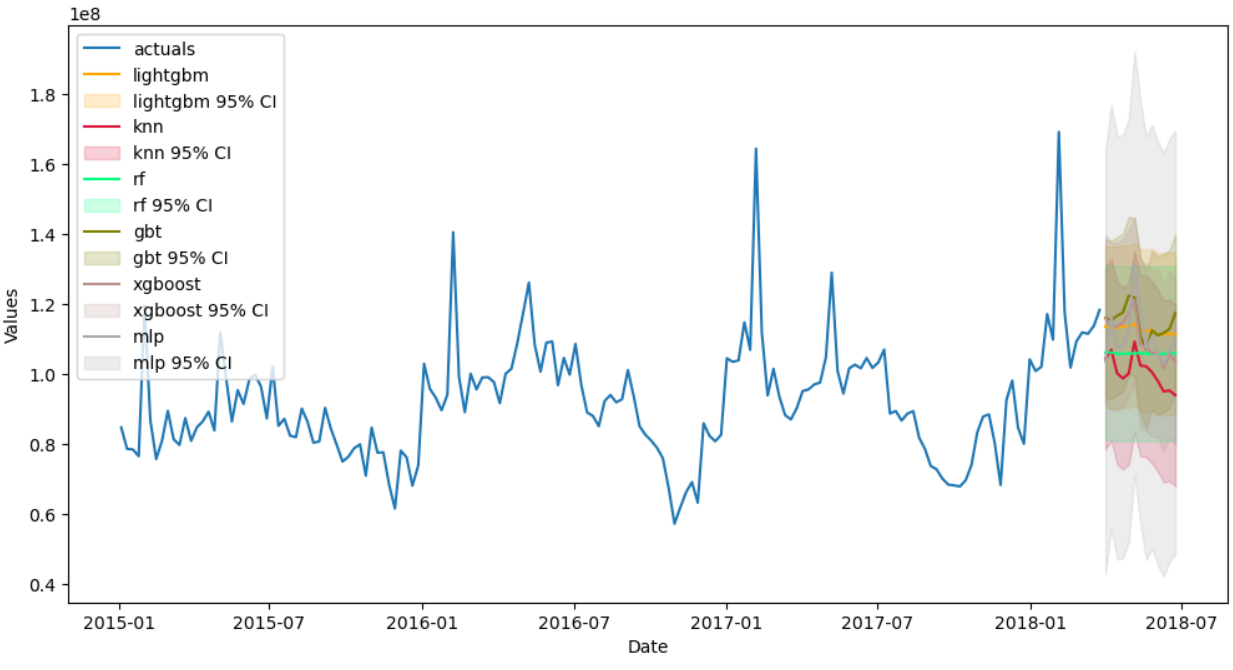
[28]:
```
f.set_estimator('mlp')
f.manual_forecast(hidden_layer_sizes=(50,50),solver='lbfgs',dynamic_testing=13)
```

```
Finished loading model, total used 100 iterations
```

[29]:
```
f.plot_test_set(
    ci=True,
    models=['rf','gbt','xgboost','lightgbm','catboost','knn','mlp'],
    order_by='TestSetRMSE'
)
plt.show()
```



[30]:
```
f.plot(ci=True,models=['rf','gbt','xgboost','lightgbm','knn','mlp'],order_by='TestSetRMSE
↪')
plt.show()
```



1.1.4.3 Stacking Models

Sklearn Stacking Model

[31]:
```
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import ElasticNet
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
```

[32]:
```
f.add_sklearn_estimator(StackingRegressor,'stacking')
```

[33]:
```
estimators = [
    ('elasticnet',ElasticNet(alpha=0.2)),
    ('xgboost',XGBRegressor(gamma=1)),
    ('gbt',GradientBoostingRegressor(max_depth=2)),
]

final_estimator = LGBMRegressor()

f.set_estimator('stacking')
f.manual_forecast(
    estimators=estimators,
    final_estimator=final_estimator,
    dynamic_testing=13
)
```

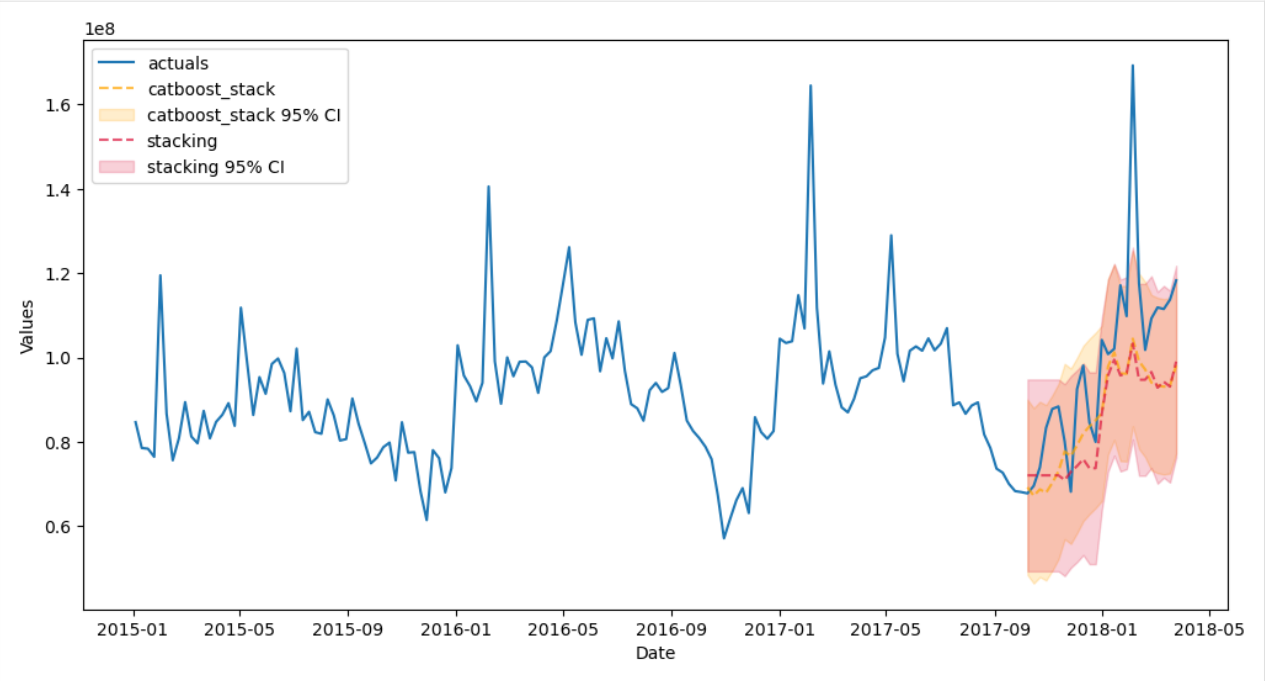
```
Finished loading model, total used 100 iterations
```

Scalecast Stacking Model

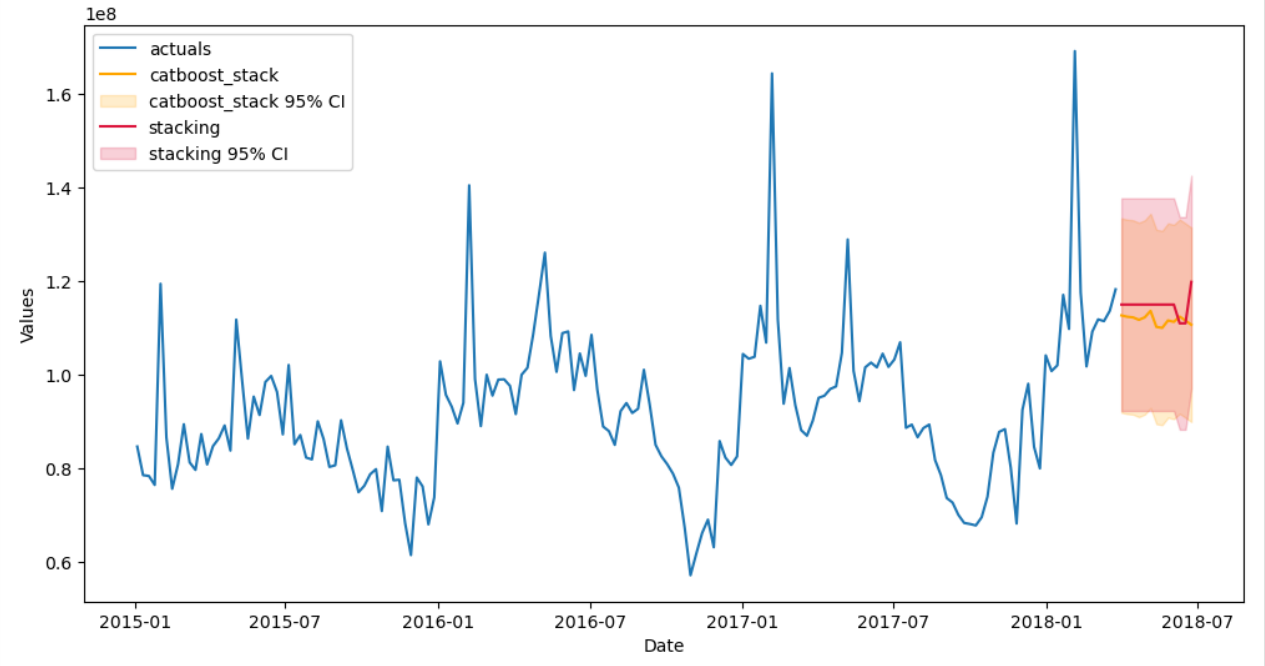
[34]:
```
f.add_signals(['elasticnet','lightgbm','xgboost','knn'],train_only=True)
f.set_estimator('catboost')
f.manual_forecast(call_me = 'catboost_stack',verbose=False)
```

```
Finished loading model, total used 100 iterations
Finished loading model, total used 100 iterations
```

[35]:
```
f.plot_test_set(models=['stacking','catboost_stack'],ci=True,order_by='TestSetRMSE')
plt.show()
```



[36]:
```
f.plot(models=['stacking','catboost_stack'],ci=True,order_by='TestSetRMSE')
plt.show()
```



1.1.4.4 ARIMA

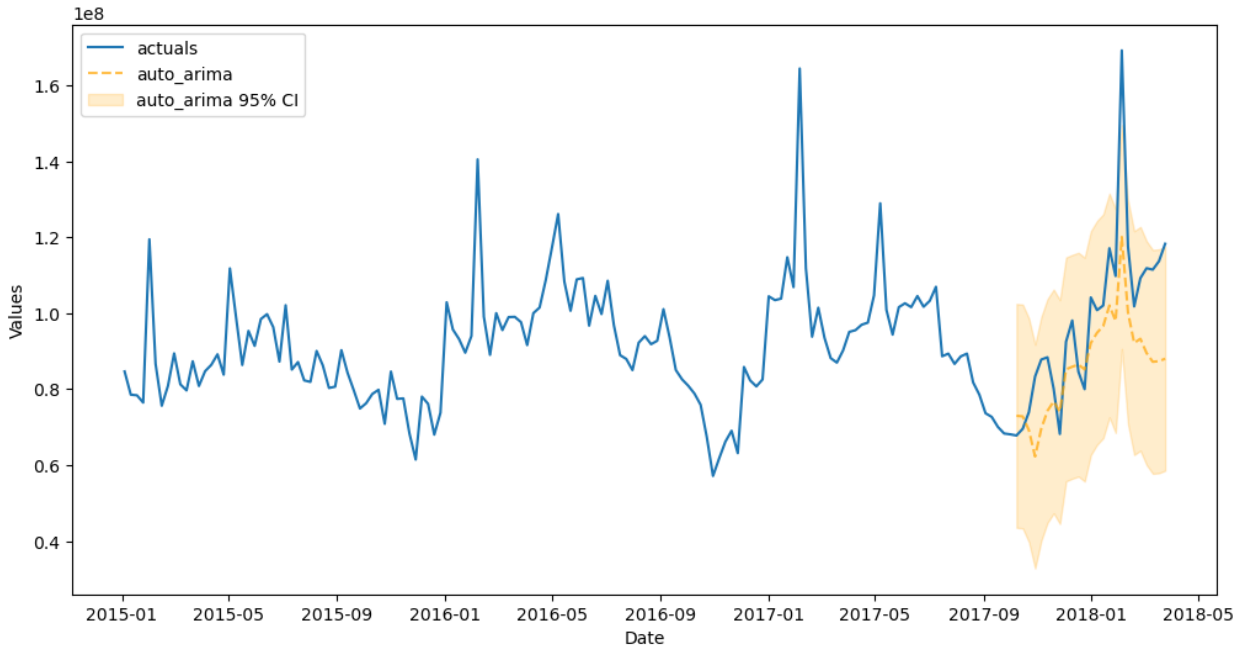
- auxmodels.auto_arima

[37]:
```
from scalecast.auxmodels import auto_arima
```

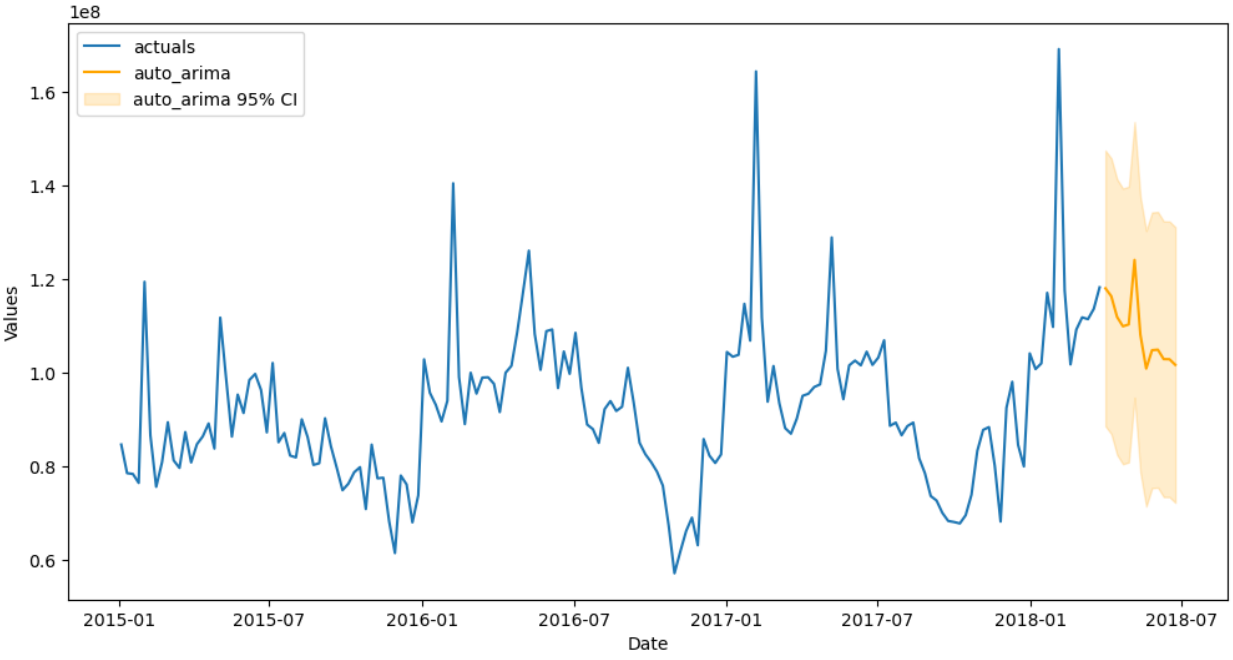
[38]:
```
auto_arima(f,m=52)
```

```
Finished loading model, total used 100 iterations
Finished loading model, total used 100 iterations
```

[39]:
```
f.plot_test_set(models='auto_arima',ci=True)
plt.show()
```



[40]:
```
f.plot(models='auto_arima',ci=True)
plt.show()
```



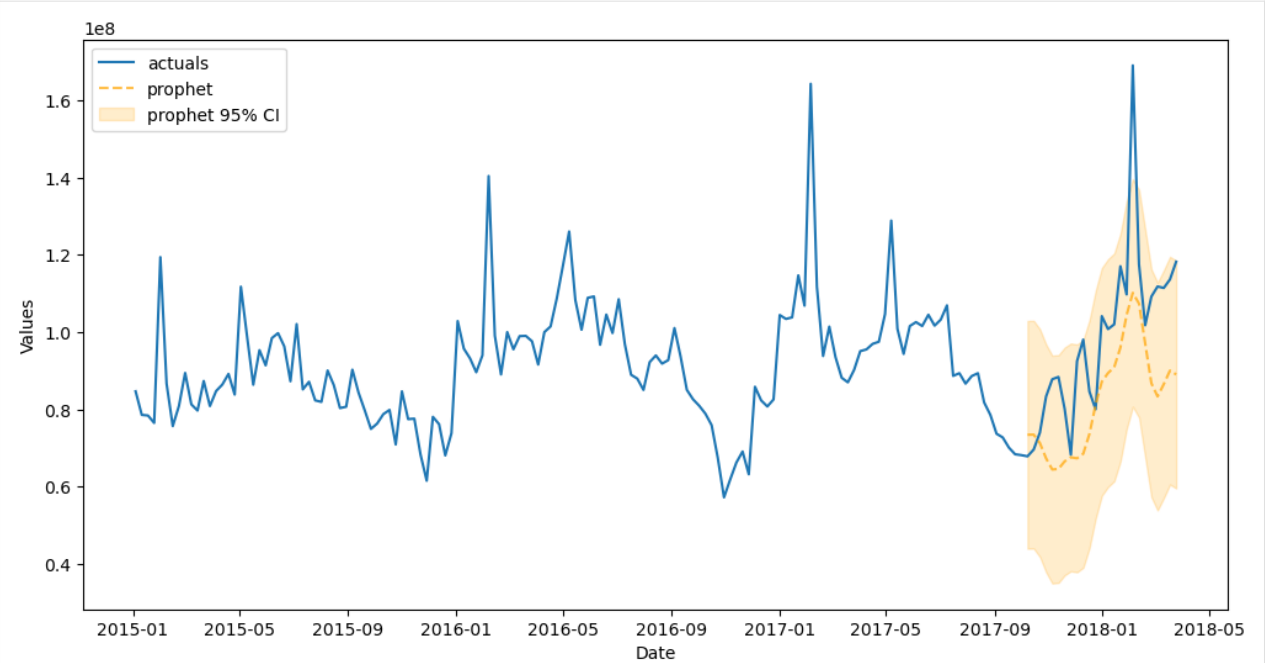
1.1.4.5 Prophet

[41]:
```
f.set_estimator('prophet')
f.manual_forecast()
```

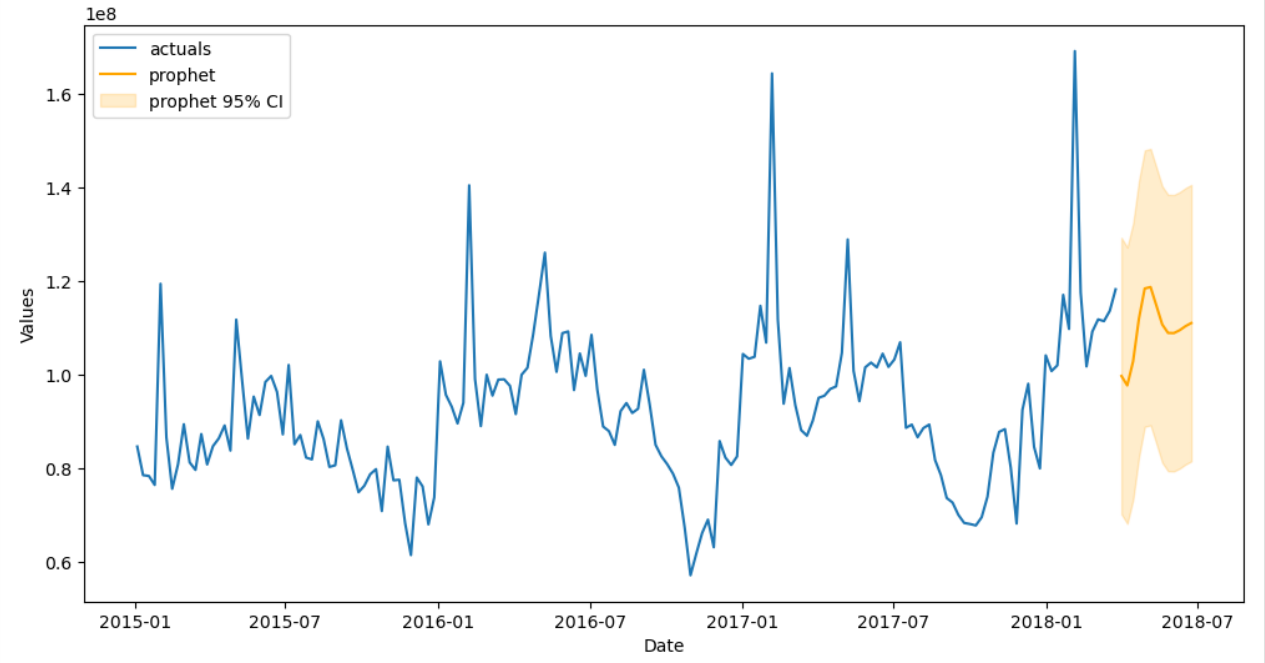
```
Finished loading model, total used 100 iterations
Finished loading model, total used 100 iterations
```

```
22:14:17 - cmdstanpy - INFO - Chain [1] start processing
22:14:17 - cmdstanpy - INFO - Chain [1] done processing
22:14:17 - cmdstanpy - INFO - Chain [1] start processing
22:14:17 - cmdstanpy - INFO - Chain [1] done processing
```

[42]:
```
f.plot_test_set(models='prophet',ci=True)
plt.show()
```



[43]:
```
f.plot(models='prophet',ci=True)
plt.show()
```



Other univariate models available: TBATS, Holt-Winters Exponential Smoothing, LSTM, RNN, Silverkite, Theta.

Working on: N-Beats, N-Hits, Genetic Algorithm.

1.2 Multivariate Forecasting

1.2.1 Load the `MVForecaster` Object

- This object extends the univariate approach to several series, with many of the same plotting and reporting features available.
- Documentation

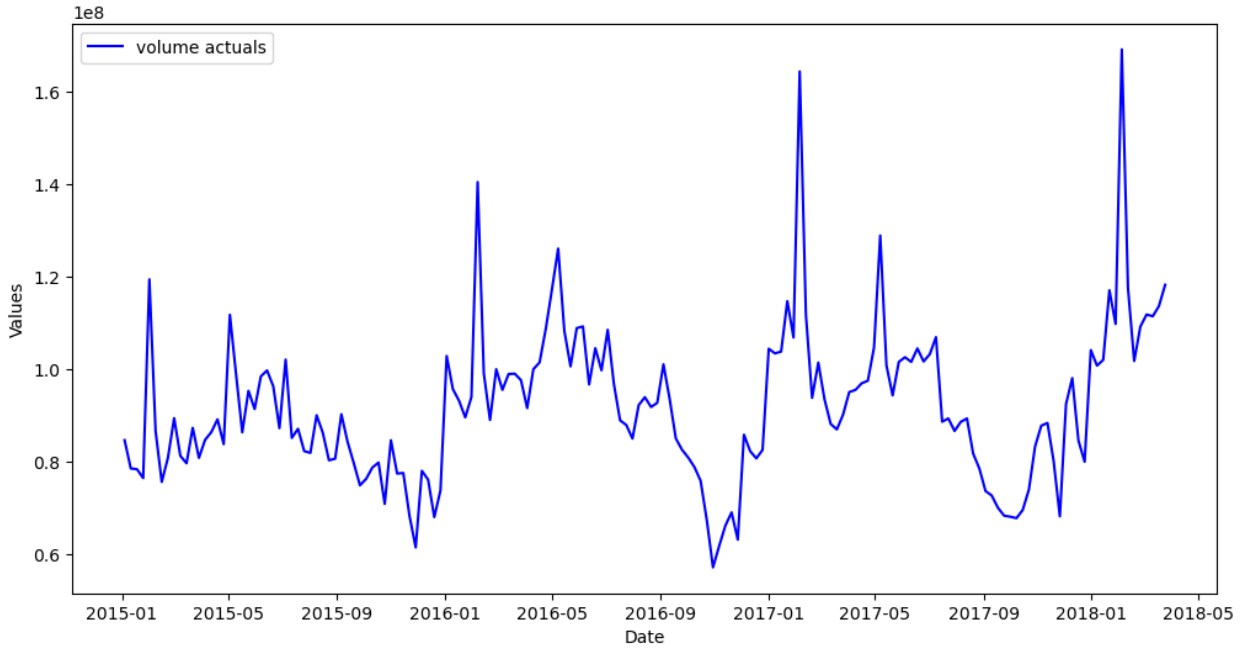
[44]:
```
from scalecast.MVForecaster import MVForecaster
```

[45]:
```
price = data.groupby('Date')['AveragePrice'].mean()

fvol = Forecaster(y=volume,current_dates=volume.index,future_dates=13)
fprice = Forecaster(y=price,current_dates=price.index,future_dates=13)

fvol.add_time_trend()
fvol.add_seasonal_regressors('week',raw=False,sincos=True)

mvf = MVForecaster(
    fvol,
    fprice,
    merge_Xvars='union',
    names=['volume','price'],
)

mvf
```

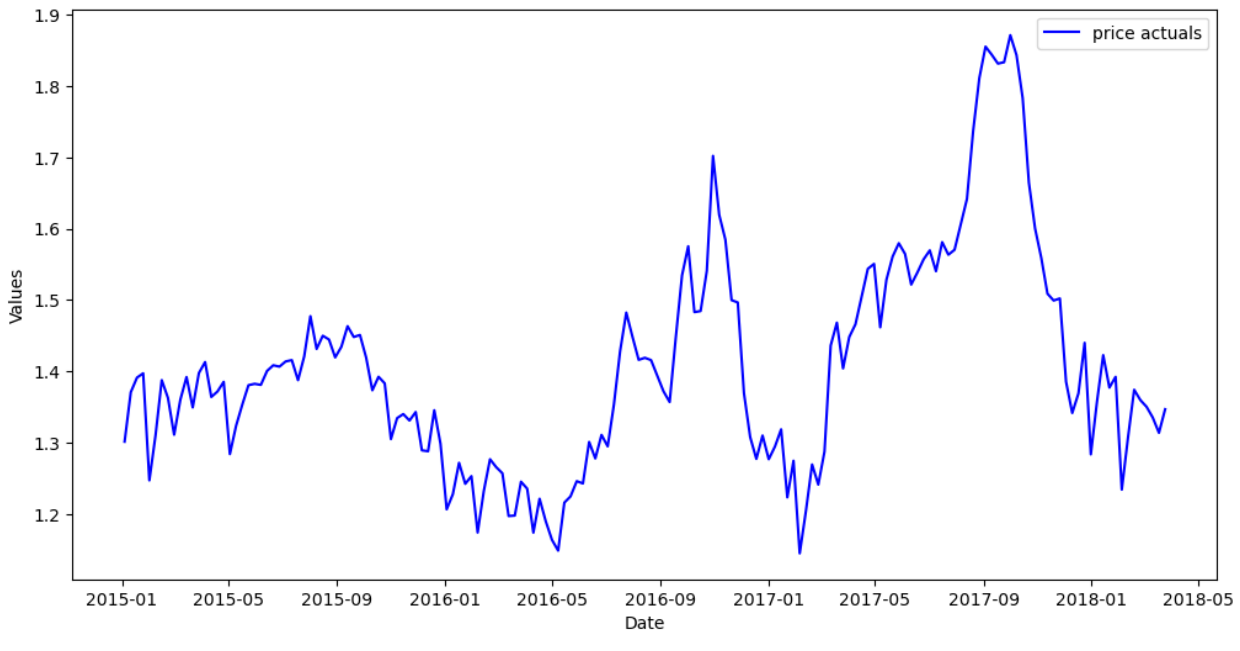
[45]:
```
MVForecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    N_series=2
    SeriesNames=['volume', 'price']
    ForecastLength=13
    Xvars=['t', 'weeksin', 'weekcos']
    TestLength=0
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    OptimizeOn=mean
    GridsFile=MVGrids
)
```

1.2.2 Exploratory Data Analysis

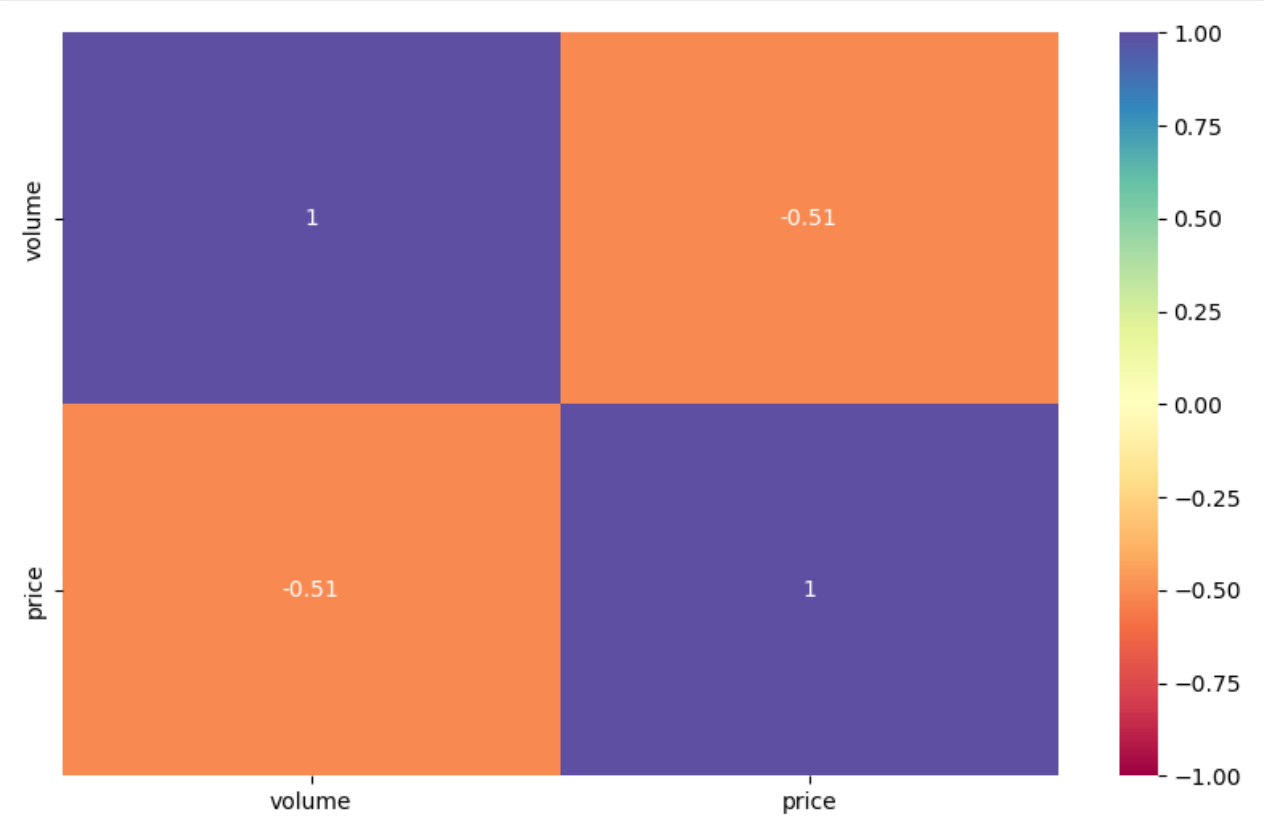
[46]:
```
mvf.plot(series='volume')
plt.show()
```



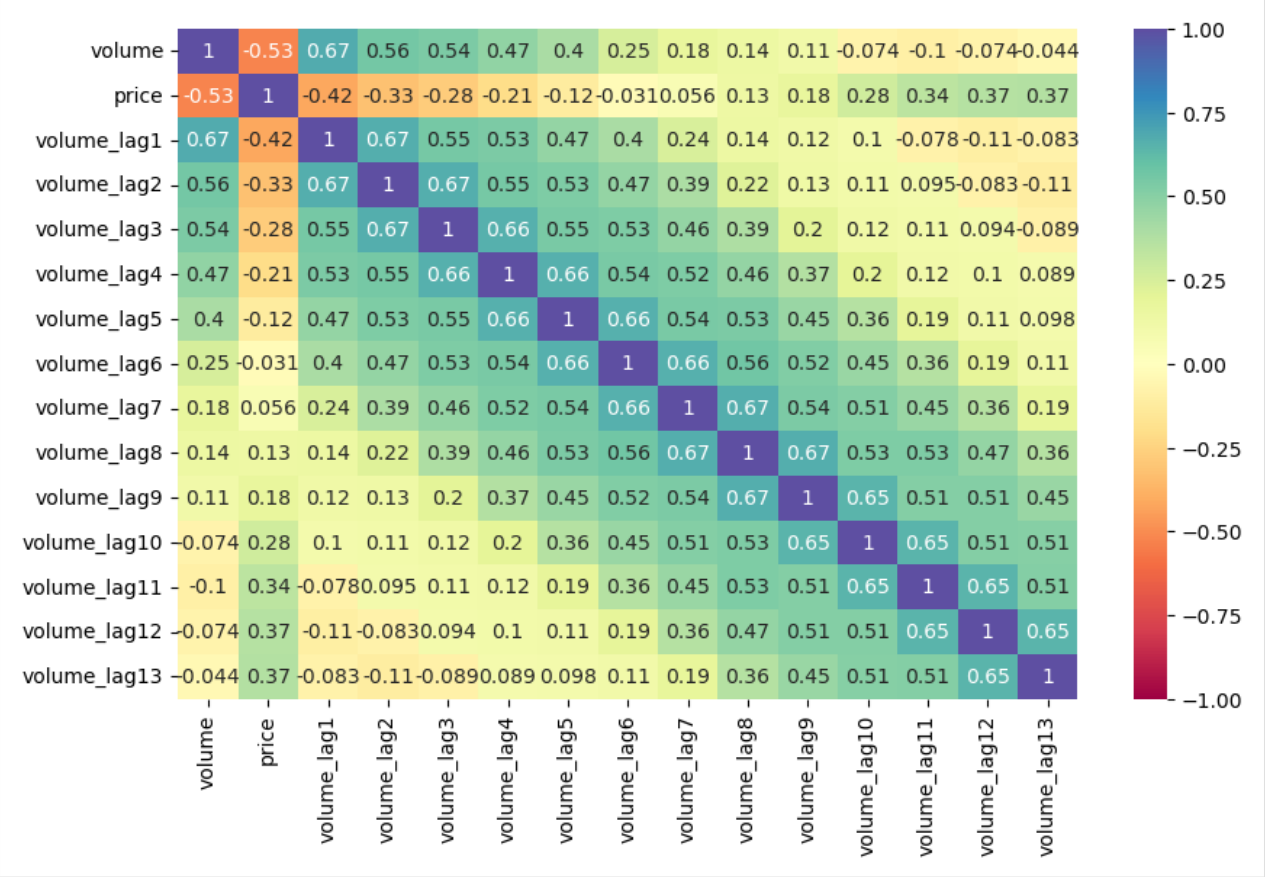
[47]:
```
mvf.plot(series='price')
plt.show()
```



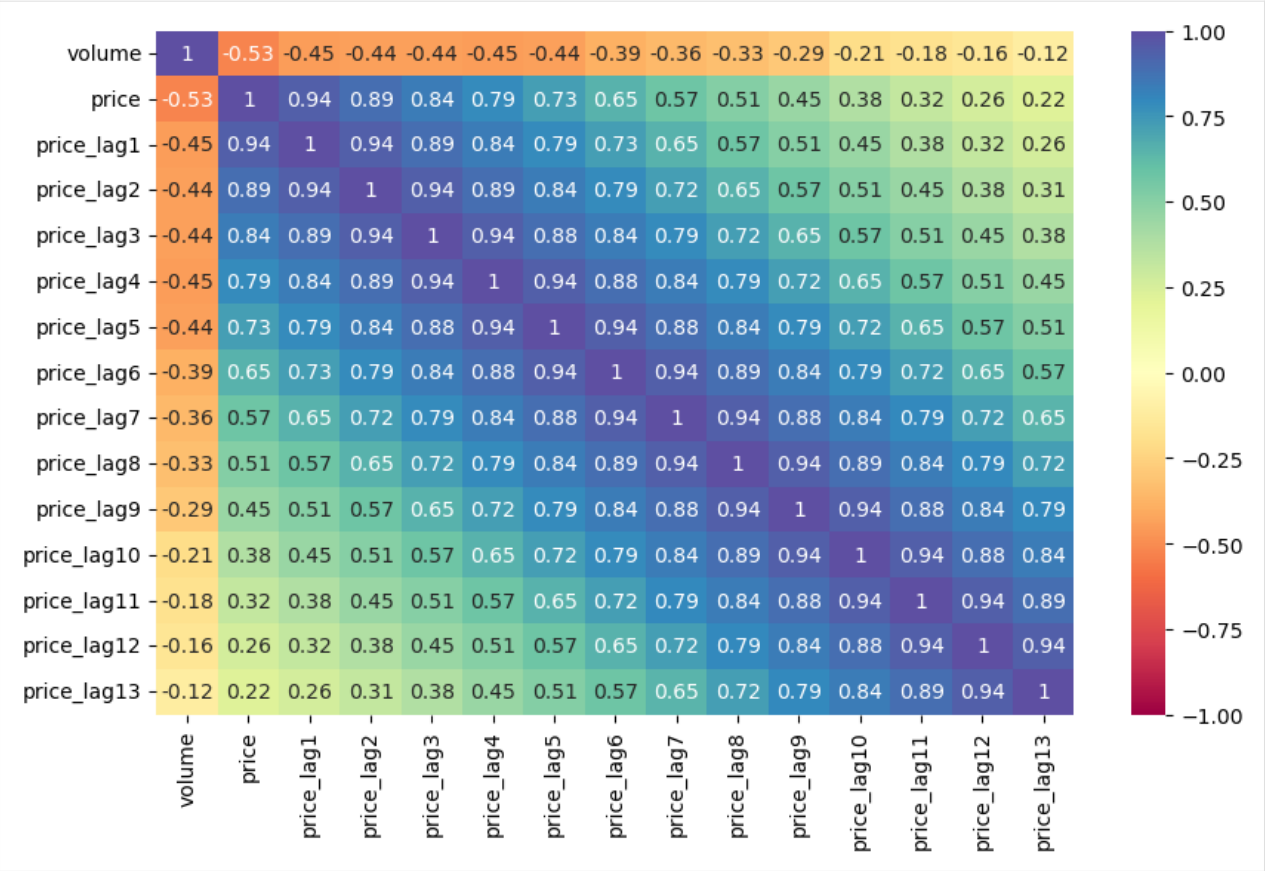
[48]:
```
mvf.corr(disp='heatmap',cmap='Spectral',annot=True,vmin=-1,vmax=1)
plt.show()
```



[49]:
```
mvf.corr_lags(y='price',x='volume',disp='heatmap',cmap='Spectral',annot=True,vmin=-1,
↪vmax=1,lags=13)
plt.show()
```



[50]:
```
mvf.corr_lags(y='volume',x='price',disp='heatmap',cmap='Spectral',annot=True,vmin=-1,
↪vmax=1,lags=13)
plt.show()
```



1.2.3 Parameterize the `MVForecaster` Object

- Starting in scalecast version 0.16.0, you can skip model testing by setting a test length of 0.
- In this example, all models will be tested on the last 15% of the observed values in the dataset.
- We will also have model optimization select hyperparemeters based on what predicts the volume series, rather than the price series, or an average of the two (which is the default), best.
- Custom optimization functions are available.

[51]:
```
mvf.set_test_length(.15)
mvf.set_optimize_on('volume') # we care more about predicting volume and price is just␣
↪used to make those predictions more accurate
# by default, the optimizer uses an average scoring of all series in the MVForecaster␣
↪object
mvf.eval_cis() # tell object to evaluate cis

mvf
```

[51]:
```
MVForecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    N_series=2
```

(continues on next page)

(continued from previous page)

```
    SeriesNames=['volume', 'price']
    ForecastLength=13
    Xvars=['t', 'weeksin', 'weekcos']
    TestLength=25
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    OptimizeOn=volume
    GridsFile=MVGrids
)
```

1.2.4 Run Models

- Uses scikit-learn models and APIs only.
- See the adapted VECM model for this object.

1.2.4.1 ElasticNet

[52]:
```
mvf.set_estimator('elasticnet')
mvf.manual_forecast(alpha=0.2,dynamic_testing=13,lags=13)
```

1.2.4.2 XGBoost

[53]:
```
mvf.set_estimator('xgboost')
mvf.manual_forecast(gamma=1,dynamic_testing=13,lags=13)
```

1.2.4.3 MLP Stack

- auxmodels.mlp_stack

[54]:
```
from scalecast.auxmodels import mlp_stack
```

[55]:
```
mvf.export('model_summaries')
```

[55]:
```
   Series ModelNickname   Estimator                 Xvars     HyperParams  \
0  volume    elasticnet  elasticnet  [t, weeksin, weekcos]  {'alpha': 0.2}
1  volume       xgboost     xgboost  [t, weeksin, weekcos]    {'gamma': 1}
2   price    elasticnet  elasticnet  [t, weeksin, weekcos]  {'alpha': 0.2}
3   price       xgboost     xgboost  [t, weeksin, weekcos]    {'gamma': 1}

   Lags  Observations  DynamicallyTested  TestSetLength  ValidationMetric  \
0    13           169                 13             25               NaN
1    13           169                 13             25               NaN
2    13           169                 13             25               NaN
3    13           169                 13             25               NaN
```

(continues on next page)

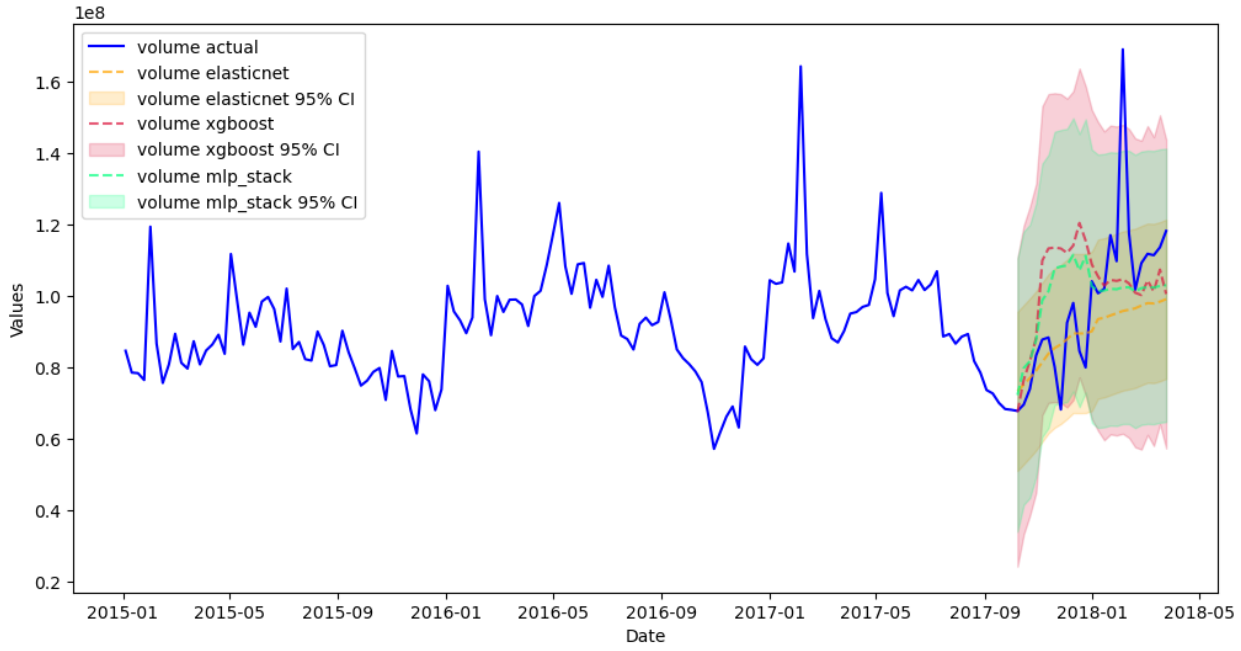
(continued from previous page)

```
   ValidationMetricValue  InSampleRMSE  InSampleMAPE   InSampleMAE  \
0                    NaN  1.197603e+07      0.088917  8.206334e+06
1                    NaN  3.416022e+02      0.000003  2.591667e+02
2                    NaN  1.560728e-01      0.084080  1.199107e-01
3                    NaN  1.141005e-01      0.058215  8.255353e-02

   InSampleR2   TestSetRMSE  TestSetMAPE    TestSetMAE  TestSetR2
0    0.486109  1.869490e+07     0.118172  1.282436e+07   0.248765
1    1.000000  2.260606e+07     0.173554  1.644149e+07  -0.098447
2    0.000000  1.522410e-01     0.071282  1.088633e-01  -0.048569
3    0.465533  1.291758e-01     0.057186  8.791148e-02   0.245089
```

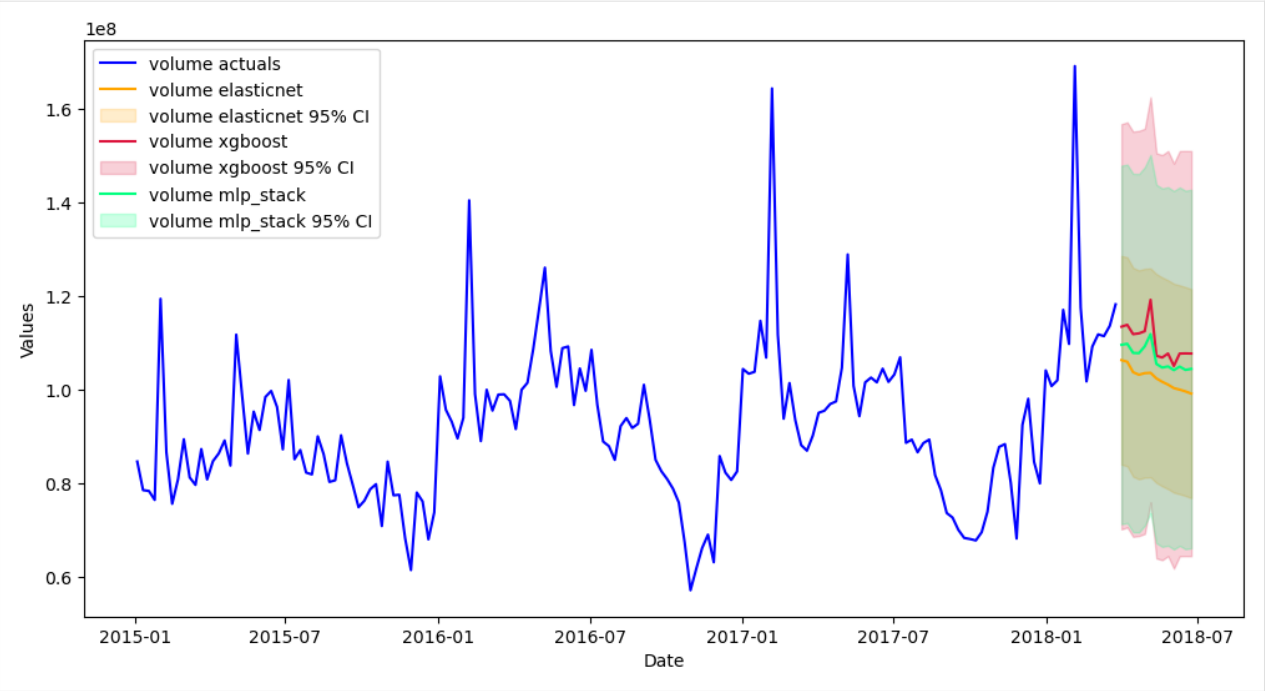
[56]:
```
mlp_stack(mvf,model_nicknames=['elasticnet','xgboost'],lags=13)
```

[57]:
```
mvf.set_best_model(determine_best_by='TestSetRMSE')
```

[58]:
```
mvf.plot_test_set(ci=True,series='volume',put_best_on_top=True)
plt.show()
```

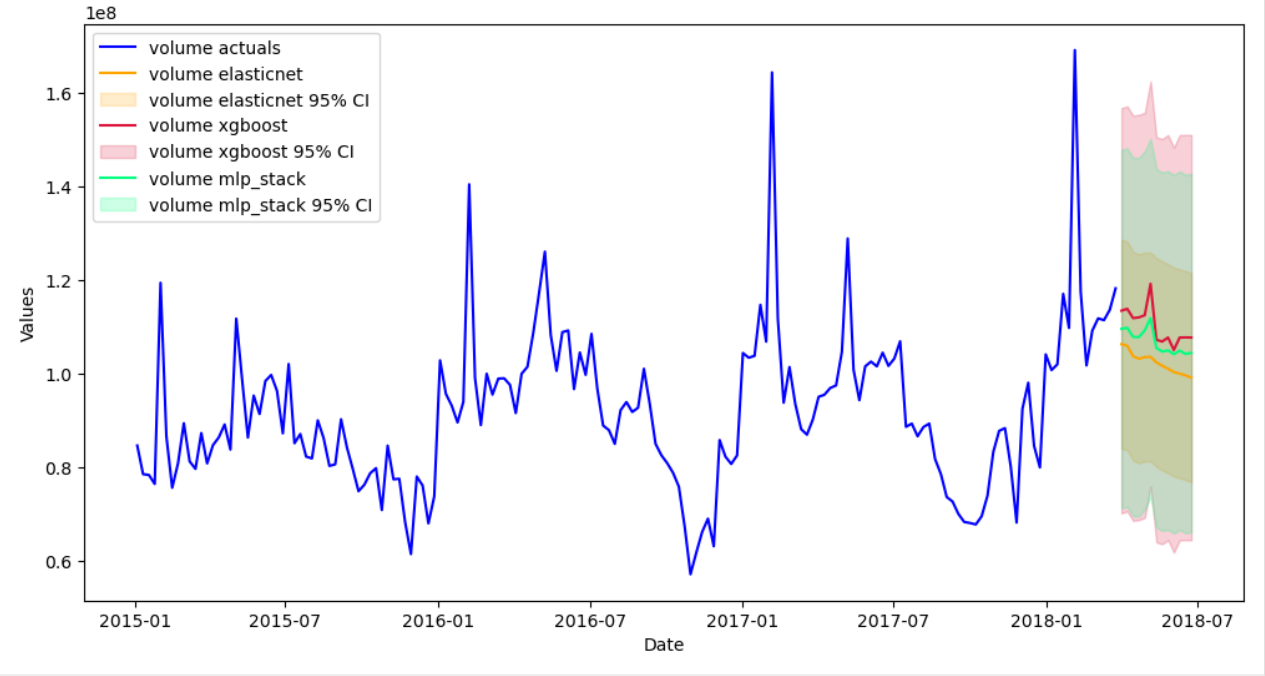


[59]:
```
mvf.plot(ci=True,series='volume',put_best_on_top=True)
plt.show()
```



Probabalistic forecasting for creating confidence intervals is currently being worked on in the MVForecaster object, but until that is done, the backtested interval also works well:

[60]:
```
mvf.plot(ci=True,series='volume',put_best_on_top=True)
plt.show()
```



1.2.5 Break Back into Forecaster Objects

- You can then add univariate models to these objects to compare with the models run multivariate.

[61]:
```
from scalecast.util import break_mv_forecaster
```

[62]:
```
fvol, fprice = break_mv_forecaster(mvf)
```

[63]:
```
fvol
```

[63]:
```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=13
    Xvars=[]
    TestLength=25
    ValidationMetric=rmse
    ForecastsEvaluated=['elasticnet', 'xgboost', 'mlp_stack']
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[64]:
```
fprice
```

[64]:
```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=13
    Xvars=[]
    TestLength=25
    ValidationMetric=rmse
    ForecastsEvaluated=['elasticnet', 'xgboost', 'mlp_stack']
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

1.3 Transformations

- One of the most effective way to boost forecasting power is with transformations.
- Transformations include:
- Log
- Scaling
  - Standard
  - MinMax

- Differencing
- Detrending
- Custom Functions
- All transformations have a corresponding revert function.
- See the blog post.

[65]:
```
from scalecast.SeriesTransformer import SeriesTransformer
```

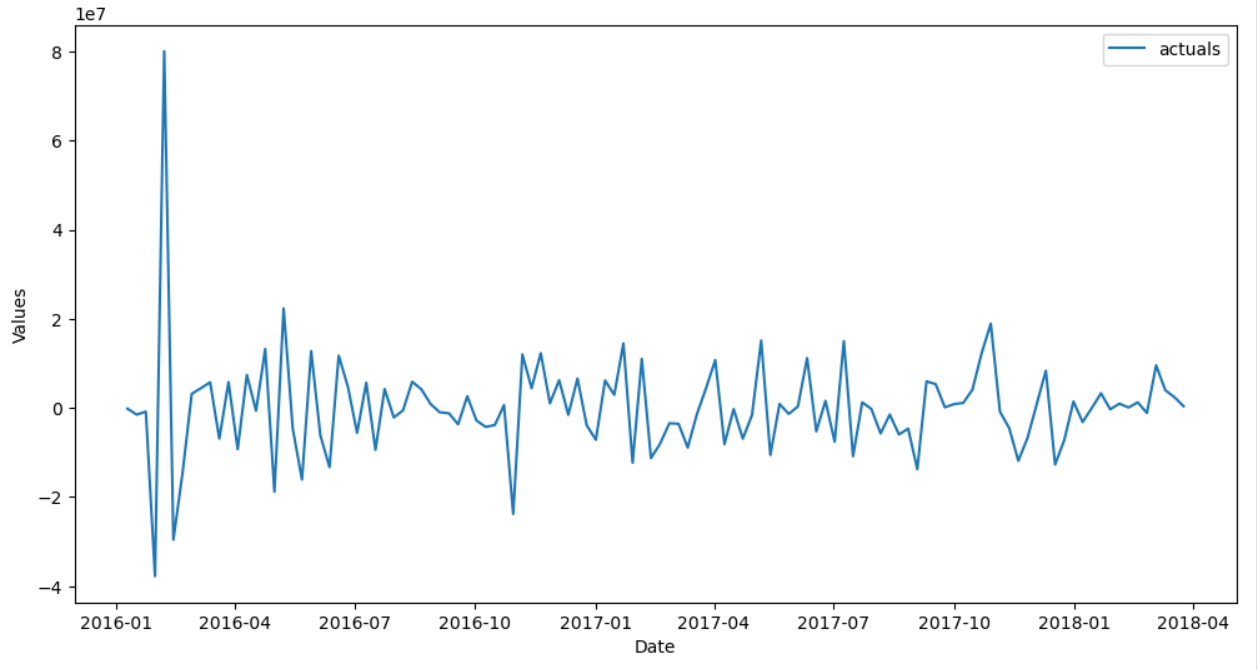
[66]:
```
f_trans = Forecaster(y=volume,current_dates=volume.index,future_dates=13)
```

[67]:
```
f_trans.set_test_length(.15)
f_trans.set_validation_length(13)
```

[68]:
```
transformer = SeriesTransformer(f_trans)
```

[69]:
```
# these will all be reverted later after forecasts have been called
f_trans = transformer.DiffTransform(1)
f_trans = transformer.DiffTransform(52)
f_trans = transformer.DetrendTransform()
```

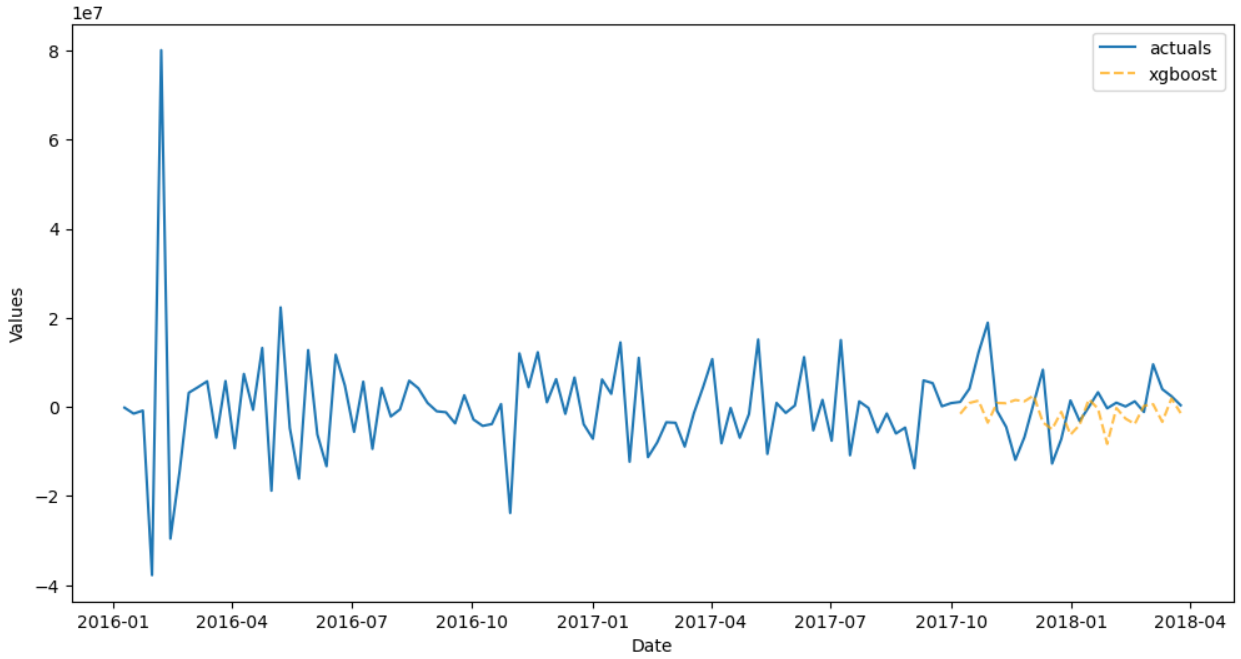
[70]:
```
f_trans.plot()
plt.show()
```



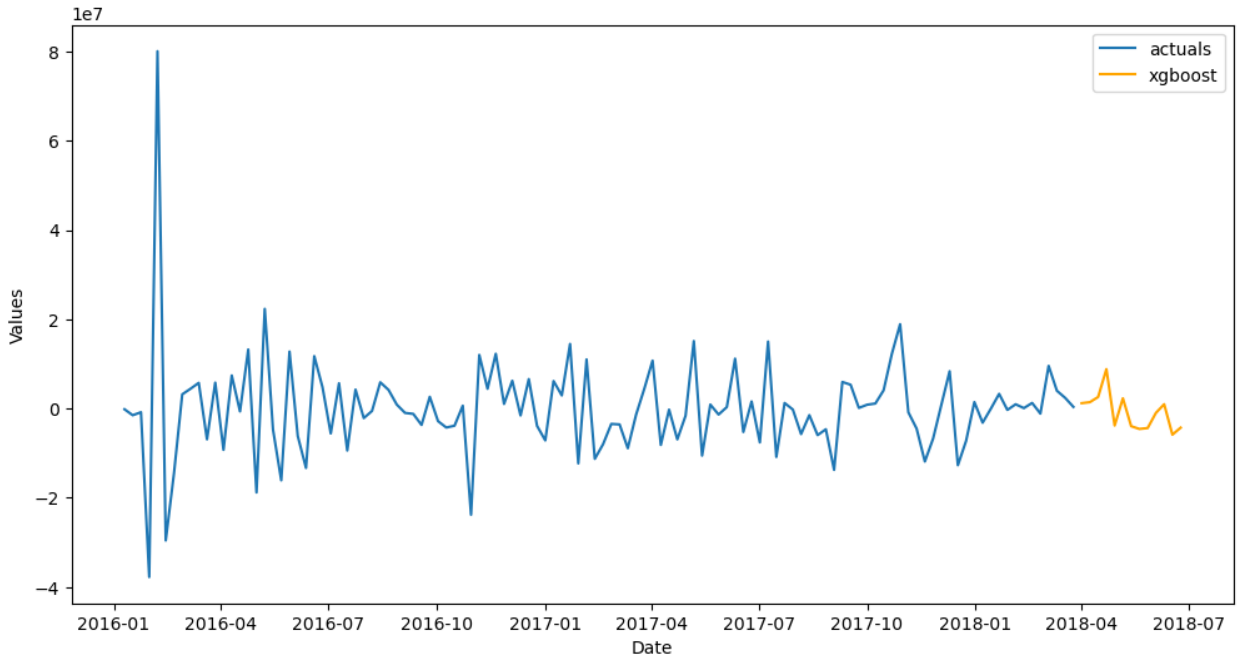
[71]:
```
f_trans.add_time_trend()
f_trans.add_seasonal_regressors('week',sincos=True,raw=False)
f_trans.add_ar_terms(13)
```

[72]:
```
f_trans.set_estimator('xgboost')
f_trans.manual_forecast(gamma=1,dynamic_testing=13)
```

[73]:
```
f_trans.plot_test_set()
plt.show()
```

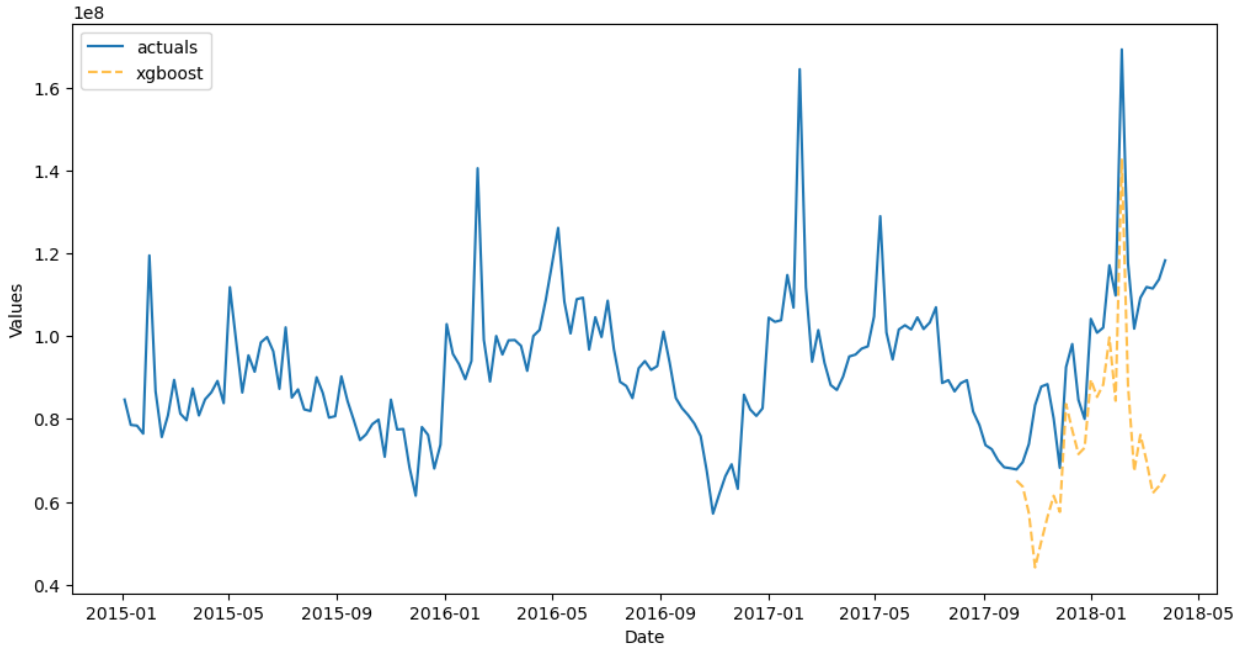


[74]:
```
f_trans.plot()
plt.show()
```

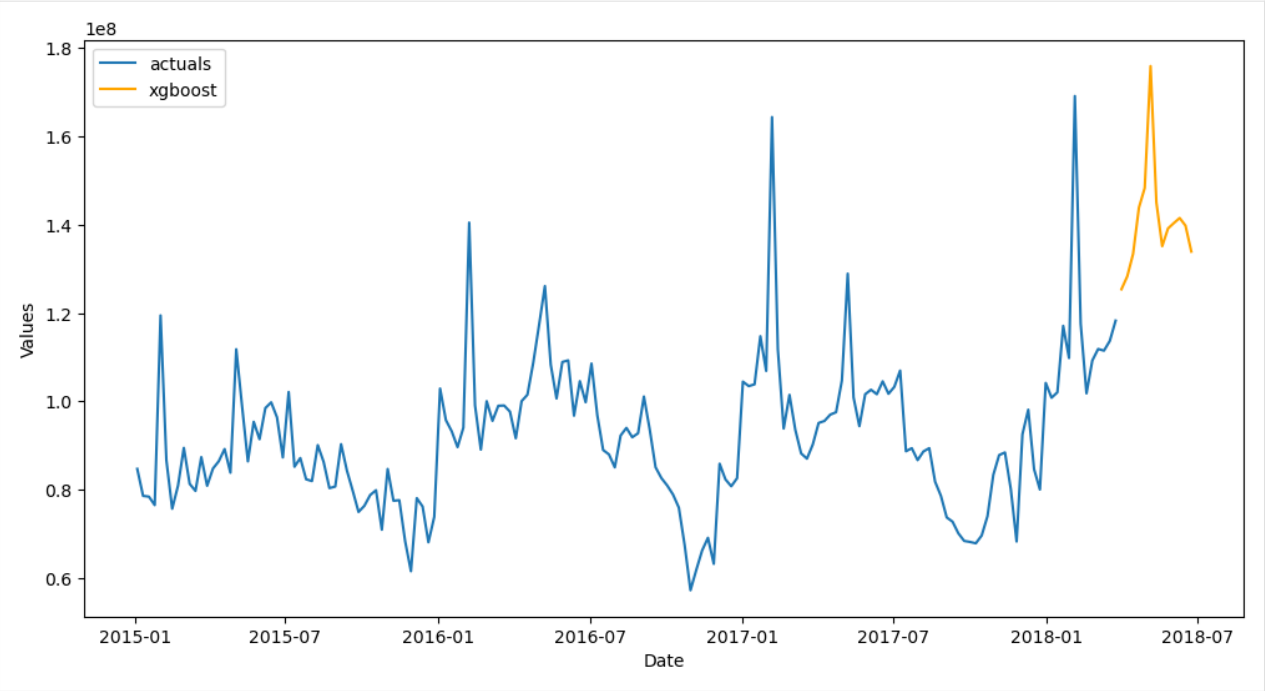


[75]:
```
# call revert functions in the opposite order as how they were called when transforming
f_trans = transformer.DetrendRevert()
f_trans = transformer.DiffRevert(52)
f_trans = transformer.DiffRevert(1)
```

[76]:
```
f_trans.plot_test_set()
plt.show()
```



[77]:
```
f_trans.plot()
plt.show()
```



1.4 Pipelines

- These are objects similar to scikit-learn pipelines that offer readable and streamlined code for transforming, forecasting, and reverting.
- See the Pipeline object documentation.

[78]:
```
from scalecast.Pipeline import Transformer, Reverter, Pipeline, MVPipeline
```

[79]:
```
f_pipe = Forecaster(y=volume,current_dates=volume.index,future_dates=13)
f_pipe.set_test_length(.15)
```

[80]:
```
def forecaster(f):
    f.add_time_trend()
    f.add_seasonal_regressors('week',raw=False,sincos=True)
    f.add_ar_terms(13)
    f.set_estimator('lightgbm')
    f.manual_forecast(max_depth=2)
```

[81]:
```
transformer = Transformer(
    transformers = [
        ('DiffTransform',1),
        ('DiffTransform',52),
        ('DetrendTransform',)
    ]
)

reverter = Reverter(
```

(continues on next page)

(continued from previous page)

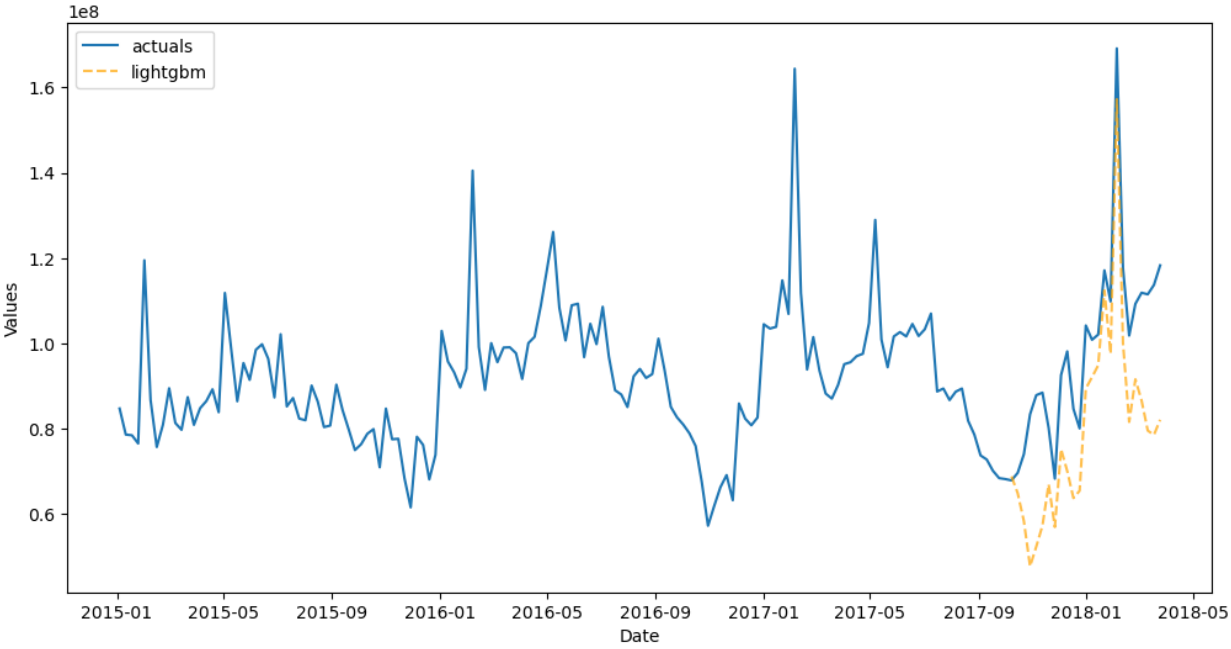
```
    reverters = [
        ('DetrendRevert',),
        ('DiffRevert',52),
        ('DiffRevert',1)
    ],
    base_transformer = transformer,
)
```

[82]:
```
reverter
```

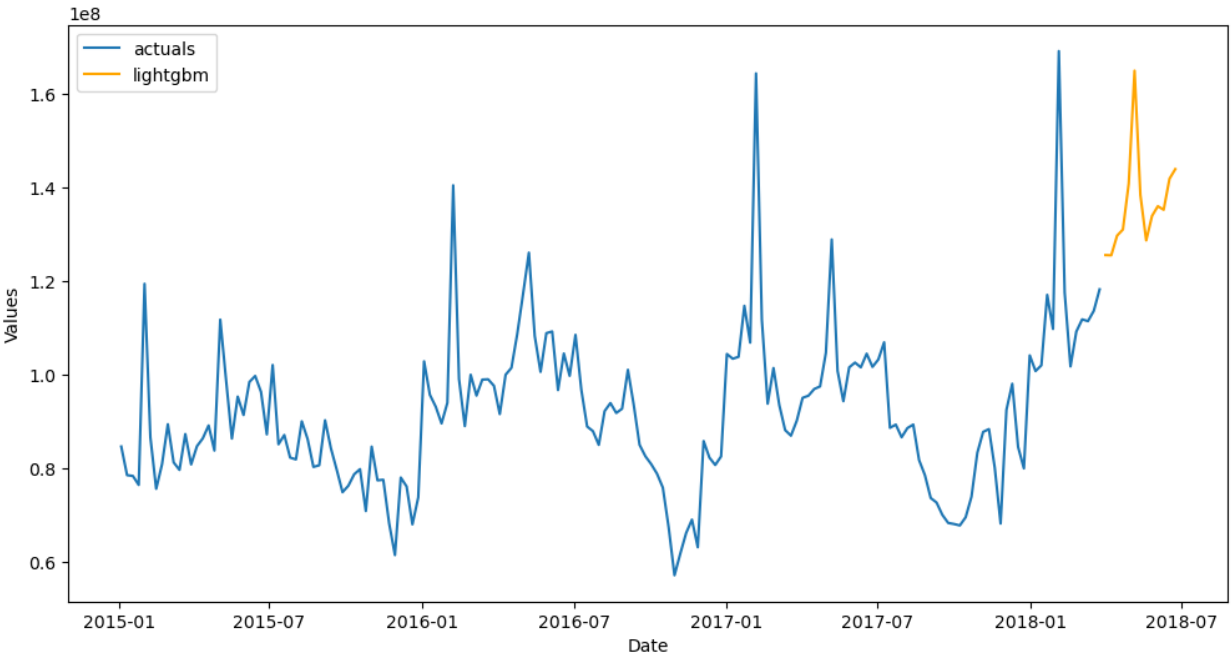
[82]:
```
Reverter(
  reverters = [
    ('DetrendRevert',),
    ('DiffRevert', 52),
    ('DiffRevert', 1)
  ],
  base_transformer = Transformer(
  transformers = [
    ('DiffTransform', 1),
    ('DiffTransform', 52),
    ('DetrendTransform',)
  ]
)
)
```

[83]:
```
pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter),
    ]
)

f_pipe = pipeline.fit_predict(f_pipe)
```

[84]:
```
f_pipe.plot_test_set()
plt.show()
```



[85]:
```
f_pipe.plot()
plt.show()
```



1.5 Fully Automated Pipelines

- We can automate the construction of pipelines, the selection of input variables, and tuning of models with cross validation on a grid search for each model using files in the working directory called `Grids.py` for univariate forecasting and `MVGrids.py` for multivariate. Default grids can be downloaded from scalecast.

1.5.1 Automated Univariate Pipelines

- Documentation

[86]:
```
from scalecast import GridGenerator
from scalecast.util import find_optimal_transformation
```

[87]:
```
GridGenerator.get_example_grids(overwrite=True)
```

[88]:
```
f_pipe_aut = Forecaster(y=volume,current_dates=volume.index,future_dates=13)
f_pipe_aut.set_test_length(.15)
```

[89]:
```
def forecaster_aut(f,models):
    f.auto_Xvar_select(
        estimator='elasticnet',
        monitor='TestSetMAE',
        alpha=0.2,
        irr_cycles = [26],
    )
    f.tune_test_forecast(
        models,
        cross_validate=True,
        k=3,
        # dynamic tuning = 13 means we will hopefully find a model that is optimized to␣
↪predict 13 steps
        dynamic_tuning=13,
        dynamic_testing=13,
    )
    f.set_estimator('combo')
    f.manual_forecast()
```

1.5.2 util.find_optimal_transformation

In this function, the following transformations are searched for:
- Detrending
- Box-Cox
- First Differencing
- Seasonal Differencing
- Scaling

The optimal set of transformations are returned based on best estimated out-of-sample performance on the test set. Therefore, running this function introduces leakage into the test set, but it can still be a good addition to an automated pipeline, depending on the application. Which and the order of transfomations to search through are configurable.

How performance is measured, the parameters specific to a given transformation, and several other paramters are also configurable. See the documentation.

[90]:
```
transformer_aut, reverter_aut = find_optimal_transformation(
    f_pipe_aut,
    lags = 13,
    m = 52,
    monitor = 'mae',
    estimator = 'elasticnet',
    alpha = 0.2,
    test_length = 13,
    num_test_sets = 3,
    space_between_sets = 4,
    verbose = True,
) # returns a Transformer and Reverter object that can be plugged into a larger pipeline
```

```
All transformation tries will use 13 lags.
Last transformer tried:
[]
Score (mae): 17933061.20374991
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (mae): 23964157.165726673
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (mae): 17174376.36667074
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (mae): 24467364.037868027
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪)]
Score (mae): 11573053.425807403
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1)]
Score (mae): 9478522.651025781
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1), ('DiffTransform', 52)]
Score (mae): 11116081.856823219
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1), ('ScaleTransform',)]
Score (mae): 9583504.942193026
--------------------------------------------------
Last transformer tried:
```

(continues on next page)

(continued from previous page)

```
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1), ('MinMaxTransform',)]
Score (mae): 9583504.942193048
--------------------------------------------------
Final Selection:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1)]
```

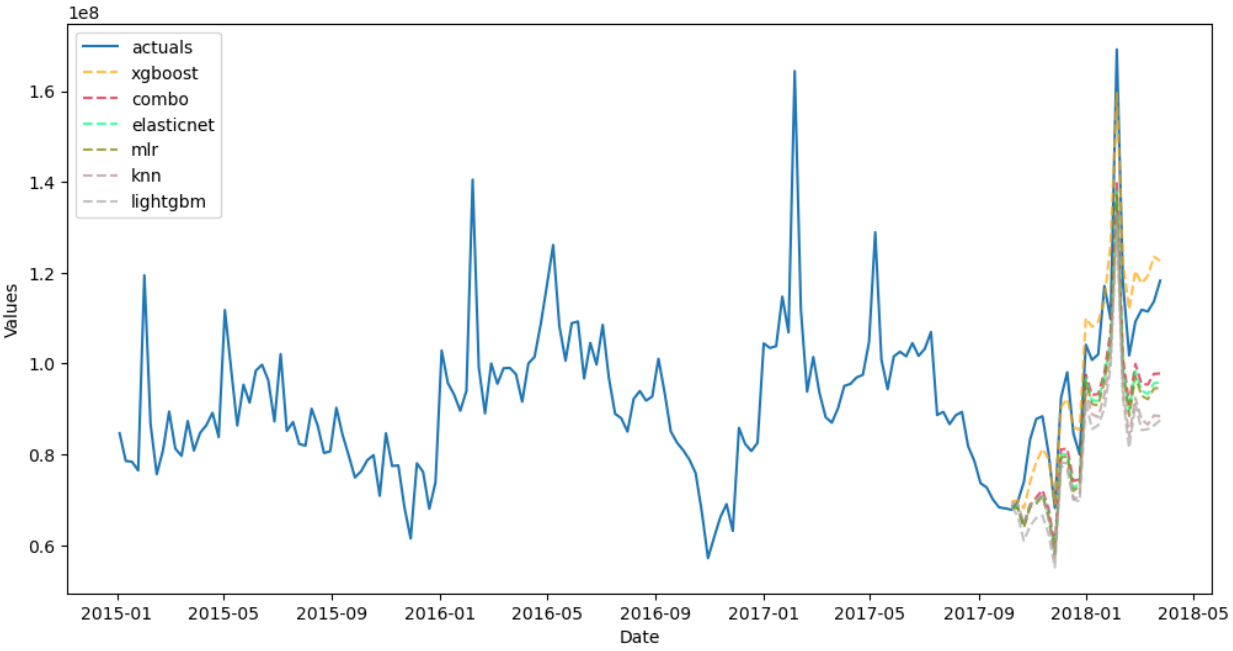
[91]:
```
pipeline_aut = Pipeline(
    steps = [
        ('Transform',transformer_aut),
        ('Forecast',forecaster_aut),
        ('Revert',reverter_aut),
    ]
)

f_pipe_aut = pipeline_aut.fit_predict(
    f_pipe_aut,
    models=[
        'mlr',
        'elasticnet',
        'xgboost',
        'lightgbm',
        'knn',
    ],
)
```

```
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
```

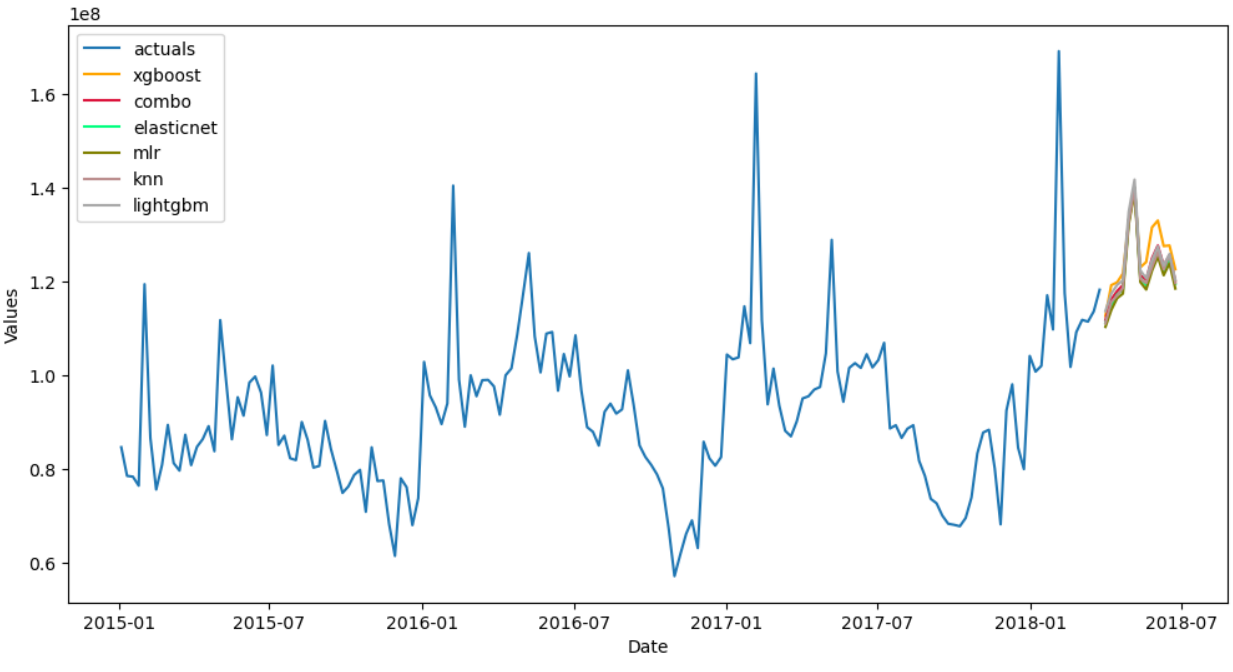
[92]:
```
f_pipe_aut
```

[92]:
```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=13
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5']
    TestLength=25
    ValidationMetric=rmse
    ForecastsEvaluated=['mlr', 'elasticnet', 'xgboost', 'lightgbm', 'knn', 'combo']
    CILevel=None
    CurrentEstimator=combo
    GridsFile=Grids
)
```

[93]:
```
f_pipe_aut.plot_test_set(order_by='TestSetRMSE')
plt.show()
```



[94]:
```
f_pipe_aut.plot(order_by='TestSetRMSE')
plt.show()
```



1.5.3 Backtest Univariate Pipeline

You may be interested to know beyond a single test-set metric, how well your pipeline performs out-of-sample. Backtesting can help answer that by iterating through the entire pipeline several times and testing the procedure each time. It can also help make expanding confidence intervals. See the documentation.

[95]:
```
from scalecast.util import backtest_metrics
```

[96]:
```
uv_backtest_results = pipeline_aut.backtest(
    f_pipe_aut,
    n_iter = 3,
    jump_back = 13,
    cis = False,
    models=[
        'mlr',
        'elasticnet',
        'xgboost',
        'lightgbm',
        'knn',
    ],
)
```

```
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
```

After obtaining the results from the backtest, we can see the average performance over each iteration using the `util.backtest_metrics` function. See the documentation.

[97]:
```
pd.options.display.float_format = '{:,.4f}'.format
backtest_metrics(
    uv_backtest_results,
    mets=['smape','rmse','bias'],
)
```

[97]:

Model	Metric	Iter0	Iter1	Iter2
mlr	smape	0.1130	0.2317	0.1142
	rmse	15,488,744.6687	19,538,227.8483	11,910,709.6300
	bias	-165,115,026.5519	-207,268,257.9313	103,757,419.4097
elasticnet	smape	0.1237	0.1962	0.1428
	rmse	16,732,592.7781	16,625,313.2954	14,338,412.2989

(continues on next page)

(continued from previous page)

```
           bias   -177,765,519.3932 -169,974,287.0231 145,808,728.0805
xgboost    smape             0.0978            0.2221           0.1710
           rmse     13,607,549.8360   18,573,088.5135  18,136,645.4700
           bias   -139,025,946.3310 -187,065,120.1406 180,813,791.1197
lightgbm   smape             0.1248            0.2178           0.1483
           rmse     16,803,405.5145   18,415,746.1087  15,074,267.8093
           bias   -176,880,323.1388 -195,683,280.2443 149,272,223.4129
knn        smape             0.2044            0.1896           0.1439
           rmse     23,802,323.7447   16,441,684.0561  14,418,665.8740
           bias   -278,780,122.9043 -174,089,027.5994 145,204,593.3703
combo      smape             0.1304            0.2109           0.1438
           rmse     17,124,663.5858   17,869,415.4842  14,723,986.4866
           bias   -187,513,387.6638 -186,815,994.5877 144,971,351.0786

                            Average
Model      Metric
mlr        smape             0.1529
           rmse     15,645,894.0490
           bias    -89,541,955.0245
elasticnet smape             0.1542
           rmse     15,898,772.7908
           bias    -67,310,359.4453
xgboost    smape             0.1636
           rmse     16,772,427.9399
           bias    -48,425,758.4506
lightgbm   smape             0.1636
           rmse     16,764,473.1442
           bias    -74,430,459.9901
knn        smape             0.1793
           rmse     18,220,891.2249
           bias   -102,554,852.3778
combo      smape             0.1617
           rmse     16,572,688.5189
           bias    -76,452,677.0576
```

1.5.4 Automated Multivariate Pipelines

- See the MVPipeline object documentation.

[98]:
```
GridGenerator.get_mv_grids(overwrite=True)
```

[99]:
```
fvol_aut = Forecaster(
    y=volume,
    current_dates=volume.index,
    future_dates=13,
    test_length = .15,
)
fprice_aut = Forecaster(
    y=price,
    current_dates=price.index,
```

(continues on next page)

(continued from previous page)

```
    future_dates=13,
    test_length = .15,
)
```

[100]:
```
def add_vars(f,**kwargs):
    f.add_seasonal_regressors(
        'month',
        'quarter',
        'week',
        raw=False,
        sincos=True
    )

def mvforecaster(mvf,models):
    mvf.set_optimize_on('volume')
    mvf.tune_test_forecast(
        models,
        cross_validate=True,
        k=2,
        rolling=True,
        dynamic_tuning=13,
        dynamic_testing=13,
        limit_grid_size = .2,
        error = 'warn',
    )
```

[101]:
```
transformer_vol, reverter_vol = find_optimal_transformation(
    fvol_aut,
    lags = 13,
    m = 52,
    monitor = 'mae',
    estimator = 'elasticnet',
    alpha = 0.2,
    test_length = 13,
    num_test_sets = 3,
    space_between_sets = 4,
    verbose = True,
)
```

```
All transformation tries will use 13 lags.
Last transformer tried:
[]
Score (mae): 17933061.20374991
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (mae): 23964157.165726673
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (mae): 17174376.36667074
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (mae): 24467364.037868027
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪)]
Score (mae): 11573053.425807403
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1)]
Score (mae): 9478522.651025781
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1), ('DiffTransform', 52)]
Score (mae): 11116081.856823219
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1), ('ScaleTransform',)]
Score (mae): 9583504.942193026
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1), ('MinMaxTransform',)]
Score (mae): 9583504.942193048
--------------------------------------------------
Final Selection:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1)]
```

[102]:
```
transformer_price, reverter_price = find_optimal_transformation(
    fprice_aut,
    lags = 13,
    m = 52,
    monitor = 'mae',
    estimator = 'elasticnet',
    alpha = 0.2,
    test_length = 13,
    num_test_sets = 3,
    space_between_sets = 4,
    verbose = True,
)
```

```
All transformation tries will use 13 lags.
Last transformer tried:
[]
Score (mae): 0.06804292152560591
--------------------------------------------------
Last transformer tried:
```

(continues on next page)

(continued from previous page)

```
[('DetrendTransform', {'loess': True})]
Score (mae): 0.35311292233215247
------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (mae): 0.18654572266988656
------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (mae): 0.407907120834863
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'})]
Score (mae): 0.04226554848615107
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', <function find_optimal_
↪transformation.<locals>.boxcox_tr at 0x7fcff088d040>, {'lmbda': -0.5})]
Score (mae): 0.047544210787943963
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', <function find_optimal_
↪transformation.<locals>.boxcox_tr at 0x7fcff088d040>, {'lmbda': 0})]
Score (mae): 0.04573807786929981
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', <function find_optimal_
↪transformation.<locals>.boxcox_tr at 0x7fcff088d040>, {'lmbda': 0.5})]
Score (mae): 0.04392809054109548
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 1)]
Score (mae): 0.08609820028223651
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 52)]
Score (mae): 0.05863628346364504
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('ScaleTransform',)]
Score (mae): 0.038603715602285725
------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('MinMaxTransform',)]
Score (mae): 0.042265548486150994
------------------------------------------------
Final Selection:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('ScaleTransform',)]
```

[103]:
```
mvpipeline = MVPipeline(
    steps = [
        ('Transform',[transformer_vol,transformer_price]),
```

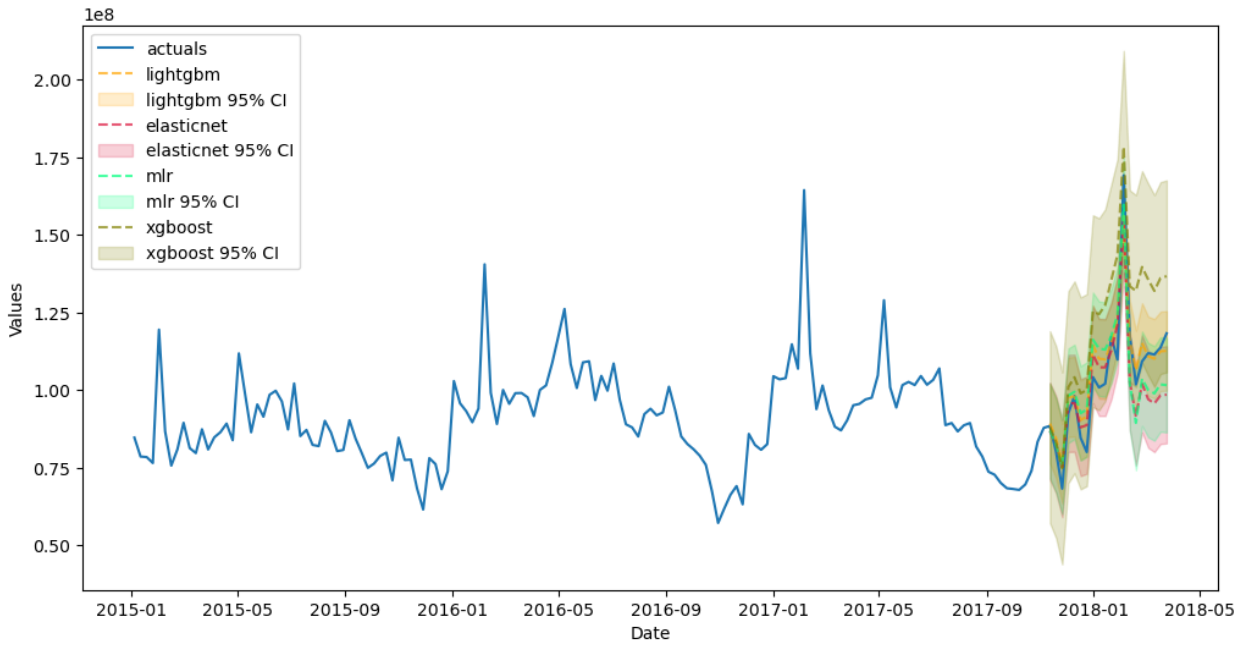
(continues on next page)

(continued from previous page)

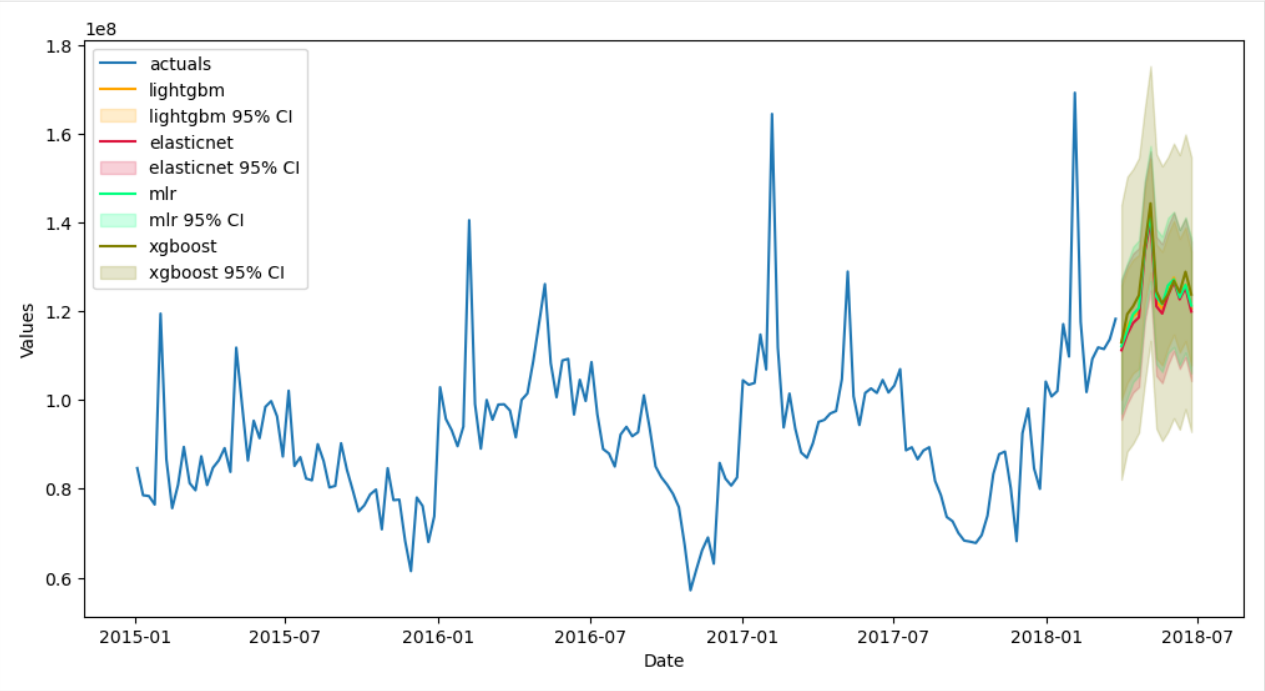
```
        ('Add Xvars',[add_vars]*2),
        ('Forecast',mvforecaster),
        ('Revert',[reverter_vol,reverter_price]),
    ],
    test_length = 20,
    cis = True,
    names = ['volume','price'],
)

fvol_aut, fprice_aut = mvpipeline.fit_predict(
    fvol_aut,
    fprice_aut,
    models=[
        'mlr',
        'elasticnet',
        'xgboost',
        'lightgbm',
    ],
) # returns a tuple of Forecaster objects
```

```
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
```

[104]:
```
fvol_aut.plot_test_set(order_by='TestSetRMSE',ci=True)
plt.show()
```



[105]:
```
fvol_aut.plot(order_by='TestSetRMSE',ci=True)
plt.show()
```



1.5.5 Backtest Multivariate Pipeline

Like univariate pipelines, multivariate pipelines can also be backtested. Info about each model and series becomes possible to compare. See the documentation.

[106]:
```
# recreate Forecaster objects to bring dates back lost from taking seasonal differences
fvol_aut = Forecaster(
    y=volume,
    current_dates=volume.index,
    future_dates=13,
)
fprice_aut = Forecaster(
    y=price,
    current_dates=price.index,
    future_dates=13,
)
```

[107]:
```
mv_backtest_results = mvpipeline.backtest(
    fvol_aut,
    fprice_aut,
    n_iter = 3,
    jump_back = 13,
    test_length = 0,
    cis = False,
    models=[
        'mlr',
        'elasticnet',
        'xgboost',
        'lightgbm',
```

(continues on next page)

(continued from previous page)

```
    ],
)
```

```
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
```

[108]:
```
backtest_metrics(
    mv_backtest_results,
    mets=['smape','rmse','bias'],
    names = ['Volume','Price'],
)
```

[108]:

			Iter0	Iter1	Iter2	\
Series	Model	Metric				
Volume	mlr	smape	0.0544	0.1176	0.1582	
		rmse	9,607,094.0997	10,760,756.5379	15,963,833.7533	
		bias	-69,742,732.2916	-87,711,912.6597	157,394,405.8767	
	elasticnet	smape	0.0662	0.1606	0.1398	
		rmse	10,759,298.7685	14,132,016.6443	14,134,913.2323	
		bias	-82,997,021.2184	-139,537,507.0051	140,719,790.4827	
	xgboost	smape	0.1315	0.1447	0.0713	
		rmse	20,090,059.9856	13,817,186.7925	7,510,114.4244	
		bias	189,709,643.4190	-129,669,530.7265	71,329,272.6822	
	lightgbm	smape	0.1278	0.1161	0.1720	
		rmse	16,974,627.6371	10,539,060.5819	17,427,536.4398	
		bias	-182,115,273.5654	-76,693,584.9027	184,015,627.3148	
Price	mlr	smape	0.0523	0.0762	0.0808	
		rmse	0.0811	0.1307	0.1782	
		bias	0.9321	1.0663	-1.5867	
	elasticnet	smape	0.0287	0.0606	0.1631	
		rmse	0.0461	0.1043	0.2838	
		bias	-0.1421	0.5670	-3.3627	
	xgboost	smape	0.0656	0.0513	0.1053	
		rmse	0.1000	0.1024	0.2100	
		bias	1.1779	0.1599	-2.2494	
	lightgbm	smape	0.0287	0.0630	0.1588	
		rmse	0.0459	0.1215	0.2799	
		bias	-0.2103	-0.1697	-3.2847	

			Average
Series	Model	Metric	
Volume	mlr	smape	0.1101
		rmse	12,110,561.4636
		bias	-20,079.6915
	elasticnet	smape	0.1222
		rmse	13,008,742.8817
		bias	-27,271,579.2469
	xgboost	smape	0.1158
		rmse	13,805,787.0675

(continues on next page)

(continued from previous page)

```
                  bias    43,789,795.1249
       lightgbm   smape             0.1386
                  rmse    14,980,408.2196
                  bias   -24,931,077.0511
Price  mlr        smape             0.0698
                  rmse              0.1300
                  bias              0.1372
       elasticnet smape             0.0841
                  rmse              0.1447
                  bias             -0.9793
       xgboost    smape             0.0740
                  rmse              0.1375
                  bias             -0.3038
       lightgbm   smape             0.0835
                  rmse              0.1491
                  bias             -1.2215
```

Through backtesting, we can see that the multivariate approach out-performed the univariate approach. Very cool!

1.6 Scaled Automated Forecasting

- We can scale the fully automated approach to many series where we can then access all results through plotting with Jupyter widgets and export functions.
- We produce a separate forecast for avocado sales in each region in our dataset.
- This is done with a univariate approach, but cleverly using the code in this notebook, it could be transformed into a multivariate process where volume and price are forecasted together.

[109]:
```
from scalecast.notebook import results_vis
from tqdm.notebook import tqdm
```

[110]:
```
def forecaster_scaled(f,models):
    f.auto_Xvar_select(
        estimator='elasticnet',
        monitor='TestSetMAE',
        alpha=0.2,
        irr_cycles = [26],
    )
    f.tune_test_forecast(
        models,
        dynamic_testing=13,
    )
    f.set_estimator('combo')
    f.manual_forecast()
```

[111]:
```
results_dict = {}
for region in tqdm(data.region.unique()):
    series = data.loc[data['region'] == region].groupby('Date')['Total Volume'].sum()
    f_i = Forecaster(
        y = series,
```

(continues on next page)

(continued from previous page)

```
        current_dates = series.index,
        future_dates = 13,
        test_length = .15,
        validation_length = 13,
        cis = True,
    )
    transformer_i, reverter_i = find_optimal_transformation(
        f_i,
        lags = 13,
        m = 52,
        monitor = 'mae',
        estimator = 'elasticnet',
        alpha = 0.2,
        test_length = 13,
    )
    pipeline_i = Pipeline(
        steps = [
            ('Transform',transformer_i),
            ('Forecast',forecaster_scaled),
            ('Revert',reverter_i),
        ]
    )
    f_i = pipeline_i.fit_predict(
        f_i,
        models=[
            'mlr',
            'elasticnet',
            'xgboost',
            'lightgbm',
            'knn',
        ],
    )
    results_dict[region] = f_i
```

```
  0%|          | 0/54 [00:00<?, ?it/s]
```

```
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
```

(continues on next page)

(continued from previous page)

```
Finished loading model, total used 200 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
```

(continues on next page)

(continued from previous page)

```
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 10 iterations
Finished loading model, total used 10 iterations
Finished loading model, total used 10 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 100 iterations
Finished loading model, total used 100 iterations
Finished loading model, total used 100 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
```

(continues on next page)

(continued from previous page)

```
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
```

Run the next two functions locally to see the full functionality of these widgets.

[112]:
```
results_vis(results_dict,'test')
```

```
Dropdown(description='Time Series:', options=('Albany', 'Atlanta', 'BaltimoreWashington',
↪ 'Boise', 'Boston', '...
```

```
Dropdown(description='No. Models', options=(1, 2, 3, 4, 5, 6), value=1)
```

```
Dropdown(description='View Confidence Intervals', options=(False, True), value=False)
```

```
Dropdown(description='Order By', index=7, options=('InSampleMAE', 'InSampleMAPE',
↪'InSampleR2', 'InSampleRMSE'...
```

```
Button(description='Select Time Series', style=ButtonStyle())
```

```
Output()
```

[113]:
```
results_vis(results_dict)
```

```
Dropdown(description='Time Series:', options=('Albany', 'Atlanta', 'BaltimoreWashington',
↪ 'Boise', 'Boston', '...
```

```
Dropdown(description='No. Models', options=(1, 2, 3, 4, 5, 6), value=1)
```

```
Dropdown(description='View Confidence Intervals', options=(False, True), value=False)
```

```
Dropdown(description='Order By', index=7, options=('InSampleMAE', 'InSampleMAPE',
↪'InSampleR2', 'InSampleRMSE'...
```

```
Button(description='Select Time Series', style=ButtonStyle())
```

```
Output()
```

1.7 Exporting Results

[114]:
```
from scalecast.multiseries import export_model_summaries
```

1.7.1 Exporting Results from a Single `Forecaster` Object

[115]:
```
results = f.export(cis=True,models=['mlr','lasso','ridge'])
results.keys()
```

[115]:
```
dict_keys(['model_summaries', 'lvl_fcsts', 'lvl_test_set_predictions'])
```

[116]:
```
for k, df in results.items():
    print(f'{k} has these columns:',*df.columns,'-'*25,sep='\n')
```

```
model_summaries has these columns:
ModelNickname
Estimator
Xvars
HyperParams
Observations
DynamicallyTested
TestSetLength
CILevel
ValidationMetric
ValidationMetricValue
models
weights
best_model
InSampleRMSE
InSampleMAPE
InSampleMAE
InSampleR2
```

(continues on next page)

(continued from previous page)

```
TestSetRMSE
TestSetMAPE
TestSetMAE
TestSetR2
-------------------------
lvl_fcsts has these columns:
DATE
mlr
mlr_upperci
mlr_lowerci
lasso
lasso_upperci
lasso_lowerci
ridge
ridge_upperci
ridge_lowerci
-------------------------
lvl_test_set_predictions has these columns:
DATE
actual
mlr
mlr_upperci
mlr_lowerci
lasso
lasso_upperci
lasso_lowerci
ridge
ridge_upperci
ridge_lowerci
-------------------------
```

[117]:
```
results['model_summaries'][['ModelNickname','HyperParams','TestSetRMSE','InSampleRMSE']]
```

[117]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	mlr	{}	16,818,489.5277	10,231,495.9303
1	lasso	{'alpha': 0.2}	16,818,489.9828	10,231,495.9303
2	ridge	{'alpha': 0.2}	16,827,165.4875	10,265,352.0145

Other export functions:
Forecaster.export_Xvars_df
Forecaster.export_feature_importance
Forecaster.export_fitted_vals
Forecaster.export_summary_stats
Forecaster.export_validation_grid

Other plotting functions:
Forecaster.plot_fitted
Forecaster.plot_periodogram

1.7.2 Exporting Results from a Single `MVForecaster` Object

[118]:
```
mvresults = mvf.export(cis=True,models=['elasticnet','xgboost'])
mvresults.keys()
```

[118]:
```
dict_keys(['model_summaries', 'lvl_fcsts', 'lvl_test_set_predictions'])
```

[119]:
```
for k, df in mvresults.items():
    print(f'{k} has these columns:',*df.columns,'-'*25,sep='\n')
```

```
model_summaries has these columns:
Series
ModelNickname
Estimator
Xvars
HyperParams
Lags
Observations
DynamicallyTested
TestSetLength
ValidationMetric
ValidationMetricValue
OptimizedOn
MetricOptimized
best_model
InSampleRMSE
InSampleMAPE
InSampleMAE
InSampleR2
TestSetRMSE
TestSetMAPE
TestSetMAE
TestSetR2
-------------------------
lvl_fcsts has these columns:
DATE
volume_elasticnet_lvl_fcst
volume_elasticnet_lvl_fcst_upper
volume_elasticnet_lvl_fcst_lower
volume_xgboost_lvl_fcst
volume_xgboost_lvl_fcst_upper
volume_xgboost_lvl_fcst_lower
price_elasticnet_lvl_fcst
price_elasticnet_lvl_fcst_upper
price_elasticnet_lvl_fcst_lower
price_xgboost_lvl_fcst
price_xgboost_lvl_fcst_upper
price_xgboost_lvl_fcst_lower
-------------------------
lvl_test_set_predictions has these columns:
DATE
volume_actuals
volume_elasticnet_lvl_ts
volume_elasticnet_lvl_ts_upper
```

(continues on next page)

(continued from previous page)

```
volume_elasticnet_lvl_ts_lower
volume_xgboost_lvl_ts
volume_xgboost_lvl_ts_upper
volume_xgboost_lvl_ts_lower
price_actuals
price_elasticnet_lvl_ts
price_elasticnet_lvl_ts_upper
price_elasticnet_lvl_ts_lower
price_xgboost_lvl_ts
price_xgboost_lvl_ts_upper
price_xgboost_lvl_ts_lower
-------------------------
```

[120]:
```
mvresults['model_summaries'][['Series','ModelNickname','HyperParams','Lags','TestSetRMSE
↪','InSampleRMSE']]
```

[120]:

	Series	ModelNickname	HyperParams	Lags	TestSetRMSE	InSampleRMSE
0	volume	elasticnet	{'alpha': 0.2}	13	18,694,896.5159	11,976,031.0554
1	volume	xgboost	{'gamma': 1}	13	22,606,056.1177	341.6022
2	price	elasticnet	{'alpha': 0.2}	13	0.1522	0.1561
3	price	xgboost	{'gamma': 1}	13	0.1292	0.1141

Other export functions:
Forecaster.export_fitted_vals
Forecaster.export_validation_grid

Other plotting functions:
MVForecaster.plot_fitted

1.7.3 Exporting Results from a Dictionary of `Forecaster` Objects

[121]:
```
all_results = export_model_summaries(results_dict)
all_results[['ModelNickname','Series','Xvars','HyperParams','TestSetRMSE','InSampleRMSE
↪']].sample(10)
```

[121]:

	ModelNickname	Series	\
178	knn	Northeast	
17	combo	BaltimoreWashington	
56	xgboost	CincinnatiDayton	
309	lightgbm	TotalUS	
115	elasticnet	Indianapolis	
132	mlr	LosAngeles	
23	combo	Boise	
141	lightgbm	Louisville	
290	xgboost	StLouis	
112	knn	Houston	

	Xvars	\

(continues on next page)

(continued from previous page)

```
178  [AR1, AR2, AR3, AR4, AR5, AR6, AR7, AR8, AR9, ...
17                                                 None
56   [AR1, AR2, AR3, AR4, AR5, AR6, AR7, AR8, AR9, ...
309  [weeksin, weekcos, monthsin, monthcos, AR1, AR...
115  [t, AR1, AR2, AR3, AR4, AR5, AR6, AR7, AR8, AR...
132  [weeksin, weekcos, monthsin, monthcos, quarter...
23                                                 None
141  [weeksin, weekcos, monthsin, monthcos, AR1, AR...
290  [AR1, AR2, AR3, AR4, AR5, AR6, AR7, AR8, AR9, ...
112  [lnt, weeksin, weekcos, monthsin, monthcos, AR...

                                        HyperParams    TestSetRMSE  \
178                              {'n_neighbors': 54}   724,041.5710
17                                                {}   154,661.0027
56   {'n_estimators': 200, 'scale_pos_weight': 5, '...    29,716.4527
309  {'n_estimators': 250, 'boosting_type': 'dart',... 8,587,458.9970
115  {'alpha': 2.0, 'l1_ratio': 0, 'normalizer': 'm...    61,612.3263
132                           {'normalizer': 'scale'}   538,249.2891
23                                                {}    12,828.5334
141  {'n_estimators': 150, 'boosting_type': 'dart',...    17,403.8036
290  {'n_estimators': 250, 'scale_pos_weight': 5, '...    44,612.6516
112                              {'n_neighbors': 69}   298,081.3679

       InSampleRMSE
178    733,517.0246
17     140,291.4529
56       7,782.8633
309 15,516,549.5493
115     37,420.2720
132    450,855.7099
23       8,090.6574
141     22,637.4852
290         87.0324
112    250,425.0721
```

[ ]:

CHAPTER

TWO

ANOMALY DETECTION

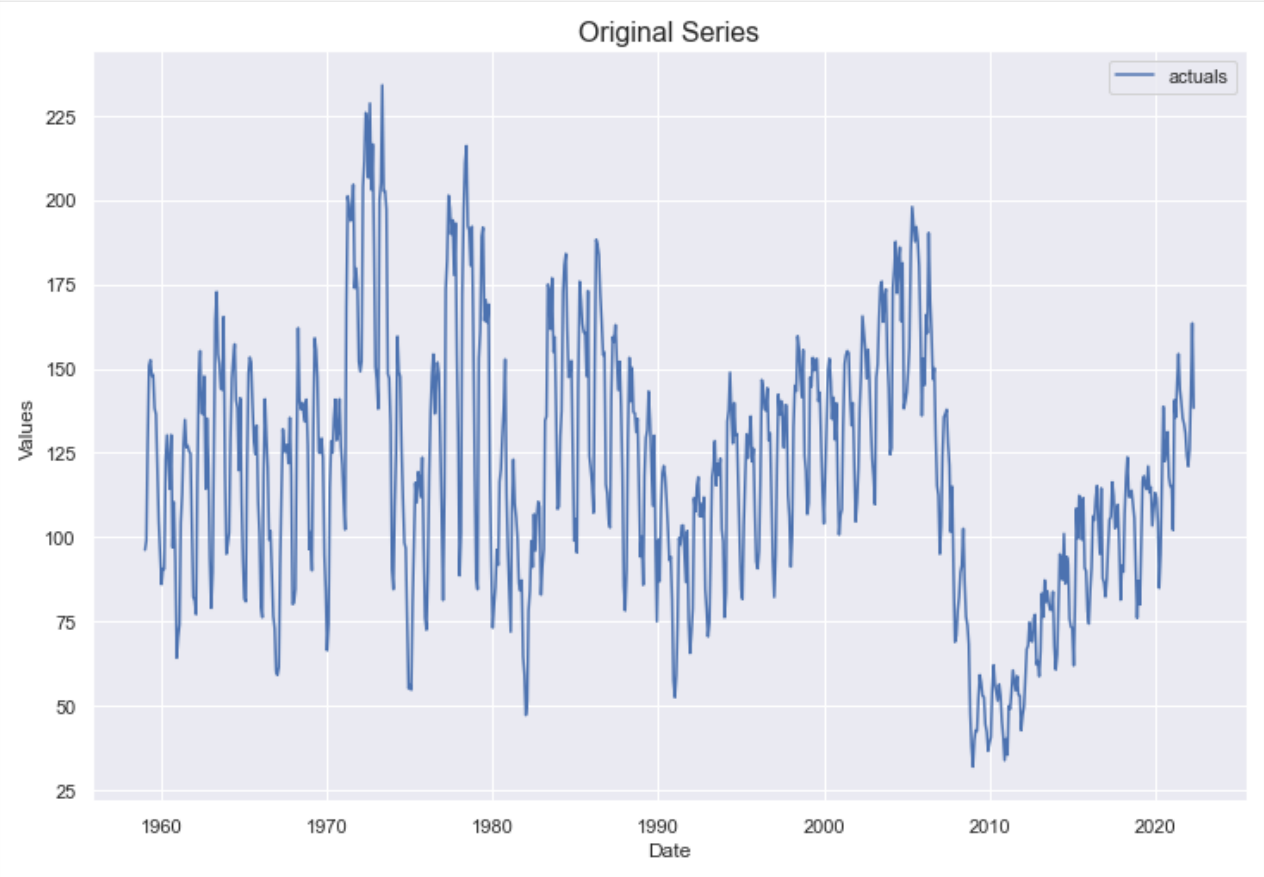
The issue with anomaly detection in time series is that an anomaly is not always well-defined. This notebook introduces several techniques native to scalecast that can be used to identify anomalies, but it is ultimately up to the user to determine what is and isn't actually an anomaly.

[1]:
```
import pandas as pd
import pandas_datareader as pdr
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scalecast.Forecaster import Forecaster
from scalecast.AnomalyDetector import AnomalyDetector
from scalecast.SeriesTransformer import SeriesTransformer
```

[2]:
```
sns.set(rc={'figure.figsize':(12,8)})
```

[3]:
```
df = pdr.get_data_fred(
    'HOUSTNSA',
    start='1959-01-01',
    end='2022-05-01',
).reset_index()
```

[4]:
```
f = Forecaster(
    y=df['HOUSTNSA'],
    current_dates=df['DATE']
)

f.plot()
plt.title('Original Series',size=16)
plt.show()
```

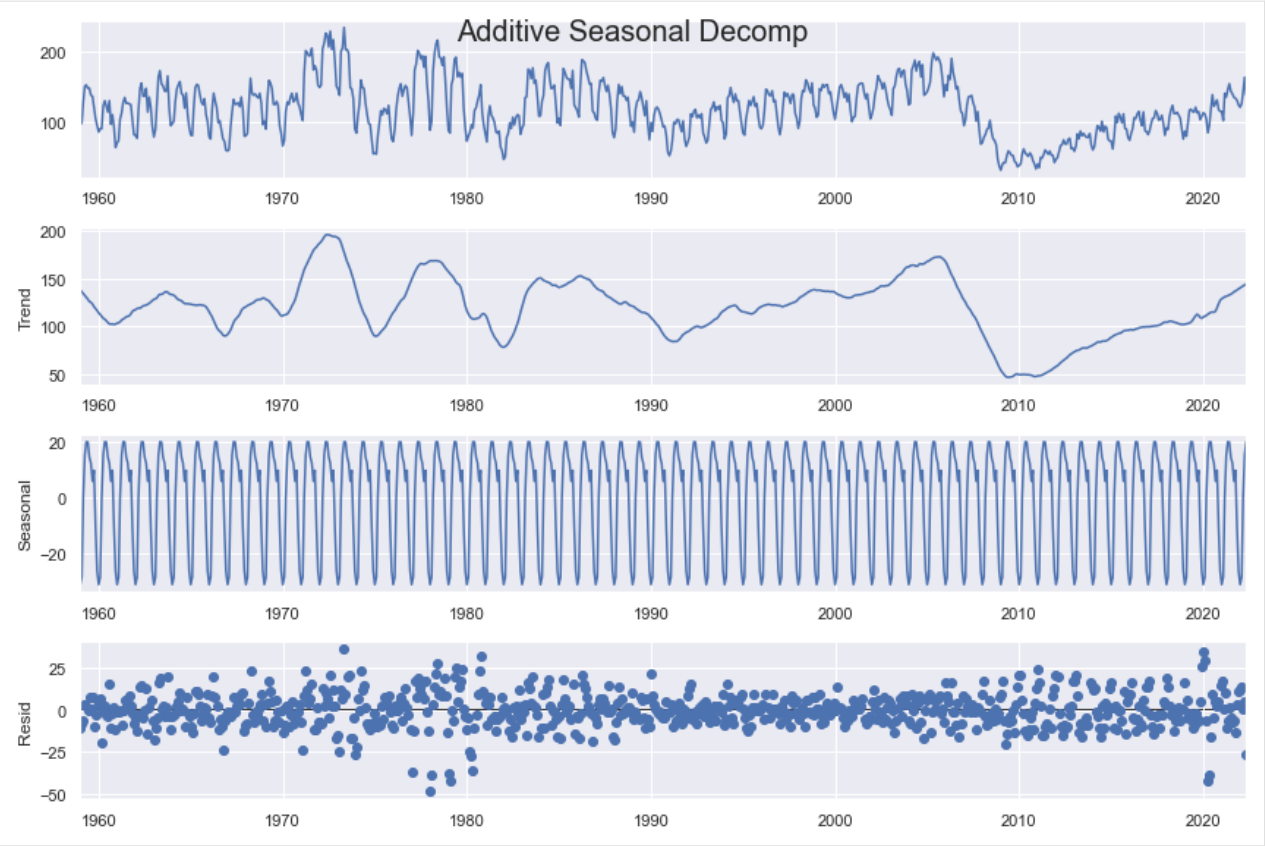


2.1 Naive Detect

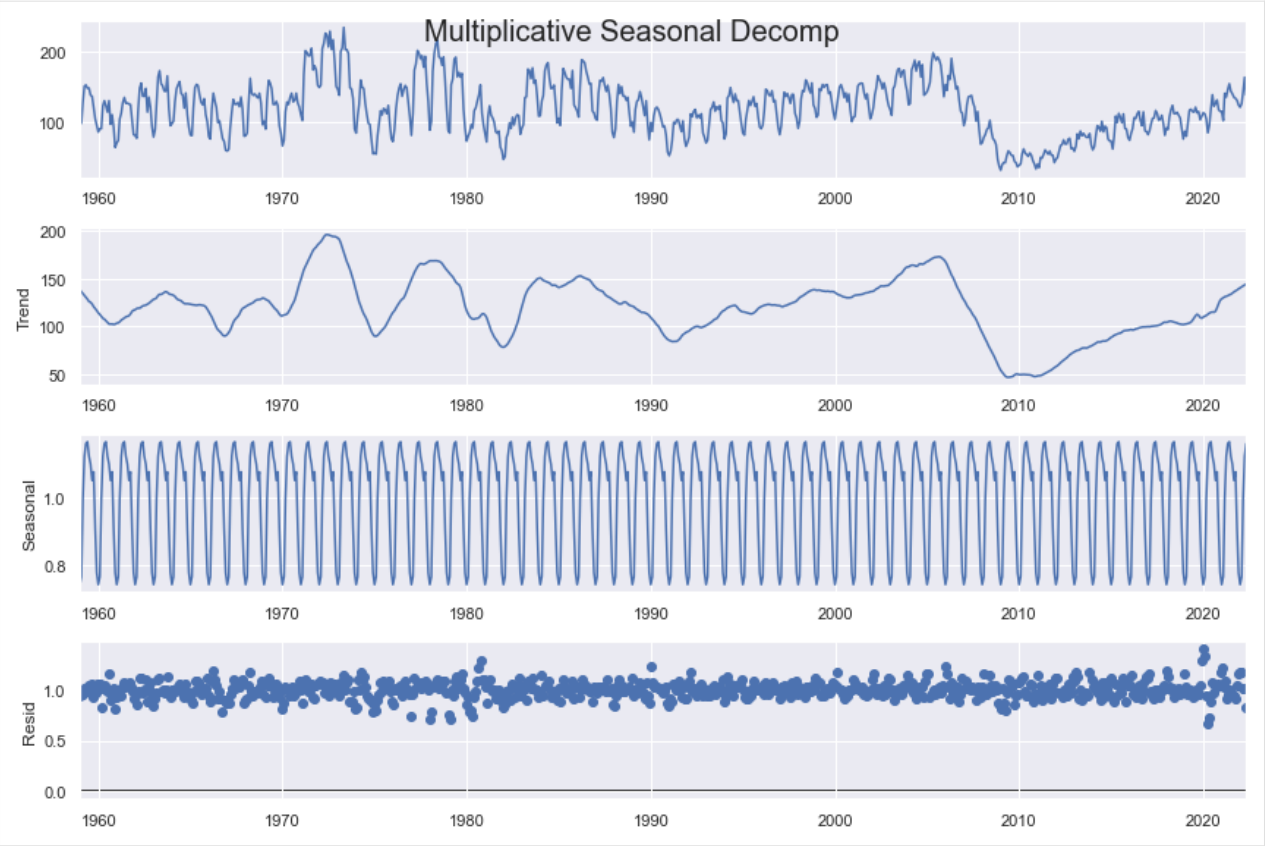
This algorithm functions by decomposing a given series into a trend, seasonal, and residual part testing the residuals' distance from the mean to identify anomalies. It is computationally cheap, but if the series isn't well-explained by the decomposition process, or the residual is not independent and/or not normal, this can be a poor way to identify anomalies.

[5]:
```
detector1 = AnomalyDetector(f)
```

[6]:
```
f.seasonal_decompose(extrapolate_trend='freq').plot()
plt.suptitle('Additive Seasonal Decomp',size=20)
plt.show()
```

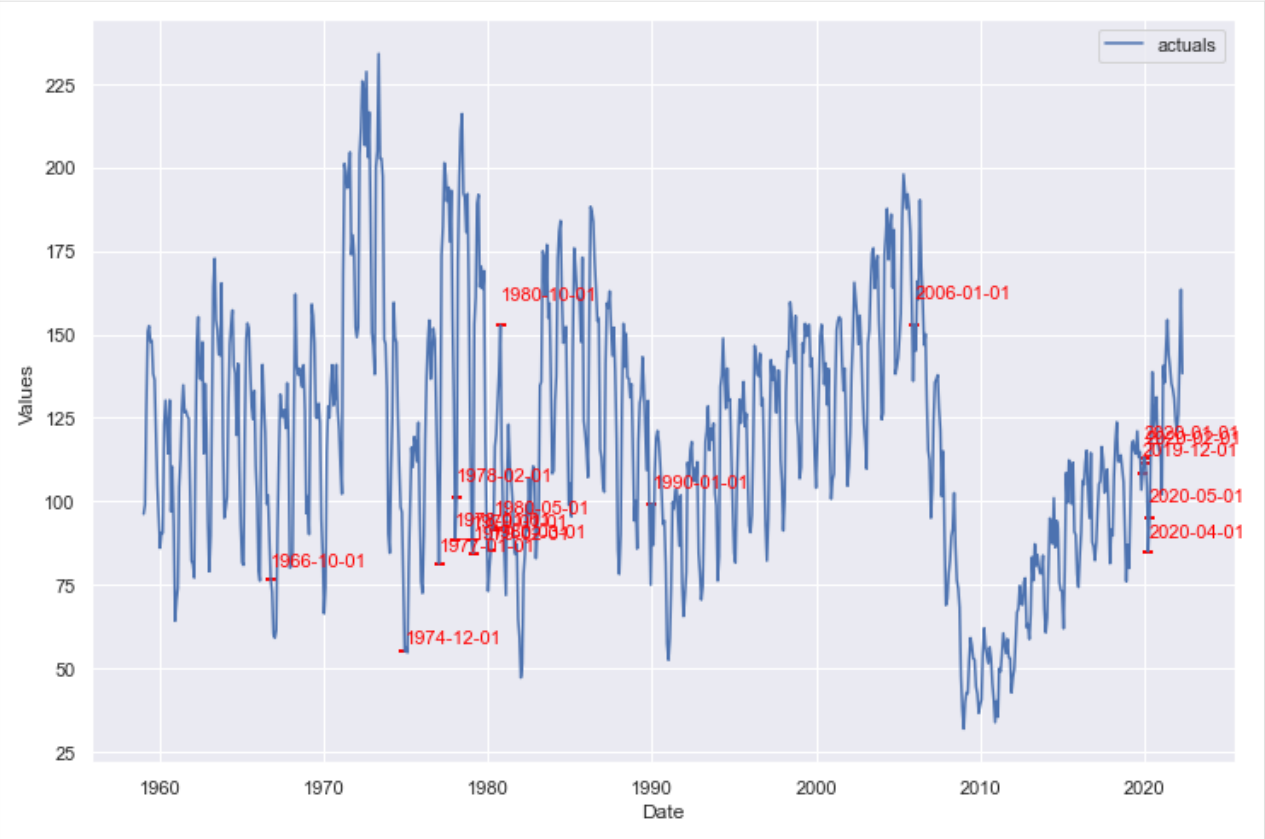


[7]:
```
f.seasonal_decompose(
    model='multiplicative',
    extrapolate_trend='freq'
).plot()
plt.suptitle('Multiplicative Seasonal Decomp',size=20)
plt.show()
```



The key parameters in the function below include the type of decomposition (additive or multiplicative), as well as the confidence level. The higher the confidence level, the fewer anomalies will be detected. The type of decomposition should conform with whichever one creates the most ideal residuals (should be normally abnd randomly distributed, independent, and homoskedastic).

[8]:
```
detector1.NaiveDetect(
    extrapolate_trend='freq',
    model='multiplicative',
    cilevel=.99,
)
detector1.plot_anom(label=True)
plt.show()
```



This method identified many anomalies and perhaps isn't the best method for this series, which exhibits irregular cycles and trends.

2.2 Monte Carlo Detect

This is a Monte Carlo simulated method that produces possible future paths for the data over some time span and uses a confidence interval to evaluate what percent of the time the actual data fell above or below the simulated paths. To make this method work properly, we should transform the distribution so that we can make it as stationary and normally distributed as possible. Therefore, extra care will be taken to prepare the data before applying the method.

To get started, we can transform the dataset so that a log transformation is taken to normalize some of the skew in the dataset, then a seasonal difference of 12 months is applied on the data, then the first difference of the resulting series is applied.

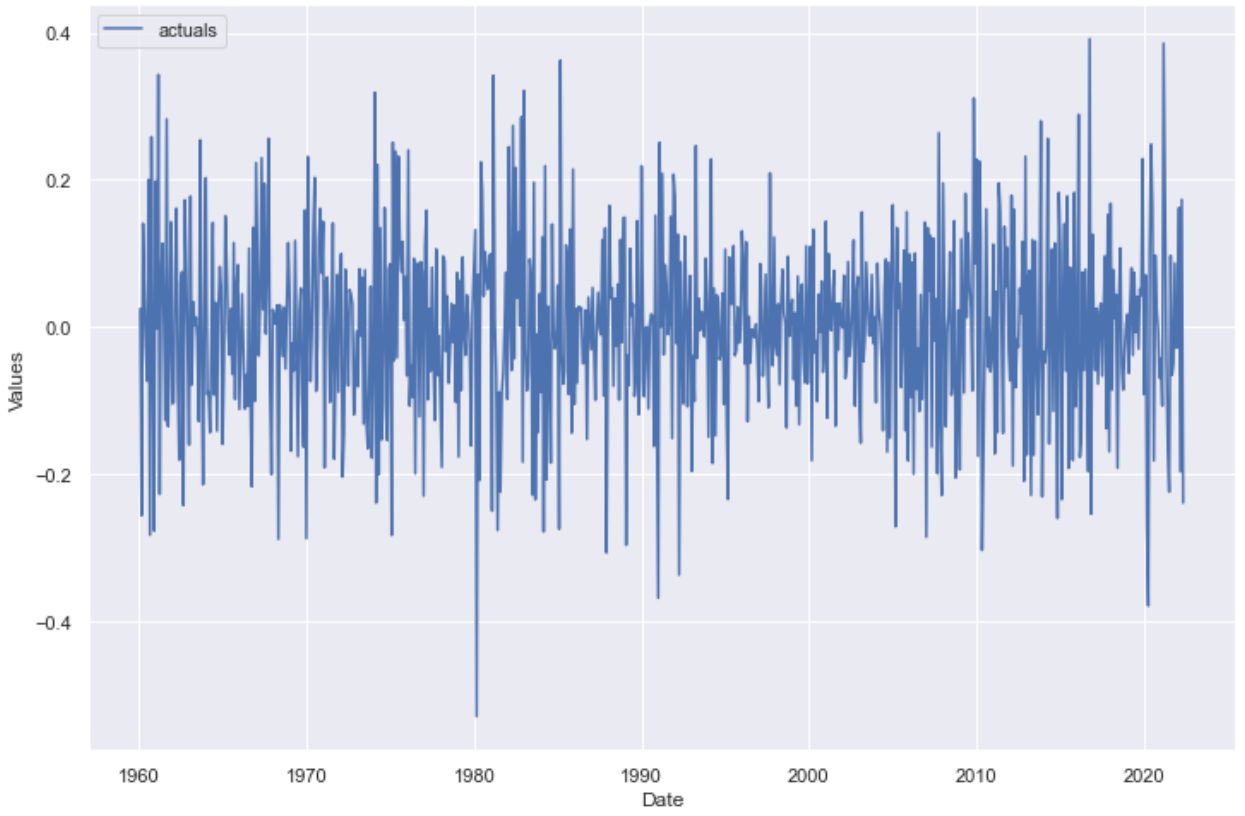
[9]:
```
f_transform = f.copy()

transformer = SeriesTransformer(f_transform)

f_transform = transformer.LogTransform()      # log transform
f_transform = transformer.DiffTransform(12)   # seasonally adjust
f_transform = transformer.DiffTransform(1)    # first difference

detector2 = AnomalyDetector(f_transform)
```

All of these transformations, as well as any models run after the transformations were called, are revertible by calling their reversion counterpart methods in revserse order, demonstrated (by not applied) by the code below:

```
# how to revert all transformations
f_transform = transformer.DiffRevert(1)
f_transform = transformer.DiffRevert(12)
f_transform = transformer.LogRevert()
```

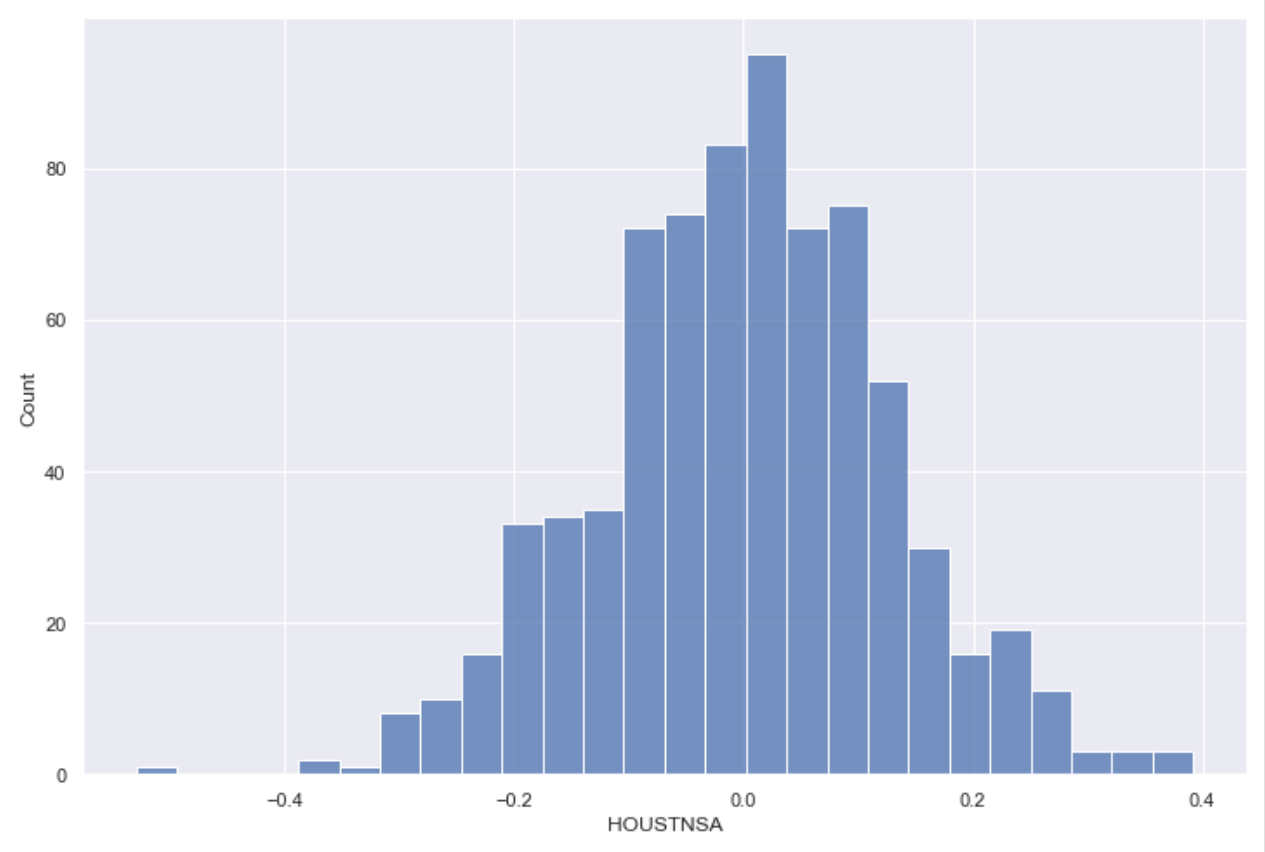
Now, we can see the resulting series:

[10]:
```
f_transform.plot()
plt.show()
```



Call a histogram to make sure it looks normal:

[11]:
```
sns.histplot(f_transform.y)
plt.show()
```



Call a statistical test to ensure its normality:

[12]:
```
critical_pval = 0.05
print('-'*100)
k2, p = stats.normaltest(f_transform.y.values)
print("D'Agostino and Pearson's test result for normality:")
print('the test-stat value is: {:.2f}'.format(k2))
print('the p-value is {:.4f}'.format(p))
print('the series is {}'.format('not normal' if p < critical_pval else 'normal'))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
↪-----------
D'Agostino and Pearson's test result for normality:
the test-stat value is: 4.55
the p-value is 0.1028
the series is normal
----------------------------------------------------------------------------------------------------
↪-----------
```

Call a statistical test to ensure its stationarity:

[13]:
```
print('-'*100)
print('Augmented Dickey-Fuller results:')
stat, pval, _, _, _, _ = f_transform.adf_test(full_res=True,maxlag=12)
print('the test-stat value is: {:.2f}'.format(stat))
```

(continues on next page)

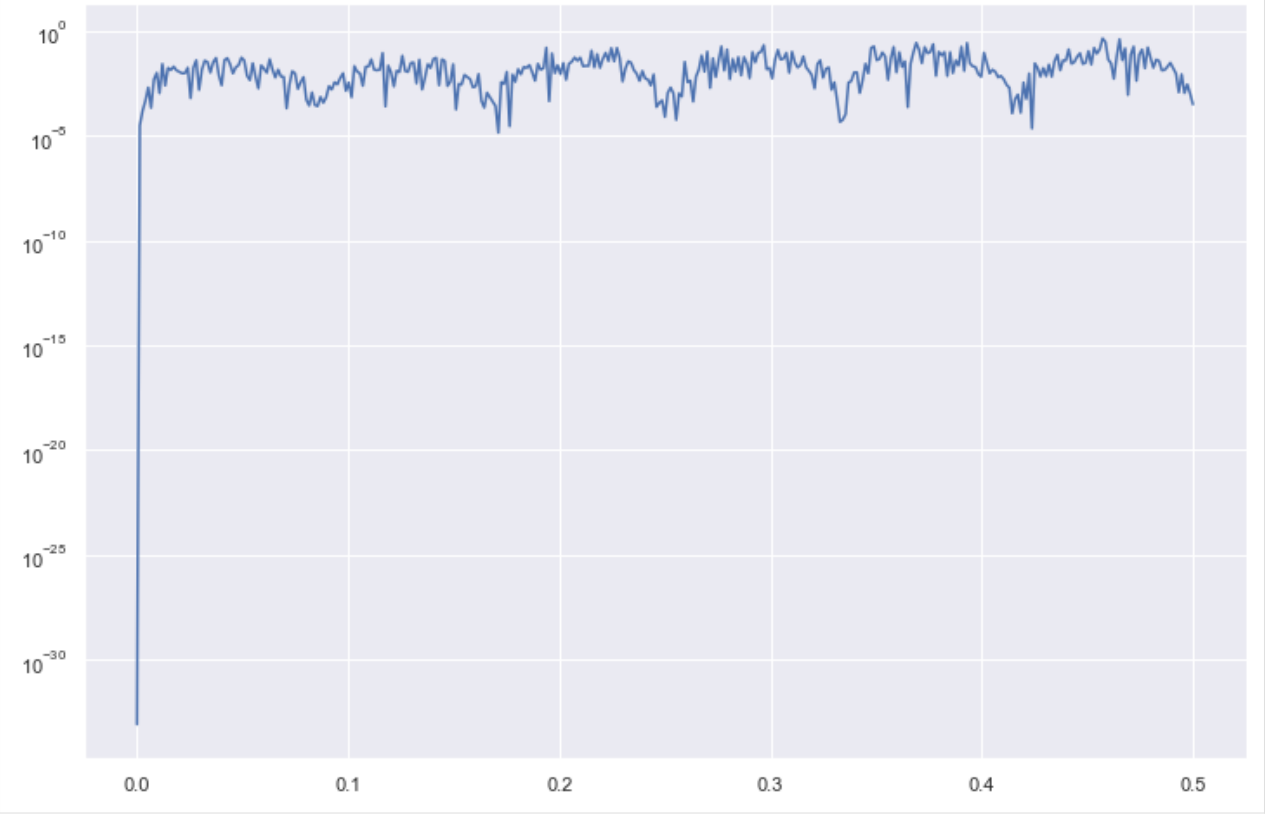
(continued from previous page)

```
print('the p-value is {:.4f}'.format(pval))
print('the series is {}'.format('stationary' if pval < critical_pval else 'not stationary
↪'))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
↪-----------
Augmented Dickey-Fuller results:
the test-stat value is: -11.72
the p-value is 0.0000
the series is stationary
----------------------------------------------------------------------------------------------------
↪-----------
```

Now, we can check out its periodogram to make sure no further adjustments are needed to the data:

[14]:
```
a, b = f_transform.plot_periodogram()
plt.semilogy(a, b)
plt.show()
```



This shows that there may be some cycles within the data that show stronger effects than other cycles, but for the most part, the periodogram is pretty stable. We can say that the transformations we took work for our purposes.

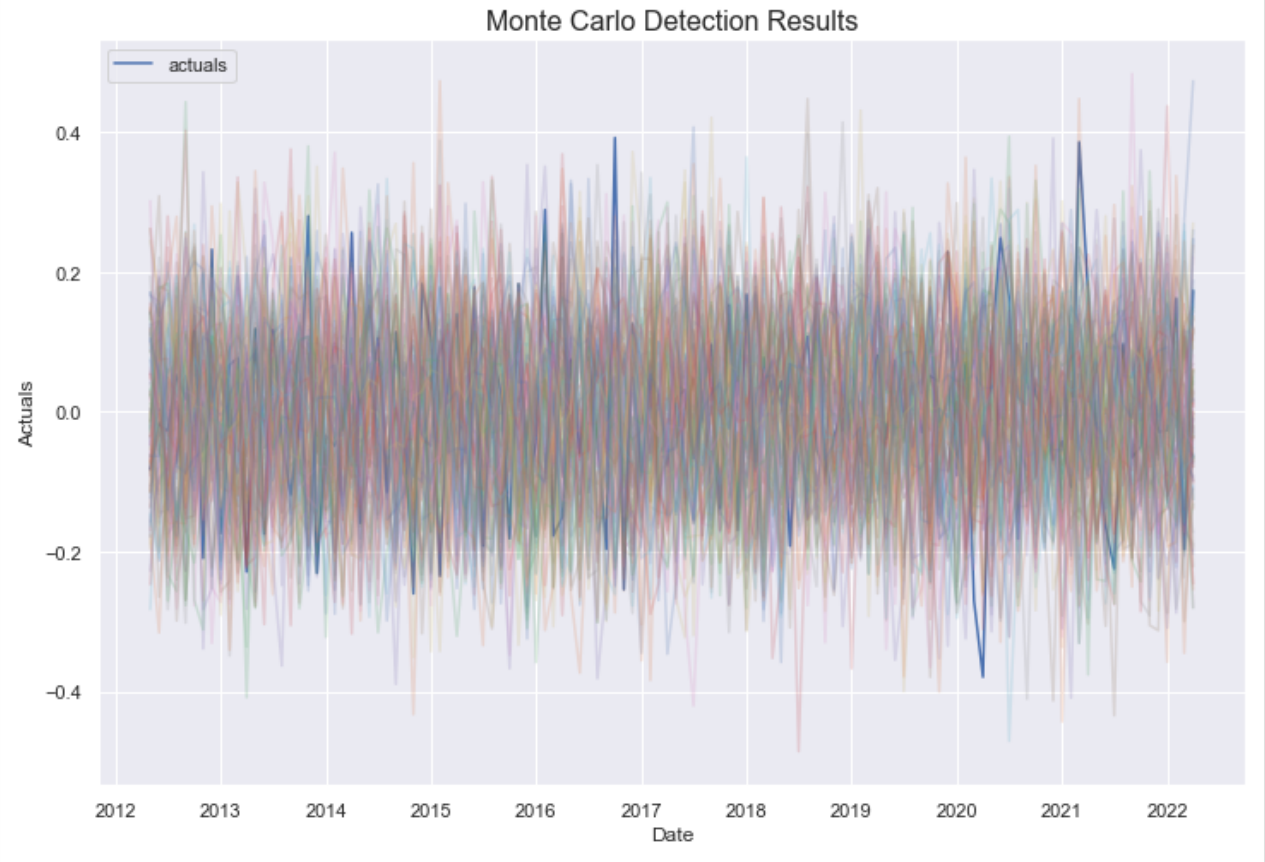
Now we can call the method and plot the results:

[15]:
```
detector2.MonteCarloDetect(
    start_at = '2012-05-01',
```

(continues on next page)

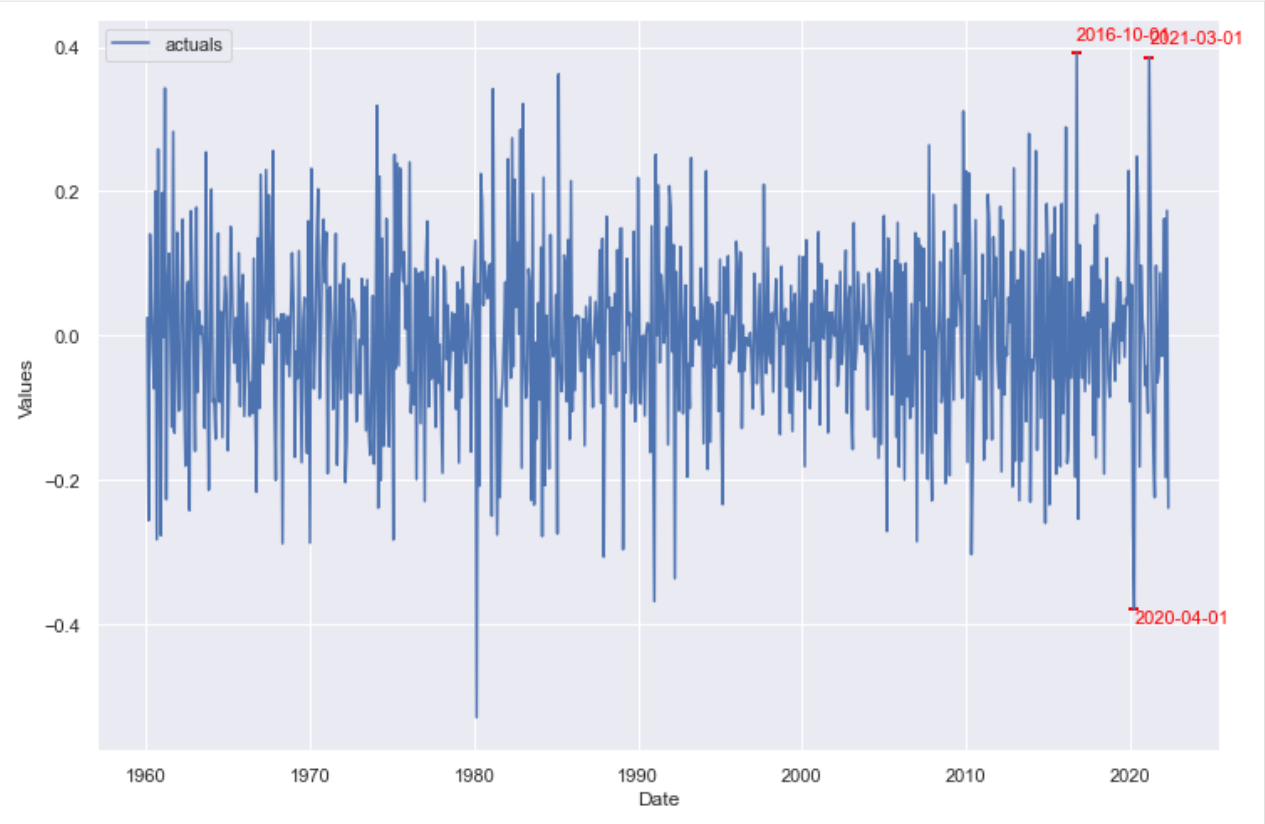
(continued from previous page)

```
    stop_at = '2022-05-01',
    cilevel=.99,
)

detector2.plot_mc_results()
plt.title('Monte Carlo Detection Results',size=16)
plt.show()
```



The blue line is the series' actual values and the colored lines are the results of each of 100 monte carlo simulations applied on the last 10 years of data available in the set. The few points that clearly stick above or below all colored lines will most likely be labeled as anomalies.

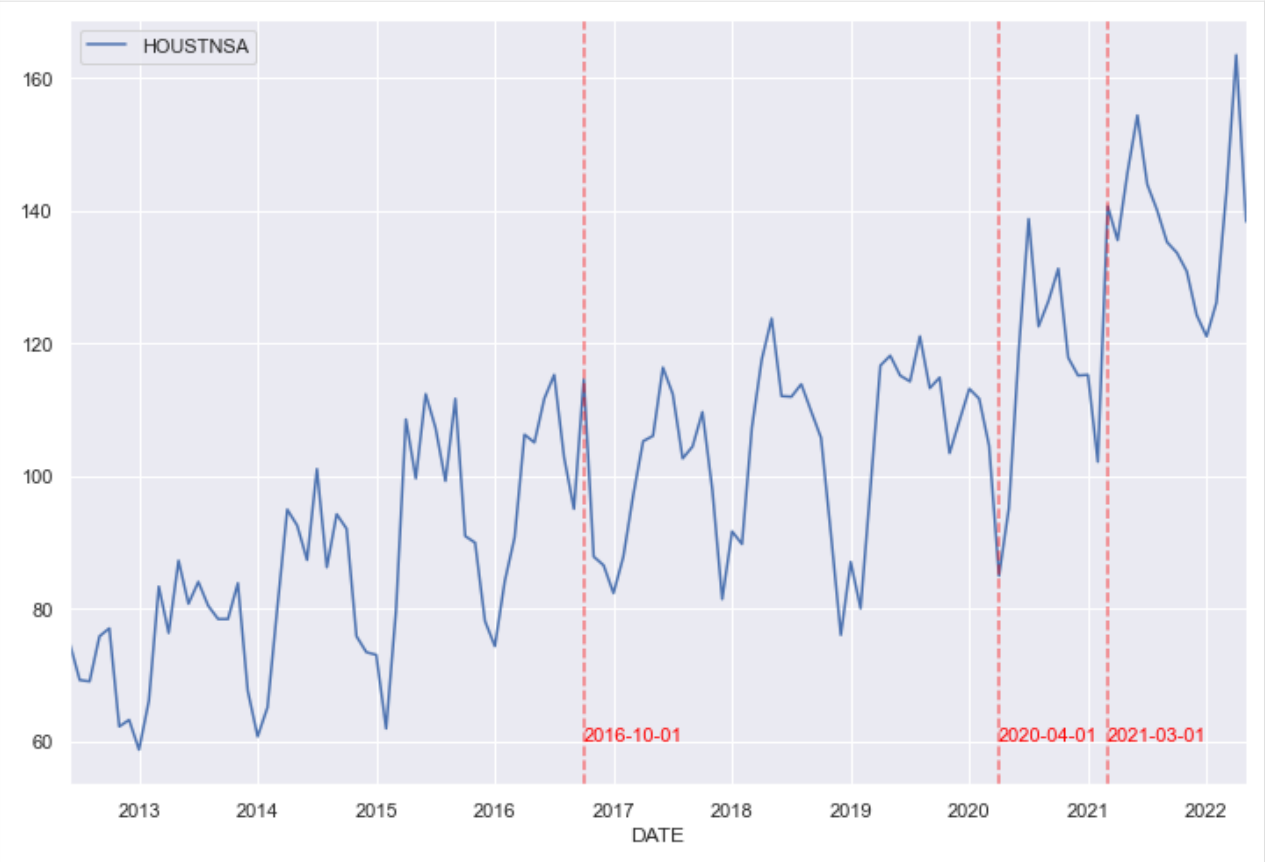
[16]:
```
detector2.plot_anom()
plt.show()
```



We see three points identified: - Oct. 2016
- April 2020 (first month of the COVID pandemic)
- March 2021

We can plot those points back onto our original series:

[17]:
```
_, ax = plt.subplots()

df.iloc[-120:,:].plot(x='DATE',y='HOUSTNSA',ax=ax)

for i,v in zip(detector2.labeled_anom.index,detector2.labeled_anom):
    if v == 1:
        ax.axvline(i,linestyle='--',color='red',alpha=.5)
        ax.text(
            i+pd.Timedelta(days=2),
            60,
            s=i.strftime('%Y-%m-%d'),
            color='red',
            size=11
        )

plt.show()
```



Let's dig into these three points a little more closely.

[18]:
```
df2 = df.copy()
df2['year'] = df2.reset_index().DATE.dt.year
df2['month'] = df2.reset_index().DATE.dt.month
df2['pct_chg'] = df2['HOUSTNSA'].pct_change()
```

[19]:
```
avg_chg_month = df2.groupby('month')['pct_chg'].mean().reset_index()
avg_chg_month['pct_chg_std'] = df2.groupby('month')['pct_chg'].std().values
avg_chg_month['pct_chg_2016'] = df2.loc[df2['year'] == 2016,'pct_chg'].values
avg_chg_month['pct_chg_2020'] = df2.loc[df2['year'] == 2020,'pct_chg'].values
avg_chg_month['pct_chg_2021'] = df2.loc[df2['year'] == 2021,'pct_chg'].values

def highlight_rows(row):

    lightgreen = 'background-color: lightgreen;'
    highlight = 'background-color: lightcoral;'
    default = ''

    if row['month'] == 3:
        return [default,lightgreen,lightgreen,default,default,highlight]
    elif row['month'] == 4:
        return [default,lightgreen,lightgreen,default,highlight,default]
    elif row['month'] == 10:
        return [default,lightgreen,lightgreen,highlight,default,default]
```

(continues on next page)

(continued from previous page)

```
    else:
        return [default,lightgreen,lightgreen,default,default,default]

avg_chg_month.style.apply(
    highlight_rows,
    axis=1
).format(
    {var:'{:.1%}'.format for var in avg_chg_month.iloc[:,1:]}
)
```

[19]:
```
<pandas.io.formats.style.Styler at 0x1606589d4c0>
```

The reason October 2016 was identified was due to how much more it increased in value from the previous month compared to Octobers from other years. The April 2020 point, as is logical, is from a precipitous decrease in housing starts after the COVID-19 pandemic began. The March 2021 point is the hardest to logically explain, but could be from a large year-over-year increase–a stronger than expected recovery from COVID.

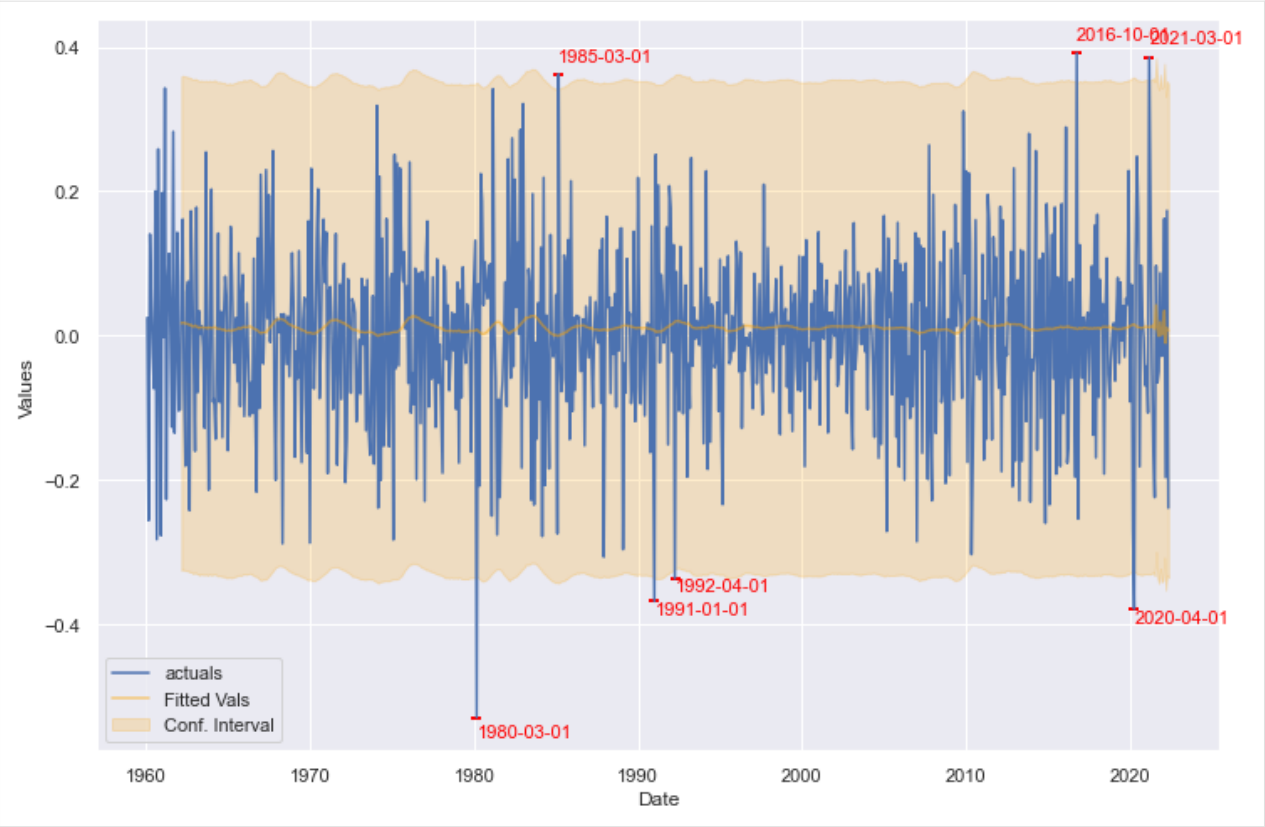
2.3 LSTM Detect

Another technique to identify anomalies includes using an estimator native to scalecast to create bootstrapped confidence intervals around fitted values. All actual values that fall outside of the generated confidence intervals are considered anomalies. This concept is very simple to understand. Let's see how the LSTM model performs in such a process on the same transformed series we used in the Monte Carlo detect.

[18]:
```
detector3 = AnomalyDetector(f_transform)
```

[19]:
```
detector3.EstimatorDetect(
    estimator='lstm',
    future_dates=12, # how many forecast steps to train into the lstm? default is 1
    cilevel=.99,
    lags=24,
    epochs=10,
    lstm_layer_sizes=(64,64,64),
    dropout=(0,0,0),
    verbose=0,
)
```

[20]:
```
detector3.plot_anom()
plt.show()
```



The LSTM model does not create a tight fit for this series, but it knows its performance is bad and creates large confidence intervals. As such, we see the same three points identified from the Monte Carlo technique, as well as an additional four points identified from early parts in the series' history. Due to the random nature of this model, results may vary each time it is run.

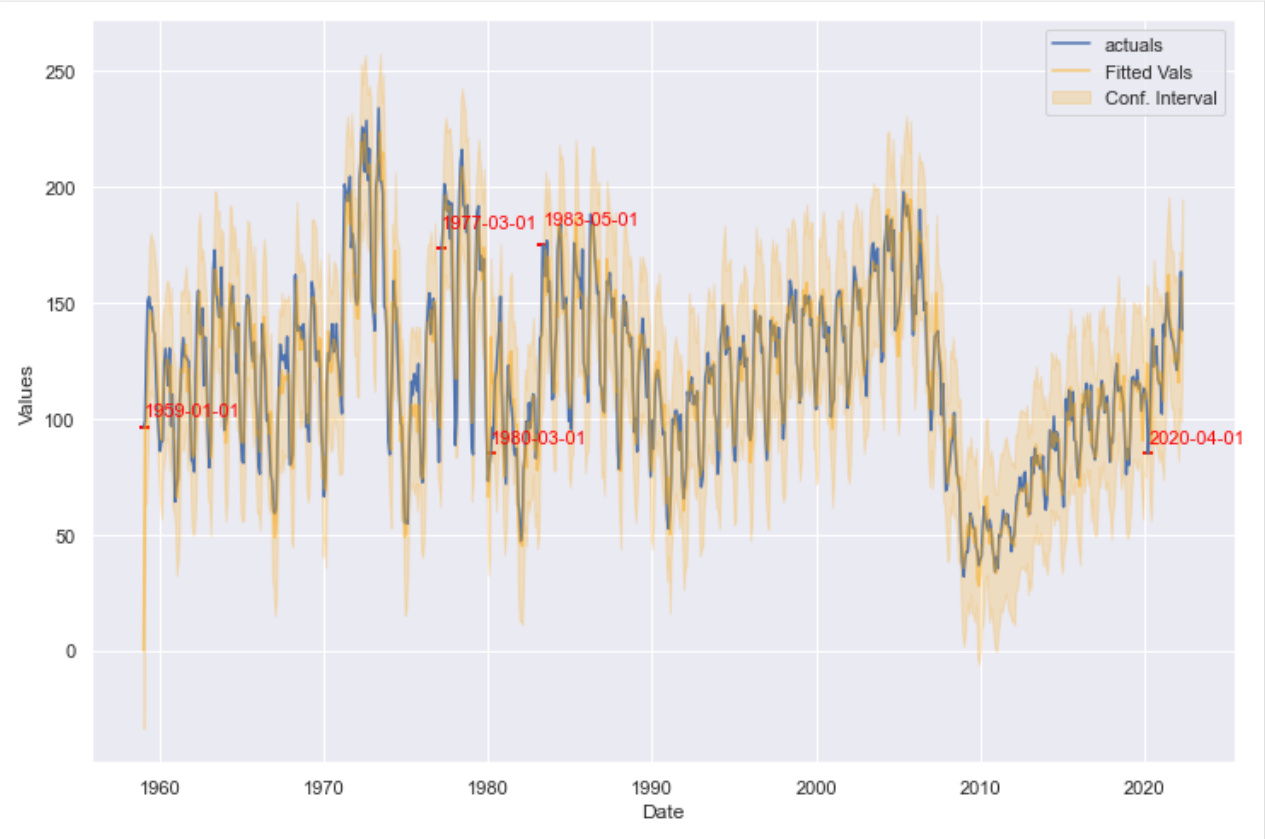
2.4 ARIMA Detect

Finally, we use the same technique as described above, but change the underlying estimator to ARIMA. We apply the model on the non-transformed series and see the results.

[21]:
```
detector4 = AnomalyDetector(f)
```

[22]:
```
detector4.EstimatorDetect(
    estimator='arima',
    cilevel=.99,
    order=(2,1,2),
    seasonal_order=(2,0,0,12),
)
```

[23]:
```
detector4.plot_anom()
plt.show()
```



The ARIMA model creates a much tighter fit than the LSTM model, but creates tighter confidence intervals as well. According to this method, October 2016 and March 2021 are no longer anomalies, but April 2020 remains. It also identifies several points earlier in the series' history.

[ ]:

CHAPTER

THREE

ARIMA

An introduction to ARIMA forecasting with scalecast.

- data: https://www.kaggle.com/datasets/rakannimer/air-passengers
- blog post: https://towardsdatascience.com/forecast-with-arima-in-python-more-easily-with-scalecast-35125fc7dc2e

[1]:
```
import pandas as pd
import numpy as np
from scalecast.Forecaster import Forecaster
from scalecast.auxmodels import auto_arima
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

[2]:
```
df = pd.read_csv('AirPassengers.csv')
f = Forecaster(
    y=df['#Passengers'],
    current_dates=df['Month'],
    future_dates = 12,
    test_length = .2,
    cis = True,
)
f
```

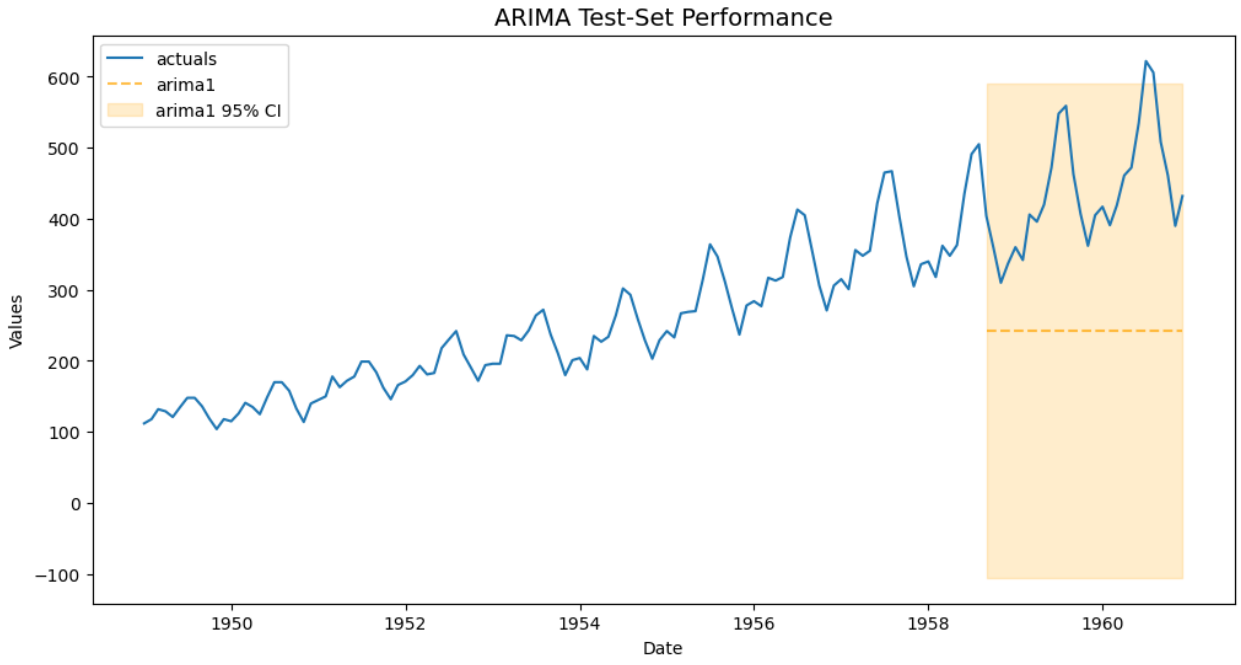
[2]:
```
Forecaster(
    DateStartActuals=1949-01-01T00:00:00.000000000
    DateEndActuals=1960-12-01T00:00:00.000000000
    Freq=MS
    N_actuals=144
    ForecastLength=12
    Xvars=[]
    TestLength=28
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

3.1 Naive Simple Approach

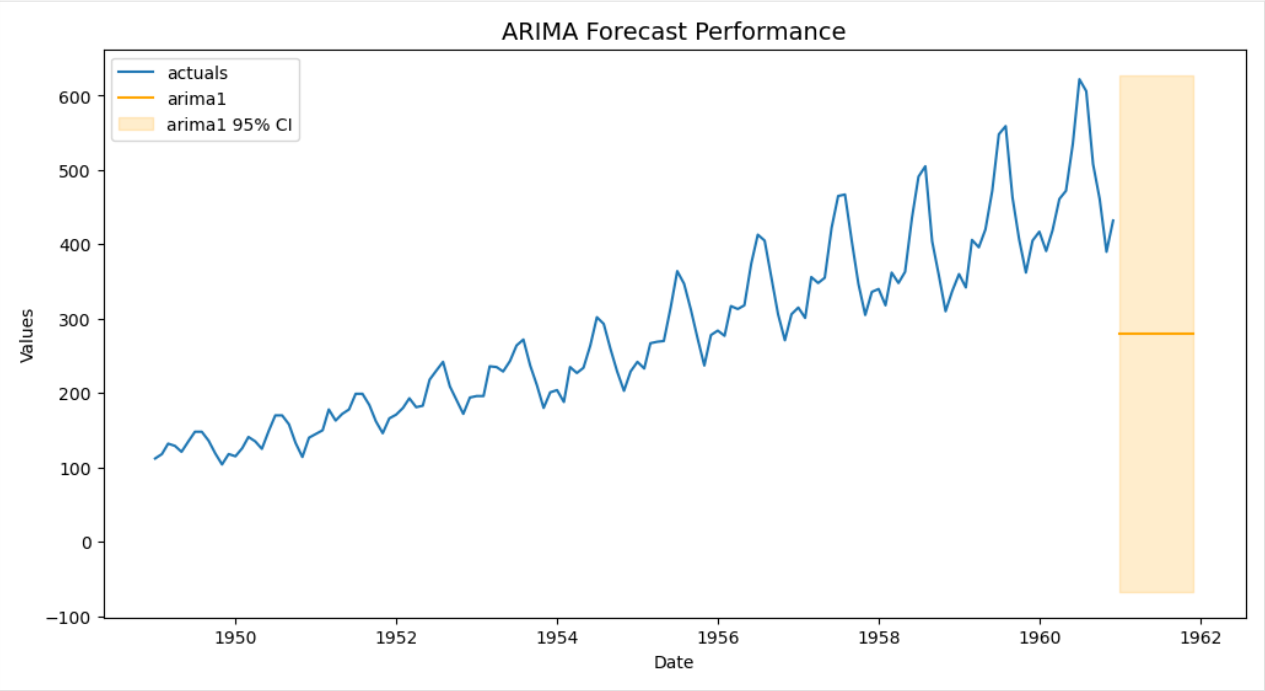
- this is not meant to be a demonstration of a model that is expected to be accurate
- it is meant to show the mechanics of using scalecast

[3]:
```
f.set_estimator('arima')
f.manual_forecast(call_me='arima1')
```

[4]:
```
f.plot_test_set(ci=True)
plt.title('ARIMA Test-Set Performance',size=14)
plt.show()
```



[5]:
```
f.plot(ci=True)
plt.title('ARIMA Forecast Performance',size=14)
plt.show()
```

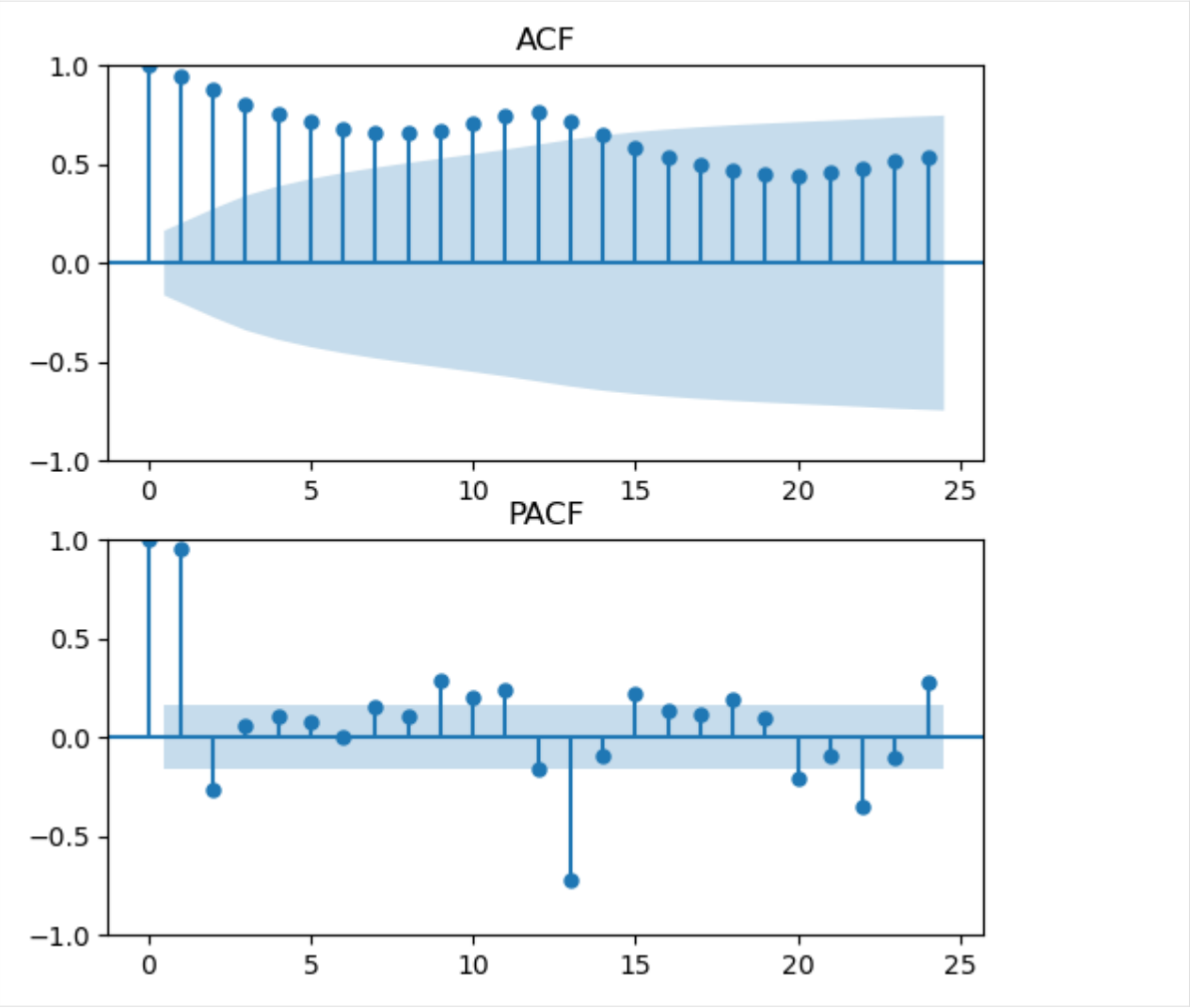


3.2 Human Interpretation Iterative Approach

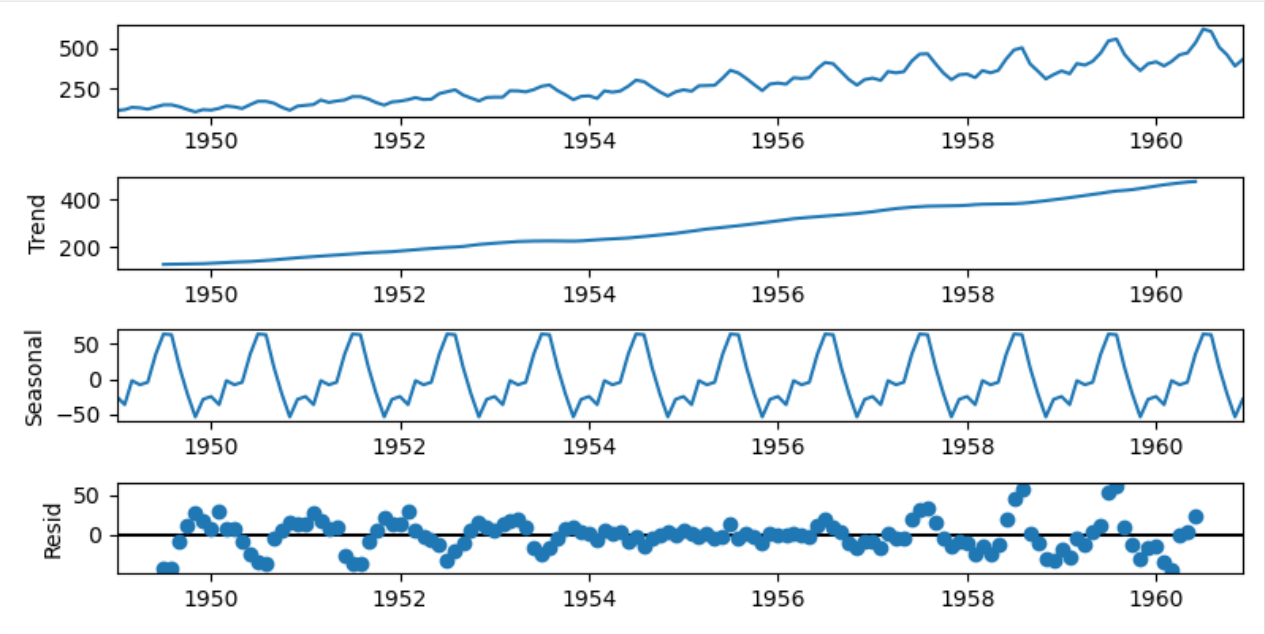
- this is a non-automated approach to ARIMA forecasting where model specification depends on human-interpretation of statistical results and charts

[6]:
```
figs, axs = plt.subplots(2, 1,figsize=(6,6))
f.plot_acf(ax=axs[0],title='ACF',lags=24)
f.plot_pacf(ax=axs[1],title='PACF',lags=24)
plt.show()
```

```
/Users/uger7/opt/anaconda3/envs/scalecast-env/lib/python3.8/site-packages/statsmodels/
↪graphics/tsaplots.py:348: FutureWarning: The default method 'yw' can produce PACF␣
↪values outside of the [-1,1] interval. After 0.13, the default will change␣
↪tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



[7]:
```
plt.rc("figure",figsize=(8,4))
f.seasonal_decompose().plot()
plt.show()
```

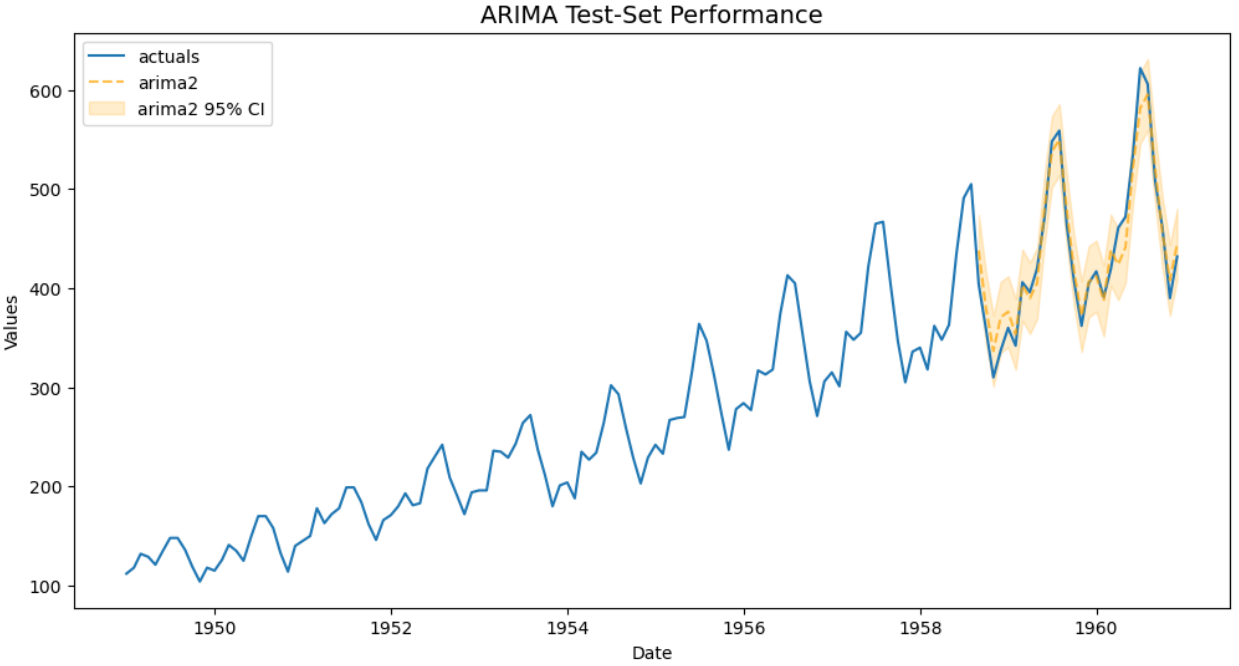


[8]:
```
stat, pval, _, _, _, _ = f.adf_test(full_res=True)
print(stat)
print(pval)
```

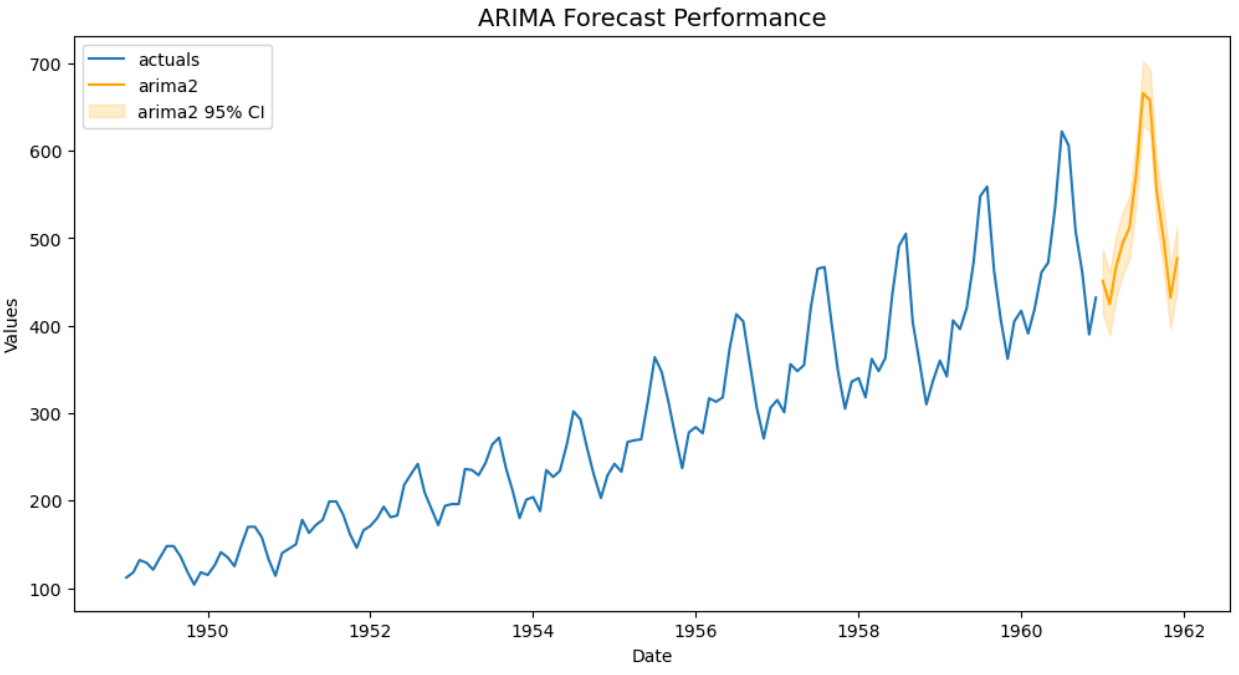
```
0.8153688792060442
0.9918802434376409
```

[9]:
```
f.manual_forecast(order=(1,1,1),seasonal_order=(2,1,1,12),call_me='arima2')
```

[10]:
```
f.plot_test_set(ci=True,models='arima2')
plt.title('ARIMA Test-Set Performance',size=14)
plt.show()
```



[11]:
```
f.plot(ci=True,models='arima2')
plt.title('ARIMA Forecast Performance',size=14)
plt.show()
```



[12]:
```
f.regr.summary()
```

[12]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
```

(continues on next page)

(continued from previous page)

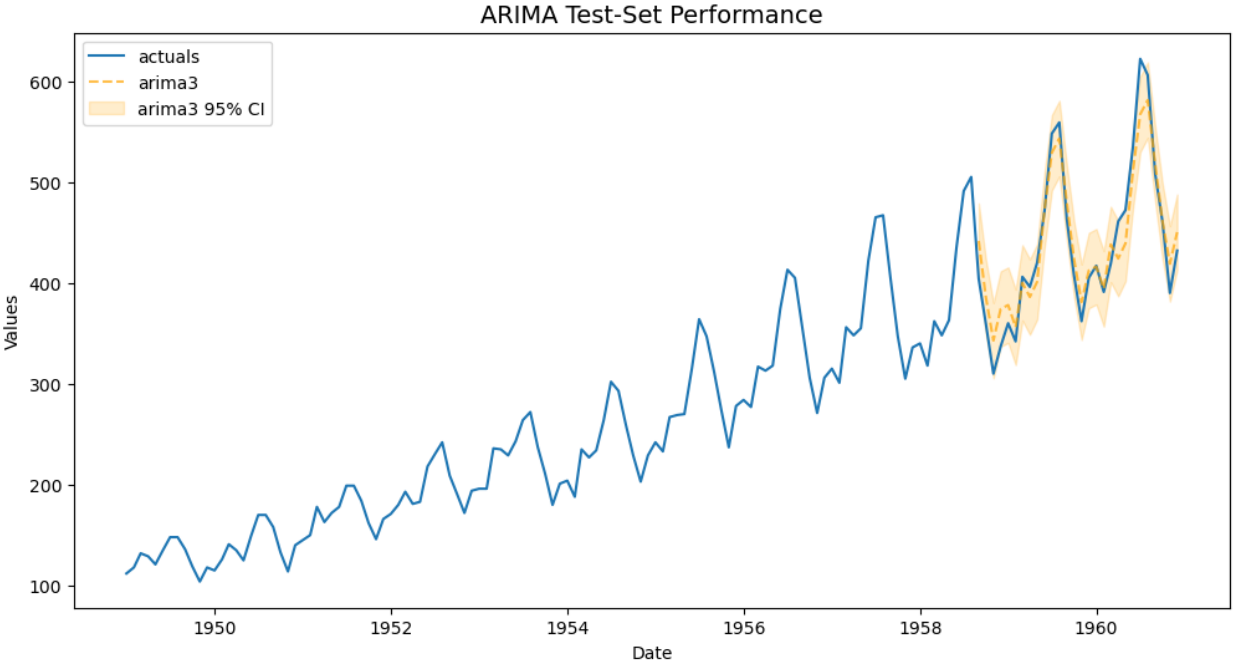
```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                  144
Model:             ARIMA(1, 1, 1)x(2, 1, 1, 12)   Log Likelihood                -501.929
Date:                          Mon, 10 Apr 2023   AIC                           1015.858
Time:                                  18:47:23   BIC                           1033.109
Sample:                                       0   HQIC                          1022.868
                                            - 144
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0731      0.273     -0.268      0.789      -0.608       0.462
ma.L1         -0.3572      0.249     -1.436      0.151      -0.845       0.130
ar.S.L12       0.6673      0.160      4.182      0.000       0.355       0.980
ar.S.L24       0.3308      0.099      3.341      0.001       0.137       0.525
ma.S.L12      -0.9711      1.086     -0.895      0.371      -3.099       1.157
sigma2       111.1028     98.277      1.131      0.258     -81.517     303.722
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 7.72
Prob(Q):                              0.99   Prob(JB):                         0.02
Heteroskedasticity (H):               2.77   Skew:                             0.08
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.18
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```

3.3 Auto-ARIMA Approach

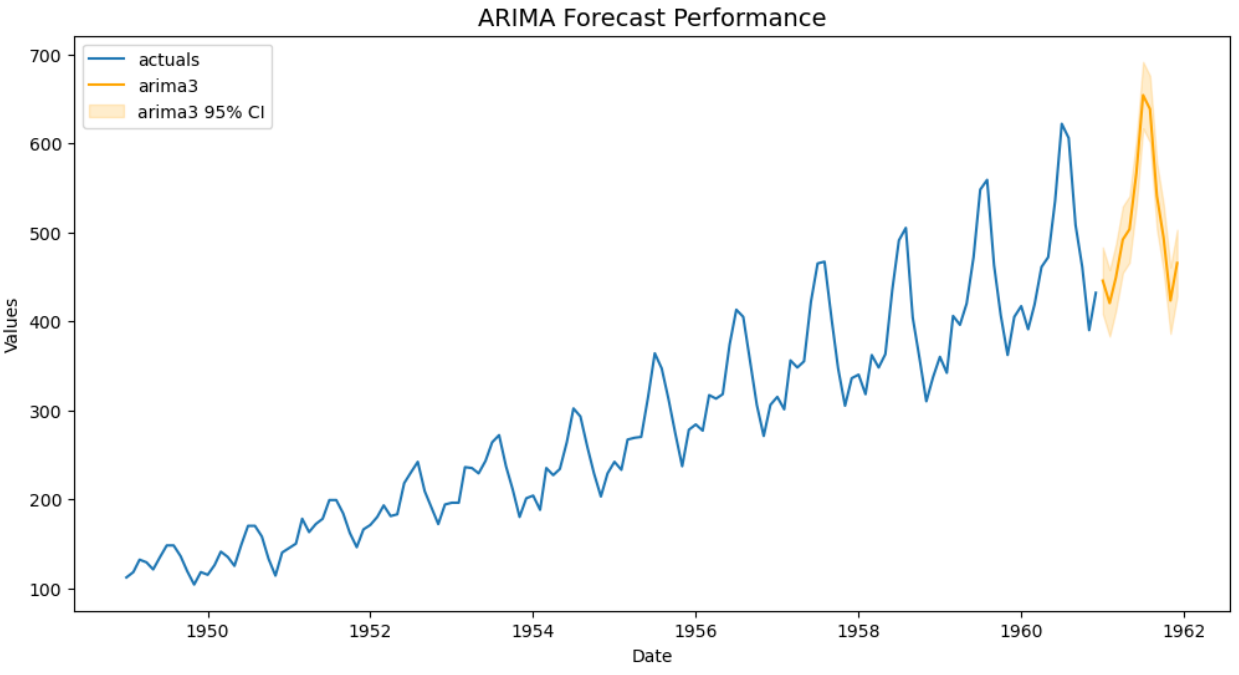
```
pip install pmdarima
```

[13]:
```
auto_arima(
    f,
    m=12,
    call_me='arima3',
)
```

[14]:
```
f.plot_test_set(ci=True,models='arima3')
plt.title('ARIMA Test-Set Performance',size=14)
plt.show()
```



[15]:
```
f.plot(ci=True,models='arima3')
plt.title('ARIMA Forecast Performance',size=14)
plt.show()
```



[16]:
```
f.regr.summary()
```

[16]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
```

(continues on next page)

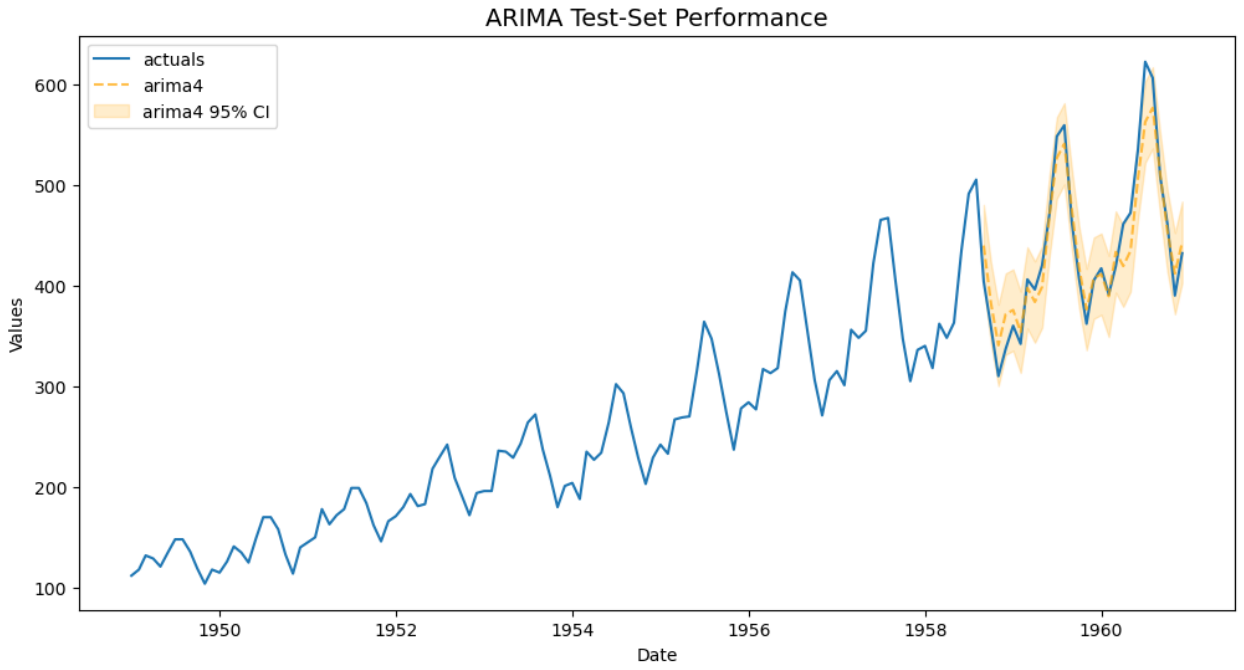
(continued from previous page)

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                  y   No. Observations:                  144
Model:             ARIMA(2, 1, 1)x(0, 1, [], 12)   Log Likelihood                -504.923
Date:                           Mon, 10 Apr 2023   AIC                           1017.847
Time:                                   18:47:55   BIC                           1029.348
Sample:                                         0   HQIC                          1022.520
                                              - 144
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.5960      0.085      6.986      0.000       0.429       0.763
ar.L2          0.2143      0.091      2.343      0.019       0.035       0.394
ma.L1         -0.9819      0.038    -25.599      0.000      -1.057      -0.907
sigma2       129.3177     14.557      8.883      0.000     100.786     157.850
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 7.68
Prob(Q):                              0.98   Prob(JB):                         0.02
Heteroskedasticity (H):               2.33   Skew:                            -0.01
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.19
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```

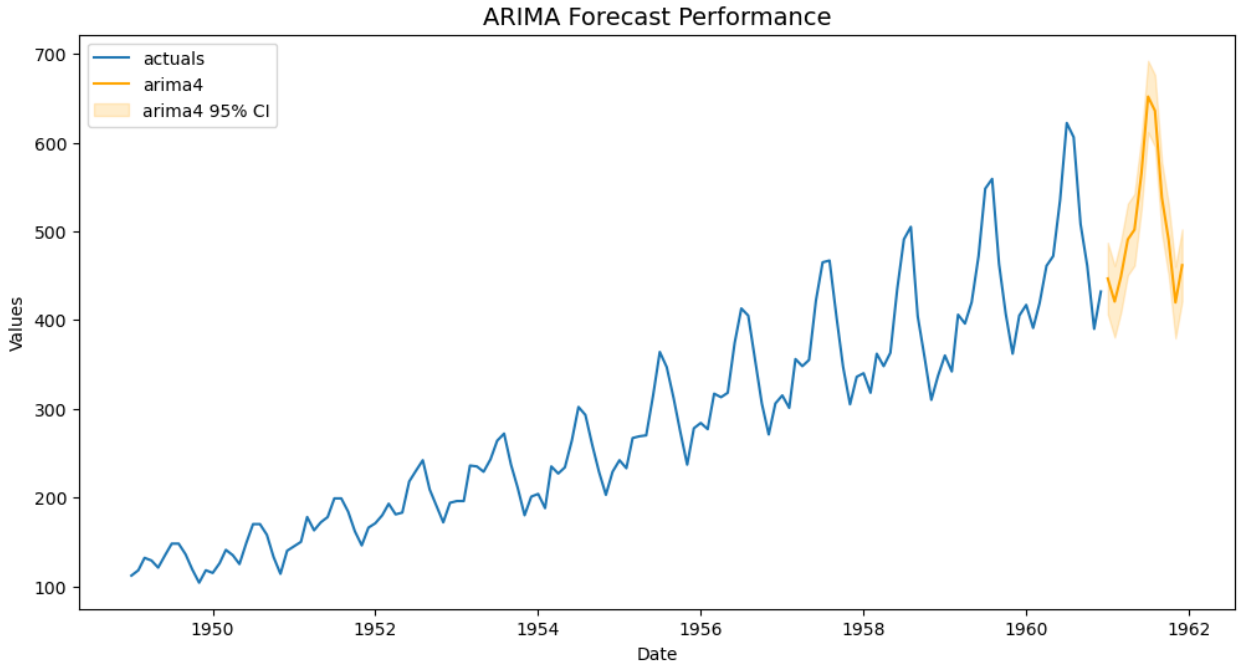
3.4 Grid Search Approach

[17]:
```
f.set_validation_length(12)
grid = {
    'order':[
        (1,1,1),
        (1,1,0),
        (0,1,1),
    ],
    'seasonal_order':[
        (2,1,1,12),
        (1,1,1,12),
        (2,1,0,12),
        (0,1,0,12),
    ],
}

f.ingest_grid(grid)
f.tune()
f.auto_forecast(call_me='arima4')
```

[18]:
```
f.plot_test_set(ci=True,models='arima4')
plt.title('ARIMA Test-Set Performance',size=14)
plt.show()
```



[19]:
```
f.plot(ci=True,models='arima4')
plt.title('ARIMA Forecast Performance',size=14)
plt.show()
```



[20]:
```
f.regr.summary()
```

[20]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                  144
Model:             ARIMA(0, 1, 1)x(0, 1, [], 12)   Log Likelihood                -508.319
Date:                           Mon, 10 Apr 2023   AIC                           1020.639
Time:                                   18:50:43   BIC                           1026.389
Sample:                                         0   HQIC                          1022.975
                                              - 144
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.3184      0.063     -5.038      0.000      -0.442      -0.195
sigma2       137.2653     15.024      9.136      0.000     107.818     166.713
===================================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 5.46
Prob(Q):                              0.95   Prob(JB):                         0.07
Heteroskedasticity (H):               2.37   Skew:                             0.02
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.00
===================================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
"""
```

3.5 Export Results

[21]:
```
pd.options.display.max_colwidth = 100
results = f.export(to_excel=True,excel_name='arima_results.xlsx',determine_best_by=
↪'TestSetMAPE')
summaries = results['model_summaries']
summaries[['ModelNickname','HyperParams','InSampleMAPE','TestSetMAPE']]
```

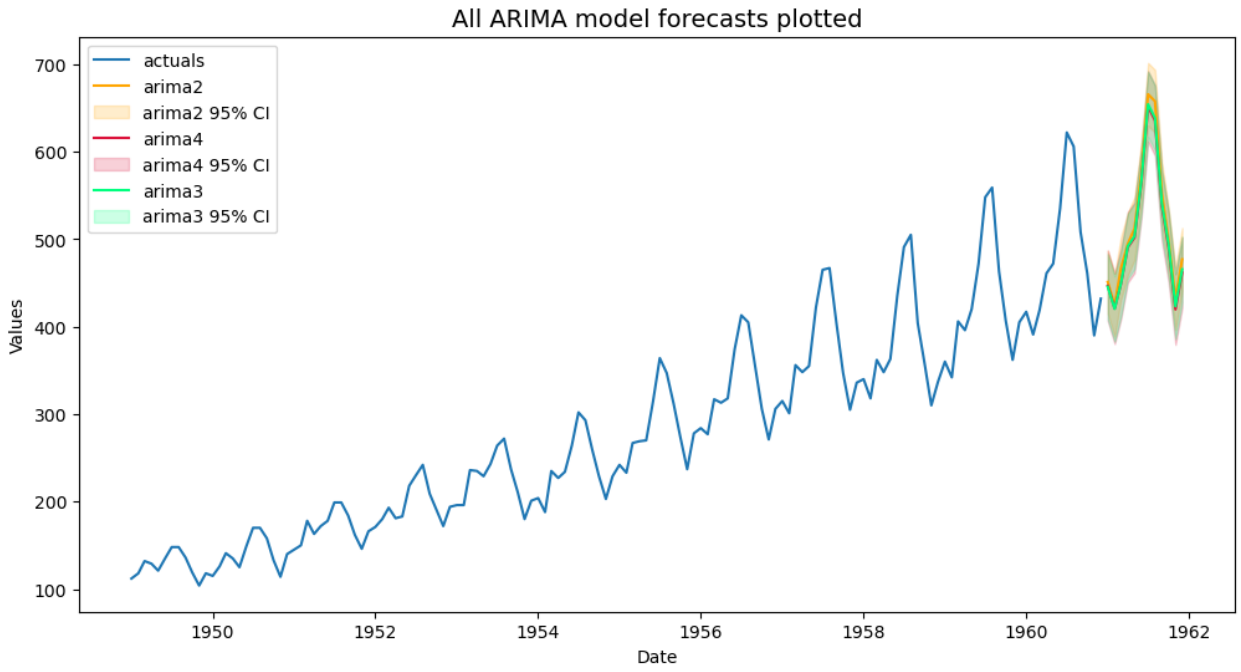
[21]:
```
  ModelNickname  \
0        arima2
1        arima4
2        arima3
3        arima1

                                                                HyperParams  \
0                {'order': (1, 1, 1), 'seasonal_order': (2, 1, 1, 12)}
1                {'order': (0, 1, 1), 'seasonal_order': (0, 1, 0, 12)}
2  {'order': (2, 1, 1), 'seasonal_order': (0, 1, 0, 12), 'trend': None}
3                                                                   {}

   InSampleMAPE  TestSetMAPE
0      0.044448     0.037170
```

(continues on next page)

(continued from previous page)

```
1       0.046529      0.044054
2       0.045081      0.045936
3       0.442457      0.430066
```

[22]:
```
f.plot(ci=True,models=['arima2','arima3','arima4'],order_by='TestSetMAPE')
plt.title('All ARIMA model forecasts plotted',size=14)
plt.show()
```



[ ]:

CHAPTER

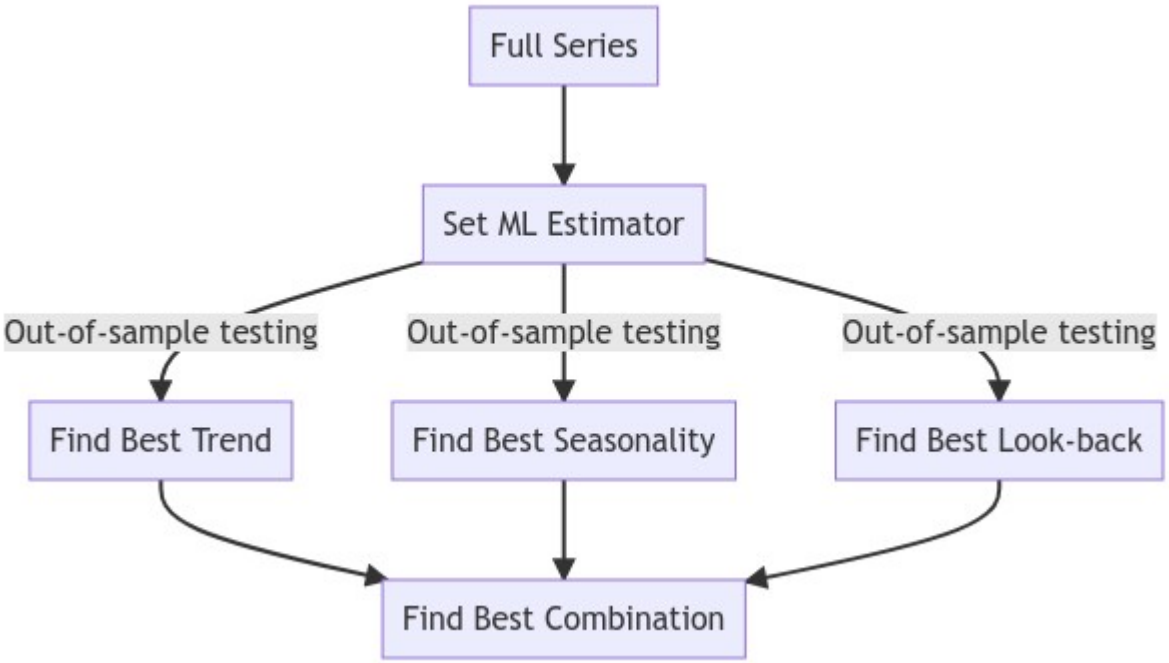
FOUR

AUTO FEATURE SELECTION

- What model combinations can be used to find the ideal features to model a time series with and also to make the best forecasts?
- See Auto Model Specification with ML Techniques for Time Series

[1]:
```
import pandas as pd
import numpy as np
from scalecast.Forecaster import Forecaster
from scalecast.util import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm.notebook import tqdm
```

Diagram to demonstrate the auto_Xvar_select() method:



4.1 Use 50 Random Series from M4 Hourly

[2]:
```
models = (
    'mlr',
    'elasticnet',
    'gbt',
    'knn',
    'svr',
    'mlp',
)

Hourly = pd.read_csv(
    'Hourly-train.csv',
    index_col=0,
).sample(50)

Hourly_test = pd.read_csv(
    'Hourly-test.csv',
    index_col=0,
)
Hourly_test = Hourly_test.loc[Hourly.index]

info = pd.read_csv(
    'M4-info.csv',
    index_col=0,
    parse_dates=['StartingDate'],
    dayfirst=True,
)

results = pd.DataFrame(
    index = models,
    columns = models,
).fillna(0)
```

4.2 Run all Model Combos and Evaluate SMAPE

[3]:
```
for i in tqdm(Hourly.index):
    y = Hourly.loc[i].dropna()
    sd = info.loc[i,'StartingDate']
    fcst_horizon = info.loc[i,'Horizon']
    cd = pd.date_range(
        start = sd,
        freq = 'H',
        periods = len(y),
    )
    f = Forecaster(
        y = y,
        current_dates = cd,
        future_dates = fcst_horizon,
        test_length = fcst_horizon, # for finding xvars
```

(continues on next page)

(continued from previous page)

```
    )
    # extension of this analysis - take transformations
    for xvm in models:
        for fcstm in models:
            f2 = f.deepcopy()
            f2.auto_Xvar_select(
                estimator = xvm,
                monitor='TestSetRMSE',
                max_ar = 48,
                exclude_seasonalities = ['quarter','month','week','day'],
            )
            f2.set_estimator(fcstm)
            f2.manual_forecast(dynamic_testing=False)
            point_fcst = f2.export('lvl_fcsts')[fcstm]
            results.loc[xvm,fcstm] += metrics.smape(
                Hourly_test.loc[i].dropna().values,
                point_fcst.values,
            )
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

4.3 View Results

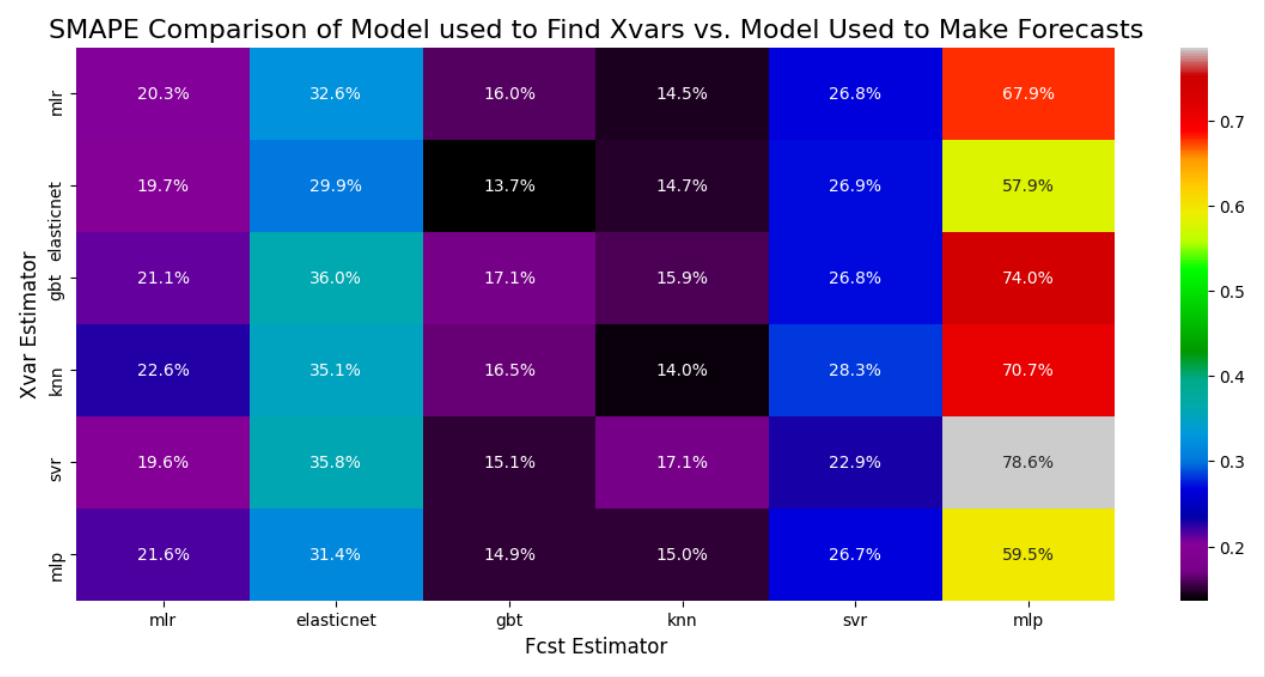
4.3.1 SMAPE Combos

[4]:
```
results = results / 50
results
```

[4]:

	mlr	elasticnet	gbt	knn	svr	mlp
mlr	0.202527	0.326240	0.160442	0.144705	0.267519	0.679007
elasticnet	0.197014	0.299116	0.136523	0.147124	0.269238	0.579156
gbt	0.210620	0.360102	0.171011	0.159181	0.268353	0.740443
knn	0.225529	0.350825	0.165057	0.140396	0.282870	0.707482
svr	0.195592	0.358184	0.151398	0.171485	0.229054	0.785514
mlp	0.216396	0.314216	0.149370	0.149949	0.266556	0.595177

[5]:
```
fig, ax = plt.subplots(figsize=(14,6))
sns.heatmap(results,cmap='nipy_spectral',ax=ax,annot=True,fmt='.1%')
plt.ylabel('Xvar Estimator',size=12)
plt.xlabel('Fcst Estimator',size=12)
plt.title('SMAPE Comparison of Model used to Find Xvars vs. Model Used to Make Forecasts
↪',size=16)
plt.show()
```



4.3.2 Best Models at Making Forecasts on Average

[6]:
```
# Fcst estimators
results.mean().sort_values()
```

[6]:
```
knn           0.152140
gbt           0.155634
mlr           0.207946
svr           0.263932
elasticnet    0.334780
mlp           0.681130
dtype: float64
```

4.3.3 Best Models at Finding Xvars on Average

[7]:
```
# Xvar estimators
results.mean(axis=1).sort_values()
```

[7]:
```
elasticnet    0.271362
mlp           0.281944
mlr           0.296740
knn           0.312027
svr           0.315204
gbt           0.318285
dtype: float64
```

[ ]:

CHAPTER

FIVE

BACKTESTED DYNAMIC CONFIDENCE INTERVALS

This notebook demonstrates how to create an expanding confidence interval using conformal intervals and backtesting. It expands what is demonstrated in the Confidence Intervals Notebook. Requires `scalecast>=0.18.1`.

We overwrite the static naive intervals produced by scalecast by default with dynamic expanding intervals obtained from backtesting.

See the article.

[1]:
```
import pandas as pd
import numpy as np
from scalecast.Forecaster import Forecaster
from scalecast.util import (
    metrics,
    backtest_metrics,
    backtest_for_resid_matrix,
    get_backtest_resid_matrix,
    overwrite_forecast_intervals,
)
from scalecast.Pipeline import Pipeline, Transformer, Reverter
import pandas_datareader as pdr
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

[2]:
```
val_len = 24
fcst_len = 24
```

- Link to data: https://fred.stlouisfed.org/series/HOUSTNSA

[3]:
```
housing = pdr.get_data_fred('HOUSTNSA',start='1900-01-01',end='2021-06-01')
housing.head()
```

[3]:
```
            HOUSTNSA
DATE
1959-01-01      96.2
1959-02-01      99.0
1959-03-01     127.7
1959-04-01     150.8
1959-05-01     152.5
```

Hold out a test set since scalecast uses its native test set to determine interval widths, therefore overfitting the interval to the test set.

[4]:
```
starts_sep = housing.iloc[-fcst_len:,0]
starts = housing.iloc[:-fcst_len,0]
```

[5]:
```
f = Forecaster(
    y=starts,
    current_dates=starts.index,
    future_dates=fcst_len,
    test_length=val_len,
    validation_length=val_len,
    cis=True,
)
```

5.1 Step 1: Build and fit_predict() Pipeline

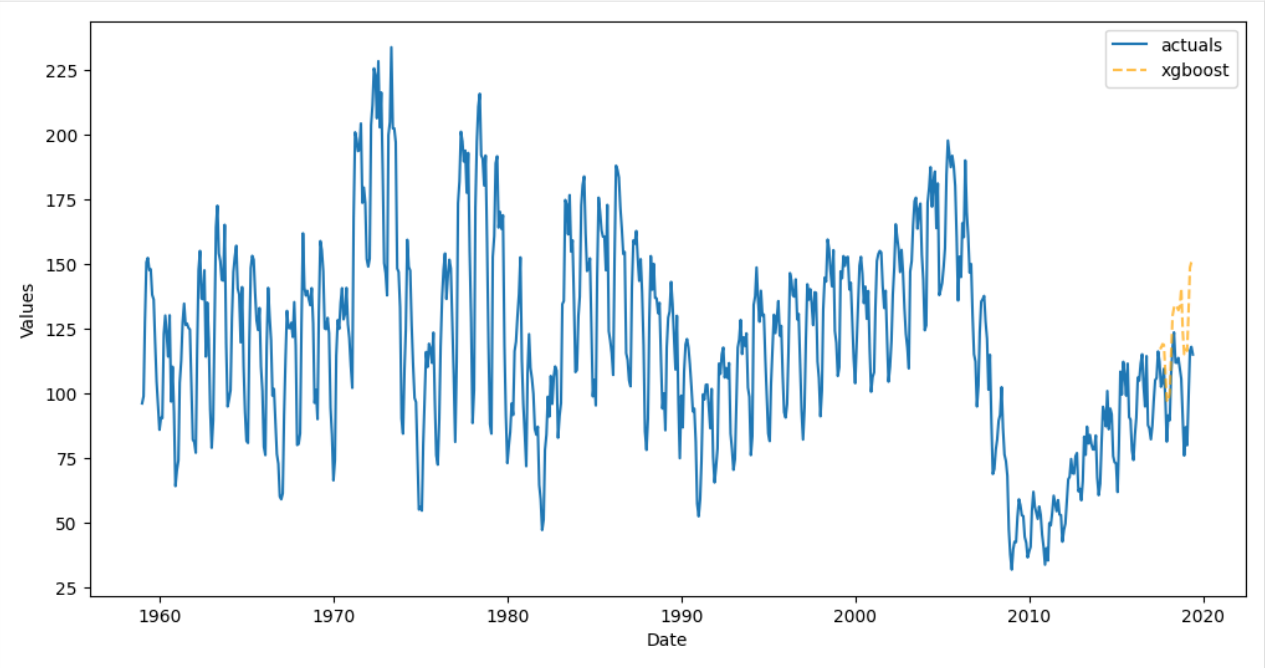
[6]:
```
transformer = Transformer(['DiffTransform'])
reverter = Reverter(['DiffRevert'],transformer)
```

[7]:
```
def forecaster(f):
    f.add_ar_terms(100)
    f.add_seasonal_regressors('month')
    f.set_estimator('xgboost')
    f.manual_forecast()
```

[8]:
```
pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter)
    ]
)
```

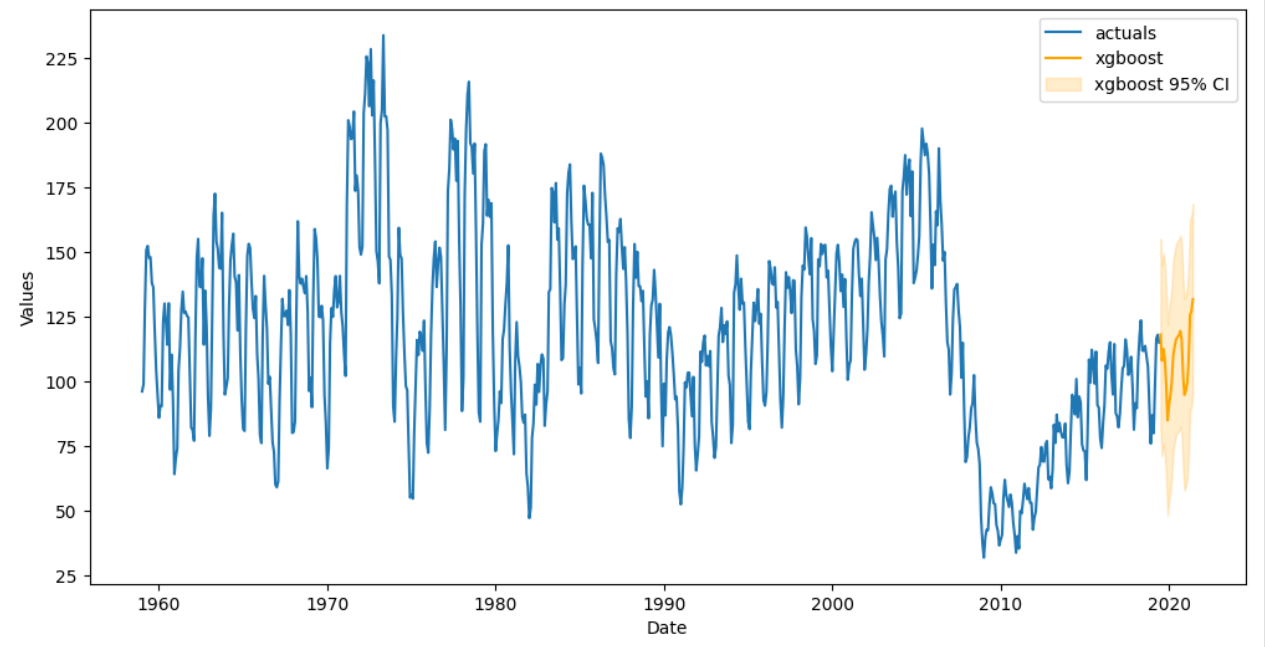
[9]:
```
f = pipeline.fit_predict(f)
```

[10]:
```
f.plot_test_set();
```



5.1.1 Score the default Interval

[11]:
```
f.plot(ci=True);
```

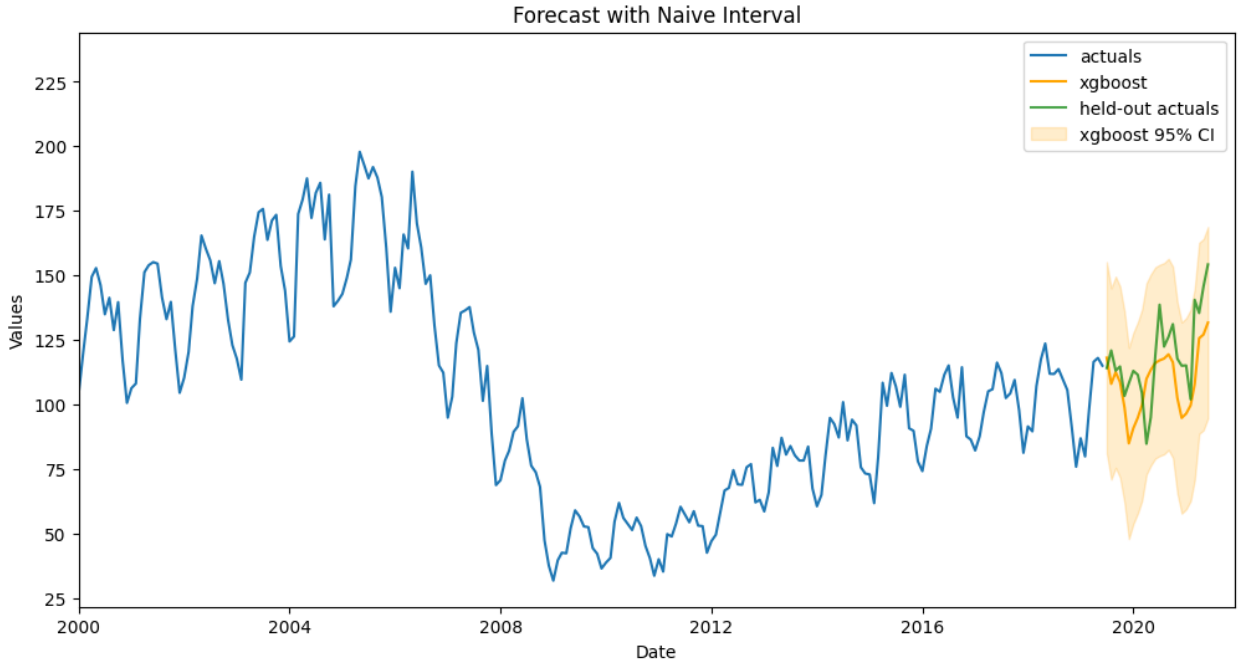


[12]:
```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(ci=True,models='top_1',order_by='TestSetRMSE',ax=ax)
sns.lineplot(
    y = 'HOUSTNSA',
```

(continues on next page)

(continued from previous page)

```
    x = 'DATE',
    data = starts_sep.reset_index(),
    ax = ax,
    label = 'held-out actuals',
    color = 'green',
    alpha = 0.7,
)
plt.xlim(pd.Timestamp('2000-01-01'),pd.Timestamp('2021-12-01'))
plt.title('Forecast with Naive Interval')
plt.show()
```



[13]:
```
print(
    'All confidence intervals for every step'
    ' are {:.2f} units away from the point predictions.'.format(
        f.history['xgboost']['UpperCI'][0] - f.history['xgboost']['Forecast'][0]
    )
)
print(
    'The interval contains {:.2%} of the actual values'.format(
        np.sum(
            [
                1 if a <= uf and a >= lf else 0
                for a, uf, lf in zip(
                    starts_sep,
                    f.history['xgboost']['UpperCI'],
                    f.history['xgboost']['LowerCI']
                )
            ]
        ) / len(starts_sep)
    )
```

(continues on next page)

(continued from previous page)

```
)
```

```
All confidence intervals for every step are 37.01 units away from the point predictions.
The interval contains 100.00% of the actual values
```

5.1.2 Score default interval

[14]:
```
metrics.msis(
    a = starts_sep,
    uf = f.history['xgboost']['UpperCI'],
    lf = f.history['xgboost']['LowerCI'],
    obs = f.y,
    m = 12,
)
```

[14]:
```
4.026554265078705
```

5.2 Step 2: Backtest Pipeline

- Iterations need to be at least 20 for 95% intervals.
- Length of each prediction in the backtest should match our desired forecast length.

[15]:
```
%%time
backtest_results = backtest_for_resid_matrix(
    f,
    pipeline=pipeline,
    alpha = .05, # default
    jump_back = 1, # default
)
```

```
CPU times: total: 1min 40s
Wall time: 28.5 s
```

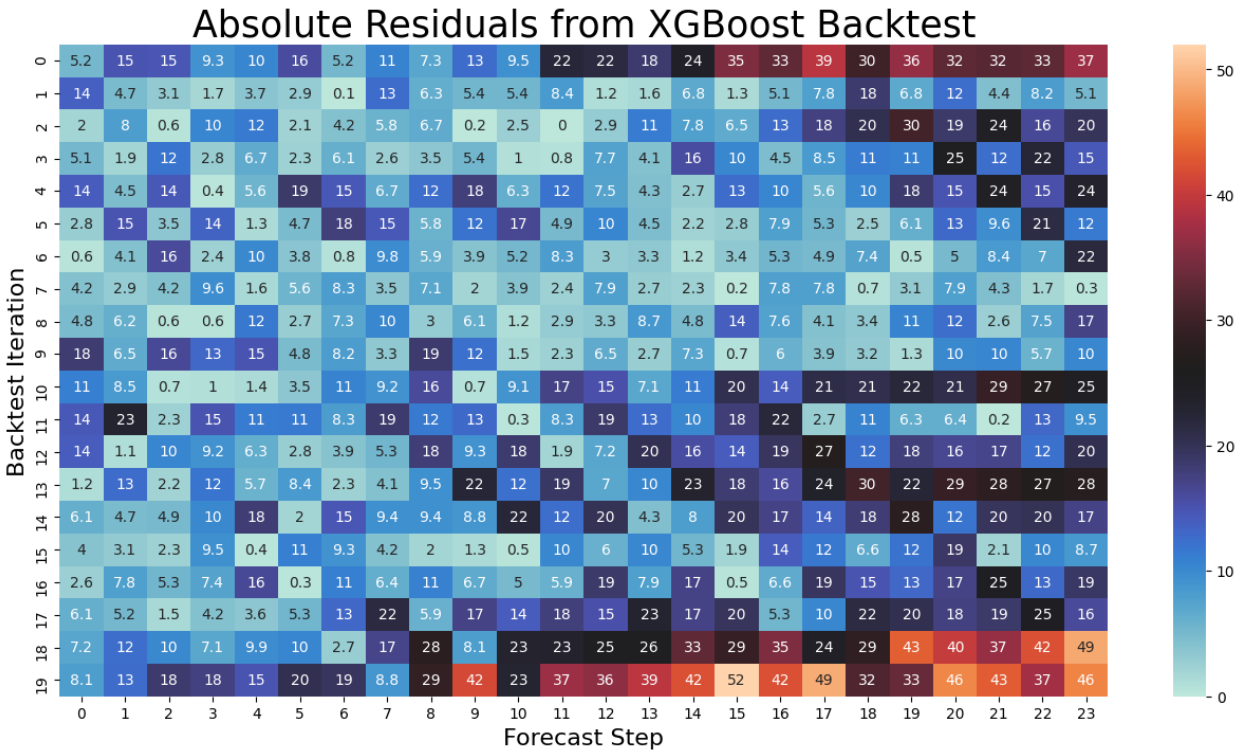
5.3 Step 3: Build Residual Matrix

- Result is matrix shaped 20x24, each row a backtest iteration, each column a forecast step, each value a residual.

[16]:
```
backtest_resid_matrix = get_backtest_resid_matrix(backtest_results)
```

5.3.1 Residual Analytics

[17]:
```
pd.options.display.max_columns = None
fig, ax = plt.subplots(figsize=(16,8))
mat = pd.DataFrame(np.abs(backtest_resid_matrix[0]['xgboost']))
sns.heatmap(
    mat.round(1),
    annot = True,
    ax = ax,
    cmap = sns.color_palette("icefire", as_cmap=True)
)
plt.ylabel('Backtest Iteration',size=16)
plt.xlabel('Forecast Step',size = 16)
plt.title('Absolute Residuals from XGBoost Backtest',size=25)
plt.show()
```

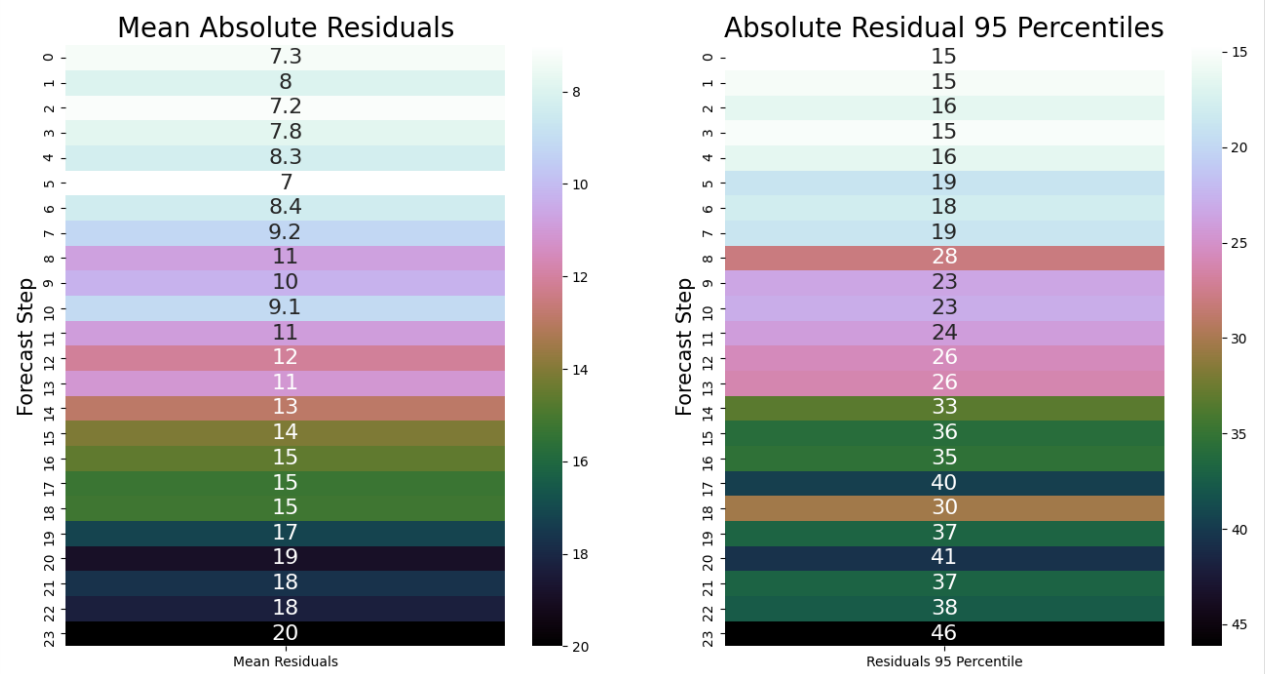


[18]:
```
fig, ax = plt.subplots(1,2,figsize=(16,8))
sns.heatmap(
    pd.DataFrame({'Mean Residuals':mat.mean().round(1)}),
    annot = True,
    cmap = 'cubehelix_r',
    ax = ax[0],
    annot_kws={"fontsize": 16},
)
cbar = ax[0].collections[0].colorbar
cbar.ax.invert_yaxis()
ax[0].set_title('Mean Absolute Residuals',size=20)
ax[0].set_ylabel('Forecast Step',size=15)
```

(continues on next page)

(continued from previous page)

```
ax[0].set_xlabel('')
sns.heatmap(
    pd.DataFrame({'Residuals 95 Percentile':np.percentile(mat, q=95, axis = 0)}),
    annot = True,
    cmap = 'cubehelix_r',
    ax = ax[1],
    annot_kws={"fontsize": 16},
)
cbar = ax[1].collections[0].colorbar
cbar.ax.invert_yaxis()
ax[1].set_title('Absolute Residual 95 Percentiles',size=20)
ax[1].set_ylabel('Forecast Step',size=15)
ax[1].set_xlabel('')
plt.show()
```

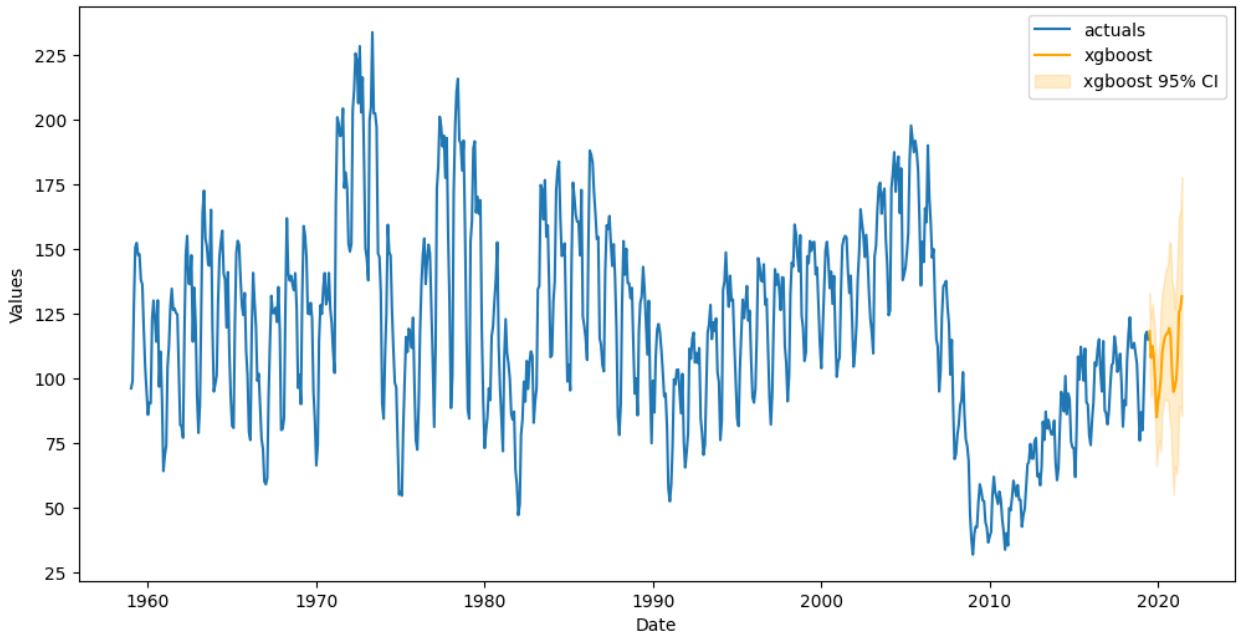


Each step of the forecast will be plus/minus the 95 percentile of the absolute residuals (plot on right).

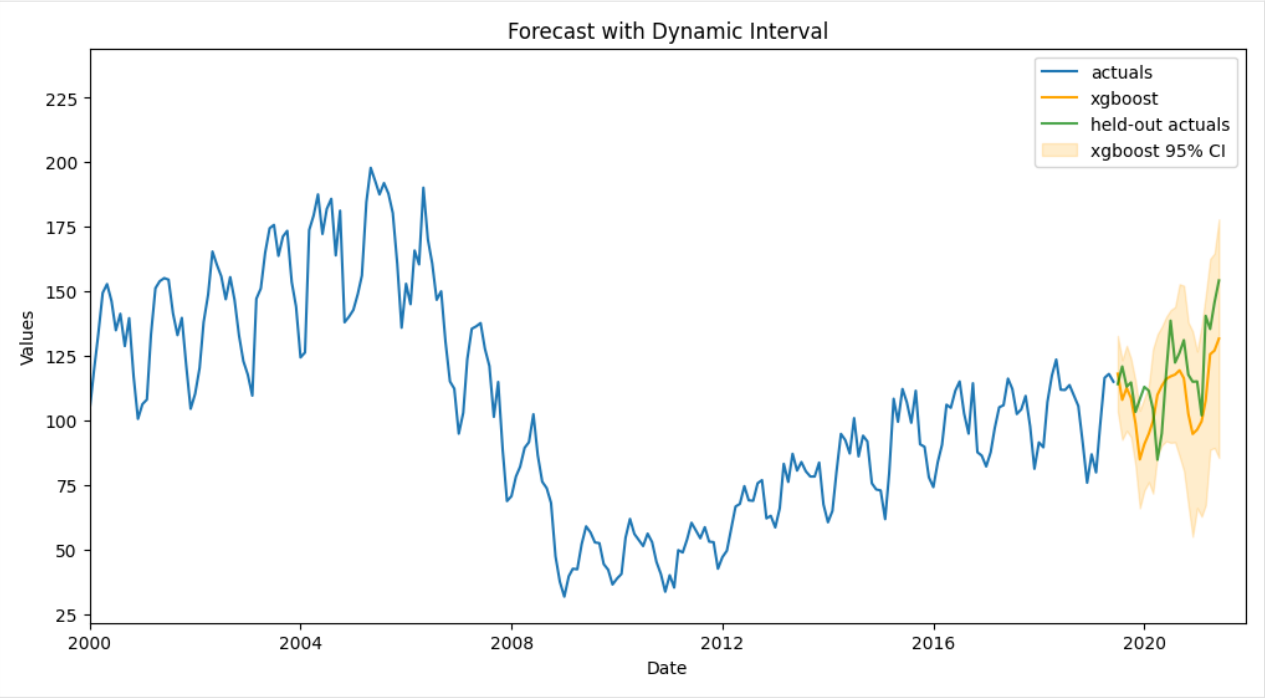
5.4 Step 4: Overwrite Naive Interval with Dynamic Interval

[19]:
```
overwrite_forecast_intervals(f,backtest_resid_matrix=backtest_resid_matrix)
```

[20]:
```
f.plot(ci=True);
```



[21]:
```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(ci=True,models='top_1',order_by='TestSetRMSE',ax=ax)
sns.lineplot(
    y = 'HOUSTNSA',
    x = 'DATE',
    data = starts_sep.reset_index(),
    ax = ax,
    label = 'held-out actuals',
    color = 'green',
    alpha = 0.7,
)
plt.xlim(pd.Timestamp('2000-01-01'),pd.Timestamp('2021-12-01'))
plt.title('Forecast with Dynamic Interval')
plt.show()
```



5.5 Score dynamic interval

[22]:
```
metrics.msis(
    a = starts_sep,
    uf = f.history['xgboost']['UpperCI'],
    lf = f.history['xgboost']['LowerCI'],
    obs = f.y,
    m = 12,
)
```

[22]:
```
3.919628517087574
```

It is a small improvement, but still an improvement!

[23]:
```
print(
    'The intervals are on average {:.2f} units away from the point predictions.'.format(
        np.mean(np.percentile(mat, q=95, axis = 0))
    )

)
print(
    'The interval contains {:.2%} of the actual values'.format(
        np.sum(
            [
                1 if a <= uf and a >= lf else 0
                for a, uf, lf in zip(
                    starts_sep,
                    f.history['xgboost']['UpperCI'],
                    f.history['xgboost']['LowerCI']
```

(continues on next page)

(continued from previous page)

```
                )
            ]
        ) / len(starts_sep)
    )
)
```

```
The intervals are on average 27.36 units away from the point predictions.
The interval contains 87.50% of the actual values
```

5.6 Other Backtest Uses

- We can also use the backtest results to report average error metrics over 20 out-of-sample sets.

[24]:
```
backtest_metrics(backtest_results,mets=['rmse','mae','bias'])[['Average']]
```

[24]:
```
                   Average
Model   Metric
xgboost rmse     14.083597
        mae      12.144780
        bias    239.430706
```

[25]:
```
# actual rmse on out-of-sample data
metrics.rmse(starts_sep,f.history['xgboost']['Forecast'])
```

[25]:
```
16.208569138706174
```

[ ]:
```

```

CHAPTER

SIX

COMBO MODELING

An introduction to the native ensemble/combo model in scalecast.

[1]:
```
import pandas as pd
import pandas_datareader as pdr
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from dateutil.relativedelta import relativedelta
from scalecast.Forecaster import Forecaster
from scalecast.SeriesTransformer import SeriesTransformer
from scalecast.Pipeline import Transformer, Reverter
from scalecast import GridGenerator
```

Download data from FRED (https://fred.stlouisfed.org/series/HOUSTNSA). This data is interesting due to its strong seasonality and irregular cycles. It measures monthly housing starts in the USA since 1959. Predicting this metric with some series that measures demand for houses could be an interesting extension to be able to explain housing prices. It is a common example series that scalecast uses.

[2]:
```
GridGenerator.get_example_grids(overwrite=True)
df = pdr.get_data_fred('HOUSTNSA',start='1900-01-01',end='2022-12-31')
f = Forecaster(
    y=df['HOUSTNSA'],
    current_dates=df.index,
    future_dates = 24,
    test_length = .1,
)
f
```

[2]:
```
Forecaster(
    DateStartActuals=1959-01-01T00:00:00.000000000
    DateEndActuals=2022-12-01T00:00:00.000000000
    Freq=MS
    N_actuals=768
    ForecastLength=24
    Xvars=[]
    TestLength=76
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
```

(continues on next page)

(continued from previous page)

```
)
```

6.1 Preprocess Data

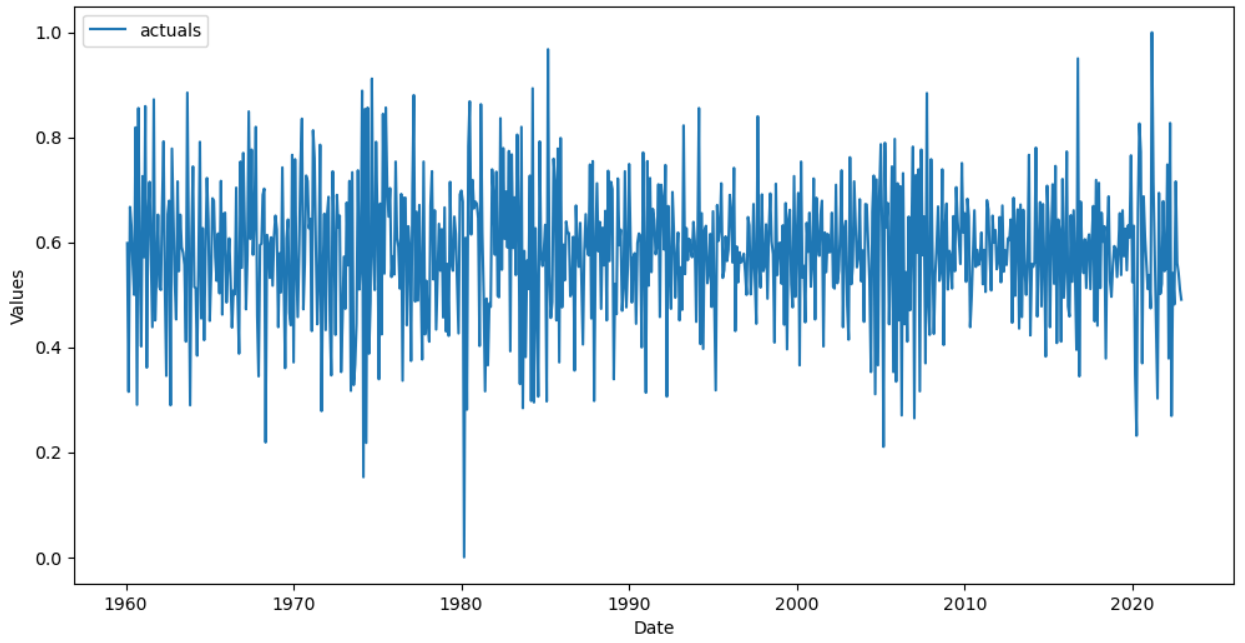
- Difference, seasonal difference, and scale the data.
- Create the object that will revert these transformations.

[3]:
```
# create transformations to model stationary data
transformer = Transformer(
    transformers = [
        ('DiffTransform',1),
        ('DiffTransform',12),
        ('MinMaxTransform',),
    ]
)

reverter = Reverter(
    reverters = [
        ('MinMaxRevert',),
        ('DiffRevert',12),
        ('DiffRevert',1)
    ],
    base_transformer = transformer,
)

reverter
```

[3]:
```
Reverter(
  reverters = [
    ('MinMaxRevert',),
    ('DiffRevert', 12),
    ('DiffRevert', 1)
  ],
  base_transformer = Transformer(
  transformers = [
    ('DiffTransform', 1),
    ('DiffTransform', 12),
    ('MinMaxTransform',)
  ]
)
)
```

[4]:
```
# transform the series by calling the Transformer.fit_transform() method
f = transformer.fit_transform(f)
```

[5]:
```
# plot the results
f.plot();
```



[6]:
```
# add regressors
f.add_ar_terms(24)
f
```

[6]:
```
Forecaster(
    DateStartActuals=1960-02-01T00:00:00.000000000
    DateEndActuals=2022-12-01T00:00:00.000000000
    Freq=MS
    N_actuals=755
    ForecastLength=24
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11',
↪ 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21', 'AR22
↪', 'AR23', 'AR24']
    TestLength=76
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

6.2 Evaluate Forecasting models

[7]:
```
# evaluate some models
f.tune_test_forecast(
    [
        'elasticnet',
        'lasso',
        'ridge',
```

(continues on next page)

(continued from previous page)

```
        'gbt',
        'lightgbm',
        'xgboost',
    ],
    dynamic_testing = 24,
    limit_grid_size = .2,
)
```

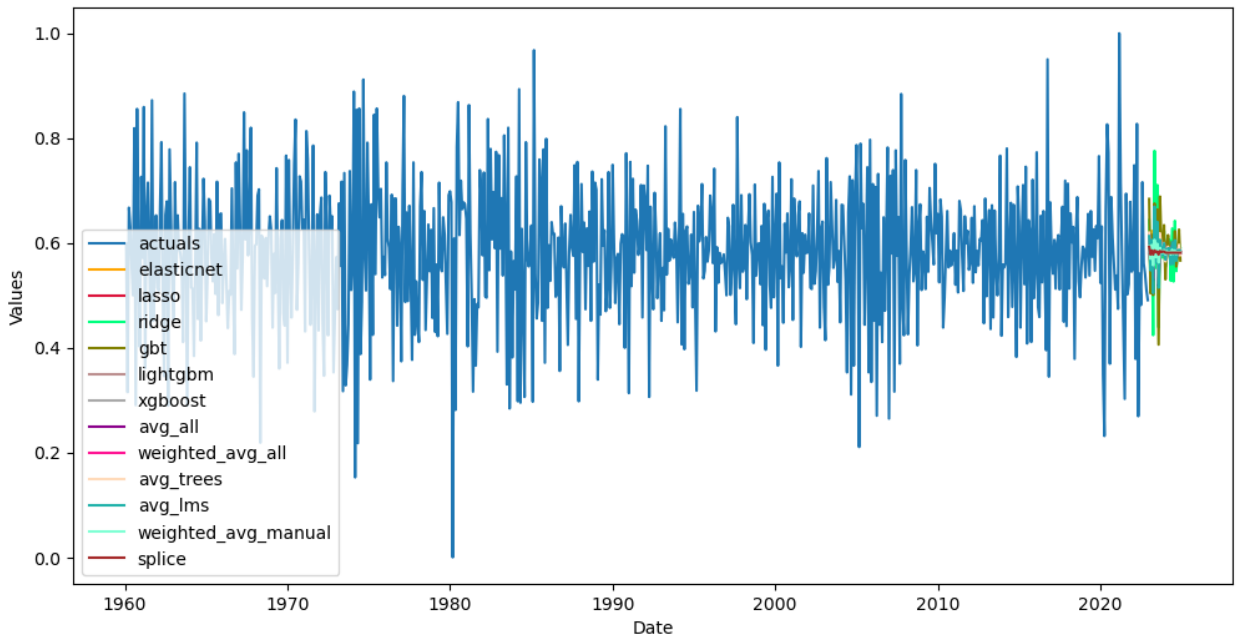
```
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
```

6.3 Combine Evaluated Models

- Below, we see several combination options that scalecast offers. These are just examples and not meant to try to find the absolute best combination for the data.

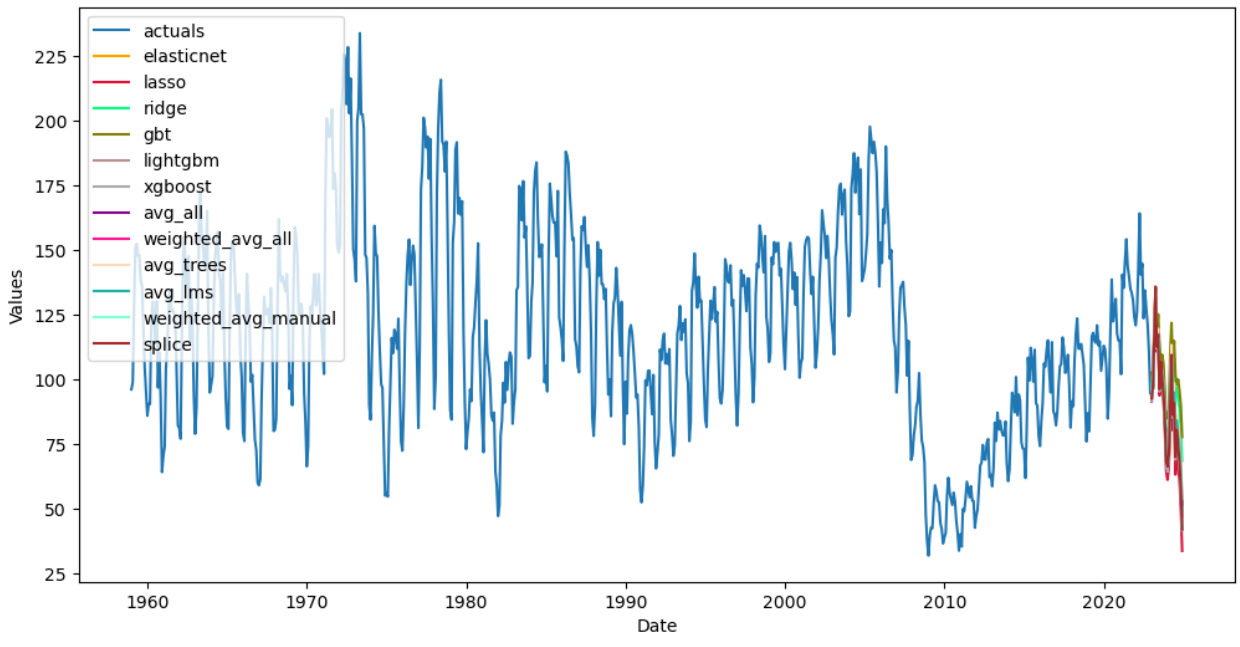
[8]:
```
f.set_estimator('combo')
# simple average of all models
f.manual_forecast(call_me = 'avg_all')
# weighted average of all models where the weights are determined from validation (not␣
↪test) performance
f.manual_forecast(
    how = 'weighted',
    determine_best_by = 'ValidationMetricValue',
    call_me = 'weighted_avg_all',
)
# simple average of a select set of models
f.manual_forecast(models = ['xgboost','gbt','lightgbm'],call_me = 'avg_trees')
# weighted average of a select set of models where the weights are determined from␣
↪validation (not test) performance
f.manual_forecast(models = ['elasticnet','lasso','ridge'],call_me = 'avg_lms')
# weighted average of a select set of models where the weights are manually passed
# weights do not have to add to 1 and they will be rebalanced to do so
f.manual_forecast(
    how = 'weighted',
    models = ['xgboost','elasticnet','lightgbm'],
    weights = (3,2,1),
    determine_best_by = None,
    call_me = 'weighted_avg_manual',
)
# splice (not many other libraries do this) - splice the future point forecasts of two␣
↪or more models together
f.manual_forecast(
    how='splice',
    models = ['elasticnet','lightgbm'],
    splice_points = ['2023-01-01'],
    call_me = 'splice',
)
```

[9]:
```
# plot the forecasts at the series' transformed level
f.plot();
```

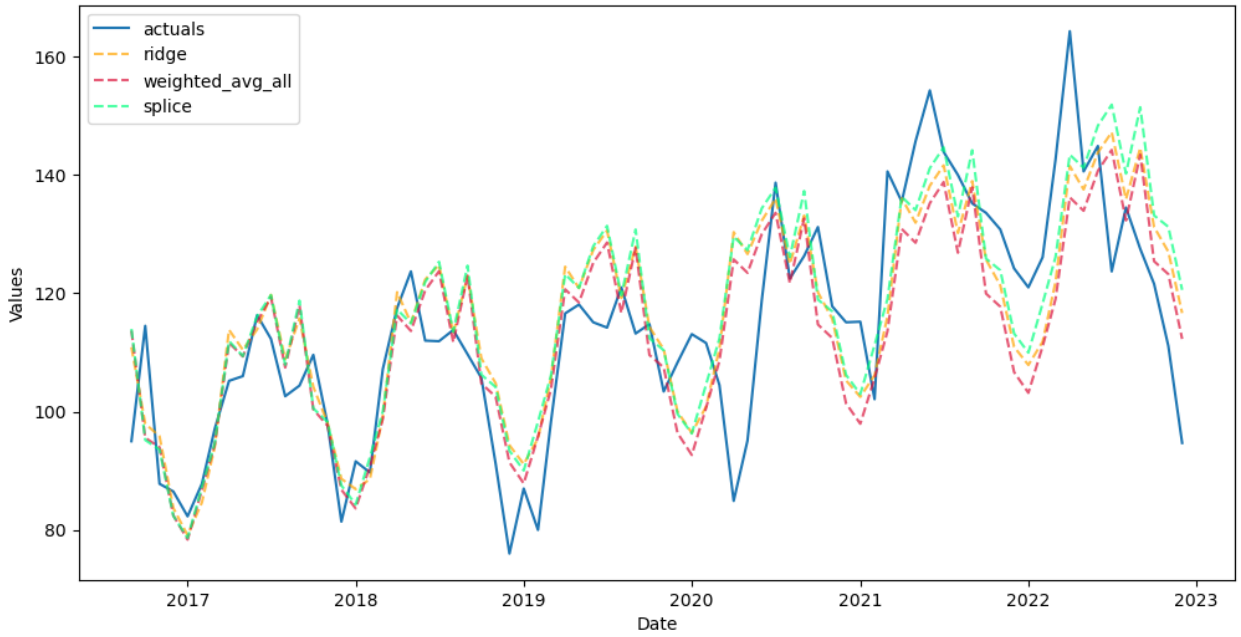


[10]:
```
# revert the transformation
f = reverter.fit_transform(f)
```

[11]:
```
# plot the forecasts at the series' original level
f.plot();
```



[12]:
```
# view performance on test set
f.plot_test_set(models = 'top_3',order_by = 'TestSetRMSE',include_train = False);
```



[ ]:

CHAPTER

SEVEN

CONFIDENCE INTERVALS

This notebook overviews scalecast intervals, which are scored with Mean Scaled Interval Score (MSIS). Lower scores are better. This notebook requires `scalecast>=0.18.0`.

Scalecast uses a naive conformal interval, created from holding out a test-set of data and setting interval ranges from the percentiles of the test-set residuals. This simple approach allows every estimator type to get the same kind of interval. As we will see by scoring the intervals and comparing them to ARIMA intervals from statsmodels, the results are good.

To evaluate the intervals, we leave out a section of each series to score out-of-sample. This is usually not necessary for scalecast, as all models are tested automatically, but the confidence intervals can overfit on any test set stored in the `Forecaster` or `MVForecaster` object due to leakage that occurs when constructing these intervals. Scalecast intervals are compared to ARIMA intervals on the same series in the last section of this notebook and the scalecast intervals, on the whole, perform better. The series used in this example are ordered from easiest-to-hardest to forecast.

Easy Distribution-Free Conformal Intervals for Time Series

Sections:

[1]:
```
import pandas as pd
import numpy as np
from scalecast.Forecaster import Forecaster
from scalecast import GridGenerator
from scalecast.util import metrics
from scalecast.notebook import tune_test_forecast
from scalecast.SeriesTransformer import SeriesTransformer
import matplotlib.pyplot as plt
import seaborn as sns
import time
from tqdm.notebook import tqdm
```

[2]:
```
import warnings
warnings.simplefilter(action='ignore', category=DeprecationWarning)
```

[3]:
```
models = (
    'mlr',
    'elasticnet',
    'ridge',
    'knn',
    'xgboost',
    'lightgbm',
    'gbt',
) # these are all scikit-learn models or APIs
# this will be used later to fill in results
results_template = pd.DataFrame(index=models)
```

[4]:
```
def score_cis(results, fcsts, ci_name, actuals, obs, val_len, models=models, m_=1):
    for m in models:
        results.loc[m,ci_name] = metrics.msis(
            a = actuals,
            uf = fcsts[m+'_upperci'],
            lf = fcsts[m+'_lowerci'],
            obs = obs,
            m = m_,
        )
    return results
```

[5]:
```
GridGenerator.get_example_grids()
```

7.1 Daily Website Visitors

- Link to data: https://www.kaggle.com/datasets/bobnau/daily-website-visitors
- We will use a length of 180 observations (about half a year) both to tune and test the models
- We want to optimize the forecasts and confidence intervals for a 60-day forecast horizon

[6]:
```
val_len = 180
fcst_len = 60
```

[7]:
```
data = pd.read_csv('daily-website-visitors.csv',parse_dates=['Date']).set_index('Date')
data.head()
```

[7]:

Date	Row	Day	Day.Of.Week	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
2014-09-14	1	Sunday	1	2,146	1,582	1430	152
2014-09-15	2	Monday	2	3,621	2,528	2297	231
2014-09-16	3	Tuesday	3	3,698	2,630		
2014-09-17	4	Wednesday	4	3,667	2,614		
2014-09-18	5	Thursday	5	3,316	2,366		

(continues on next page)

(continued from previous page)

```
2014-09-16              2352              278
2014-09-17              2327              287
2014-09-18              2130              236
```

[8]:
```
visits_sep = data['First.Time.Visits'].iloc[-fcst_len:]
visits = data['First.Time.Visits'].iloc[:-fcst_len]
```

[9]:
```
f=Forecaster(
    y=visits,
    current_dates=visits.index,
    future_dates=fcst_len,
    test_length = val_len,
    validation_length = val_len, # for hyperparameter tuning
    cis = True, # set to True at initialization to always evaluate cis
)
f.auto_Xvar_select(
    estimator='elasticnet',
    alpha=.2,
    max_ar=100,
    monitor='ValidationMetricValue',
)
f
```

[9]:
```
Forecaster(
    DateStartActuals=2014-09-14T00:00:00.000000000
    DateEndActuals=2020-06-20T00:00:00.000000000
    Freq=D
    N_actuals=2107
    ForecastLength=60
    Xvars=['t', 'AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10',
↪'AR11', 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21',
↪ 'AR22', 'AR23', 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30', 'AR31', 'AR32
↪', 'AR33', 'AR34', 'AR35', 'AR36', 'AR37', 'AR38', 'AR39', 'AR40', 'AR41', 'AR42',
↪'AR43', 'AR44', 'AR45', 'AR46', 'AR47', 'AR48', 'AR49', 'AR50', 'AR51', 'AR52', 'AR53',
↪ 'AR54', 'AR55', 'AR56', 'AR57', 'AR58', 'AR59', 'AR60', 'AR61', 'AR62', 'AR63', 'AR64
↪', 'AR65', 'AR66', 'AR67', 'AR68', 'AR69', 'AR70', 'AR71', 'AR72', 'AR73', 'AR74',
↪'AR75', 'AR76', 'AR77', 'AR78', 'AR79', 'AR80', 'AR81', 'AR82', 'AR83', 'AR84', 'AR85',
↪ 'AR86', 'AR87', 'AR88', 'AR89', 'AR90', 'AR91', 'AR92', 'AR93', 'AR94', 'AR95', 'AR96
↪', 'AR97', 'AR98', 'AR99', 'AR100']
    TestLength=180
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[10]:
```
tune_test_forecast(
    f,
    models,
)
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
Finished loading model, total used 250 iterations
```

[11]:
```
ms = f.export('model_summaries',determine_best_by='TestSetRMSE')
ms[['ModelNickname','TestSetRMSE','InSampleRMSE']]
```

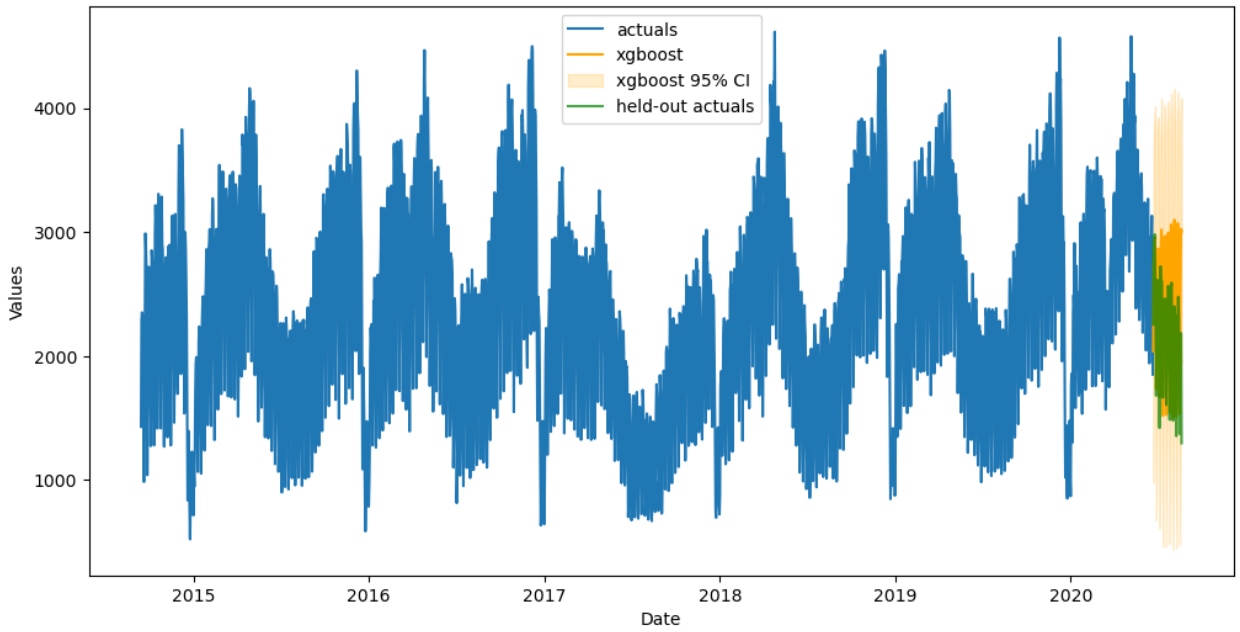
[11]:

	ModelNickname	TestSetRMSE	InSampleRMSE
0	xgboost	532.374846	14.496801
1	lightgbm	582.261645	112.543232
2	knn	788.704950	306.828988
3	elasticnet	794.649818	186.616620
4	ridge	798.711718	187.026325
5	mlr	803.931314	185.620629
6	gbt	1171.782785	147.478748

We will demonstrate how the confidence intervals change as they are re-evaluated using the best model according to the test RMSE.

7.1.1 Evaluate Interval

[12]:
```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(ci=True,models='top_1',order_by='TestSetRMSE',ax=ax)
sns.lineplot(
    y = 'First.Time.Visits',
    x = 'Date',
    data = visits_sep.reset_index(),
    ax = ax,
    label = 'held-out actuals',
    color = 'green',
    alpha = 0.7,
)
plt.show()
```



[13]:
```
# export test-set preds and confidence intervals
fcsts1 = f.export("lvl_fcsts",cis=True)
fcsts1.head()
```

[13]:
```
        DATE          mlr  mlr_upperci  mlr_lowerci   elasticnet  \
0 2020-06-21  2190.555279  3464.907755   916.202804  2194.962019
1 2020-06-22  2835.674158  4110.026633  1561.321683  2823.011263
2 2020-06-23  2975.510723  4249.863198  1701.158248  2944.846465
3 2020-06-24  3038.258157  4312.610632  1763.905681  3006.932564
4 2020-06-25  3032.580568  4306.933043  1758.228093  3011.122478

   elasticnet_upperci  elasticnet_lowerci        ridge  ridge_upperci  \
0         3452.970789          936.953250  2191.868075    3464.239858
1         4081.020033         1565.002493  2831.886812    4104.258594
2         4202.855235         1686.837696  2972.183687    4244.555469
3         4264.941333         1748.923794  3033.992818    4306.364600
4         4269.131247         1753.113708  3021.274091    4293.645873

   ridge_lowerci  ...  knn_lowerci      xgboost  xgboost_upperci  \
0     919.496293  ...   812.176316  2039.648438      3098.496887
1    1559.515029  ...  1612.281579  2743.368896      3802.217346
2    1699.811904  ...  1804.176316  2851.803711      3910.652161
3    1761.621035  ...  1673.860526  2905.132568      3963.981018
4    1748.902309  ...  1475.386842  2955.912598      4014.761047

   xgboost_lowerci     lightgbm  lightgbm_upperci  lightgbm_lowerci  \
0       980.799988  2198.858468       3232.664729       1165.052207
1      1684.520447  2820.575637       3854.381898       1786.769376
2      1792.955261  2819.654176       3853.460437       1785.847915
3      1846.284119  2824.992206       3858.798467       1791.185945
4      1897.064148  2961.588543       3995.394805       1927.782282
```

(continues on next page)

(continued from previous page)

```
           gbt  gbt_upperci  gbt_lowerci
0  2125.968510  4021.476092   230.460929
1  2729.050006  4624.557587   833.542425
2  2880.262061  4775.769642   984.754480
3  2930.293895  4825.801476  1034.786314
4  3029.007375  4924.514956  1133.499794

[5 rows x 22 columns]
```

The values in the below table are mean scaled interval scores for confidence intervals. Lower scores are better.

[14]:
```
results = score_cis(
    results_template.copy(),
    fcsts1,
    'Daily Visitors',
    visits_sep,
    visits,
    val_len = val_len,
)
results
```

[14]:
```
            Daily Visitors
mlr               5.632456
elasticnet        5.548336
ridge             5.622506
knn               5.639715
xgboost           5.597762
lightgbm          5.196714
gbt               8.319963
```

7.2 Housing Starts

- Link to data: https://fred.stlouisfed.org/series/HOUSTNSA
- We will use a length of 96 observations (8 years) both to tune and test the models
- We want to optimize the forecasts and confidence intervals for a 24-month forecast horizon

[15]:
```
import pandas_datareader as pdr
```

[16]:
```
val_len = 96
fcst_len = 24
```

[17]:
```
housing = pdr.get_data_fred('HOUSTNSA',start='1900-01-01',end='2021-06-01')
housing.head()
```

[17]:
```
            HOUSTNSA
DATE
1959-01-01      96.2
1959-02-01      99.0
1959-03-01     127.7
```

(continues on next page)

(continued from previous page)

```
1959-04-01     150.8
1959-05-01     152.5
```

[18]:
```
starts_sep = housing.iloc[-fcst_len:,0]
starts = housing.iloc[:-fcst_len,0]
```

[19]:
```
f = Forecaster(
    y=starts,
    current_dates=starts.index,
    future_dates=fcst_len,
    test_length=val_len,
    validation_length=val_len,
    cis=True,
)
# difference the data for stationary modeling
transformer = SeriesTransformer(f)
f = transformer.DiffTransform(1)
# find best xvars to forecast with
f.auto_Xvar_select(
    estimator='elasticnet',
    alpha=.2,
    max_ar=100,
    monitor='ValidationMetricValue', # not test set
)
f
```

[19]:
```
Forecaster(
    DateStartActuals=1959-02-01T00:00:00.000000000
    DateEndActuals=2019-06-01T00:00:00.000000000
    Freq=MS
    N_actuals=725
    ForecastLength=24
    Xvars=['monthsin', 'monthcos', 'quartersin', 'quartercos', 'AR1', 'AR2', 'AR3', 'AR4
↪', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11', 'AR12', 'AR13', 'AR14', 'AR15',
↪'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21', 'AR22', 'AR23', 'AR24', 'AR25', 'AR26',
↪ 'AR27', 'AR28', 'AR29', 'AR30', 'AR31', 'AR32', 'AR33', 'AR34', 'AR35', 'AR36', 'AR37
↪', 'AR38', 'AR39', 'AR40', 'AR41', 'AR42', 'AR43', 'AR44', 'AR45', 'AR46', 'AR47',
↪'AR48', 'AR49', 'AR50', 'AR51', 'AR52', 'AR53', 'AR54', 'AR55', 'AR56', 'AR57', 'AR58',
↪ 'AR59', 'AR60', 'AR61', 'AR62', 'AR63', 'AR64', 'AR65', 'AR66', 'AR67', 'AR68', 'AR69
↪', 'AR70', 'AR71', 'AR72', 'AR73', 'AR74', 'AR75', 'AR76', 'AR77', 'AR78', 'AR79',
↪'AR80', 'AR81', 'AR82', 'AR83', 'AR84', 'AR85', 'AR86', 'AR87', 'AR88', 'AR89', 'AR90',
↪ 'AR91', 'AR92', 'AR93', 'AR94', 'AR95', 'AR96', 'AR97', 'AR98', 'AR99', 'AR100']
    TestLength=96
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[20]:
```
tune_test_forecast(
    f,
```

(continues on next page)

(continued from previous page)

```
    models,
    dynamic_testing = fcst_len,
)
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
```

[21]:
```
# rever the difftransform
f = transformer.DiffRevert(1)
```

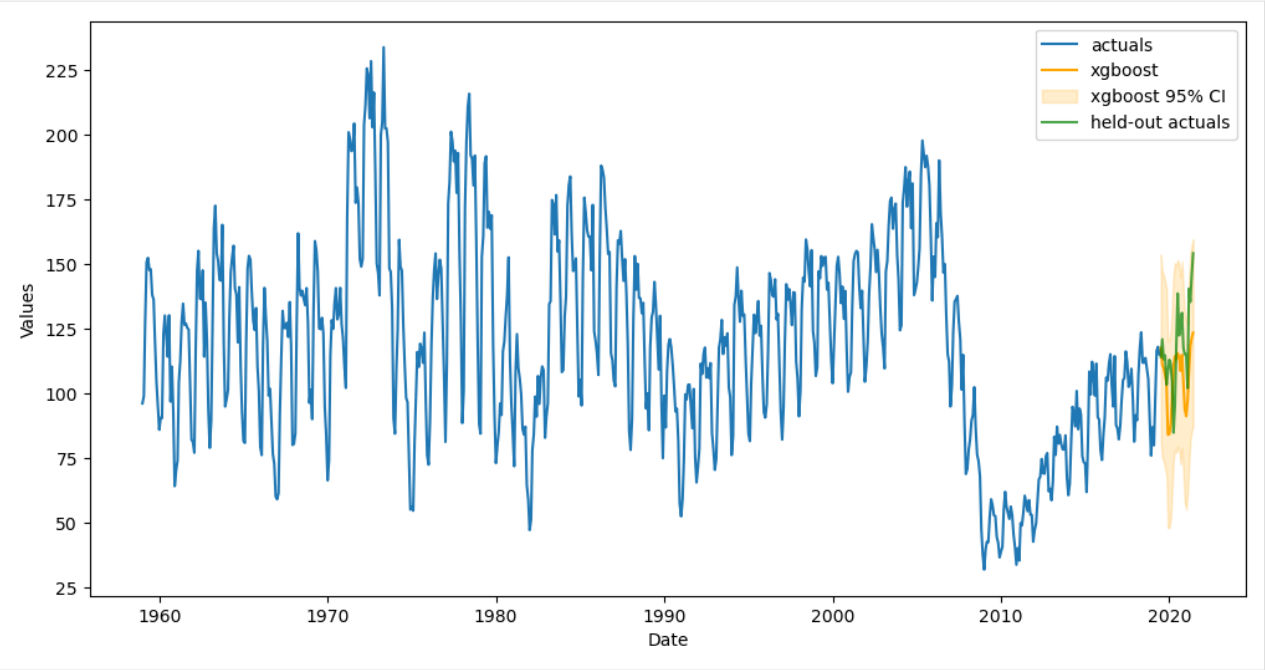
[22]:
```
ms = f.export('model_summaries',determine_best_by='TestSetRMSE')
ms[['ModelNickname','TestSetRMSE','InSampleRMSE']]
```

[22]:

	ModelNickname	TestSetRMSE	InSampleRMSE
0	xgboost	21.402469	3.112793
1	lightgbm	23.160361	27.660879
2	mlr	30.059747	71.624715
3	gbt	38.185841	38.657789
4	knn	42.432505	146.502991
5	ridge	46.457215	55.969584
6	elasticnet	53.111686	38.237499

7.2.1 Evaluate Interval

[23]:
```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(ci=True,models='top_1',order_by='TestSetRMSE',ax=ax)
sns.lineplot(
    y = 'HOUSTNSA',
    x = 'DATE',
    data = starts_sep.reset_index(),
    ax = ax,
    label = 'held-out actuals',
    color = 'green',
    alpha = 0.7,
)
plt.show()
```



[24]:
```
housing_fcsts1 = f.export("lvl_fcsts",cis=True)
housing_results = score_cis(
    results_template.copy(),
    housing_fcsts1,
    'Housing Starts',
    starts_sep,
    starts,
    val_len = val_len,
    m_ = 12, # monthly seasonality
)
housing_results
```

[24]:

	Housing Starts
mlr	5.661675
elasticnet	8.247183
ridge	7.432564
knn	6.552641
xgboost	3.930183
lightgbm	4.289493
gbt	5.884677

[25]:
```
results['Housing Starts'] = housing_results['Housing Starts']
results
```

[25]:

	Daily Visitors	Housing Starts
mlr	5.632456	5.661675
elasticnet	5.548336	8.247183
ridge	5.622506	7.432564
knn	5.639715	6.552641
xgboost	5.597762	3.930183
lightgbm	5.196714	4.289493
gbt	8.319963	5.884677

7.3 Avocado Sales

- Link to data: https://www.kaggle.com/datasets/neuromusic/avocado-prices
- We will use a length of 20 observations both to tune and test the models (95% CIs require at least 20 observations in the test set)
- We want to optimize the forecasts and confidence intervals for a 20-week forecast horizon

[26]:
```
# change display settings
pd.options.display.float_format = '{:,.2f}'.format
```

[27]:
```
val_len = 20
fcst_len = 20
```

[28]:
```
avocados = pd.read_csv('avocado.csv',parse_dates = ['Date'])
volume = avocados.groupby('Date')['Total Volume'].sum()
```

[29]:
```
volume.reset_index().head()
```

[29]:
```
        Date   Total Volume
0 2015-01-04  84,674,337.20
1 2015-01-11  78,555,807.24
2 2015-01-18  78,388,784.08
3 2015-01-25  76,466,281.07
4 2015-02-01 119,453,235.25
```

[30]:
```
volume_sep = volume.iloc[-fcst_len:]
volume = volume.iloc[:-fcst_len]
```

[31]:
```
f = Forecaster(
    y = volume,
    current_dates = volume.index,
    future_dates = fcst_len,
    test_length = val_len,
    validation_length = val_len,
    cis = True,
)

# difference the data for stationary modeling
transformer = SeriesTransformer(f)
f = transformer.DiffTransform(1)
f = transformer.DiffTransform(52) # seasonal differencing
# find best xvars
f.auto_Xvar_select(
    estimator='elasticnet',
    alpha=.2,
    max_ar=26,
    monitor='ValidationMetricValue', # not test set
    decomp_trend=False,
)
f
```

[31]:
```
Forecaster(
    DateStartActuals=2016-01-10T00:00:00.000000000
    DateEndActuals=2017-11-05T00:00:00.000000000
    Freq=W-SUN
    N_actuals=96
    ForecastLength=20
    Xvars=['weeksin', 'weekcos', 'AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8',
↪'AR9']
    TestLength=20
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[32]:
```
tune_test_forecast(
    f,
    models,
    dynamic_testing = fcst_len,
)
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
```

[33]:
```
# revert differencing
f = transformer.DiffRevert(52)
f = transformer.DiffRevert(1)
```

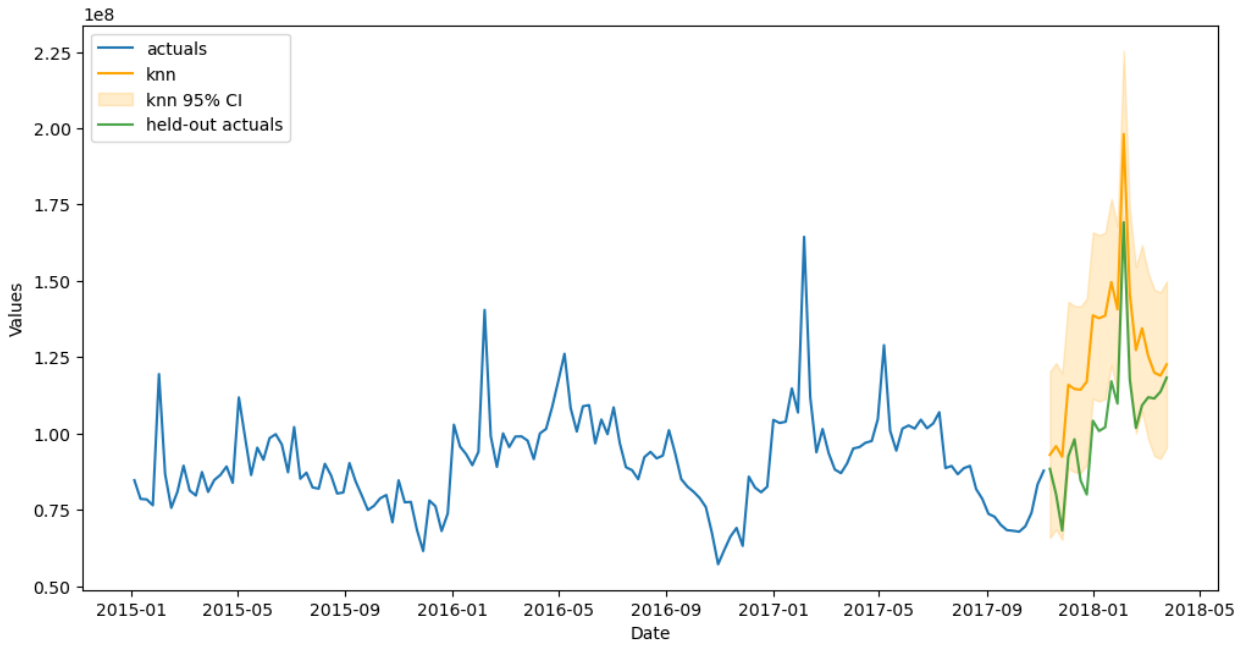
[34]:
```
ms = f.export('model_summaries',determine_best_by='TestSetRMSE')
ms[['ModelNickname','TestSetRMSE','InSampleRMSE']]
```

[34]:

	ModelNickname	TestSetRMSE	InSampleRMSE
0	knn	12,792,100.70	14,090,078.03
1	ridge	13,457,205.75	23,224,298.55
2	elasticnet	13,532,378.91	23,130,014.21
3	mlr	13,532,379.31	23,130,012.59
4	lightgbm	15,265,692.24	21,365,608.04
5	gbt	20,539,987.16	18,023,127.21
6	xgboost	21,487,561.26	12,876.01

7.3.1 Evaluate Interval

[35]:
```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(ci=True,models='top_1',order_by='TestSetRMSE',ax=ax)
sns.lineplot(
    y = 'Total Volume',
    x = 'Date',
    data = volume_sep.reset_index(),
    ax = ax,
    label = 'held-out actuals',
    color = 'green',
    alpha = 0.7,
)
plt.show()
```



[36]:
```
avc_fcsts1 = f.export("lvl_fcsts",cis=True)
avc_results = score_cis(
    results_template.copy(),
    avc_fcsts1,
    'Avocados',
    volume_sep,
    volume,
    val_len = val_len,
    models = models,
)
avc_results
```

[36]:

	Avocados
mlr	8.55
elasticnet	8.55
ridge	8.48
knn	19.70

(continues on next page)

(continued from previous page)

```
xgboost        13.74
lightgbm       19.81
gbt            10.65
```

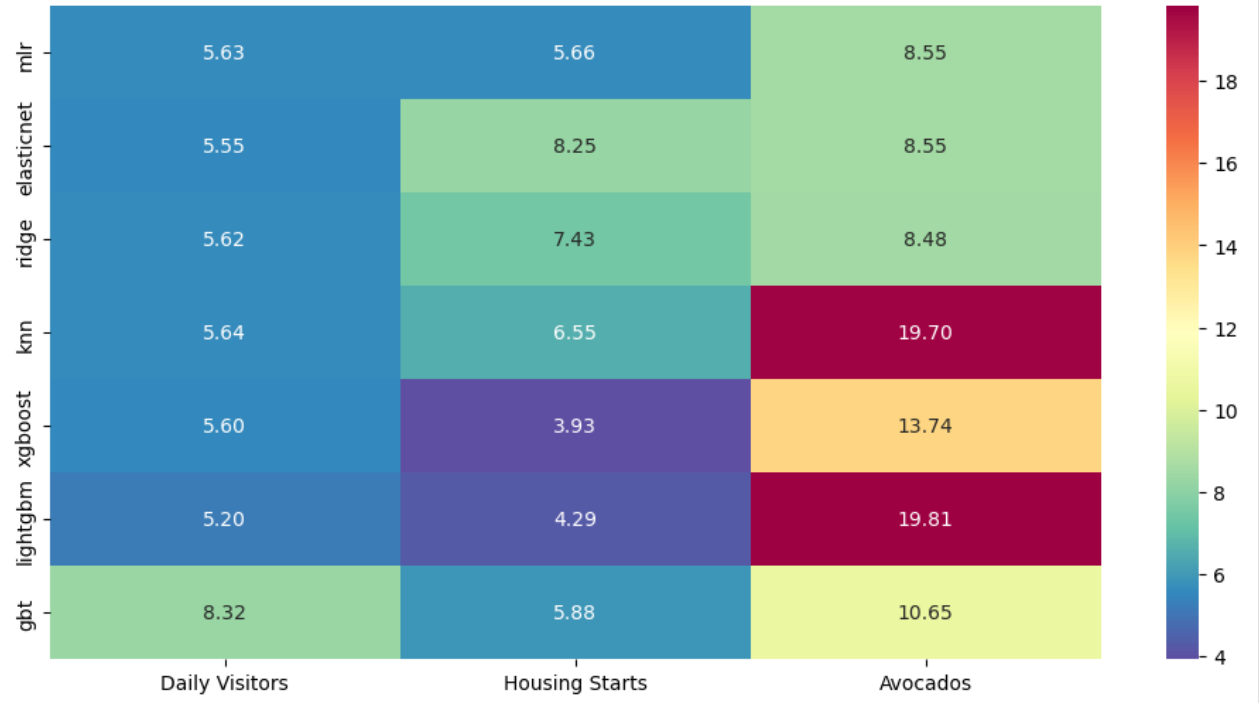
[37]:
```
results['Avocados'] = avc_results['Avocados']
results
```

[37]:

	Daily Visitors	Housing Starts	Avocados
mlr	5.63	5.66	8.55
elasticnet	5.55	8.25	8.55
ridge	5.62	7.43	8.48
knn	5.64	6.55	19.70
xgboost	5.60	3.93	13.74
lightgbm	5.20	4.29	19.81
gbt	8.32	5.88	10.65

7.4 All Aggregated Results

[38]:
```
_, ax = plt.subplots(figsize=(12,6))
sns.heatmap(
    results,
    annot=True,
    fmt='.2f',
    cmap="Spectral_r",
    ax=ax
)
plt.show()
```



For the most part, the linear models set the best intervals and the boosted tree models were very good or very bad, with GBT more towards the bad side.

7.5 Benchmark Against StatsModels ARIMA

- Confidence intervals come from StatsModels but the auto-ARIMA process is from PMDARIMA.

[39]:
```
from scalecast.auxmodels import auto_arima
```

[40]:
```
all_series = {
    # series,out-of-sample series,seasonal step
    'visitors':[visits,visits_sep,1],
    'housing starts':[starts,starts_sep,12],
    'avocados':[volume,volume_sep,1]
}
arima_conformal_results = pd.DataFrame()
arima_sm_results = pd.DataFrame()
```

[41]:
```
for k, v in all_series.items():
    print(k)
    f = Forecaster(
        y=v[0],
        current_dates=v[0].index,
        future_dates=len(v[1]),
        test_length = len(v[1]),
        cis=True,
    )
    auto_arima(f,m=v[2])
    arima_results = f.export("lvl_fcsts",cis=True)
    # scalecast intervals
    arima_conformal_results.loc[k,'MSIS'] = metrics.msis(
        a = v[1].values,
        uf = arima_results['auto_arima_upperci'].values,
        lf = arima_results['auto_arima_lowerci'].values,
        obs = v[0].values,
        m = v[2],
    )
    # statsmodels intervals
    cis = f.regr.get_forecast(len(v[1])).conf_int()
    arima_sm_results.loc[k,'MSIS'] = metrics.msis(
        a = v[1].values,
        uf = cis.T[1],
        lf = cis.T[0],
        obs = v[0].values,
        m = v[2],
    )
```

```
visitors
housing starts
avocados
```

7.5.1 MSIS Results - ARIMA Scalecast

[42]:
```
# results from the scalecast intervals
arima_conformal_results
```

[42]:
```
                MSIS
visitors        4.97
housing starts 24.90
avocados       23.15
```

[43]:
```
arima_conformal_results.mean()
```

[43]:
```
MSIS   17.67
dtype: float64
```

7.5.2 MSIS Results - ARIMA StatsModels

[44]:
```
# results from the statsmodels intervals
arima_sm_results
```

[44]:
```
                MSIS
visitors        5.80
housing starts  5.62
avocados       19.94
```

[45]:
```
arima_sm_results.mean()
```

[45]:
```
MSIS   10.46
dtype: float64
```

[46]:
```
all_results = results.copy()
all_results.loc['arima (conformal)'] =  arima_conformal_results.T.values[0]
all_results.loc['arima (statsmodels) - benchmark'] = arima_sm_results.T.values[0]
all_results = all_results.T
all_results
```

[46]:
```
                mlr  elasticnet  ridge   knn  xgboost  lightgbm   gbt  \
Daily Visitors 5.63        5.55   5.62  5.64     5.60      5.20  8.32
Housing Starts 5.66        8.25   7.43  6.55     3.93      4.29  5.88
Avocados       8.55        8.55   8.48 19.70    13.74     19.81 10.65

                arima (conformal)  arima (statsmodels) - benchmark
Daily Visitors               4.97                             5.80
Housing Starts              24.90                             5.62
Avocados                    23.15                            19.94
```

[47]:
```
def highlight_rows(row):
    ret_row = ['']*all_results.shape[1]
    for i, c in enumerate(all_results.iloc[:,:-1]):
        if row[c] < row['arima (statsmodels) - benchmark']:
            ret_row[i] = 'background-color: lightgreen;'
        else:
```

(continues on next page)

(continued from previous page)

```
            ret_row[i] = 'background-color: lightcoral;'
    return ret_row

all_results.style.apply(
    highlight_rows,
    axis=1,
)
```

[47]:
```
<pandas.io.formats.style.Styler at 0x7ff0e916eee0>
```

The above table shows which scalecast intervals performed better or worse than the ARIMA interval. Green scores are better, red are worse. The scalecast conformal intervals were on the whole better, but not always. When comparing ARIMA to ARIMA, the confidence intervals from statsmodels performed slightly worse on one series, slightly better on another, and significantly better on the remaining one. The rest of the models run through scalecast usually beat the ARIMA intervals on all datasets, except housing starts. XGBoost and Lightgbm were the only models to always beat the intervals from StatsModels. On the whole, the scalecast interval has a nice showing against the more traditional interval from ARIMA in the StatsModels package.

[ ]:

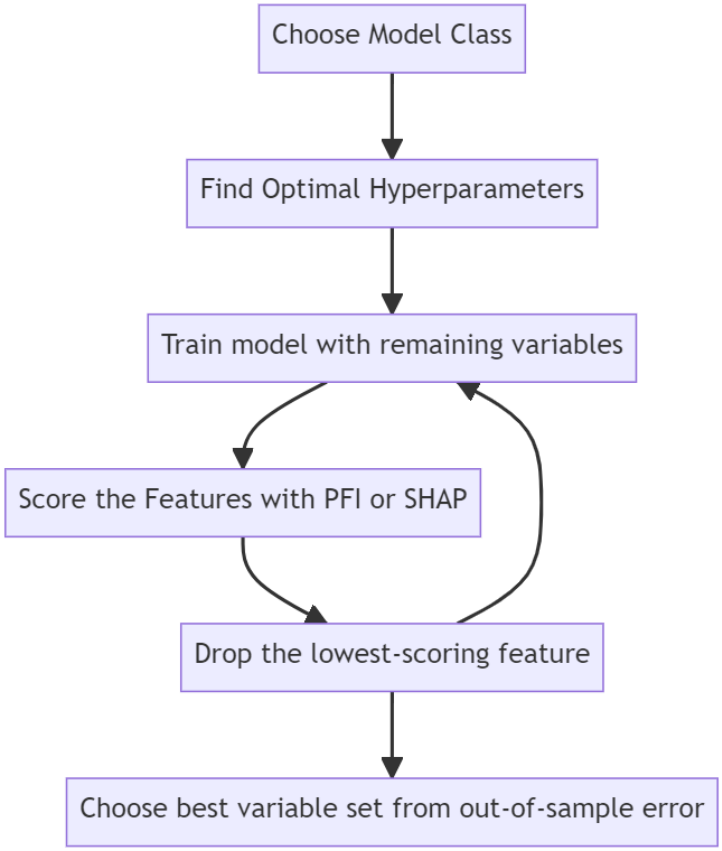
CHAPTER

EIGHT

FEATURE REDUCTION

This notebook demonstrates how to use various feature selection methods using scalecast. The idea behind this example is to reduce the variables stored in the `Forecaster` object to an optimal subset after adding many potentially helpful features to the object.

- Link to data
- See the blog post
- Function documentation link



[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

(continues on next page)

(continued from previous page)

```
from scalecast.Forecaster import Forecaster
from scalecast.util import plot_reduction_errors
```

[2]:
```
sns.set(rc={"figure.figsize": (12, 8)})
```

[3]:
```
def prepare_fcst(f, test_length=120, fcst_length=120, validation_length=120):
    """ adds all variables and sets the test length/forecast length in the object

    Args:
        f (Forecaster): the Forecaster object.
        test_length (int or float): the test length as a size or proportion.
        fcst_length (int): the forecast horizon.

    Returns:
        (Forecaster) the processed object.
    """
    f.generate_future_dates(fcst_length)
    f.set_test_length(test_length)
    f.set_validation_length(validation_length)
    f.add_seasonal_regressors("month", "quarter", raw=False, sincos=True)
    f.set_validation_metric("mae")
    for i in np.arange(12, 289, 12):  # 12-month cycles from 12 to 288 months
        f.add_cycle(i)
    f.add_ar_terms(120)  # AR 1-120
    f.add_AR_terms((20, 12))  # seasonal AR up to 20 years, spaced one year apart
    f.add_time_trend()
    f.add_seasonal_regressors("year")
    return f


def export_results(f):
    """ returns a dataframe with all model results given a Forecaster object.

    Args:
        f (Forecaster): the Forecaster object.

    Returns:
        (DataFrame) the dataframe with the pertinent results.
    """
    results = f.export("model_summaries", determine_best_by="TestSetMAE")
    results["N_Xvars"] = results["Xvars"].apply(lambda x: len(x))
    return results[
        [
            "ModelNickname",
            "TestSetMAE",
            "InSampleMAE",
            "TestSetR2",
            "InSampleR2",
            "DynamicallyTested",
            "N_Xvars",
        ]
    ]
```

8.1 Load Forecaster Object

- we choose 120 periods (10 years) for all validation and forecasting
- 10 years of observervations to tune model hyperparameters, 10 years to test, and a forecast horizon of 10 years

[4]:
```
df = pd.read_csv("Sunspots.csv", index_col=0, names=["Date", "Target"], header=0)
f = Forecaster(y=df["Target"], current_dates=df["Date"])
prepare_fcst(f)
```

[4]:
```
Forecaster(
    DateStartActuals=1749-01-31T00:00:00.000000000
    DateEndActuals=2021-01-31T00:00:00.000000000
    Freq=M
    N_actuals=3265
    ForecastLength=120
    Xvars=['monthsin', 'monthcos', 'quartersin', 'quartercos', 'cycle12sin', 'cycle12cos
↪', 'cycle24sin', 'cycle24cos', 'cycle36sin', 'cycle36cos', 'cycle48sin', 'cycle48cos',
↪'cycle60sin', 'cycle60cos', 'cycle72sin', 'cycle72cos', 'cycle84sin', 'cycle84cos',
↪'cycle96sin', 'cycle96cos', 'cycle108sin', 'cycle108cos', 'cycle120sin', 'cycle120cos',
↪ 'cycle132sin', 'cycle132cos', 'cycle144sin', 'cycle144cos', 'cycle156sin',
↪'cycle156cos', 'cycle168sin', 'cycle168cos', 'cycle180sin', 'cycle180cos', 'cycle192sin
↪', 'cycle192cos', 'cycle204sin', 'cycle204cos', 'cycle216sin', 'cycle216cos',
↪'cycle228sin', 'cycle228cos', 'cycle240sin', 'cycle240cos', 'cycle252sin', 'cycle252cos
↪', 'cycle264sin', 'cycle264cos', 'cycle276sin', 'cycle276cos', 'cycle288sin',
↪'cycle288cos', 'AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10',
↪'AR11', 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21',
↪ 'AR22', 'AR23', 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30', 'AR31', 'AR32
↪', 'AR33', 'AR34', 'AR35', 'AR36', 'AR37', 'AR38', 'AR39', 'AR40', 'AR41', 'AR42',
↪'AR43', 'AR44', 'AR45', 'AR46', 'AR47', 'AR48', 'AR49', 'AR50', 'AR51', 'AR52', 'AR53',
↪ 'AR54', 'AR55', 'AR56', 'AR57', 'AR58', 'AR59', 'AR60', 'AR61', 'AR62', 'AR63', 'AR64
↪', 'AR65', 'AR66', 'AR67', 'AR68', 'AR69', 'AR70', 'AR71', 'AR72', 'AR73', 'AR74',
↪'AR75', 'AR76', 'AR77', 'AR78', 'AR79', 'AR80', 'AR81', 'AR82', 'AR83', 'AR84', 'AR85',
↪ 'AR86', 'AR87', 'AR88', 'AR89', 'AR90', 'AR91', 'AR92', 'AR93', 'AR94', 'AR95', 'AR96
↪', 'AR97', 'AR98', 'AR99', 'AR100', 'AR101', 'AR102', 'AR103', 'AR104', 'AR105', 'AR106
↪', 'AR107', 'AR108', 'AR109', 'AR110', 'AR111', 'AR112', 'AR113', 'AR114', 'AR115',
↪'AR116', 'AR117', 'AR118', 'AR119', 'AR120', 'AR132', 'AR144', 'AR156', 'AR168', 'AR180
↪', 'AR192', 'AR204', 'AR216', 'AR228', 'AR240', 't', 'year']
    TestLength=120
    ValidationMetric=mae
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[5]:
```
print("starting out with {} variables".format(len(f.get_regressor_names())))
```

```
starting out with 184 variables
```

8.2 Reducing with L1 Regularization

- L1 regularization reduces out least important variables using a lasso model
- Not computationally expensive
- `overwrite=False` is used to not replace variables currently in the `Forecaster` object

[6]:
```
f.reduce_Xvars(overwrite=False, dynamic_testing=False)
```

[7]:
```
lasso_reduced_vars = f.reduced_Xvars[:]
print(f"lasso reduced to {len(lasso_reduced_vars)} variables")
```

```
lasso reduced to 59 variables
```

[8]:
```
print(*lasso_reduced_vars, sep="\n")
```

```
monthcos
quartersin
cycle12cos
cycle24sin
cycle24cos
cycle36cos
cycle48cos
cycle60sin
cycle60cos
cycle72sin
cycle84sin
cycle84cos
cycle96sin
cycle96cos
cycle108sin
cycle108cos
cycle120sin
cycle120cos
cycle132sin
cycle132cos
cycle144sin
cycle156sin
cycle168cos
cycle180sin
cycle192cos
cycle204sin
cycle216sin
cycle216cos
cycle228sin
cycle252sin
cycle252cos
cycle264cos
cycle288sin
cycle288cos
AR1
AR2
AR3
```

(continues on next page)

(continued from previous page)

```
AR4
AR5
AR6
AR8
AR9
AR10
AR11
AR18
AR21
AR27
AR28
AR29
AR34
AR35
AR92
AR102
AR108
AR111
AR113
AR116
AR117
AR216
```

8.3 Reducing with Permutation Feature Importance

- uses the ELI5 package to score feature importances
- eliminates the least-scoring variable (with some scalecast adjustment to account for collinearity), re-trains the model, and evaluates the new error
- by default, the Xvars that return the best error score (according to any metric you want to monitor) is chosen as the optimal set of variables
- to save computation resources, not every variable combination is tried (unlike other reduction techniques)
- to save computation resources, we can set `dynamic_testing = False` and we can set a mininum number of variables to use (the square root of the total number of observations is chosen in this example)
- we can use any sklearn model to reduce variables using this method and we select MLR, KNN, GBT, and SVR in this example

[9]:
```
f.reduce_Xvars(
    method="pfi",
    estimator="mlr",
    keep_at_least="sqrt",
    grid_search=False,
    normalizer="minmax",
    overwrite=False,
)
```

```
2023-04-11 12:03:53.245099: I tensorflow/core/platform/cpu_feature_guard.cc:193] This␣
↪TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use␣
↪the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
```

(continues on next page)

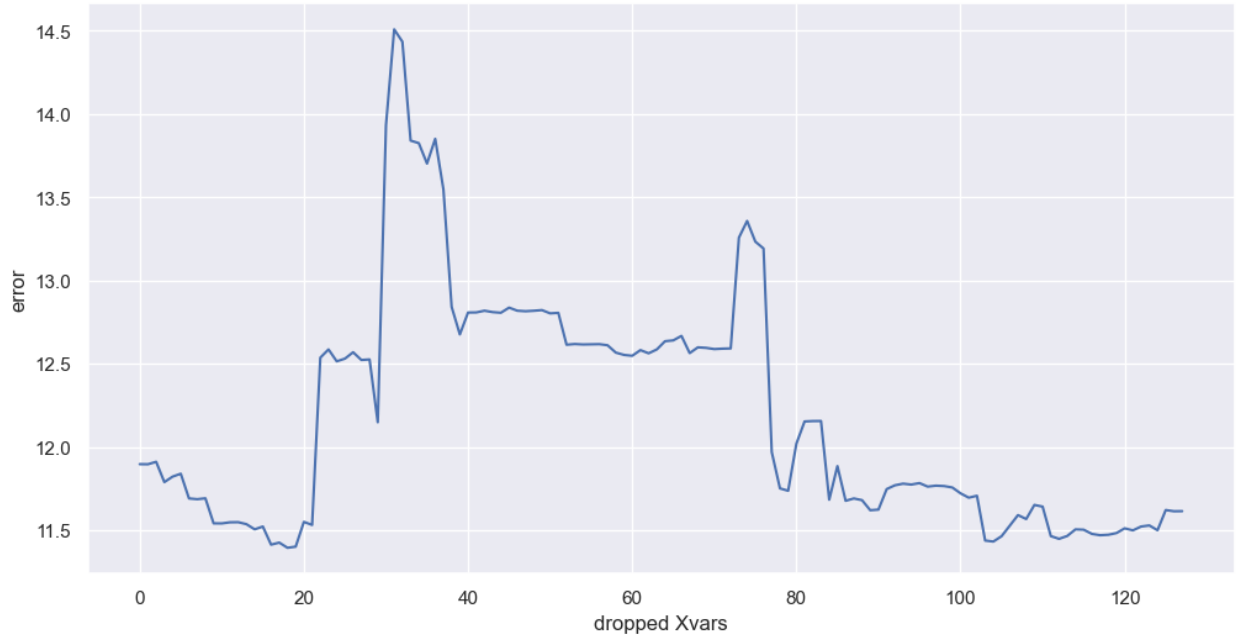
(continued from previous page)

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler␣
↪flags.
```

[10]:
```
mlr_reduced_vars = f.reduced_Xvars[:]
print(f"mlr with pfi reduced to {len(mlr_reduced_vars)} variables")
```

```
mlr with pfi reduced to 166 variables
```

[11]:
```
plot_reduction_errors(f)
plt.show()
```



The above graph shows that the MLR performs best when about 50 variables are reduced, according to the MAE on the validation set. The order the variables were dropped was based on permutation feature importance. Clearly, too many variables had been added initially and reducing them out made a significant positive difference to the model's performance. Note, the error metric monitored is not from the test set, but from a different out-of-sample slice of data. Later models use the test set to validate the decisions that were made while monitoring this validation metric.

[12]:
```
def reduce_and_save_results(f, model, results):
    f.reduce_Xvars(
        method="pfi",
        estimator=model,
        keep_at_least="sqrt",
        grid_search=True,
        dynamic_testing=False,
        overwrite=False,
    )
    results[model] = [
        f.reduced_Xvars,
        f.pfi_dropped_vars,
        f.pfi_error_values,
        f.reduction_hyperparams,
```

(continues on next page)

(continued from previous page)

```
    ]
```

[ ]:
```
results = {
    "mlr": [
        f.reduced_Xvars,
        f.pfi_dropped_vars,
        f.pfi_error_values,
        f.reduction_hyperparams,
    ]
}

for model in ("knn", "gbt", "svr"):
    print(f"reducing with {model}")
    reduce_and_save_results(f, model, results)
```

```
reducing with knn
```

[ ]:
```
for k, v in results.items():
    print(f"the {k} model chose {len(v[0])} variables")
    sns.lineplot(x=np.arange(0, len(v[1]) + 1, 1), y=v[2], label=k)
plt.xlabel("dropped Xvars")
plt.ylabel("error")
plt.show()
```

Out of the four models we tried using pfi, the gbt model scored the best on the validation set with 136 variables. We will re-train that model with each of the reduced set of variables and see which reduction performs best on the test set. We will use the best-performing model class from the four we tried: gbt. There is no perfect way to choose a set of variables and it is clear that the gbt model performs only marginally different depending on how many variables are dropped, probably because of how the underlying decision tree model can give lower weight to less-important features natively.

[ ]:
```
knn_reduced_vars = results["knn"][0]
gbt_reduced_vars = results["gbt"][0]
svr_reduced_vars = results["svr"][0]
```

8.4 Training a Model Class with All Reduced Variable Sets

[ ]:
```
selected_model = "gbt"
hp = results[selected_model][3]
f.set_estimator(selected_model)
f.manual_forecast(**hp, Xvars="all", call_me=selected_model + "_all_vars")
f.manual_forecast(
    **hp, Xvars=lasso_reduced_vars, call_me=selected_model + "_l1_reduced_vars"
)
f.manual_forecast(
    **hp, Xvars=mlr_reduced_vars, call_me=selected_model + "pfi-mlr_reduced_vars"
)
f.manual_forecast(
    **hp, Xvars=knn_reduced_vars, call_me=selected_model + "pfi-knn_reduced_vars"
)
```

(continues on next page)

(continued from previous page)

```
f.manual_forecast(
    **hp, Xvars=gbt_reduced_vars, call_me=selected_model + "pfi-gbt_reduced_vars"
)
f.manual_forecast(
    **hp, Xvars=svr_reduced_vars, call_me=selected_model + "pfi-svr_reduced_vars"
)
```

[ ]:
```
export_results(f)
```

Although the gbt model with 136 variables performed best out of all tried models on the validation set, on the test set, the 98 variables selected by KNN led to the best performance, followed by using all variables and then the 36 selected from the L1 reduction. This illustrates that even if a reduction decision is made with one estimator class, it can still be useful for another model class.

[ ]:
```
f.plot_test_set(ci=True, order_by="TestSetMAE", include_train=240)
plt.show()
```

8.5 Backtest Best Model

Backtesting shows the average performance of a given model across the last 10 (by default) forecast horizons, using only data before each 120-period forecast to train the model. It's a good way to see if your model generalizes well or just got lucky on the particular test set you fed to it.

[ ]:
```
f.backtest("gbtpfi-knn_reduced_vars")
f.export_backtest_metrics("gbtpfi-knn_reduced_vars")
```

[ ]:
```
f.plot(models="gbtpfi-knn_reduced_vars", ci=True)
plt.show()
```

8.6 Export Reduced Dataset

[ ]:
```
reduced_dataset = f.export_Xvars_df()[['DATE'] + knn_reduced_vars]
reduced_dataset.head(25)
```

8.7 See how often each var was dropped

[ ]:
```
pd.options.display.max_rows = None
all_xvars = f.get_regressor_names()
final_dropped = pd.DataFrame({"Var": all_xvars})
for i, v in f.export("model_summaries").iterrows():
    model = v["ModelNickname"]
    Xvars = v["Xvars"]
    dropped_vars = [x for x in f.get_regressor_names() if x not in Xvars]
    if not dropped_vars:
        continue
```

(continues on next page)

(continued from previous page)

```
    tmp_dropped = pd.DataFrame(
        {"Var": dropped_vars, f"dropped in {model}": [1] * len(dropped_vars)}
    )
    final_dropped = final_dropped.merge(tmp_dropped, on="Var", how="left").fillna(0)
final_dropped["total times dropped"] = final_dropped.iloc[:, 1:].sum(axis=1)
final_dropped = final_dropped.loc[final_dropped["total times dropped"] > 0]
final_dropped = final_dropped.sort_values("total times dropped", ascending=False)
final_dropped = final_dropped.reset_index(drop=True)
final_dropped.iloc[:, 1:] = final_dropped.iloc[:, 1:].astype(int)
final_dropped
```

[ ]:

CHAPTER

NINE

FORECASTING COVID-AFFECTED DATA

This notebook showcases some scalecast techniques that can be used to forecast on series heavily affected by the COVID-19 pandemic. Special thanks to Zohoor Nezhad Halafi for helping with this notebook! Connect with her on LinkedIn.

[1]:
```
import pandas as pd
import numpy as np
from scalecast.Forecaster import Forecaster
from scalecast.MVForecaster import MVForecaster
from scalecast.SeriesTransformer import SeriesTransformer
from scalecast.AnomalyDetector import AnomalyDetector
from scalecast.ChangepointDetector import ChangepointDetector
from scalecast.util import plot_reduction_errors, break_mv_forecaster, metrics
from scalecast import GridGenerator
from scalecast.multiseries import export_model_summaries
from scalecast.auxmodels import mlp_stack, auto_arima
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
import seaborn as sns
import os
from tqdm.notebook import tqdm
import pickle

sns.set(rc={'figure.figsize':(12,8)})
```

[2]:
```
GridGenerator.get_example_grids()
```

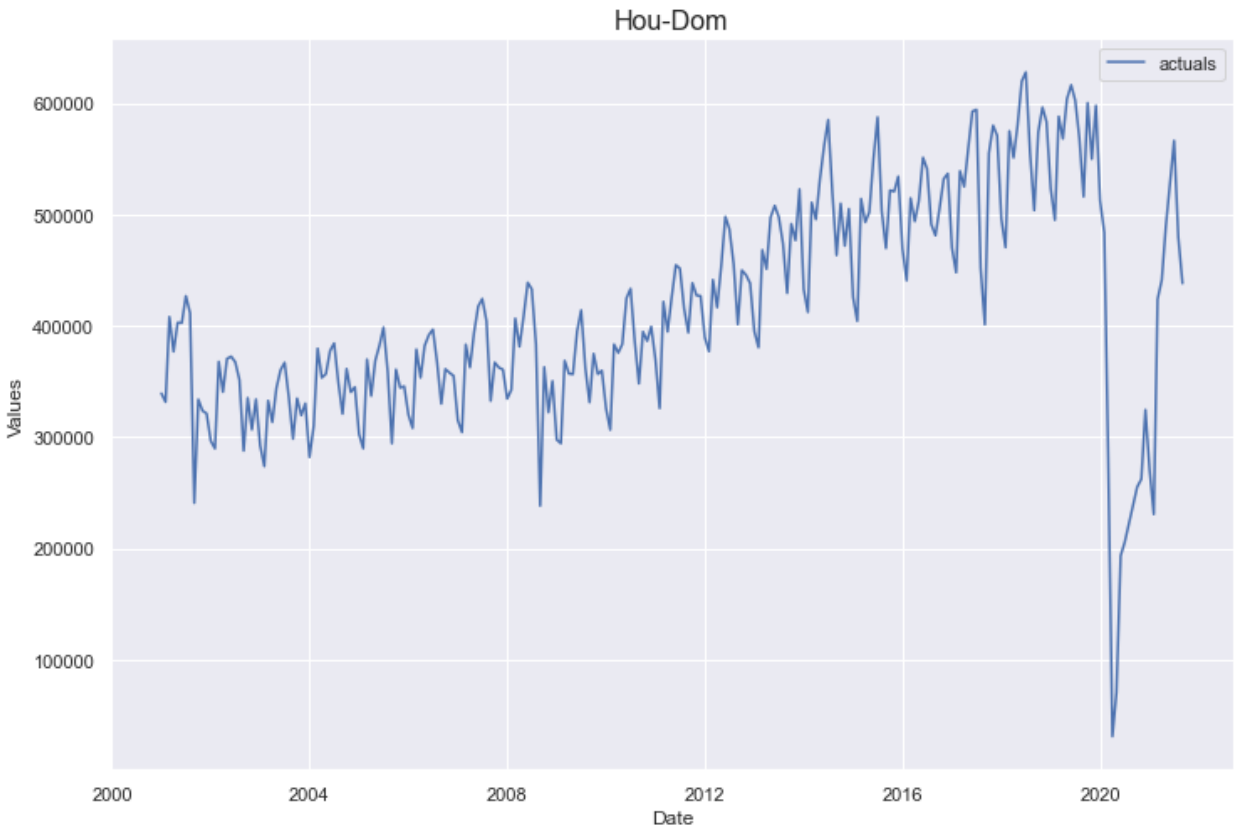
[3]:
```
airline_series = ['Hou-Dom','IAH-DOM','IAH-Int']
fcst_horizon = 4

data = {
    l:pd.read_csv(
        os.path.join('data',l+'.csv')
    ) for l in airline_series
}

fdict = {
    l:Forecaster(
        y=df['PASSENGERS'],
        current_dates=df['Date'],
        future_dates=fcst_horizon,
```

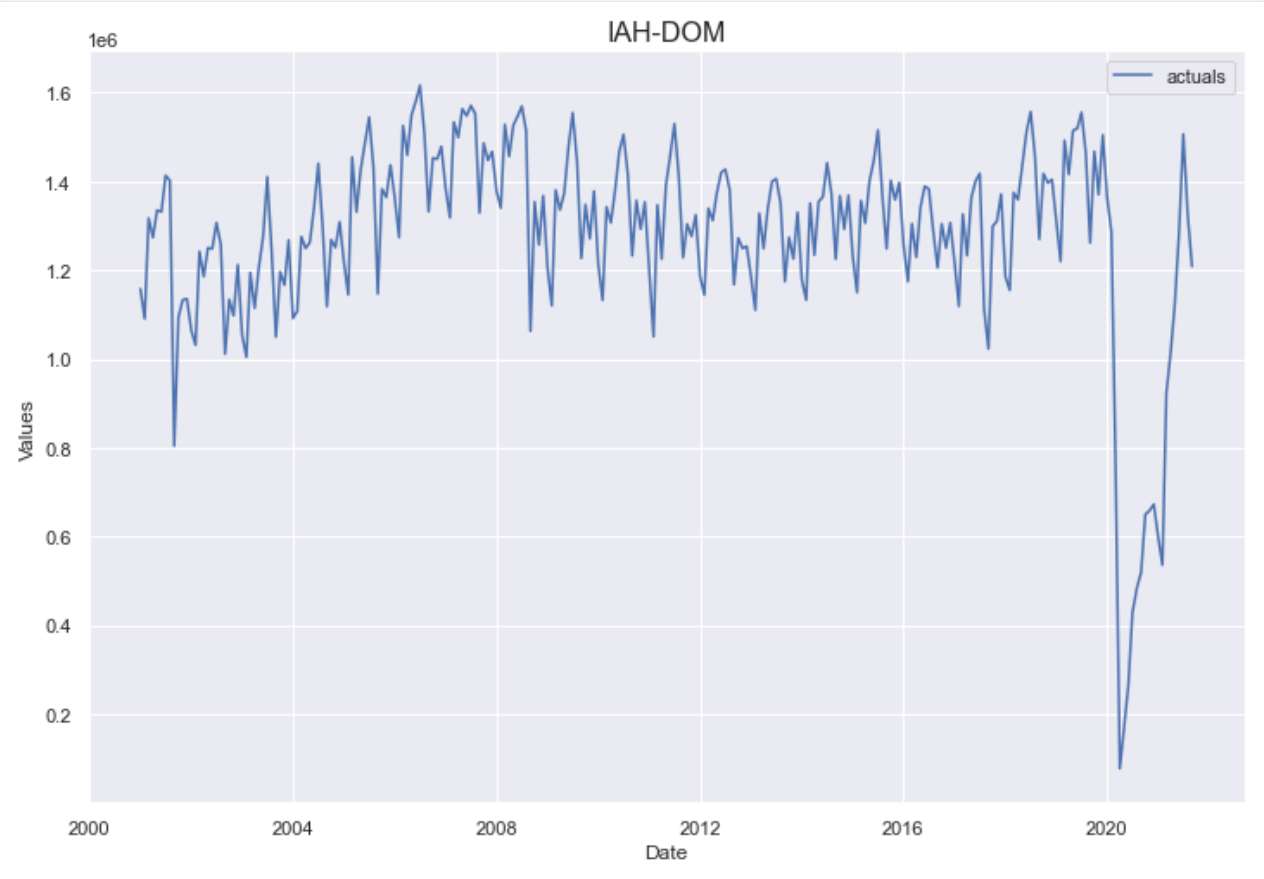
(continues on next page)

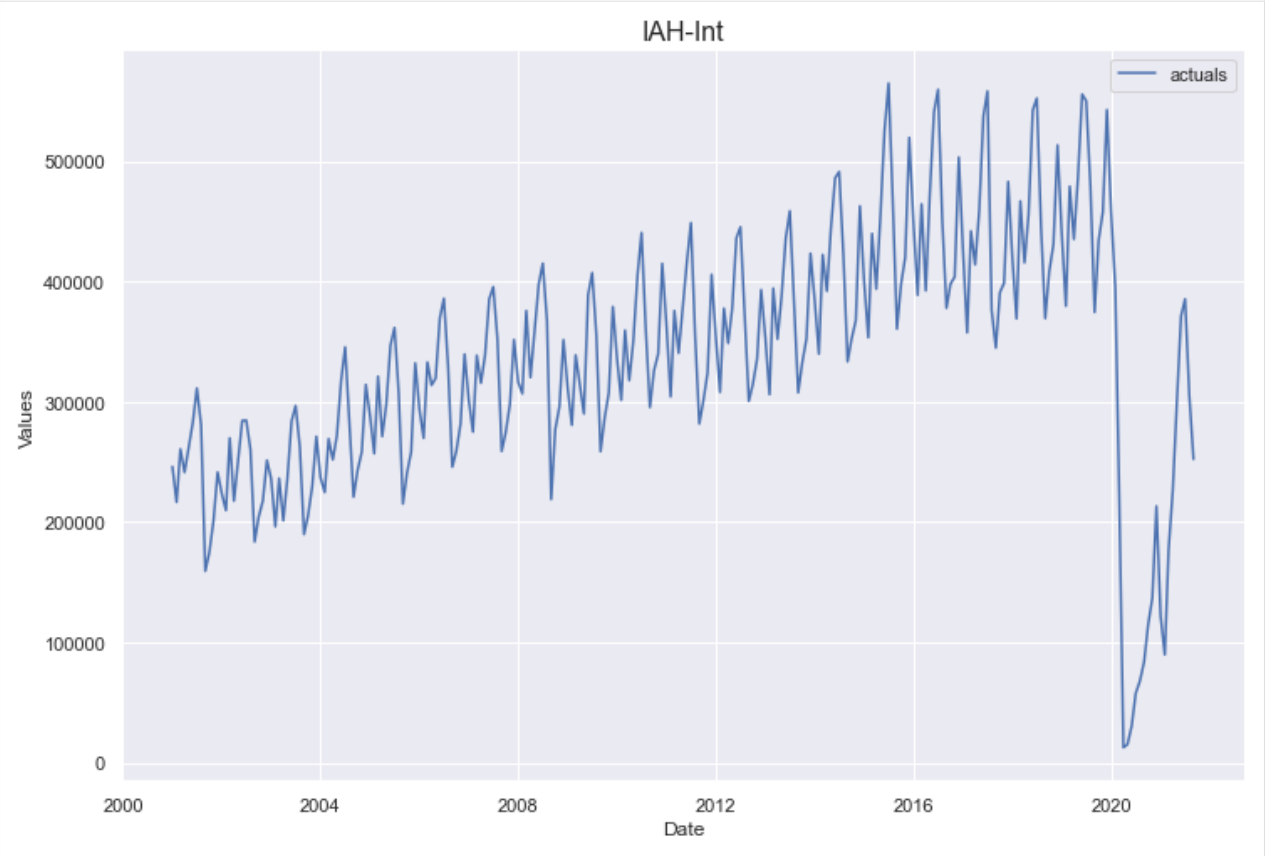
(continued from previous page)

```
    ) for l,df in data.items()
}
```

[4]:
```
for l, f in fdict.items():
    f.plot()
    plt.title(l,size=16)
    plt.show()
```



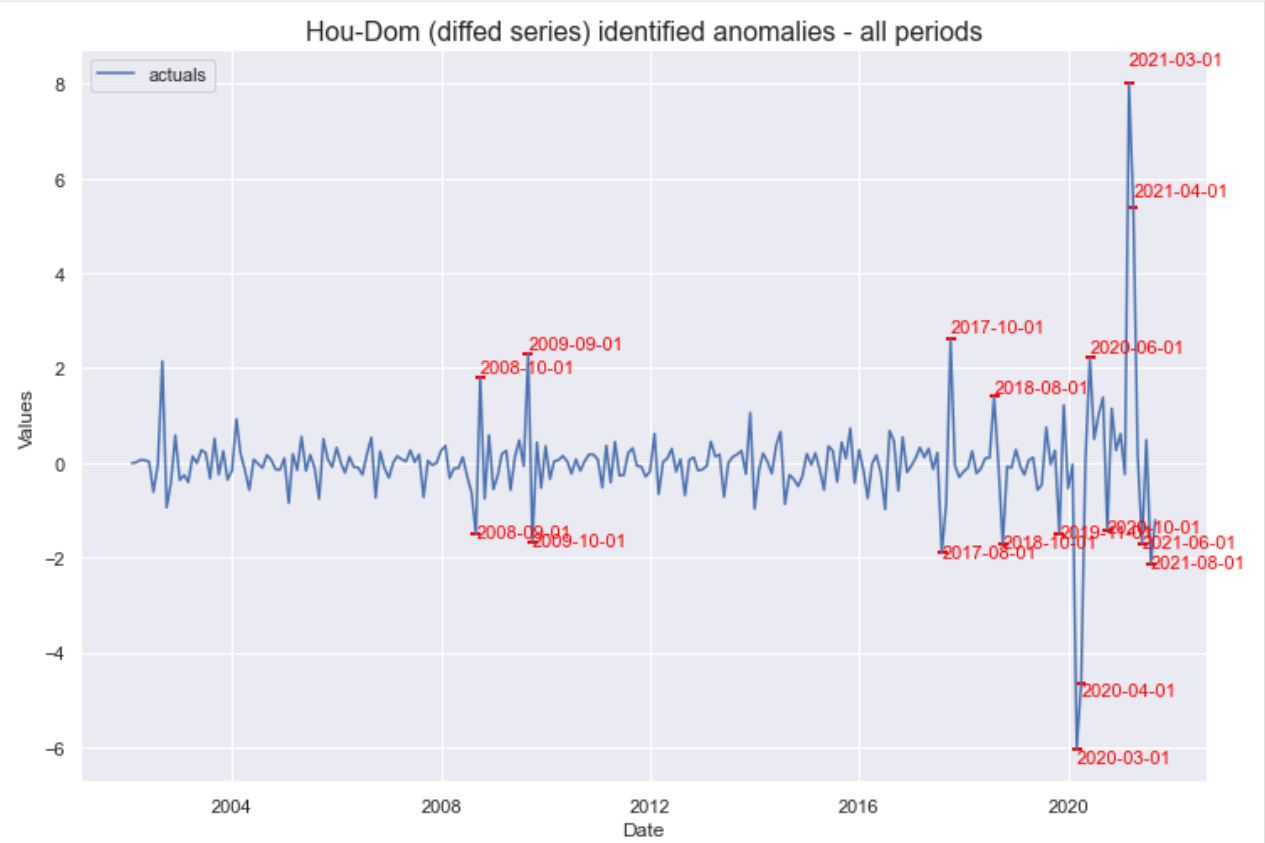

IAH-DOM
1e6
actuals
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Values
2000
2004
2008
2012
2016
2020
Date



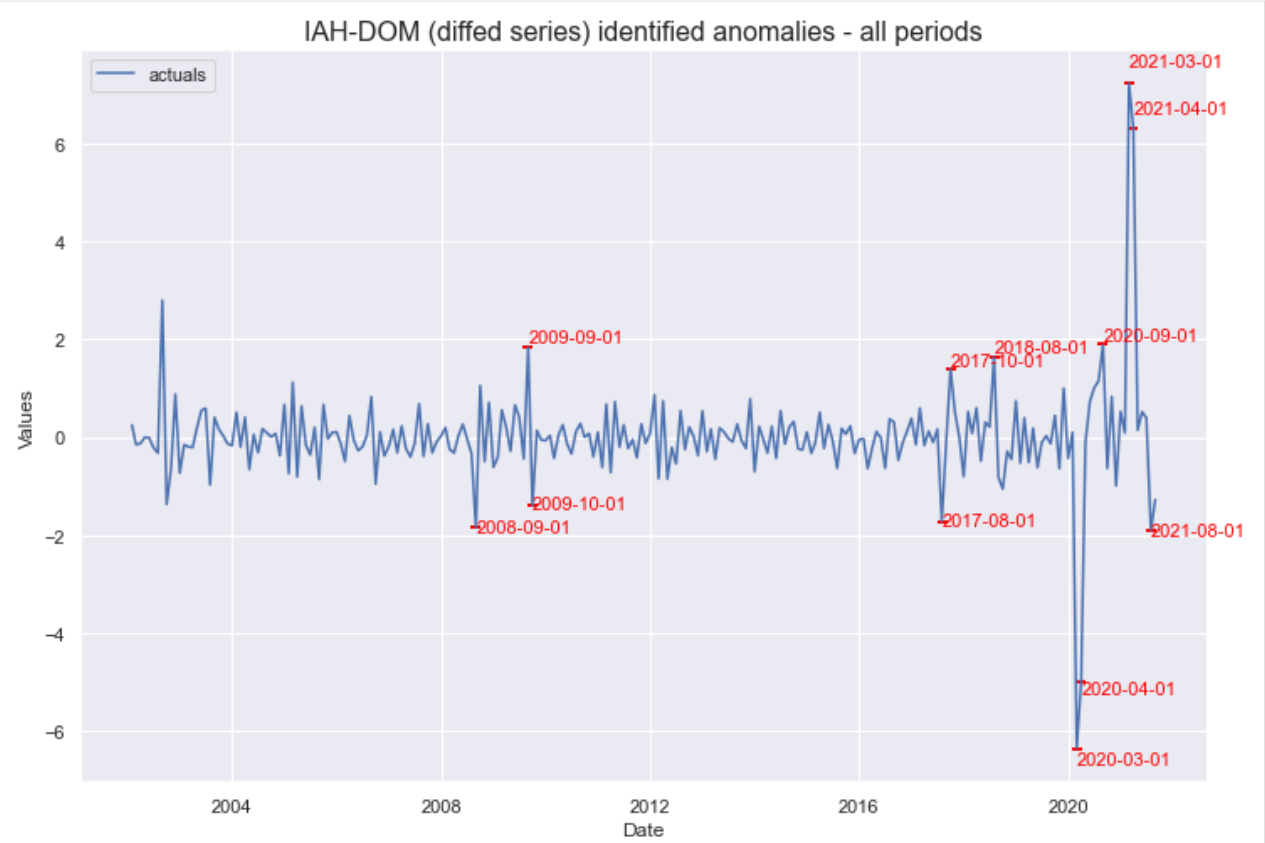
9.1 Add Anomalies to Model

[5]:
```
# scan on whole dataset to compare
for l, f in fdict.items():
    tr = SeriesTransformer(f)
    tr.DiffTransform(1)
    tr.DiffTransform(12)
    f2 = tr.ScaleTransform()
    ad = AnomalyDetector(f2)
    ad.MonteCarloDetect_sliding(36,36)
    ad.plot_anom()
    plt.title(f'{l} (diffed series) identified anomalies - all periods',size=16)
    plt.show()
```

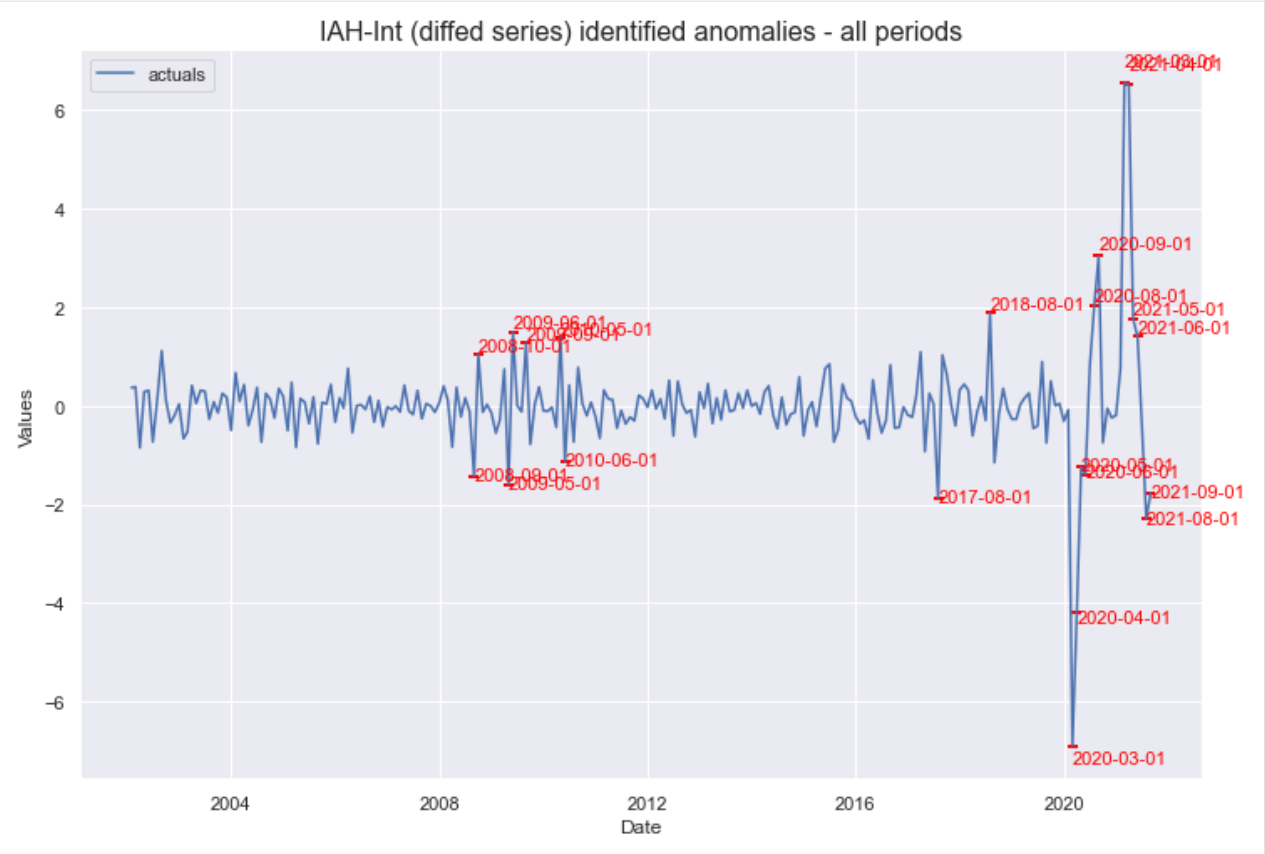
```
scanning from obs 36 to obs 72
scanning from obs 72 to obs 108
scanning from obs 108 to obs 144
scanning from obs 144 to obs 180
scanning from obs 180 to obs 216
scanning from obs 216 to obs 252
```




```
scanning from obs 36 to obs 72
scanning from obs 72 to obs 108
scanning from obs 108 to obs 144
scanning from obs 144 to obs 180
scanning from obs 180 to obs 216
scanning from obs 216 to obs 252
```

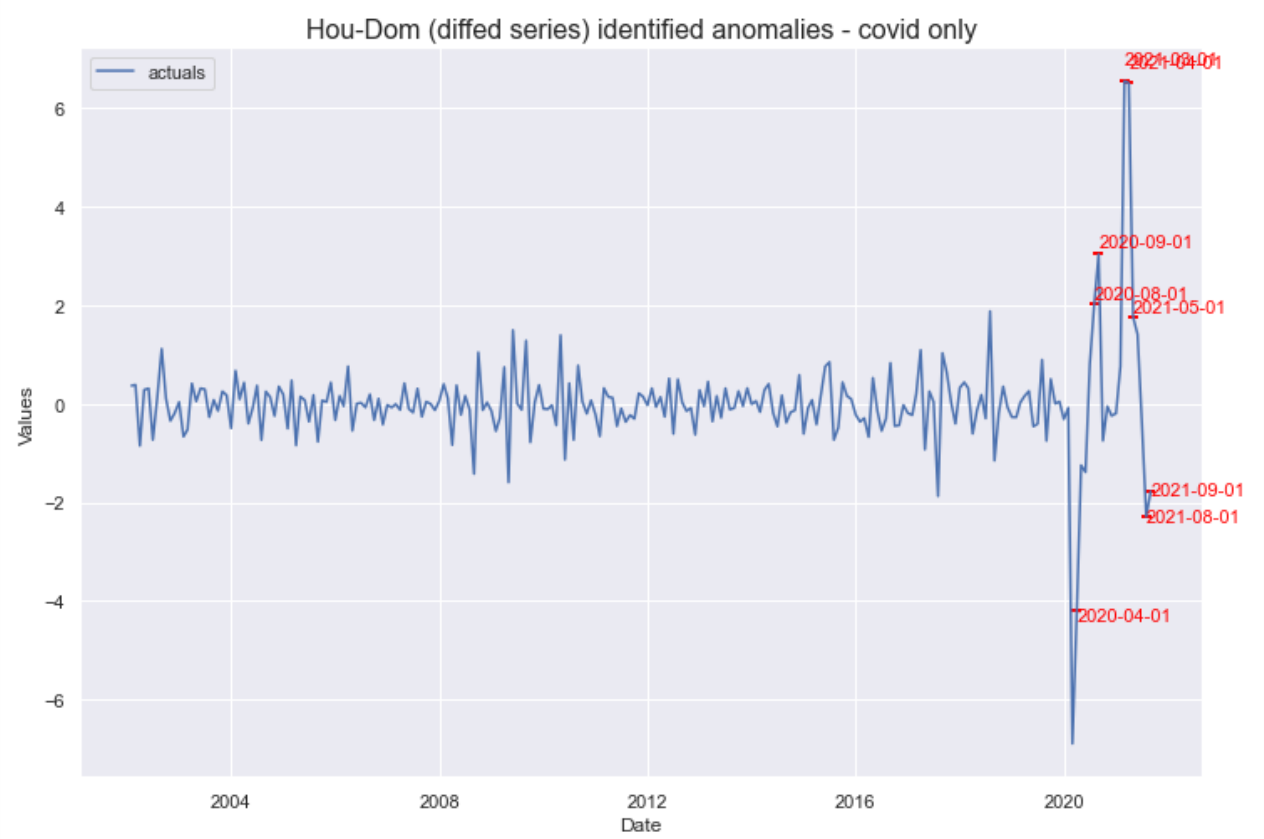


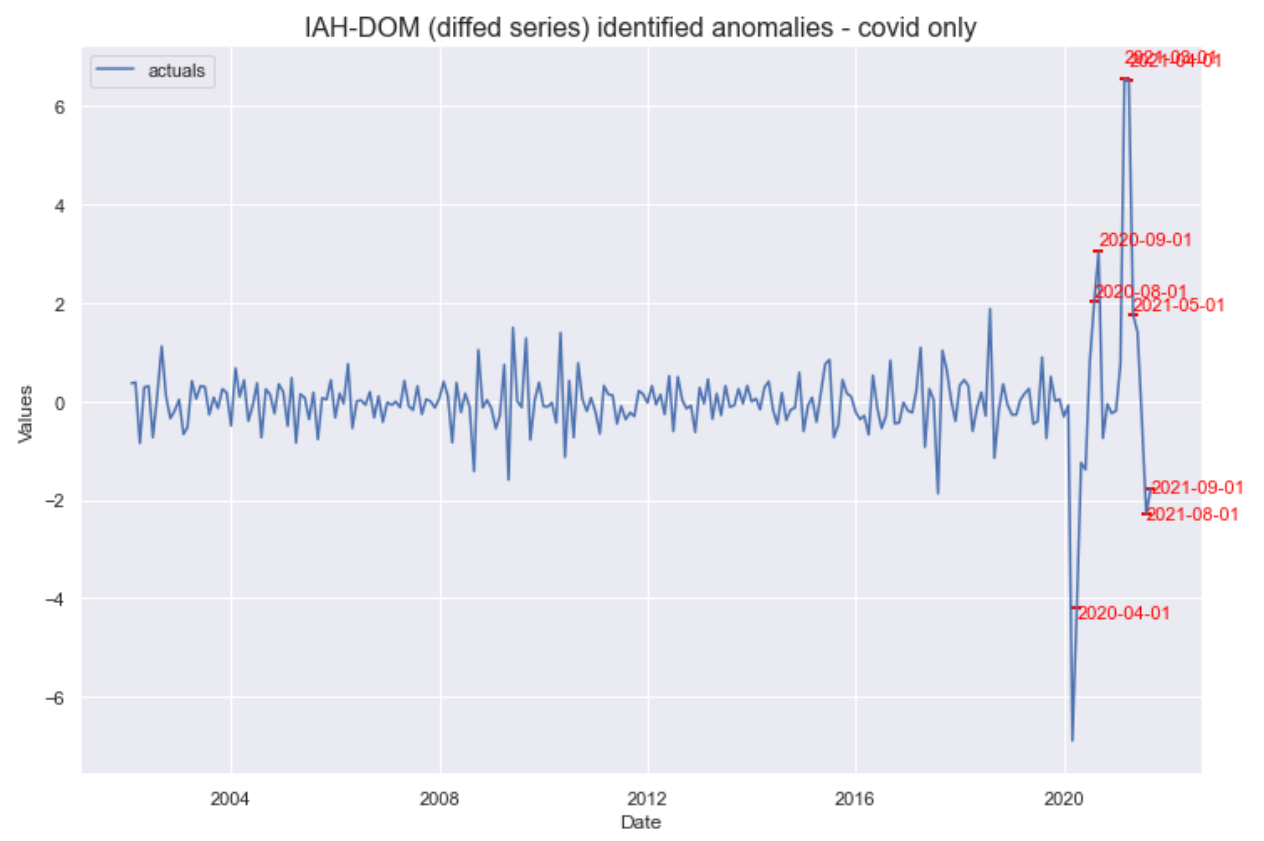

```
scanning from obs 36 to obs 72
scanning from obs 72 to obs 108
scanning from obs 108 to obs 144
scanning from obs 144 to obs 180
scanning from obs 180 to obs 216
scanning from obs 216 to obs 252
```

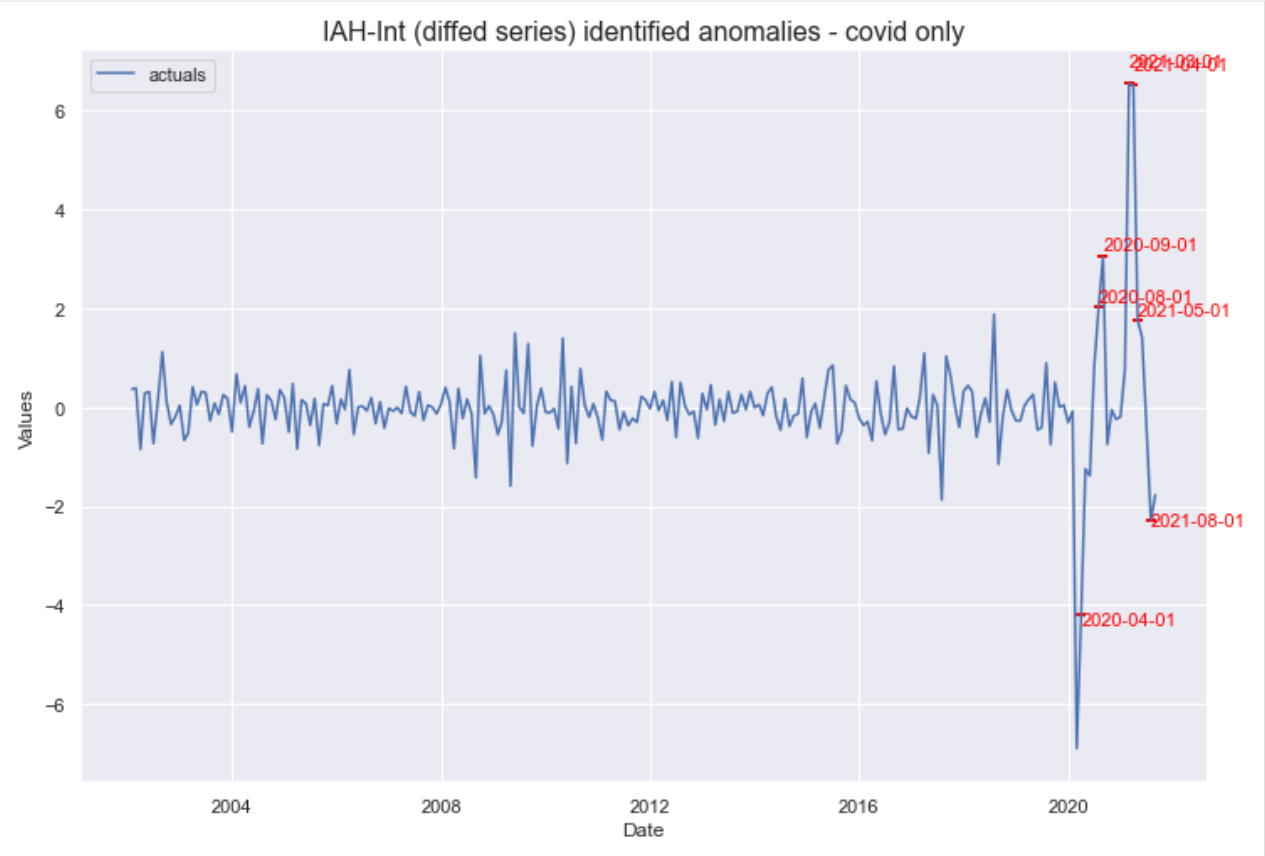


- Sept 11 2001
- Recession sept 2008
- Hurricane Harvey August 2017

[6]:
```
# scan over only covid periods
for l, f in fdict.items():
    ad.MonteCarloDetect(start_at='2020-04-01',stop_at='2021-09-01')
    ad.plot_anom()
    plt.title(f'{l} (diffed series) identified anomalies - covid only',size=16)
    plt.show()
    f = ad.WriteAnomtoXvars(f=f,drop_first=True)
    f.add_other_regressor(start = '2001-09-01', end='2001-09-01', called='2001')
    f.add_other_regressor(start = '2008-09-01', end='2008-09-01', called='Great Recession
↪')
    f.add_other_regressor(start = '2017-08-01', end='2017-08-01', called='Harvey')
    fdict[l] = f
```

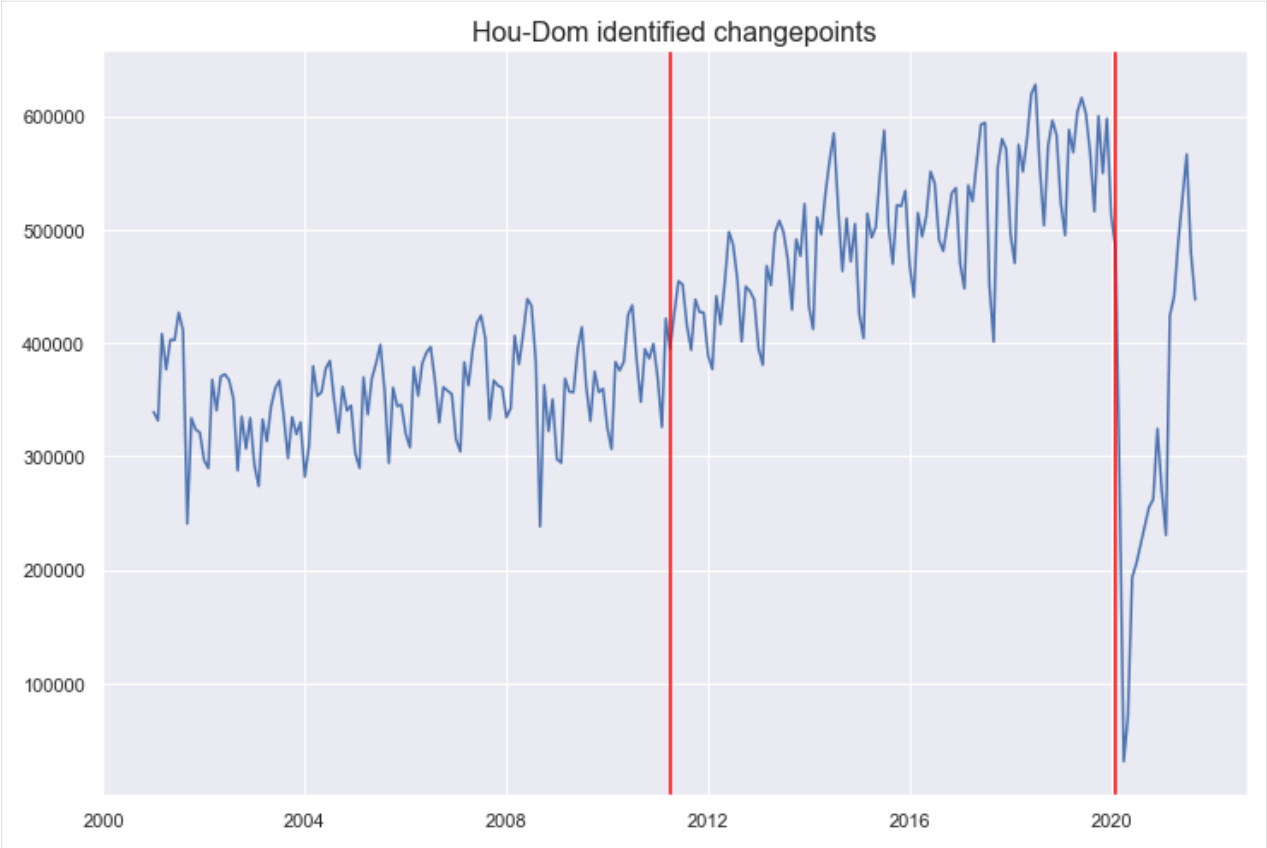

Hou-Dom (diffed series) identified anomalies - covid only
actuals
2021-08-01
2021-04-01
2020-09-01
2020-08-01
2021-05-01
2021-09-01
2021-08-01
2020-04-01
Values
Date
2004
2008
2012
2016
2020

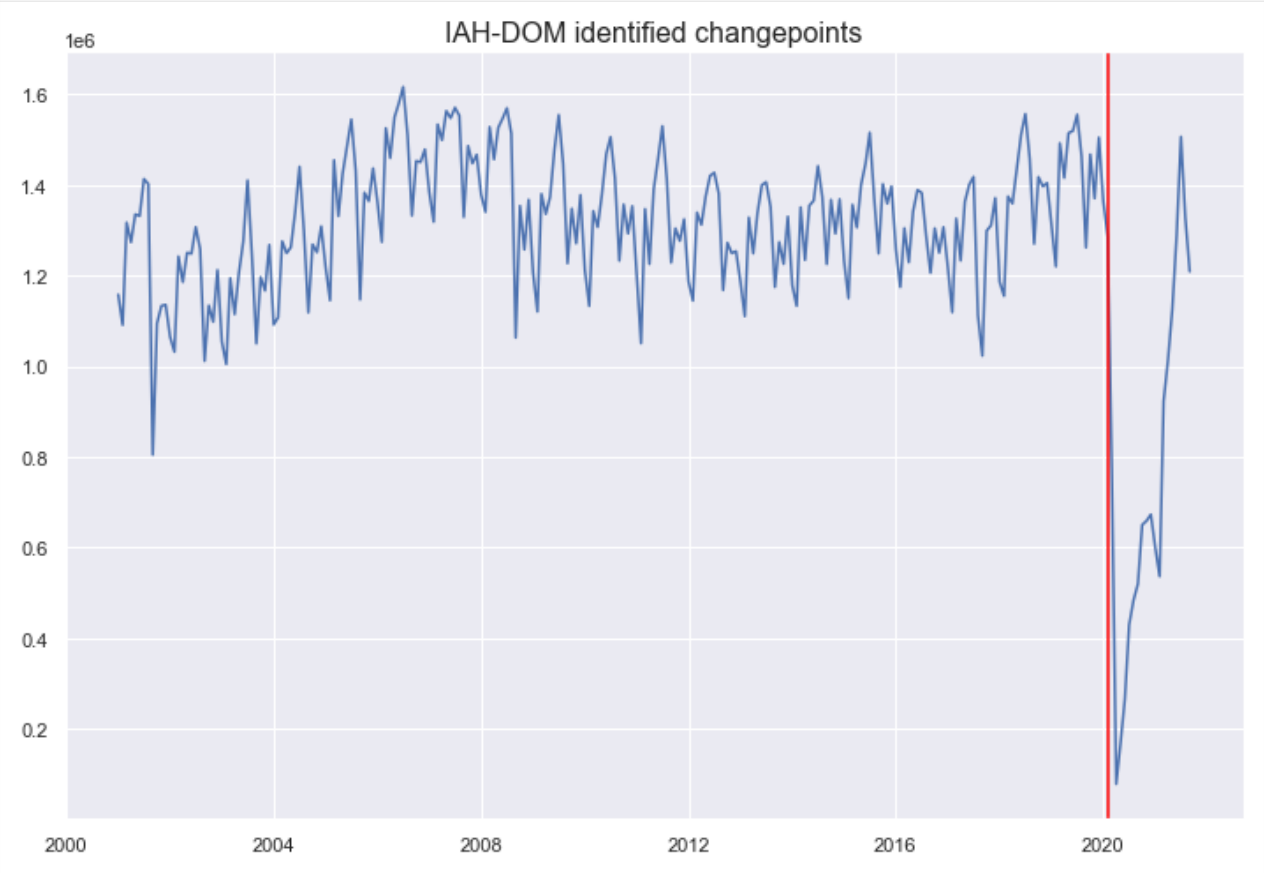

IAH-DOM (diffed series) identified anomalies - covid only
actuals
2021-08-01
2021-04-01
2020-09-01
2020-08-01
2021-05-01
2021-09-01
2021-08-01
2020-04-01
Values
Date
2004
2008
2012
2016
2020

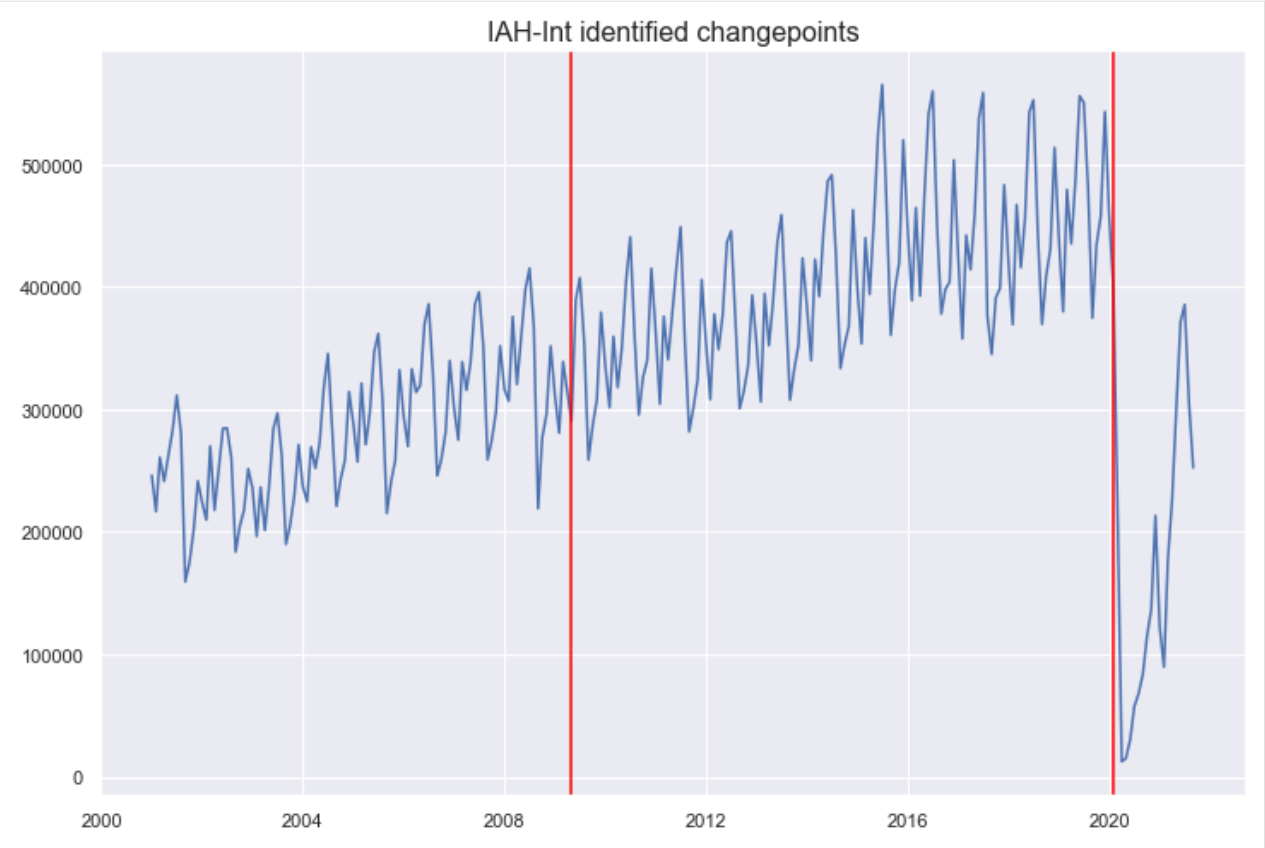


9.2 Add Changepoints to Model

[7]:
```
for l, f in fdict.items():
    cd = ChangepointDetector(f)
    cd.DetectCPCUSUM()
    cd.plot()
    plt.title(f'{l} identified changepoints',size=16)
    plt.show()
    f = cd.WriteCPtoXvars()
    fdict[l] = f
```


Hou-Dom identified changepoints
600000
500000
400000
300000
200000
100000
2000
2004
2008
2012
2016
2020


IAH-DOM identified changepoints
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
2000
2004
2008
2012
2016
2020



9.3 Choose Xvars

[8]:
```
regressors = pd.read_csv('data/Regressors.csv',parse_dates=['Date'],index_col=0)
for dt in f.future_dates:
    regressors.loc[dt] = [0]*regressors.shape[1]

for c in regressors:
    series = regressors[c]
    if adfuller(series)[1] >= 0.01:
        regressors[c] = regressors[c].diff()
regressors = regressors.fillna(0) # just for now
regressors.head()
```

[8]:
```
            Number of available seats Domestic and international level  \
Date
2001-01-01                                                  0.0
2001-02-01                                              -7491574.0
2001-03-01                                               8507261.0
2001-04-01                                              -1273374.0
2001-05-01                                               2815795.0

            Number of domestic flights  Number of International  flights  \
Date
2001-01-01                         0.0                                0.0
```

(continues on next page)

(continued from previous page)

```
2001-02-01                          0.0                                 0.0
2001-03-01                          0.0                                 0.0
2001-04-01                          0.0                                 0.0
2001-05-01                          0.0                                 0.0

            Revenue Passenger-miles/Domestic  \
Date
2001-01-01                               0.0
2001-02-01                               0.0
2001-03-01                               0.0
2001-04-01                               0.0
2001-05-01                               0.0

            Revenue Passenger-miles/International  TSI/Passengers  \
Date
2001-01-01                                    0.0             0.0
2001-02-01                                    0.0             0.0
2001-03-01                                    0.0             0.5
2001-04-01                                    0.0             0.6
2001-05-01                                    0.0             0.5

            TSI/Freight  Trade, Transportation, and Utilities  \
Date
2001-01-01          0.0                                  0.00
2001-02-01         -0.1                                  0.51
2001-03-01          0.2                                 -0.12
2001-04-01         -1.5                                 -0.23
2001-05-01          1.5                                  0.57

            All employees Information  Employees In financial activities  ...  \
Date                                                                        ...
2001-01-01                       0.00                                0.00  ...
2001-02-01                       0.11                                0.10  ...
2001-03-01                      -0.12                                0.13  ...
2001-04-01                      -0.12                               -0.05  ...
2001-05-01                      -0.58                               -0.01  ...

            Employees in Leisure and Hospitality  \
Date
2001-01-01                                  0.00
2001-02-01                                  0.57
2001-03-01                                  0.38
2001-04-01                                 -0.18
2001-05-01                                  0.18

            All Employees,Education and Health Services  \
Date
2001-01-01                                         0.00
2001-02-01                                         0.81
2001-03-01                                         0.60
2001-04-01                                         0.72
2001-05-01                                         1.61
```

(continues on next page)

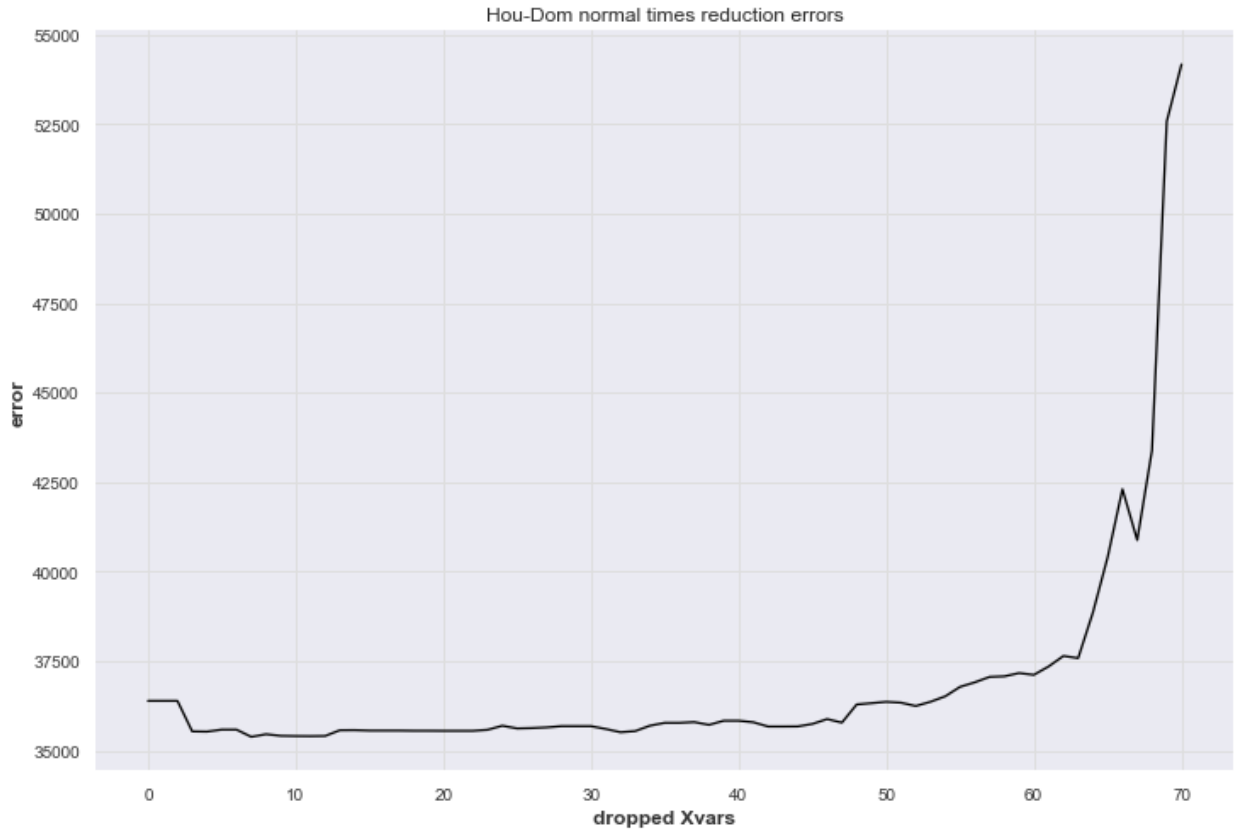
(continued from previous page)

```
            All Employee,Mining and Logging: Oil and Gas Extraction  \
Date
2001-01-01                                               0.00
2001-02-01                                               0.36
2001-03-01                                              -0.02
2001-04-01                                              -0.07
2001-05-01                                               0.73

            Employees in Other Services  Unemployement Rate in Houston Area  \
Date
2001-01-01                         0.00                                 4.1
2001-02-01                        -0.13                                 3.9
2001-03-01                         0.33                                 4.1
2001-04-01                         0.29                                 4.0
2001-05-01                        -0.22                                 4.2

            Texas Business CyclesIndex  \
Date
2001-01-01                         0.0
2001-02-01                         0.3
2001-03-01                         0.1
2001-04-01                         0.0
2001-05-01                        -0.2

            PCE on Durable goods(Billions of Dollars, Monthly, Seasonally Adjusted␣
↪Annual Rate)  \
Date
2001-01-01                                               0.0
2001-02-01                                              22.4
2001-03-01                                             -13.8
2001-04-01                                             -16.8
2001-05-01                                               7.7

            PCE on Recreational activities  Recession  WTO Oil Price
Date
2001-01-01                             0.0          0           0.00
2001-02-01                             0.0          0           0.03
2001-03-01                             0.0          0          -2.37
2001-04-01                             0.0          1           0.17
2001-05-01                             0.0          1           1.23

[5 rows x 21 columns]
```

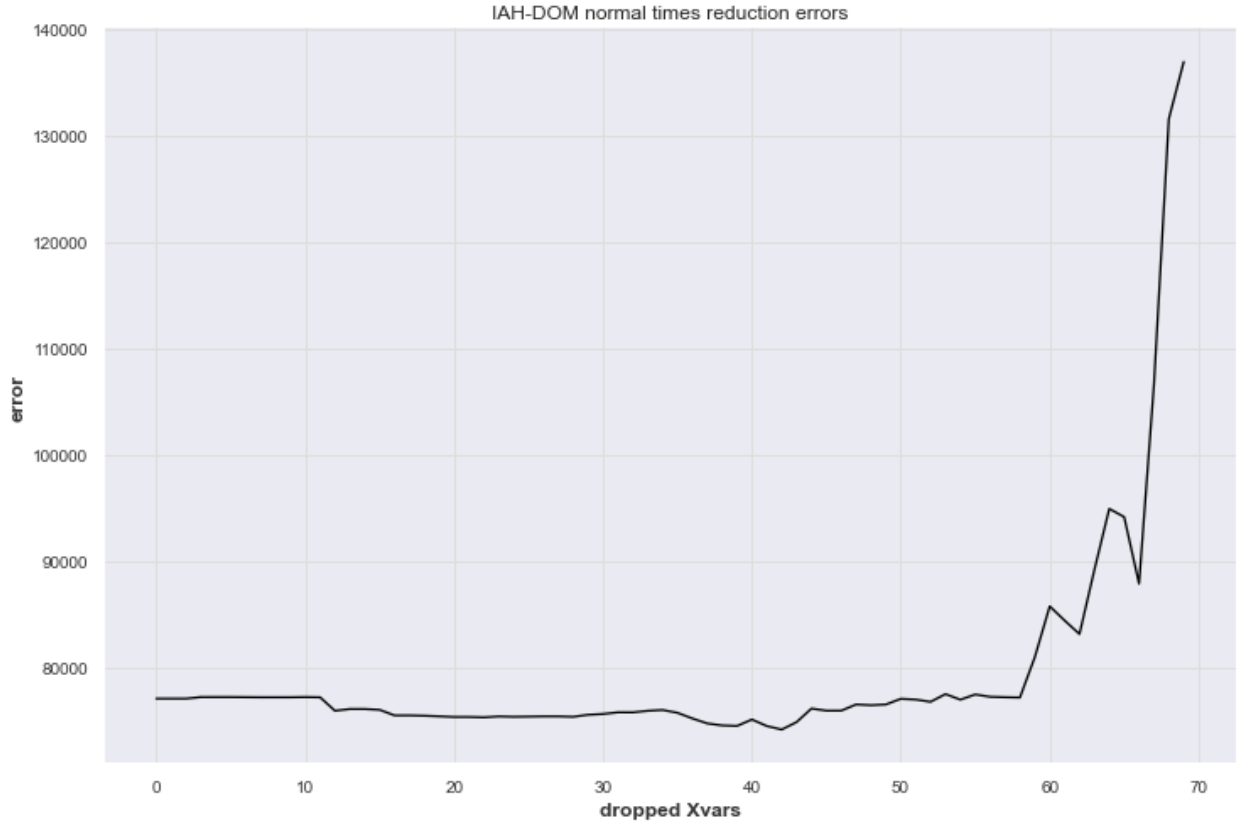
9.4 Variable Reduction

[9]:
```
dropped = pd.DataFrame(columns=fdict.keys())

# first to get regressors for regular times
for l, f in fdict.items():
    print(f'reducing {l}')
    f2 = f.deepcopy()
    f2.add_ar_terms(36)
    f.add_seasonal_regressors(
        'month',
        raw=False,
        sincos=True
    )
    f2.add_time_trend()
    f2.diff()
    f2.ingest_Xvars_df(
        regressors.reset_index(),
        date_col='Date',
    )
    f2.reduce_Xvars(
        method='pfi',
        estimator='elasticnet',
        cross_validate=True,
        cvkwargs={'k':3},
        dynamic_tuning=fcst_horizon, # optimize on fcst_horizon worth of predictions
        overwrite=False,
    )
    for x in f2.pfi_dropped_vars:
        dropped.loc[x,l] = 1
    for x in f2.reduced_Xvars:
        dropped.loc[x,l] = 0

    plot_reduction_errors(f2)
    plt.title(f'{l} normal times reduction errors',size=12)
    plt.show()
```

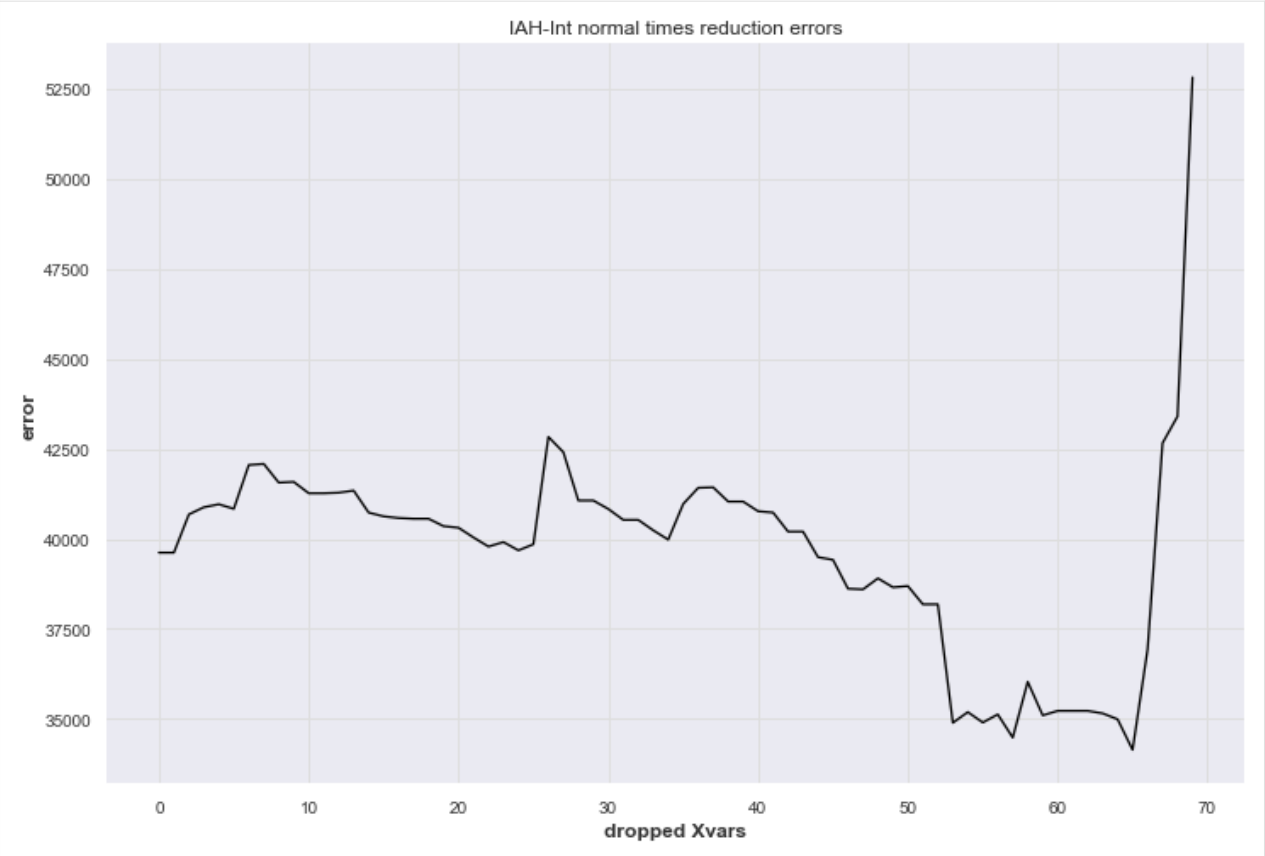
```
reducing Hou-Dom
```




```
reducing IAH-DOM
```




```
reducing IAH-Int
```



9.4.1 Which variables were dropped most?

[10]:
```
dropped = dropped.fillna(0).drop('total times dropped',axis=1,errors='ignore')
dropped['total times dropped'] = dropped.sum(axis=1)
dropped.sort_values(['total times dropped'],ascending=False).head(25)
```

[10]:

	Hou-Dom	IAH-DOM	IAH-Int	\
2001	1	1	1	
PCE on Recreational activities	1	1	1	
AR16	1	1	1	
AR11	1	1	1	
Anomaly_2021-05-01	1	1	1	
Unemployement Rate in Houston Area	1	1	1	
AR33	0	1	1	
AR10	0	1	1	
AR5	0	1	1	
Harvey	0	1	1	
WTO Oil Price	0	1	1	
AR13	0	1	1	
AR15	0	1	1	
AR21	0	1	1	
TSI/Freight	0	1	1	
Anomaly_2021-09-01	1	1	0	
Anomaly_2021-03-01	0	1	1	

(continues on next page)

(continued from previous page)

```
AR3                                                0        1        1
All Employees,Education and Health Services        0        1        1
Employees In financial activities                  0        1        1
AR34                                               0        1        1
AR2                                                0        1        1
AR22                                               0        1        1
AR27                                               0        1        1
Anomaly_2020-08-01                                 0        1        1

                                           total times dropped
2001                                                         3
PCE on Recreational activities                               3
AR16                                                         3
AR11                                                         3
Anomaly_2021-05-01                                           3
Unemployement Rate in Houston Area                           3
AR33                                                         2
AR10                                                         2
AR5                                                          2
Harvey                                                       2
WTO Oil Price                                                2
AR13                                                         2
AR15                                                         2
AR21                                                         2
TSI/Freight                                                  2
Anomaly_2021-09-01                                           2
Anomaly_2021-03-01                                           2
AR3                                                          2
All Employees,Education and Health Services                  2
Employees In financial activities                            2
AR34                                                         2
AR2                                                          2
AR22                                                         2
AR27                                                         2
Anomaly_2020-08-01                                           2
```

9.4.2 Which variables were dropped least?

[11]:
```
dropped.sort_values(['total times dropped']).head(40)
```

[11]:
```
                                           Hou-Dom  IAH-DOM  IAH-Int  \
AR36                                             0        0        0
Revenue Passenger-miles/Domestic                 0        0        0
AR18                                             0        0        0
Number of domestic flights                       0        0        0
Number of International  flights                0        0        0
cp2                                              0        0        1
AR17                                             0        0        1
All employees Information                        0        0        1
Employees in Other Services                      0        0        1
Employees in Leisure and Hospitality             0        0        1
```

(continues on next page)

(continued from previous page)

Anomaly_2020-04-01	0	0	1
PCE on Durable goods(Billions of Dollars, Month...	0	0	1
AR25	0	0	1
AR31	0	0	1
AR6	0	0	1
Trade, Transportation, and Utilities	0	0	1
AR29	0	0	1
AR24	0	0	1
AR26	0	0	1
AR19	0	0	1
Revenue Passenger-miles/International	0	0	1
Recession	0	0	1
AR14	0	0	1
Texas Business CyclesIndex	0	0	1
AR12	0	0	1
AR1	0	0	1
TSI/Passengers	0	0	1
AR30	0	0	1
Number of available seats Domestic and internat...	0	0	1
Anomaly_2021-03-01	0	1	1
AR22	0	1	1
All Employees,Education and Health Services	0	1	1
AR2	0	1	1
Employees In financial activities	0	1	1
TSI/Freight	0	1	1
AR34	0	1	1
Anomaly_2020-08-01	0	1	1
AR21	0	1	1
Anomaly_2021-09-01	1	1	0
AR9	0	1	1

	total times dropped
AR36	0
Revenue Passenger-miles/Domestic	0
AR18	0
Number of domestic flights	0
Number of International flights	0
cp2	1
AR17	1
All employees Information	1
Employees in Other Services	1
Employees in Leisure and Hospitality	1
Anomaly_2020-04-01	1
PCE on Durable goods(Billions of Dollars, Month...	1
AR25	1
AR31	1
AR6	1
Trade, Transportation, and Utilities	1
AR29	1
AR24	1
AR26	1
AR19	1

(continues on next page)

(continued from previous page)

```
Revenue Passenger-miles/International                    1
Recession                                                1
AR14                                                     1
Texas Business CyclesIndex                               1
AR12                                                     1
AR1                                                      1
TSI/Passengers                                           1
AR30                                                     1
Number of available seats Domestic and internat...       1
Anomaly_2021-03-01                                       2
AR22                                                     2
All Employees,Education and Health Services              2
AR2                                                      2
Employees In financial activities                        2
TSI/Freight                                              2
AR34                                                     2
Anomaly_2020-08-01                                       2
AR21                                                     2
Anomaly_2021-09-01                                       2
AR9                                                      2
```

9.4.3 What about AR terms?

[12]:
```
ar_dropped = dropped.reset_index()
ar_dropped = ar_dropped.loc[ar_dropped['index'].str.contains('AR')]
ar_dropped['ar'] = ar_dropped['index'].apply(lambda x: int(x[2:]))
ar_dropped.sort_values('ar')
```

[12]:

	index	Hou-Dom	IAH-DOM	IAH-Int	total times dropped	ar
53	AR1	0	0	1	1	1
60	AR2	0	1	1	2	2
25	AR3	0	1	1	2	3
9	AR4	0	1	1	2	4
28	AR5	0	1	1	2	5
37	AR6	0	0	1	1	6
19	AR7	0	1	1	2	7
24	AR8	0	1	1	2	8
7	AR9	0	1	1	2	9
27	AR10	0	1	1	2	10
4	AR11	1	1	1	3	11
66	AR12	0	0	1	1	12
31	AR13	0	1	1	2	13
58	AR14	0	0	1	1	14
32	AR15	0	1	1	2	15
3	AR16	1	1	1	3	16
49	AR17	0	0	1	1	17
51	AR18	0	0	0	0	18
56	AR19	0	0	1	1	19
13	AR20	0	1	1	2	20
33	AR21	0	1	1	2	21
61	AR22	0	1	1	2	22

(continues on next page)

(continued from previous page)

```
12  AR23        0        1        1                    2  23
70  AR24        0        0        1                    1  24
40  AR25        0        0        1                    1  25
55  AR26        0        0        1                    1  26
26  AR27        0        1        1                    2  27
10  AR28        0        1        1                    2  28
36  AR29        0        0        1                    1  29
62  AR30        0        0        1                    1  30
38  AR31        0        0        1                    1  31
14  AR32        0        1        1                    2  32
34  AR33        0        1        1                    2  33
52  AR34        0        1        1                    2  34
23  AR35        0        1        1                    2  35
69  AR36        0        0        0                    0  36
```

9.4.3.1 Subjectively make selections

[13]:
```
final_Xvars_selected = [
    x for x in dropped.loc[dropped['total times dropped'] == 0].index if x in regressors
]

final_anom_selected = [
    x for x in dropped.loc[dropped['total times dropped'] <= 1].index if x.startswith(
↪'Anomaly') or x in ('Harvey','2001','Great Recession') or x.startswith('cp')
]

lags = {
    'Hou-Dom':[1,12,24,36],
    'IAH-DOM':[1,12,24,36],
    'IAH-Int':[1,12,24,36],
}
```

[14]:
```
for l, f in fdict.items():
    f.drop_Xvars(*[x for x in f.get_regressor_names() if x not in final_anom_selected])
    f.add_seasonal_regressors('month',raw=False,sincos=True)
    f.diff()
```

9.5 Multivariate Forecasting

[15]:
```
mvfdict = fdict.copy()

for x in final_Xvars_selected:
    if x in regressors:
        mvfdict[x] = Forecaster(
            y=regressors[x].values[:-4],
            current_dates = regressors.index.values[:-4],
        )
        lags[x] = [1]
```

(continues on next page)

(continued from previous page)

```
for l, f in fdict.items():
    f.drop_Xvars(*[x for x in f.get_regressor_names() if x in regressors])

mvf = MVForecaster(*mvfdict.values(),names=mvfdict.keys())
mvf.add_optimizer_func(lambda x: x[0]*(1/3) + x[1]*(1/3) + x[2]*(1/3),'weighted') #␣
↪first three series only get weight
mvf.set_optimize_on('weighted')
mvf.set_test_length(fcst_horizon)
mvf.set_validation_metric('mae')
mvf.set_validation_length(36)
mvf
```

[15]:
```
MVForecaster(
    DateStartActuals=2001-01-01T00:00:00.000000000
    DateEndActuals=2021-09-01T00:00:00.000000000
    Freq=MS
    N_actuals=249
    N_series=6
    SeriesNames=['Hou-Dom', 'IAH-DOM', 'IAH-Int', 'Number of International  flights',
↪'Number of domestic flights', 'Revenue Passenger-miles/Domestic']
    ForecastLength=4
    Xvars=['Anomaly_2020-04-01', 'cp2', 'monthsin', 'monthcos']
    TestLength=4
    ValidationLength=36
    ValidationMetric=mae
    ForecastsEvaluated=[]
    CILevel=0.95
    BootstrapSamples=100
    CurrentEstimator=mlr
    OptimizeOn=weighted
)
```

[16]:
```
elasticnet = {
    'alpha':[i/10 for i in range(1,21)],
    'l1_ratio':[0,0.25,0.5,0.75,1],
    'normalizer':['scale','minmax',None],
    'lags':[lags],
}
knn = {
    'n_neighbors':range(2,76),
    'lags':[lags],
}

lightgbm = {
    'n_estimators':[150,200,250],
    'boosting_type':['gbdt','dart','goss'],
    'max_depth':[1,2,3],
    'learning_rate':[0.001,0.01,0.1],
    'lags':[lags],
}
mlp = {
```

(continues on next page)

(continued from previous page)

```
    'activation':['relu','tanh'],
    'hidden_layer_sizes':[(25,),(25,25,)],
    'solver':['lbfgs','adam'],
    'normalizer':['minmax','scale'],
    'lags':[lags],
}
mlr = {
    'normalizer':['scale','minmax',None],
    'lags':[lags],
}
sgd={
    'penalty':['l2','l1','elasticnet'],
    'l1_ratio':[0,0.15,0.5,0.85,1],
    'learning_rate':['invscaling','constant','optimal','adaptive'],
    'lags':[lags],
}
svr={
    'kernel':['linear'],
    'C':[.5,1,2,3],
    'epsilon':[0.01,0.1,0.5],
    'lags':[lags],
}
xgboost = {
    'n_estimators':[150,200,250],
    'scale_pos_weight':[5,10],
    'learning_rate':[0.1,0.2],
    'gamma':[0,3,5],
    'subsample':[0.8,0.9],
    'lags':[lags],
}
```

[17]:
```
models = (
    'knn',
    'mlr',
    'elasticnet',
    'sgd',
    'xgboost',
    'lightgbm',
    'mlp',
)

print('begining models cross validated')
for m in tqdm(models):
    mvf.set_estimator(m)
    mvf.ingest_grid(globals()[m])
    mvf.limit_grid_size(10)
    mvf.cross_validate(k=3,dynamic_tuning=fcst_horizon)
    mvf.auto_forecast(call_me=f'{m}_cv_mv')

print('begining models tuned')
for m in tqdm(models):
    mvf.set_estimator(m)
```

(continues on next page)

(continued from previous page)

```
    mvf.ingest_grid(globals()[m])
    mvf.limit_grid_size(10)
    mvf.tune(dynamic_tuning=fcst_horizon)
    mvf.auto_forecast(call_me=f'{m}_tune_mv')
```

```
begining models cross validated
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
begining models tuned
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
```

[18]:
```
mvf.set_best_model(determine_best_by='LevelTestSetMAPE')
results = mvf.export('model_summaries')
results_sub = results.loc[results['Series'].isin(airline_series)]
results_sub[['Series','ModelNickname','LevelTestSetR2','LevelTestSetMAPE']]
```

[18]:

	Series	ModelNickname	LevelTestSetR2	LevelTestSetMAPE
0	Hou-Dom	elasticnet_tune_mv	-0.008851	0.092853
1	Hou-Dom	knn_cv_mv	-0.122960	0.100203
2	Hou-Dom	mlr_cv_mv	-4.865199	0.209266
3	Hou-Dom	elasticnet_cv_mv	-0.046781	0.090712
4	Hou-Dom	sgd_cv_mv	0.044149	0.086127
5	Hou-Dom	xgboost_cv_mv	-1.806345	0.130049
6	Hou-Dom	lightgbm_cv_mv	-0.020278	0.087836
7	Hou-Dom	mlp_cv_mv	-0.082069	0.087113
8	Hou-Dom	knn_tune_mv	-0.731346	0.106951
9	Hou-Dom	mlr_tune_mv	-4.865199	0.209266
10	Hou-Dom	sgd_tune_mv	0.069918	0.086680
11	Hou-Dom	xgboost_tune_mv	-1.687364	0.129505
12	Hou-Dom	lightgbm_tune_mv	0.106759	0.086545
13	Hou-Dom	mlp_tune_mv	0.140452	0.080000
14	IAH-DOM	elasticnet_tune_mv	-2.675066	0.130365
15	IAH-DOM	knn_cv_mv	-1.521107	0.102719
16	IAH-DOM	mlr_cv_mv	-0.700337	0.065628
17	IAH-DOM	elasticnet_cv_mv	-3.282911	0.143407
18	IAH-DOM	sgd_cv_mv	-3.305075	0.144816
19	IAH-DOM	xgboost_cv_mv	-1.703854	0.112821
20	IAH-DOM	lightgbm_cv_mv	-3.602143	0.150592
21	IAH-DOM	mlp_cv_mv	-3.669990	0.150982
22	IAH-DOM	knn_tune_mv	-0.765313	0.077754
23	IAH-DOM	mlr_tune_mv	-0.700337	0.065628
24	IAH-DOM	sgd_tune_mv	-3.141806	0.141990
25	IAH-DOM	xgboost_tune_mv	-2.161605	0.126577
26	IAH-DOM	lightgbm_tune_mv	-2.480593	0.125745
27	IAH-DOM	mlp_tune_mv	-2.759960	0.128641
28	IAH-Int	elasticnet_tune_mv	-0.019007	0.153717
29	IAH-Int	knn_cv_mv	-0.411933	0.196223
30	IAH-Int	mlr_cv_mv	-2.984908	0.283815
31	IAH-Int	elasticnet_cv_mv	-0.175874	0.148786
32	IAH-Int	sgd_cv_mv	-0.637143	0.207411
33	IAH-Int	xgboost_cv_mv	-1.073559	0.225976
34	IAH-Int	lightgbm_cv_mv	-0.132948	0.156983

(continues on next page)

(continued from previous page)

```
35  IAH-Int           mlp_cv_mv       -0.302144          0.156178
36  IAH-Int         knn_tune_mv       -0.689363          0.192302
37  IAH-Int         mlr_tune_mv       -2.984908          0.283815
38  IAH-Int         sgd_tune_mv       -0.498934          0.200455
39  IAH-Int     xgboost_tune_mv       -0.300637          0.173677
40  IAH-Int    lightgbm_tune_mv       -0.715109          0.207723
41  IAH-Int         mlp_tune_mv       -0.430382          0.193184
```

[19]:
```
models_to_stack = (
    'knn_cv_mv',
    'elasticnet_cv_mv',
    'xgboost_cv_mv',
    'lightgbm_cv_mv',
)
mlp_stack(
    mvf,
    model_nicknames = models_to_stack,
    call_me='stacking_mv',
)
```

[20]:
```
mvf.set_best_model(determine_best_by='LevelTestSetMAPE')
results = mvf.export('model_summaries')
results_sub = results.loc[results['Series'].isin(airline_series)]
results_sub[['Series','ModelNickname','LevelTestSetR2','LevelTestSetMAPE']]
```

[20]:
```
     Series       ModelNickname  LevelTestSetR2  LevelTestSetMAPE
0   Hou-Dom         stacking_mv        0.137636          0.062217
1   Hou-Dom           knn_cv_mv       -0.122960          0.100203
2   Hou-Dom           mlr_cv_mv       -4.865199          0.209266
3   Hou-Dom    elasticnet_cv_mv       -0.046781          0.090712
4   Hou-Dom           sgd_cv_mv        0.044149          0.086127
5   Hou-Dom       xgboost_cv_mv       -1.806345          0.130049
6   Hou-Dom      lightgbm_cv_mv       -0.020278          0.087836
7   Hou-Dom           mlp_cv_mv       -0.082069          0.087113
8   Hou-Dom         knn_tune_mv       -0.731346          0.106951
9   Hou-Dom         mlr_tune_mv       -4.865199          0.209266
10  Hou-Dom  elasticnet_tune_mv       -0.008851          0.092853
11  Hou-Dom         sgd_tune_mv        0.069918          0.086680
12  Hou-Dom     xgboost_tune_mv       -1.687364          0.129505
13  Hou-Dom    lightgbm_tune_mv        0.106759          0.086545
14  Hou-Dom         mlp_tune_mv        0.140452          0.080000
15  IAH-DOM         stacking_mv       -1.030366          0.097983
16  IAH-DOM           knn_cv_mv       -1.521107          0.102719
17  IAH-DOM           mlr_cv_mv       -0.700337          0.065628
18  IAH-DOM    elasticnet_cv_mv       -3.282911          0.143407
19  IAH-DOM           sgd_cv_mv       -3.305075          0.144816
20  IAH-DOM       xgboost_cv_mv       -1.703854          0.112821
21  IAH-DOM      lightgbm_cv_mv       -3.602143          0.150592
22  IAH-DOM           mlp_cv_mv       -3.669990          0.150982
23  IAH-DOM         knn_tune_mv       -0.765313          0.077754
24  IAH-DOM         mlr_tune_mv       -0.700337          0.065628
25  IAH-DOM  elasticnet_tune_mv       -2.675066          0.130365
```

(continues on next page)

(continued from previous page)

```
26  IAH-DOM         sgd_tune_mv       -3.141806          0.141990
27  IAH-DOM     xgboost_tune_mv       -2.161605          0.126577
28  IAH-DOM    lightgbm_tune_mv       -2.480593          0.125745
29  IAH-DOM         mlp_tune_mv       -2.759960          0.128641
30  IAH-Int         stacking_mv        0.725500          0.062671
31  IAH-Int           knn_cv_mv       -0.411933          0.196223
32  IAH-Int           mlr_cv_mv       -2.984908          0.283815
33  IAH-Int    elasticnet_cv_mv       -0.175874          0.148786
34  IAH-Int           sgd_cv_mv       -0.637143          0.207411
35  IAH-Int       xgboost_cv_mv       -1.073559          0.225976
36  IAH-Int      lightgbm_cv_mv       -0.132948          0.156983
37  IAH-Int           mlp_cv_mv       -0.302144          0.156178
38  IAH-Int         knn_tune_mv       -0.689363          0.192302
39  IAH-Int         mlr_tune_mv       -2.984908          0.283815
40  IAH-Int  elasticnet_tune_mv       -0.019007          0.153717
41  IAH-Int         sgd_tune_mv       -0.498934          0.200455
42  IAH-Int     xgboost_tune_mv       -0.300637          0.173677
43  IAH-Int    lightgbm_tune_mv       -0.715109          0.207723
44  IAH-Int         mlp_tune_mv       -0.430382          0.193184
```

9.6 Univariate Forecasting

[21]:
```
f1, f2, f3 = break_mv_forecaster(mvf)[:3]
fdict[airline_series[0]] = f1
fdict[airline_series[1]] = f2
fdict[airline_series[2]] = f3
```

[22]:
```
f1
```

[22]:
```
Forecaster(
    DateStartActuals=2001-02-01T00:00:00.000000000
    DateEndActuals=2021-09-01T00:00:00.000000000
    Freq=MS
    N_actuals=248
    ForecastLength=4
    Xvars=['Anomaly_2020-04-01', 'cp2', 'monthsin', 'monthcos']
    Differenced=1
    TestLength=4
    ValidationLength=36
    ValidationMetric=rmse
    ForecastsEvaluated=['knn_cv_mv', 'mlr_cv_mv', 'elasticnet_cv_mv', 'sgd_cv_mv',
↪'xgboost_cv_mv', 'lightgbm_cv_mv', 'mlp_cv_mv', 'knn_tune_mv', 'mlr_tune_mv',
↪'elasticnet_tune_mv', 'sgd_tune_mv', 'xgboost_tune_mv', 'lightgbm_tune_mv', 'mlp_tune_
↪mv', 'stacking_mv']
    CILevel=0.95
    BootstrapSamples=100
    CurrentEstimator=None
)
```

[23]:
```
models = (
    'knn',
    'mlr',
    'elasticnet',
    'sgd',
    'xgboost',
    'lightgbm',
    'mlp',
    'hwes',
    'arima',
    'prophet',
    'silverkite',
    'theta',
)

for l, f in fdict.items():
    print(f'processing {l}')
    f.add_ar_terms(1)
    f.add_AR_terms((3,6))
    f.add_AR_terms((3,12))
    auto_arima(
        f,
        m=12,
        error_action='ignore',
    )
    f.manual_forecast(
        order=f.auto_arima_params['order'],
        seasonal_order=f.auto_arima_params['seasonal_order'],
        Xvars = 'all',
        call_me='auto_arima_anom',
    )
    f.tune_test_forecast(
        models,
        limit_grid_size=10,
        cross_validate=True,
        k=3,
        dynamic_tuning=fcst_horizon,
        suffix='_cv_uv',
    )
    f.tune_test_forecast(
        models,
        limit_grid_size=10,
        cross_validate=False,
        dynamic_tuning=fcst_horizon,
        suffix='_tune_uv',
    )
```

```
processing Hou-Dom
processing IAH-DOM
processing IAH-Int
```

[24]:
```
results = export_model_summaries(
    fdict,
```

(continues on next page)

(continued from previous page)

```
    determine_best_by='LevelTestSetMAPE',
)
results[
    [
        'Series',
        'ModelNickname',
        'LevelTestSetMAPE',
    ]
].sort_values(
    [
        'Series',
        'LevelTestSetMAPE',
    ]
).head(25)
```

[24]:

	Series	ModelNickname	LevelTestSetMAPE
0	Hou-Dom	prophet_cv_uv	0.024967
1	Hou-Dom	arima_tune_uv	0.031463
2	Hou-Dom	arima_cv_uv	0.031463
3	Hou-Dom	hwes_tune_uv	0.040938
4	Hou-Dom	hwes_cv_uv	0.040938
5	Hou-Dom	auto_arima_anom	0.041796
6	Hou-Dom	prophet_tune_uv	0.046722
7	Hou-Dom	sgd_cv_uv	0.054584
8	Hou-Dom	sgd_tune_uv	0.056646
9	Hou-Dom	stacking_mv	0.062217
10	Hou-Dom	auto_arima	0.071545
11	Hou-Dom	mlr_tune_uv	0.072219
12	Hou-Dom	mlr_cv_uv	0.072219
13	Hou-Dom	elasticnet_cv_uv	0.072386
14	Hou-Dom	elasticnet_tune_uv	0.073144
15	Hou-Dom	mlp_cv_uv	0.075766
16	Hou-Dom	mlp_tune_uv	0.079230
17	Hou-Dom	mlp_tune_mv	0.080000
18	Hou-Dom	lightgbm_cv_uv	0.084266
19	Hou-Dom	sgd_cv_mv	0.086127
20	Hou-Dom	lightgbm_tune_mv	0.086545
21	Hou-Dom	sgd_tune_mv	0.086680
22	Hou-Dom	mlp_cv_mv	0.087113
23	Hou-Dom	lightgbm_tune_uv	0.087297
24	Hou-Dom	lightgbm_cv_mv	0.087836

[25]:

```
models_to_stack = (
    'knn_cv_uv',
    'sgd_cv_uv',
    'elasticnet_cv_uv',
    'xgboost_cv_uv',
)

for l, f in fdict.items():
    mlp_stack(
        f,
```

(continues on next page)

(continued from previous page)

```
        model_nicknames=models_to_stack,
        call_me='stacking_uv'
    )
```

[26]:
```
for l, f in fdict.items():
    # trains 120 total lstm models to get forecast intervals probabilistically
    f.set_estimator('rnn')
    f.proba_forecast(
        lags=48,
        layers_struct=[('LSTM',{'units':100,'dropout':0})]*2 + [('Dense',{'units':10}
↪)]*2,
        epochs=8,
        random_seed=42,
        validation_split=0.2,
        call_me = 'lstm',
    )

    f.set_estimator('combo')
    f.manual_forecast(how='simple',call_me='avg')
    f.manual_forecast(how='weighted',determine_best_by='LevelTestSetMAPE',call_me=
↪'weighted')
```

```
Epoch 1/8
5/5 [==============================] - 4s 220ms/step - loss: 0.4620 - val_loss: 0.2296
Epoch 2/8
5/5 [==============================] - 0s 50ms/step - loss: 0.1978 - val_loss: 0.0809
Epoch 3/8
5/5 [==============================] - 0s 50ms/step - loss: 0.1269 - val_loss: 0.0839
Epoch 4/8
5/5 [==============================] - 0s 53ms/step - loss: 0.1066 - val_loss: 0.0730
Epoch 5/8
5/5 [==============================] - 0s 55ms/step - loss: 0.1078 - val_loss: 0.0715
Epoch 6/8
5/5 [==============================] - 0s 57ms/step - loss: 0.1027 - val_loss: 0.0679
Epoch 7/8
5/5 [==============================] - 0s 57ms/step - loss: 0.1011 - val_loss: 0.0684
Epoch 8/8
5/5 [==============================] - 0s 57ms/step - loss: 0.1015 - val_loss: 0.0667
Epoch 1/8
5/5 [==============================] - 3s 216ms/step - loss: 0.4601 - val_loss: 0.3736
Epoch 2/8
5/5 [==============================] - 0s 53ms/step - loss: 0.3318 - val_loss: 0.2527
Epoch 3/8
5/5 [==============================] - 0s 53ms/step - loss: 0.2008 - val_loss: 0.1088
Epoch 4/8
5/5 [==============================] - 0s 57ms/step - loss: 0.1475 - val_loss: 0.0784
Epoch 5/8
5/5 [==============================] - 0s 58ms/step - loss: 0.1171 - val_loss: 0.0935
Epoch 6/8
5/5 [==============================] - 0s 58ms/step - loss: 0.1159 - val_loss: 0.0686
Epoch 7/8
5/5 [==============================] - 0s 59ms/step - loss: 0.1070 - val_loss: 0.0794
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 58ms/step - loss: 0.1058 - val_loss: 0.0712
Epoch 1/8
5/5 [==============================] - 4s 212ms/step - loss: 0.4531 - val_loss: 0.3494
Epoch 2/8
5/5 [==============================] - 0s 56ms/step - loss: 0.3395 - val_loss: 0.2977
Epoch 3/8
5/5 [==============================] - 0s 62ms/step - loss: 0.2786 - val_loss: 0.2221
Epoch 4/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1931 - val_loss: 0.1028
Epoch 5/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1221 - val_loss: 0.1066
Epoch 6/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1128 - val_loss: 0.0671
Epoch 7/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1043 - val_loss: 0.0803
Epoch 8/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1072 - val_loss: 0.0695
Epoch 1/8
5/5 [==============================] - 3s 208ms/step - loss: 0.4293 - val_loss: 0.2660
Epoch 2/8
5/5 [==============================] - 0s 53ms/step - loss: 0.2586 - val_loss: 0.2127
Epoch 3/8
5/5 [==============================] - 0s 59ms/step - loss: 0.2098 - val_loss: 0.1606
Epoch 4/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1548 - val_loss: 0.0804
Epoch 5/8
5/5 [==============================] - 0s 60ms/step - loss: 0.1092 - val_loss: 0.0948
Epoch 6/8
5/5 [==============================] - 0s 62ms/step - loss: 0.1145 - val_loss: 0.0755
Epoch 7/8
5/5 [==============================] - 0s 61ms/step - loss: 0.1030 - val_loss: 0.0713
Epoch 8/8
5/5 [==============================] - 0s 60ms/step - loss: 0.1055 - val_loss: 0.0713
Epoch 1/8
5/5 [==============================] - 4s 209ms/step - loss: 0.4792 - val_loss: 0.2554
Epoch 2/8
5/5 [==============================] - 0s 61ms/step - loss: 0.2547 - val_loss: 0.2213
Epoch 3/8
5/5 [==============================] - 0s 62ms/step - loss: 0.2354 - val_loss: 0.1948
Epoch 4/8
5/5 [==============================] - 0s 63ms/step - loss: 0.2018 - val_loss: 0.1737
Epoch 5/8
5/5 [==============================] - 0s 61ms/step - loss: 0.1835 - val_loss: 0.1632
Epoch 6/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1655 - val_loss: 0.1375
Epoch 7/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1479 - val_loss: 0.1149
Epoch 8/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1307 - val_loss: 0.0922
Epoch 1/8
5/5 [==============================] - 3s 208ms/step - loss: 0.4198 - val_loss: 0.1965
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 58ms/step - loss: 0.1934 - val_loss: 0.1305
Epoch 3/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1269 - val_loss: 0.1067
Epoch 4/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1175 - val_loss: 0.0894
Epoch 5/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1071 - val_loss: 0.0720
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1016 - val_loss: 0.0698
Epoch 7/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1033 - val_loss: 0.0673
Epoch 8/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1021 - val_loss: 0.0675
Epoch 1/8
5/5 [==============================] - 4s 212ms/step - loss: 0.3767 - val_loss: 0.2195
Epoch 2/8
5/5 [==============================] - 0s 59ms/step - loss: 0.1603 - val_loss: 0.1068
Epoch 3/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1155 - val_loss: 0.0934
Epoch 4/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1072 - val_loss: 0.0758
Epoch 5/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1023 - val_loss: 0.0729
Epoch 6/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1031 - val_loss: 0.0683
Epoch 7/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1002 - val_loss: 0.0644
Epoch 8/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1005 - val_loss: 0.0648
Epoch 1/8
5/5 [==============================] - 3s 219ms/step - loss: 0.4492 - val_loss: 0.3560
Epoch 2/8
5/5 [==============================] - 0s 60ms/step - loss: 0.3102 - val_loss: 0.2044
Epoch 3/8
5/5 [==============================] - 0s 71ms/step - loss: 0.2073 - val_loss: 0.1401
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1334 - val_loss: 0.0936
Epoch 5/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1162 - val_loss: 0.0925
Epoch 6/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1104 - val_loss: 0.0837
Epoch 7/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1038 - val_loss: 0.0777
Epoch 8/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1062 - val_loss: 0.0714
Epoch 1/8
5/5 [==============================] - 4s 211ms/step - loss: 0.4593 - val_loss: 0.2216
Epoch 2/8
5/5 [==============================] - 0s 60ms/step - loss: 0.1930 - val_loss: 0.0755
Epoch 3/8
5/5 [==============================] - 0s 61ms/step - loss: 0.1168 - val_loss: 0.0891
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1054 - val_loss: 0.0771
Epoch 5/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1080 - val_loss: 0.0758
Epoch 6/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1047 - val_loss: 0.0671
Epoch 7/8
5/5 [==============================] - 0s 70ms/step - loss: 0.0987 - val_loss: 0.0685
Epoch 8/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1010 - val_loss: 0.0668
Epoch 1/8
5/5 [==============================] - 3s 211ms/step - loss: 0.3480 - val_loss: 0.1881
Epoch 2/8
5/5 [==============================] - 0s 55ms/step - loss: 0.1496 - val_loss: 0.1327
Epoch 3/8
5/5 [==============================] - 0s 58ms/step - loss: 0.1256 - val_loss: 0.0812
Epoch 4/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1118 - val_loss: 0.0679
Epoch 5/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1041 - val_loss: 0.0699
Epoch 6/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1041 - val_loss: 0.0719
Epoch 7/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1011 - val_loss: 0.0718
Epoch 8/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1016 - val_loss: 0.0662
Epoch 1/8
5/5 [==============================] - 4s 278ms/step - loss: 0.3927 - val_loss: 0.1465
Epoch 2/8
5/5 [==============================] - 0s 62ms/step - loss: 0.1637 - val_loss: 0.1382
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1637 - val_loss: 0.0927
Epoch 4/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1179 - val_loss: 0.0952
Epoch 5/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1103 - val_loss: 0.0755
Epoch 6/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1057 - val_loss: 0.0721
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1038 - val_loss: 0.0688
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1020 - val_loss: 0.0706
Epoch 1/8
5/5 [==============================] - 4s 254ms/step - loss: 0.3792 - val_loss: 0.2446
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2609 - val_loss: 0.2121
Epoch 3/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1907 - val_loss: 0.1374
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1291 - val_loss: 0.0814
Epoch 5/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1152 - val_loss: 0.0888
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1110 - val_loss: 0.0718
Epoch 7/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1024 - val_loss: 0.0704
Epoch 8/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1031 - val_loss: 0.0699
Epoch 1/8
5/5 [==============================] - 4s 263ms/step - loss: 0.4634 - val_loss: 0.1887
Epoch 2/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1608 - val_loss: 0.0798
Epoch 3/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1264 - val_loss: 0.0769
Epoch 4/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1073 - val_loss: 0.0790
Epoch 5/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1057 - val_loss: 0.0830
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1065 - val_loss: 0.0696
Epoch 7/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1033 - val_loss: 0.0670
Epoch 8/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1027 - val_loss: 0.0732
Epoch 1/8
5/5 [==============================] - 5s 364ms/step - loss: 0.4697 - val_loss: 0.3228
Epoch 2/8
5/5 [==============================] - 0s 100ms/step - loss: 0.2379 - val_loss: 0.1059
Epoch 3/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1345 - val_loss: 0.0838
Epoch 4/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1206 - val_loss: 0.0798
Epoch 5/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1090 - val_loss: 0.0798
Epoch 6/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1082 - val_loss: 0.0723
Epoch 7/8
5/5 [==============================] - 1s 146ms/step - loss: 0.1056 - val_loss: 0.0726
Epoch 8/8
5/5 [==============================] - 0s 100ms/step - loss: 0.1040 - val_loss: 0.0767
Epoch 1/8
5/5 [==============================] - 4s 223ms/step - loss: 0.3879 - val_loss: 0.2142
Epoch 2/8
5/5 [==============================] - 0s 62ms/step - loss: 0.1915 - val_loss: 0.1533
Epoch 3/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1613 - val_loss: 0.1085
Epoch 4/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1361 - val_loss: 0.0919
Epoch 5/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1199 - val_loss: 0.0895
Epoch 6/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1102 - val_loss: 0.0791
Epoch 7/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1039 - val_loss: 0.0719
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1019 - val_loss: 0.0672
Epoch 1/8
5/5 [==============================] - 5s 252ms/step - loss: 0.4495 - val_loss: 0.2936
Epoch 2/8
5/5 [==============================] - 0s 71ms/step - loss: 0.2243 - val_loss: 0.0909
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1233 - val_loss: 0.1121
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1221 - val_loss: 0.0722
Epoch 5/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1044 - val_loss: 0.0728
Epoch 6/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1050 - val_loss: 0.0705
Epoch 7/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1017 - val_loss: 0.0683
Epoch 8/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1035 - val_loss: 0.0710
Epoch 1/8
5/5 [==============================] - 3s 214ms/step - loss: 0.3382 - val_loss: 0.2025
Epoch 2/8
5/5 [==============================] - 0s 61ms/step - loss: 0.1671 - val_loss: 0.1272
Epoch 3/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1206 - val_loss: 0.0859
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1111 - val_loss: 0.0765
Epoch 5/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1075 - val_loss: 0.0655
Epoch 6/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1050 - val_loss: 0.0676
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1006 - val_loss: 0.0729
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1005 - val_loss: 0.0657
Epoch 1/8
5/5 [==============================] - 3s 211ms/step - loss: 0.3966 - val_loss: 0.2425
Epoch 2/8
5/5 [==============================] - 0s 59ms/step - loss: 0.1635 - val_loss: 0.1310
Epoch 3/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1218 - val_loss: 0.0867
Epoch 4/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1174 - val_loss: 0.0791
Epoch 5/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1061 - val_loss: 0.0781
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1054 - val_loss: 0.0676
Epoch 7/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1003 - val_loss: 0.0674
Epoch 8/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1002 - val_loss: 0.0673
Epoch 1/8
5/5 [==============================] - 5s 273ms/step - loss: 0.4536 - val_loss: 0.2352
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 60ms/step - loss: 0.2195 - val_loss: 0.0839
Epoch 3/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1171 - val_loss: 0.0999
Epoch 4/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1121 - val_loss: 0.0683
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1038 - val_loss: 0.0733
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1033 - val_loss: 0.0707
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1004 - val_loss: 0.0659
Epoch 8/8
5/5 [==============================] - 0s 72ms/step - loss: 0.0994 - val_loss: 0.0654
Epoch 1/8
5/5 [==============================] - 5s 226ms/step - loss: 0.4240 - val_loss: 0.2890
Epoch 2/8
5/5 [==============================] - 0s 59ms/step - loss: 0.2342 - val_loss: 0.1640
Epoch 3/8
5/5 [==============================] - 0s 61ms/step - loss: 0.1377 - val_loss: 0.0749
Epoch 4/8
5/5 [==============================] - 0s 61ms/step - loss: 0.1181 - val_loss: 0.0889
Epoch 5/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1039 - val_loss: 0.0670
Epoch 6/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1030 - val_loss: 0.0728
Epoch 7/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1036 - val_loss: 0.0721
Epoch 8/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1012 - val_loss: 0.0663
Epoch 1/8
5/5 [==============================] - 4s 244ms/step - loss: 0.4628 - val_loss: 0.3032
Epoch 2/8
5/5 [==============================] - 0s 60ms/step - loss: 0.2330 - val_loss: 0.1009
Epoch 3/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1380 - val_loss: 0.1133
Epoch 4/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1185 - val_loss: 0.0787
Epoch 5/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1099 - val_loss: 0.0818
Epoch 6/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1080 - val_loss: 0.0658
Epoch 7/8
5/5 [==============================] - 0s 64ms/step - loss: 0.0994 - val_loss: 0.0718
Epoch 8/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1015 - val_loss: 0.0680
Epoch 1/8
5/5 [==============================] - 3s 215ms/step - loss: 0.4780 - val_loss: 0.3311
Epoch 2/8
5/5 [==============================] - 0s 63ms/step - loss: 0.2221 - val_loss: 0.1361
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1313 - val_loss: 0.1162
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1200 - val_loss: 0.0776
Epoch 5/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1116 - val_loss: 0.0669
Epoch 6/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1046 - val_loss: 0.0743
Epoch 7/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1020 - val_loss: 0.0691
Epoch 8/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1035 - val_loss: 0.0699
Epoch 1/8
5/5 [==============================] - 4s 253ms/step - loss: 0.4216 - val_loss: 0.1989
Epoch 2/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1909 - val_loss: 0.1219
Epoch 3/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1425 - val_loss: 0.0693
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1113 - val_loss: 0.0675
Epoch 5/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1053 - val_loss: 0.0670
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1019 - val_loss: 0.0664
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1010 - val_loss: 0.0706
Epoch 8/8
5/5 [==============================] - 0s 73ms/step - loss: 0.0997 - val_loss: 0.0702
Epoch 1/8
5/5 [==============================] - 3s 218ms/step - loss: 0.4416 - val_loss: 0.2831
Epoch 2/8
5/5 [==============================] - 0s 65ms/step - loss: 0.2329 - val_loss: 0.1571
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1408 - val_loss: 0.1208
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1209 - val_loss: 0.0977
Epoch 5/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1094 - val_loss: 0.0827
Epoch 6/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1088 - val_loss: 0.0733
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1029 - val_loss: 0.0725
Epoch 8/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1052 - val_loss: 0.0700
Epoch 1/8
5/5 [==============================] - 4s 238ms/step - loss: 0.4120 - val_loss: 0.2061
Epoch 2/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1739 - val_loss: 0.1037
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1385 - val_loss: 0.1082
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1111 - val_loss: 0.0834
Epoch 5/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1076 - val_loss: 0.0746
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1039 - val_loss: 0.0659
Epoch 7/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1000 - val_loss: 0.0690
Epoch 8/8
5/5 [==============================] - 0s 72ms/step - loss: 0.0999 - val_loss: 0.0656
Epoch 1/8
5/5 [==============================] - 4s 236ms/step - loss: 0.3936 - val_loss: 0.2807
Epoch 2/8
5/5 [==============================] - 0s 60ms/step - loss: 0.2506 - val_loss: 0.1960
Epoch 3/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1686 - val_loss: 0.0824
Epoch 4/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1244 - val_loss: 0.0909
Epoch 5/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1101 - val_loss: 0.0748
Epoch 6/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1070 - val_loss: 0.0713
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1025 - val_loss: 0.0664
Epoch 8/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1012 - val_loss: 0.0674
Epoch 1/8
5/5 [==============================] - 4s 236ms/step - loss: 0.4210 - val_loss: 0.2066
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1626 - val_loss: 0.1242
Epoch 3/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1269 - val_loss: 0.0778
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1042 - val_loss: 0.0751
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1032 - val_loss: 0.0694
Epoch 6/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1003 - val_loss: 0.0681
Epoch 7/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1008 - val_loss: 0.0663
Epoch 8/8
5/5 [==============================] - 0s 79ms/step - loss: 0.0997 - val_loss: 0.0655
Epoch 1/8
5/5 [==============================] - 4s 228ms/step - loss: 0.4332 - val_loss: 0.2339
Epoch 2/8
5/5 [==============================] - 0s 59ms/step - loss: 0.1974 - val_loss: 0.1167
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1404 - val_loss: 0.1110
Epoch 4/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1149 - val_loss: 0.1094
Epoch 5/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1116 - val_loss: 0.0835
Epoch 6/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1125 - val_loss: 0.0689
Epoch 7/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1029 - val_loss: 0.0793
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1042 - val_loss: 0.0726
Epoch 1/8
5/5 [==============================] - 3s 207ms/step - loss: 0.4108 - val_loss: 0.2654
Epoch 2/8
5/5 [==============================] - 0s 59ms/step - loss: 0.2124 - val_loss: 0.1372
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1366 - val_loss: 0.0673
Epoch 4/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1074 - val_loss: 0.0849
Epoch 5/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1105 - val_loss: 0.0704
Epoch 6/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1043 - val_loss: 0.0697
Epoch 7/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1033 - val_loss: 0.0698
Epoch 8/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1030 - val_loss: 0.0681
Epoch 1/8
5/5 [==============================] - 4s 276ms/step - loss: 0.4871 - val_loss: 0.2833
Epoch 2/8
5/5 [==============================] - 0s 74ms/step - loss: 0.2129 - val_loss: 0.0989
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1284 - val_loss: 0.0944
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1180 - val_loss: 0.0703
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1036 - val_loss: 0.0751
Epoch 6/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1037 - val_loss: 0.0702
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1005 - val_loss: 0.0693
Epoch 8/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1014 - val_loss: 0.0664
Epoch 1/8
5/5 [==============================] - 4s 218ms/step - loss: 0.4543 - val_loss: 0.3302
Epoch 2/8
5/5 [==============================] - 0s 62ms/step - loss: 0.2753 - val_loss: 0.1645
Epoch 3/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1374 - val_loss: 0.1095
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1161 - val_loss: 0.0818
Epoch 5/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1076 - val_loss: 0.0694
Epoch 6/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1018 - val_loss: 0.0734
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1014 - val_loss: 0.0679
Epoch 8/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1014 - val_loss: 0.0704
Epoch 1/8
5/5 [==============================] - 4s 253ms/step - loss: 0.4470 - val_loss: 0.2336
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 75ms/step - loss: 0.2225 - val_loss: 0.1406
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1471 - val_loss: 0.0668
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1129 - val_loss: 0.0906
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1185 - val_loss: 0.0766
Epoch 6/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1116 - val_loss: 0.0668
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1067 - val_loss: 0.0732
Epoch 8/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1045 - val_loss: 0.0728
Epoch 1/8
5/5 [==============================] - 5s 345ms/step - loss: 0.3600 - val_loss: 0.1618
Epoch 2/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1455 - val_loss: 0.0905
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1219 - val_loss: 0.0847
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1056 - val_loss: 0.0826
Epoch 5/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1071 - val_loss: 0.0687
Epoch 6/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1035 - val_loss: 0.0654
Epoch 7/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1008 - val_loss: 0.0682
Epoch 8/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1010 - val_loss: 0.0656
Epoch 1/8
5/5 [==============================] - 4s 227ms/step - loss: 0.5048 - val_loss: 0.3985
Epoch 2/8
5/5 [==============================] - 0s 67ms/step - loss: 0.3078 - val_loss: 0.2169
Epoch 3/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1968 - val_loss: 0.1200
Epoch 4/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1295 - val_loss: 0.1103
Epoch 5/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1228 - val_loss: 0.0818
Epoch 6/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1097 - val_loss: 0.0803
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1051 - val_loss: 0.0740
Epoch 8/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1092 - val_loss: 0.0728
Epoch 1/8
5/5 [==============================] - 4s 235ms/step - loss: 0.4198 - val_loss: 0.2489
Epoch 2/8
5/5 [==============================] - 0s 68ms/step - loss: 0.2026 - val_loss: 0.1621
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1482 - val_loss: 0.1045
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1201 - val_loss: 0.0866
Epoch 5/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1117 - val_loss: 0.0732
Epoch 6/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1109 - val_loss: 0.0663
Epoch 7/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1006 - val_loss: 0.0688
Epoch 8/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1002 - val_loss: 0.0659
Epoch 1/8
5/5 [==============================] - 4s 218ms/step - loss: 0.4322 - val_loss: 0.2733
Epoch 2/8
5/5 [==============================] - 0s 71ms/step - loss: 0.2340 - val_loss: 0.1413
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1437 - val_loss: 0.0971
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1162 - val_loss: 0.0941
Epoch 5/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1137 - val_loss: 0.0844
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1042 - val_loss: 0.0739
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1054 - val_loss: 0.0710
Epoch 8/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1032 - val_loss: 0.0656
Epoch 1/8
5/5 [==============================] - 4s 283ms/step - loss: 0.4392 - val_loss: 0.1988
Epoch 2/8
5/5 [==============================] - 0s 61ms/step - loss: 0.1863 - val_loss: 0.1257
Epoch 3/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1447 - val_loss: 0.0869
Epoch 4/8
5/5 [==============================] - 0s 65ms/step - loss: 0.1090 - val_loss: 0.0933
Epoch 5/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1161 - val_loss: 0.0720
Epoch 6/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1033 - val_loss: 0.0663
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1015 - val_loss: 0.0674
Epoch 8/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1003 - val_loss: 0.0641
Epoch 1/8
5/5 [==============================] - 3s 211ms/step - loss: 0.4296 - val_loss: 0.2323
Epoch 2/8
5/5 [==============================] - 0s 60ms/step - loss: 0.1646 - val_loss: 0.1102
Epoch 3/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1151 - val_loss: 0.0960
Epoch 4/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1092 - val_loss: 0.0745
Epoch 5/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1056 - val_loss: 0.0773
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1055 - val_loss: 0.0713
Epoch 7/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1032 - val_loss: 0.0714
Epoch 8/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1014 - val_loss: 0.0706
Epoch 1/8
5/5 [==============================] - 4s 270ms/step - loss: 0.4104 - val_loss: 0.1197
Epoch 2/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1379 - val_loss: 0.0930
Epoch 3/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1173 - val_loss: 0.0695
Epoch 4/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1065 - val_loss: 0.0669
Epoch 5/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1066 - val_loss: 0.0765
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1069 - val_loss: 0.0753
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1034 - val_loss: 0.0696
Epoch 8/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1003 - val_loss: 0.0678
Epoch 1/8
5/5 [==============================] - 4s 233ms/step - loss: 0.3521 - val_loss: 0.1905
Epoch 2/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1519 - val_loss: 0.1034
Epoch 3/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1183 - val_loss: 0.0824
Epoch 4/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1071 - val_loss: 0.0845
Epoch 5/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1071 - val_loss: 0.0708
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1071 - val_loss: 0.0703
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1056 - val_loss: 0.0693
Epoch 8/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1047 - val_loss: 0.0704
Epoch 1/8
5/5 [==============================] - 5s 292ms/step - loss: 0.5010 - val_loss: 0.3337
Epoch 2/8
5/5 [==============================] - 0s 68ms/step - loss: 0.2583 - val_loss: 0.1420
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1443 - val_loss: 0.1009
Epoch 4/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1182 - val_loss: 0.1119
Epoch 5/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1146 - val_loss: 0.0859
Epoch 6/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1122 - val_loss: 0.0973
Epoch 7/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1080 - val_loss: 0.0878
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1078 - val_loss: 0.0931
Epoch 1/8
5/5 [==============================] - 5s 320ms/step - loss: 0.4257 - val_loss: 0.2465
Epoch 2/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1725 - val_loss: 0.1440
Epoch 3/8
5/5 [==============================] - 0s 94ms/step - loss: 0.1413 - val_loss: 0.1158
Epoch 4/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1138 - val_loss: 0.1017
Epoch 5/8
5/5 [==============================] - 1s 108ms/step - loss: 0.1163 - val_loss: 0.0972
Epoch 6/8
5/5 [==============================] - 1s 169ms/step - loss: 0.1077 - val_loss: 0.0872
Epoch 7/8
5/5 [==============================] - 1s 109ms/step - loss: 0.1090 - val_loss: 0.0923
Epoch 8/8
5/5 [==============================] - 1s 106ms/step - loss: 0.1093 - val_loss: 0.0898
Epoch 1/8
5/5 [==============================] - 4s 229ms/step - loss: 0.4553 - val_loss: 0.1722
Epoch 2/8
5/5 [==============================] - 0s 62ms/step - loss: 0.1758 - val_loss: 0.1273
Epoch 3/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1420 - val_loss: 0.0989
Epoch 4/8
5/5 [==============================] - 1s 117ms/step - loss: 0.1249 - val_loss: 0.1008
Epoch 5/8
5/5 [==============================] - 1s 188ms/step - loss: 0.1110 - val_loss: 0.1013
Epoch 6/8
5/5 [==============================] - 0s 92ms/step - loss: 0.1116 - val_loss: 0.0906
Epoch 7/8
5/5 [==============================] - 0s 91ms/step - loss: 0.1110 - val_loss: 0.0879
Epoch 8/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1095 - val_loss: 0.0937
Epoch 1/8
5/5 [==============================] - 3s 217ms/step - loss: 0.4561 - val_loss: 0.2289
Epoch 2/8
5/5 [==============================] - 0s 74ms/step - loss: 0.2222 - val_loss: 0.1831
Epoch 3/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1762 - val_loss: 0.1572
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1512 - val_loss: 0.1140
Epoch 5/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1244 - val_loss: 0.0985
Epoch 6/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1089 - val_loss: 0.0924
Epoch 7/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1091 - val_loss: 0.0890
Epoch 8/8
5/5 [==============================] - 0s 107ms/step - loss: 0.1119 - val_loss: 0.0891
Epoch 1/8
5/5 [==============================] - 4s 220ms/step - loss: 0.4510 - val_loss: 0.2694
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1826 - val_loss: 0.1577
Epoch 3/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1341 - val_loss: 0.0952
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1209 - val_loss: 0.1013
Epoch 5/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1107 - val_loss: 0.0908
Epoch 6/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1085 - val_loss: 0.0910
Epoch 7/8
5/5 [==============================] - 0s 64ms/step - loss: 0.1070 - val_loss: 0.0860
Epoch 8/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1061 - val_loss: 0.0876
Epoch 1/8
5/5 [==============================] - 3s 222ms/step - loss: 0.4646 - val_loss: 0.2459
Epoch 2/8
5/5 [==============================] - 0s 62ms/step - loss: 0.1769 - val_loss: 0.1648
Epoch 3/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1333 - val_loss: 0.1120
Epoch 4/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1254 - val_loss: 0.0942
Epoch 5/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1122 - val_loss: 0.1037
Epoch 6/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1151 - val_loss: 0.0904
Epoch 7/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1084 - val_loss: 0.0922
Epoch 8/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1091 - val_loss: 0.0873
Epoch 1/8
5/5 [==============================] - 4s 220ms/step - loss: 0.4846 - val_loss: 0.3335
Epoch 2/8
5/5 [==============================] - 1s 124ms/step - loss: 0.2892 - val_loss: 0.2251
Epoch 3/8
5/5 [==============================] - 0s 84ms/step - loss: 0.2149 - val_loss: 0.1717
Epoch 4/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1393 - val_loss: 0.1104
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1258 - val_loss: 0.0953
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1101 - val_loss: 0.0956
Epoch 7/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1113 - val_loss: 0.0926
Epoch 8/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1102 - val_loss: 0.0887
Epoch 1/8
5/5 [==============================] - 3s 219ms/step - loss: 0.5529 - val_loss: 0.3860
Epoch 2/8
5/5 [==============================] - 0s 64ms/step - loss: 0.3421 - val_loss: 0.2278
Epoch 3/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1963 - val_loss: 0.1094
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1275 - val_loss: 0.1176
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1247 - val_loss: 0.0981
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1137 - val_loss: 0.0946
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1116 - val_loss: 0.0886
Epoch 8/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1086 - val_loss: 0.0869
Epoch 1/8
5/5 [==============================] - 4s 220ms/step - loss: 0.4849 - val_loss: 0.3383
Epoch 2/8
5/5 [==============================] - 0s 66ms/step - loss: 0.2608 - val_loss: 0.1820
Epoch 3/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1520 - val_loss: 0.1439
Epoch 4/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1365 - val_loss: 0.1055
Epoch 5/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1168 - val_loss: 0.1007
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1155 - val_loss: 0.0919
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1077 - val_loss: 0.0887
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1076 - val_loss: 0.0874
Epoch 1/8
5/5 [==============================] - 4s 219ms/step - loss: 0.5882 - val_loss: 0.3979
Epoch 2/8
5/5 [==============================] - 0s 70ms/step - loss: 0.3944 - val_loss: 0.3196
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.3007 - val_loss: 0.2020
Epoch 4/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1702 - val_loss: 0.1035
Epoch 5/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1305 - val_loss: 0.0974
Epoch 6/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1117 - val_loss: 0.0956
Epoch 7/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1145 - val_loss: 0.0932
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1084 - val_loss: 0.0984
Epoch 1/8
5/5 [==============================] - 3s 217ms/step - loss: 0.4444 - val_loss: 0.2905
Epoch 2/8
5/5 [==============================] - 0s 65ms/step - loss: 0.2042 - val_loss: 0.1075
Epoch 3/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1278 - val_loss: 0.0964
Epoch 4/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1193 - val_loss: 0.1031
Epoch 5/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1146 - val_loss: 0.0865
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1068 - val_loss: 0.0916
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1072 - val_loss: 0.0880
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1075 - val_loss: 0.0876
Epoch 1/8
5/5 [==============================] - 3s 239ms/step - loss: 0.5037 - val_loss: 0.3601
Epoch 2/8
5/5 [==============================] - 0s 70ms/step - loss: 0.2783 - val_loss: 0.2187
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1971 - val_loss: 0.1513
Epoch 4/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1363 - val_loss: 0.1029
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1162 - val_loss: 0.0950
Epoch 6/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1129 - val_loss: 0.0976
Epoch 7/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1109 - val_loss: 0.0891
Epoch 8/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1092 - val_loss: 0.0878
Epoch 1/8
5/5 [==============================] - 4s 230ms/step - loss: 0.4347 - val_loss: 0.2433
Epoch 2/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1994 - val_loss: 0.1239
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1266 - val_loss: 0.1119
Epoch 4/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1237 - val_loss: 0.0857
Epoch 5/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1156 - val_loss: 0.1002
Epoch 6/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1092 - val_loss: 0.0965
Epoch 7/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1104 - val_loss: 0.0855
Epoch 8/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1072 - val_loss: 0.0893
Epoch 1/8
5/5 [==============================] - 4s 236ms/step - loss: 0.4672 - val_loss: 0.3460
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2682 - val_loss: 0.2080
Epoch 3/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1880 - val_loss: 0.1477
Epoch 4/8
5/5 [==============================] - 0s 91ms/step - loss: 0.1397 - val_loss: 0.1021
Epoch 5/8
5/5 [==============================] - 0s 99ms/step - loss: 0.1178 - val_loss: 0.0896
Epoch 6/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1164 - val_loss: 0.0922
Epoch 7/8
5/5 [==============================] - 0s 102ms/step - loss: 0.1106 - val_loss: 0.0891
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1082 - val_loss: 0.0941
Epoch 1/8
5/5 [==============================] - 4s 260ms/step - loss: 0.4650 - val_loss: 0.3402
Epoch 2/8
5/5 [==============================] - 0s 74ms/step - loss: 0.2561 - val_loss: 0.2101
Epoch 3/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1868 - val_loss: 0.1034
Epoch 4/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1276 - val_loss: 0.0948
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1246 - val_loss: 0.0955
Epoch 6/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1088 - val_loss: 0.0968
Epoch 7/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1086 - val_loss: 0.0899
Epoch 8/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1077 - val_loss: 0.0859
Epoch 1/8
5/5 [==============================] - 4s 232ms/step - loss: 0.4851 - val_loss: 0.2090
Epoch 2/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1803 - val_loss: 0.1379
Epoch 3/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1515 - val_loss: 0.0933
Epoch 4/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1217 - val_loss: 0.1008
Epoch 5/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1137 - val_loss: 0.0982
Epoch 6/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1094 - val_loss: 0.0995
Epoch 7/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1106 - val_loss: 0.0870
Epoch 8/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1071 - val_loss: 0.0898
Epoch 1/8
5/5 [==============================] - 3s 247ms/step - loss: 0.5362 - val_loss: 0.3065
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2498 - val_loss: 0.1286
Epoch 3/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1349 - val_loss: 0.1222
Epoch 4/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1174 - val_loss: 0.0950
Epoch 5/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1112 - val_loss: 0.0963
Epoch 6/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1128 - val_loss: 0.0901
Epoch 7/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1057 - val_loss: 0.0920
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1072 - val_loss: 0.0873
Epoch 1/8
5/5 [==============================] - 3s 226ms/step - loss: 0.3716 - val_loss: 0.1545
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1517 - val_loss: 0.0954
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1158 - val_loss: 0.1110
Epoch 4/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1160 - val_loss: 0.0982
Epoch 5/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1128 - val_loss: 0.0883
Epoch 6/8
5/5 [==============================] - 1s 112ms/step - loss: 0.1098 - val_loss: 0.0931
Epoch 7/8
5/5 [==============================] - 0s 94ms/step - loss: 0.1102 - val_loss: 0.0879
Epoch 8/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1084 - val_loss: 0.0883
Epoch 1/8
5/5 [==============================] - 5s 282ms/step - loss: 0.4775 - val_loss: 0.2264
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1833 - val_loss: 0.1396
Epoch 3/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1232 - val_loss: 0.1239
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1276 - val_loss: 0.0917
Epoch 5/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1123 - val_loss: 0.0958
Epoch 6/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1100 - val_loss: 0.0873
Epoch 7/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1078 - val_loss: 0.0877
Epoch 8/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1061 - val_loss: 0.0881
Epoch 1/8
5/5 [==============================] - 5s 294ms/step - loss: 0.4750 - val_loss: 0.3055
Epoch 2/8
5/5 [==============================] - 0s 67ms/step - loss: 0.2100 - val_loss: 0.1387
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1286 - val_loss: 0.1087
Epoch 4/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1174 - val_loss: 0.0927
Epoch 5/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1140 - val_loss: 0.0927
Epoch 6/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1076 - val_loss: 0.0908
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1094 - val_loss: 0.0899
Epoch 8/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1072 - val_loss: 0.0885
Epoch 1/8
5/5 [==============================] - 5s 285ms/step - loss: 0.4185 - val_loss: 0.2925
Epoch 2/8
5/5 [==============================] - 0s 62ms/step - loss: 0.2248 - val_loss: 0.1906
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1634 - val_loss: 0.1329
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1273 - val_loss: 0.1117
Epoch 5/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1206 - val_loss: 0.0983
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1089 - val_loss: 0.0910
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1091 - val_loss: 0.0863
Epoch 8/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1057 - val_loss: 0.0870
Epoch 1/8
5/5 [==============================] - 5s 301ms/step - loss: 0.3663 - val_loss: 0.2243
Epoch 2/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1755 - val_loss: 0.1262
Epoch 3/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1199 - val_loss: 0.1040
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1199 - val_loss: 0.0972
Epoch 5/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1142 - val_loss: 0.0873
Epoch 6/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1111 - val_loss: 0.0910
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1109 - val_loss: 0.0921
Epoch 8/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1094 - val_loss: 0.0893
Epoch 1/8
5/5 [==============================] - 5s 308ms/step - loss: 0.5264 - val_loss: 0.2916
Epoch 2/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1966 - val_loss: 0.1625
Epoch 3/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1344 - val_loss: 0.1269
Epoch 4/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1255 - val_loss: 0.0911
Epoch 5/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1108 - val_loss: 0.0987
Epoch 6/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1101 - val_loss: 0.0876
Epoch 7/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1072 - val_loss: 0.0917
Epoch 8/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1066 - val_loss: 0.0869
Epoch 1/8
5/5 [==============================] - 4s 305ms/step - loss: 0.5563 - val_loss: 0.3354
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2302 - val_loss: 0.1510
Epoch 3/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1635 - val_loss: 0.1071
Epoch 4/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1216 - val_loss: 0.1167
Epoch 5/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1121 - val_loss: 0.0957
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1128 - val_loss: 0.0951
Epoch 7/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1093 - val_loss: 0.0885
Epoch 8/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1080 - val_loss: 0.0922
Epoch 1/8
5/5 [==============================] - 6s 323ms/step - loss: 0.4861 - val_loss: 0.3270
Epoch 2/8
5/5 [==============================] - 0s 71ms/step - loss: 0.2357 - val_loss: 0.1664
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1416 - val_loss: 0.1177
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1341 - val_loss: 0.1032
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1083 - val_loss: 0.0918
Epoch 6/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1131 - val_loss: 0.0954
Epoch 7/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1096 - val_loss: 0.0893
Epoch 8/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1091 - val_loss: 0.0928
Epoch 1/8
5/5 [==============================] - 4s 260ms/step - loss: 0.5360 - val_loss: 0.3559
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2752 - val_loss: 0.1948
Epoch 3/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1606 - val_loss: 0.1081
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1255 - val_loss: 0.0964
Epoch 5/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1161 - val_loss: 0.1077
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1115 - val_loss: 0.0965
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1142 - val_loss: 0.0916
Epoch 8/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1088 - val_loss: 0.0975
Epoch 1/8
5/5 [==============================] - 4s 254ms/step - loss: 0.5516 - val_loss: 0.4349
Epoch 2/8
5/5 [==============================] - 0s 63ms/step - loss: 0.3751 - val_loss: 0.2875
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.2946 - val_loss: 0.2508
Epoch 4/8
5/5 [==============================] - 0s 71ms/step - loss: 0.2303 - val_loss: 0.1590
Epoch 5/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1448 - val_loss: 0.1343
Epoch 6/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1412 - val_loss: 0.1232
Epoch 7/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1188 - val_loss: 0.1091
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1157 - val_loss: 0.1005
Epoch 1/8
5/5 [==============================] - 4s 370ms/step - loss: 0.5056 - val_loss: 0.3230
Epoch 2/8
5/5 [==============================] - 0s 81ms/step - loss: 0.2670 - val_loss: 0.1642
Epoch 3/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1688 - val_loss: 0.1187
Epoch 4/8
5/5 [==============================] - 0s 106ms/step - loss: 0.1262 - val_loss: 0.0949
Epoch 5/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1158 - val_loss: 0.1073
Epoch 6/8
5/5 [==============================] - 0s 101ms/step - loss: 0.1152 - val_loss: 0.0943
Epoch 7/8
5/5 [==============================] - 1s 128ms/step - loss: 0.1100 - val_loss: 0.0874
Epoch 8/8
5/5 [==============================] - 0s 95ms/step - loss: 0.1073 - val_loss: 0.0874
Epoch 1/8
5/5 [==============================] - 4s 221ms/step - loss: 0.4814 - val_loss: 0.2698
Epoch 2/8
5/5 [==============================] - 0s 68ms/step - loss: 0.2795 - val_loss: 0.2185
Epoch 3/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1783 - val_loss: 0.1246
Epoch 4/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1294 - val_loss: 0.1070
Epoch 5/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1281 - val_loss: 0.1037
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1070 - val_loss: 0.0908
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1095 - val_loss: 0.0894
Epoch 8/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1080 - val_loss: 0.0883
Epoch 1/8
5/5 [==============================] - 5s 288ms/step - loss: 0.4528 - val_loss: 0.2696
Epoch 2/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1771 - val_loss: 0.1441
Epoch 3/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1343 - val_loss: 0.1034
Epoch 4/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1194 - val_loss: 0.0997
Epoch 5/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1132 - val_loss: 0.0934
Epoch 6/8
5/5 [==============================] - 0s 95ms/step - loss: 0.1091 - val_loss: 0.0922
Epoch 7/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1089 - val_loss: 0.0882
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1087 - val_loss: 0.0877
Epoch 1/8
5/5 [==============================] - 4s 252ms/step - loss: 0.5027 - val_loss: 0.2607
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1905 - val_loss: 0.1532
Epoch 3/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1396 - val_loss: 0.1259
Epoch 4/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1193 - val_loss: 0.1017
Epoch 5/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1144 - val_loss: 0.0879
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1083 - val_loss: 0.0928
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1089 - val_loss: 0.0867
Epoch 8/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1066 - val_loss: 0.0857
Epoch 1/8
5/5 [==============================] - 4s 258ms/step - loss: 0.4938 - val_loss: 0.3788
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2757 - val_loss: 0.2120
Epoch 3/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1612 - val_loss: 0.1489
Epoch 4/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1382 - val_loss: 0.1211
Epoch 5/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1250 - val_loss: 0.0924
Epoch 6/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1146 - val_loss: 0.0951
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1094 - val_loss: 0.0934
Epoch 8/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1076 - val_loss: 0.0875
Epoch 1/8
5/5 [==============================] - 4s 311ms/step - loss: 0.4695 - val_loss: 0.2814
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2014 - val_loss: 0.1572
Epoch 3/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1323 - val_loss: 0.1183
Epoch 4/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1199 - val_loss: 0.1050
Epoch 5/8
5/5 [==============================] - 0s 99ms/step - loss: 0.1108 - val_loss: 0.0888
Epoch 6/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1113 - val_loss: 0.0903
Epoch 7/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1078 - val_loss: 0.0953
Epoch 8/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1089 - val_loss: 0.0864
Epoch 1/8
5/5 [==============================] - 6s 512ms/step - loss: 0.4966 - val_loss: 0.3159
Epoch 2/8
5/5 [==============================] - 0s 96ms/step - loss: 0.3002 - val_loss: 0.2229
Epoch 3/8
5/5 [==============================] - 1s 104ms/step - loss: 0.2252 - val_loss: 0.1459
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 103ms/step - loss: 0.1540 - val_loss: 0.0924
Epoch 5/8
5/5 [==============================] - 0s 92ms/step - loss: 0.1209 - val_loss: 0.0979
Epoch 6/8
5/5 [==============================] - 0s 94ms/step - loss: 0.1168 - val_loss: 0.0888
Epoch 7/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1162 - val_loss: 0.0884
Epoch 8/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1123 - val_loss: 0.0913
Epoch 1/8
5/5 [==============================] - 4s 244ms/step - loss: 0.4177 - val_loss: 0.2801
Epoch 2/8
5/5 [==============================] - 0s 87ms/step - loss: 0.2180 - val_loss: 0.1653
Epoch 3/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1400 - val_loss: 0.1016
Epoch 4/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1161 - val_loss: 0.0970
Epoch 5/8
5/5 [==============================] - 0s 92ms/step - loss: 0.1074 - val_loss: 0.0906
Epoch 6/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1082 - val_loss: 0.0918
Epoch 7/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1087 - val_loss: 0.0872
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1080 - val_loss: 0.0859
Epoch 1/8
5/5 [==============================] - 4s 258ms/step - loss: 0.4474 - val_loss: 0.1747
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1594 - val_loss: 0.1318
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1330 - val_loss: 0.1019
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1195 - val_loss: 0.0955
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1132 - val_loss: 0.0905
Epoch 6/8
5/5 [==============================] - 0s 68ms/step - loss: 0.1106 - val_loss: 0.0971
Epoch 7/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1091 - val_loss: 0.0885
Epoch 8/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1085 - val_loss: 0.0869
Epoch 1/8
5/5 [==============================] - 4s 241ms/step - loss: 0.5250 - val_loss: 0.2925
Epoch 2/8
5/5 [==============================] - 0s 63ms/step - loss: 0.2537 - val_loss: 0.1388
Epoch 3/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1612 - val_loss: 0.1164
Epoch 4/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1247 - val_loss: 0.0902
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1137 - val_loss: 0.0922
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1119 - val_loss: 0.0883
Epoch 7/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1099 - val_loss: 0.0927
Epoch 8/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1054 - val_loss: 0.0893
Epoch 1/8
5/5 [==============================] - 4s 230ms/step - loss: 0.4727 - val_loss: 0.0974
Epoch 2/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1783 - val_loss: 0.1104
Epoch 3/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1376 - val_loss: 0.0958
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1193 - val_loss: 0.1119
Epoch 5/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1121 - val_loss: 0.1027
Epoch 6/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1141 - val_loss: 0.0879
Epoch 7/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1091 - val_loss: 0.0984
Epoch 8/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1089 - val_loss: 0.0873
Epoch 1/8
5/5 [==============================] - 4s 210ms/step - loss: 0.4265 - val_loss: 0.2164
Epoch 2/8
5/5 [==============================] - 0s 60ms/step - loss: 0.1712 - val_loss: 0.1509
Epoch 3/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1374 - val_loss: 0.1068
Epoch 4/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1222 - val_loss: 0.0899
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1173 - val_loss: 0.0928
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1099 - val_loss: 0.0969
Epoch 7/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1072 - val_loss: 0.0899
Epoch 8/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1082 - val_loss: 0.0857
Epoch 1/8
5/5 [==============================] - 3s 221ms/step - loss: 0.5075 - val_loss: 0.3002
Epoch 2/8
5/5 [==============================] - 0s 64ms/step - loss: 0.2871 - val_loss: 0.2482
Epoch 3/8
5/5 [==============================] - 0s 82ms/step - loss: 0.2305 - val_loss: 0.1712
Epoch 4/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1598 - val_loss: 0.1141
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1235 - val_loss: 0.0978
Epoch 6/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1178 - val_loss: 0.1063
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1140 - val_loss: 0.0952
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1094 - val_loss: 0.0900
Epoch 1/8
5/5 [==============================] - 4s 227ms/step - loss: 0.5536 - val_loss: 0.4502
Epoch 2/8
5/5 [==============================] - 0s 72ms/step - loss: 0.3756 - val_loss: 0.1998
Epoch 3/8
5/5 [==============================] - 0s 93ms/step - loss: 0.2155 - val_loss: 0.1760
Epoch 4/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1966 - val_loss: 0.1442
Epoch 5/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1933 - val_loss: 0.1454
Epoch 6/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1865 - val_loss: 0.1267
Epoch 7/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1819 - val_loss: 0.1286
Epoch 8/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1797 - val_loss: 0.1242
Epoch 1/8
5/5 [==============================] - 4s 240ms/step - loss: 0.4798 - val_loss: 0.3670
Epoch 2/8
5/5 [==============================] - 0s 75ms/step - loss: 0.3003 - val_loss: 0.1744
Epoch 3/8
5/5 [==============================] - 0s 78ms/step - loss: 0.2073 - val_loss: 0.1636
Epoch 4/8
5/5 [==============================] - 0s 81ms/step - loss: 0.2014 - val_loss: 0.1420
Epoch 5/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1847 - val_loss: 0.1382
Epoch 6/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1846 - val_loss: 0.1315
Epoch 7/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1863 - val_loss: 0.1316
Epoch 8/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1847 - val_loss: 0.1271
Epoch 1/8
5/5 [==============================] - 4s 209ms/step - loss: 0.5232 - val_loss: 0.3507
Epoch 2/8
5/5 [==============================] - 0s 64ms/step - loss: 0.3048 - val_loss: 0.2290
Epoch 3/8
5/5 [==============================] - 0s 74ms/step - loss: 0.2401 - val_loss: 0.1498
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1992 - val_loss: 0.1597
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1957 - val_loss: 0.1383
Epoch 6/8
5/5 [==============================] - 1s 124ms/step - loss: 0.1862 - val_loss: 0.1296
Epoch 7/8
5/5 [==============================] - 0s 93ms/step - loss: 0.1834 - val_loss: 0.1303
Epoch 8/8
5/5 [==============================] - 0s 94ms/step - loss: 0.1810 - val_loss: 0.1267
Epoch 1/8
5/5 [==============================] - 4s 261ms/step - loss: 0.4982 - val_loss: 0.3606
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 66ms/step - loss: 0.2965 - val_loss: 0.2275
Epoch 3/8
5/5 [==============================] - 0s 66ms/step - loss: 0.2272 - val_loss: 0.1766
Epoch 4/8
5/5 [==============================] - 0s 71ms/step - loss: 0.2007 - val_loss: 0.1383
Epoch 5/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1938 - val_loss: 0.1332
Epoch 6/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1872 - val_loss: 0.1418
Epoch 7/8
5/5 [==============================] - 0s 63ms/step - loss: 0.1820 - val_loss: 0.1313
Epoch 8/8
5/5 [==============================] - 0s 66ms/step - loss: 0.1843 - val_loss: 0.1310
Epoch 1/8
5/5 [==============================] - 4s 238ms/step - loss: 0.5299 - val_loss: 0.3215
Epoch 2/8
5/5 [==============================] - 0s 68ms/step - loss: 0.3126 - val_loss: 0.2267
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.2429 - val_loss: 0.1552
Epoch 4/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1917 - val_loss: 0.1275
Epoch 5/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1858 - val_loss: 0.1449
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1879 - val_loss: 0.1297
Epoch 7/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1859 - val_loss: 0.1307
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1790 - val_loss: 0.1302
Epoch 1/8
5/5 [==============================] - 4s 301ms/step - loss: 0.4187 - val_loss: 0.3205
Epoch 2/8
5/5 [==============================] - 0s 84ms/step - loss: 0.2412 - val_loss: 0.1885
Epoch 3/8
5/5 [==============================] - 0s 91ms/step - loss: 0.2167 - val_loss: 0.1410
Epoch 4/8
5/5 [==============================] - 0s 94ms/step - loss: 0.1855 - val_loss: 0.1577
Epoch 5/8
5/5 [==============================] - 1s 116ms/step - loss: 0.1870 - val_loss: 0.1310
Epoch 6/8
5/5 [==============================] - 1s 112ms/step - loss: 0.1853 - val_loss: 0.1311
Epoch 7/8
5/5 [==============================] - 1s 117ms/step - loss: 0.1784 - val_loss: 0.1354
Epoch 8/8
5/5 [==============================] - 1s 113ms/step - loss: 0.1808 - val_loss: 0.1283
Epoch 1/8
5/5 [==============================] - 4s 245ms/step - loss: 0.5952 - val_loss: 0.4157
Epoch 2/8
5/5 [==============================] - 0s 71ms/step - loss: 0.3230 - val_loss: 0.2512
Epoch 3/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2258 - val_loss: 0.1597
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.2052 - val_loss: 0.1442
Epoch 5/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1840 - val_loss: 0.1402
Epoch 6/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1927 - val_loss: 0.1274
Epoch 7/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1841 - val_loss: 0.1380
Epoch 8/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1851 - val_loss: 0.1312
Epoch 1/8
5/5 [==============================] - 4s 255ms/step - loss: 0.4880 - val_loss: 0.2641
Epoch 2/8
5/5 [==============================] - 0s 81ms/step - loss: 0.2221 - val_loss: 0.2057
Epoch 3/8
5/5 [==============================] - 0s 79ms/step - loss: 0.2058 - val_loss: 0.1359
Epoch 4/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1908 - val_loss: 0.1431
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1881 - val_loss: 0.1383
Epoch 6/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1823 - val_loss: 0.1276
Epoch 7/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1794 - val_loss: 0.1322
Epoch 8/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1807 - val_loss: 0.1278
Epoch 1/8
5/5 [==============================] - 4s 240ms/step - loss: 0.4971 - val_loss: 0.3298
Epoch 2/8
5/5 [==============================] - 0s 78ms/step - loss: 0.2524 - val_loss: 0.2043
Epoch 3/8
5/5 [==============================] - 0s 77ms/step - loss: 0.2239 - val_loss: 0.1314
Epoch 4/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1897 - val_loss: 0.1434
Epoch 5/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1915 - val_loss: 0.1418
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1849 - val_loss: 0.1382
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1885 - val_loss: 0.1260
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1837 - val_loss: 0.1259
Epoch 1/8
5/5 [==============================] - 4s 237ms/step - loss: 0.4911 - val_loss: 0.2808
Epoch 2/8
5/5 [==============================] - 0s 61ms/step - loss: 0.2422 - val_loss: 0.1590
Epoch 3/8
5/5 [==============================] - 0s 74ms/step - loss: 0.2079 - val_loss: 0.1519
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1859 - val_loss: 0.1287
Epoch 5/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1826 - val_loss: 0.1341
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1821 - val_loss: 0.1304
Epoch 7/8
5/5 [==============================] - 0s 91ms/step - loss: 0.1797 - val_loss: 0.1264
Epoch 8/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1798 - val_loss: 0.1324
Epoch 1/8
5/5 [==============================] - 3s 228ms/step - loss: 0.4821 - val_loss: 0.2923
Epoch 2/8
5/5 [==============================] - 0s 70ms/step - loss: 0.2435 - val_loss: 0.2100
Epoch 3/8
5/5 [==============================] - 0s 75ms/step - loss: 0.2161 - val_loss: 0.1352
Epoch 4/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1860 - val_loss: 0.1377
Epoch 5/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1822 - val_loss: 0.1253
Epoch 6/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1830 - val_loss: 0.1278
Epoch 7/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1797 - val_loss: 0.1280
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1791 - val_loss: 0.1238
Epoch 1/8
5/5 [==============================] - 3s 234ms/step - loss: 0.4641 - val_loss: 0.2769
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2632 - val_loss: 0.2183
Epoch 3/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2270 - val_loss: 0.1384
Epoch 4/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1961 - val_loss: 0.1570
Epoch 5/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1949 - val_loss: 0.1515
Epoch 6/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1893 - val_loss: 0.1377
Epoch 7/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1828 - val_loss: 0.1284
Epoch 8/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1833 - val_loss: 0.1274
Epoch 1/8
5/5 [==============================] - 3s 254ms/step - loss: 0.5581 - val_loss: 0.4006
Epoch 2/8
5/5 [==============================] - 0s 66ms/step - loss: 0.3953 - val_loss: 0.3233
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.3392 - val_loss: 0.3120
Epoch 4/8
5/5 [==============================] - 0s 69ms/step - loss: 0.3046 - val_loss: 0.2468
Epoch 5/8
5/5 [==============================] - 0s 72ms/step - loss: 0.2650 - val_loss: 0.1999
Epoch 6/8
5/5 [==============================] - 0s 67ms/step - loss: 0.2121 - val_loss: 0.1515
Epoch 7/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1900 - val_loss: 0.1345
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1866 - val_loss: 0.1530
Epoch 1/8
5/5 [==============================] - 3s 219ms/step - loss: 0.5849 - val_loss: 0.4818
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.4327 - val_loss: 0.3128
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.2740 - val_loss: 0.1485
Epoch 4/8
5/5 [==============================] - 0s 71ms/step - loss: 0.2161 - val_loss: 0.1772
Epoch 5/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1933 - val_loss: 0.1384
Epoch 6/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1879 - val_loss: 0.1389
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1815 - val_loss: 0.1426
Epoch 8/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1832 - val_loss: 0.1342
Epoch 1/8
5/5 [==============================] - 4s 271ms/step - loss: 0.5064 - val_loss: 0.2341
Epoch 2/8
5/5 [==============================] - 1s 125ms/step - loss: 0.2361 - val_loss: 0.1721
Epoch 3/8
5/5 [==============================] - 0s 100ms/step - loss: 0.2123 - val_loss: 0.1434
Epoch 4/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1916 - val_loss: 0.1415
Epoch 5/8
5/5 [==============================] - 0s 91ms/step - loss: 0.1859 - val_loss: 0.1296
Epoch 6/8
5/5 [==============================] - 0s 101ms/step - loss: 0.1816 - val_loss: 0.1322
Epoch 7/8
5/5 [==============================] - 0s 96ms/step - loss: 0.1808 - val_loss: 0.1265
Epoch 8/8
5/5 [==============================] - 1s 106ms/step - loss: 0.1828 - val_loss: 0.1269
Epoch 1/8
5/5 [==============================] - 4s 238ms/step - loss: 0.5633 - val_loss: 0.3690
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.3807 - val_loss: 0.3072
Epoch 3/8
5/5 [==============================] - 0s 93ms/step - loss: 0.2994 - val_loss: 0.1856
Epoch 4/8
5/5 [==============================] - 1s 111ms/step - loss: 0.2083 - val_loss: 0.1478
Epoch 5/8
5/5 [==============================] - 0s 103ms/step - loss: 0.1959 - val_loss: 0.1475
Epoch 6/8
5/5 [==============================] - 1s 113ms/step - loss: 0.1881 - val_loss: 0.1373
Epoch 7/8
5/5 [==============================] - 1s 117ms/step - loss: 0.1792 - val_loss: 0.1307
Epoch 8/8
5/5 [==============================] - 1s 112ms/step - loss: 0.1807 - val_loss: 0.1298
Epoch 1/8
5/5 [==============================] - 4s 212ms/step - loss: 0.4948 - val_loss: 0.2011
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 58ms/step - loss: 0.2015 - val_loss: 0.1755
Epoch 3/8
5/5 [==============================] - 0s 61ms/step - loss: 0.2129 - val_loss: 0.1405
Epoch 4/8
5/5 [==============================] - 0s 67ms/step - loss: 0.1879 - val_loss: 0.1366
Epoch 5/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1856 - val_loss: 0.1402
Epoch 6/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1878 - val_loss: 0.1279
Epoch 7/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1803 - val_loss: 0.1265
Epoch 8/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1792 - val_loss: 0.1253
Epoch 1/8
5/5 [==============================] - 5s 262ms/step - loss: 0.4490 - val_loss: 0.1686
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2219 - val_loss: 0.1521
Epoch 3/8
5/5 [==============================] - 0s 95ms/step - loss: 0.2045 - val_loss: 0.1359
Epoch 4/8
5/5 [==============================] - 1s 127ms/step - loss: 0.1872 - val_loss: 0.1453
Epoch 5/8
5/5 [==============================] - 1s 112ms/step - loss: 0.1829 - val_loss: 0.1326
Epoch 6/8
5/5 [==============================] - 0s 93ms/step - loss: 0.1884 - val_loss: 0.1339
Epoch 7/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1801 - val_loss: 0.1358
Epoch 8/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1829 - val_loss: 0.1274
Epoch 1/8
5/5 [==============================] - 4s 288ms/step - loss: 0.5309 - val_loss: 0.3266
Epoch 2/8
5/5 [==============================] - 0s 79ms/step - loss: 0.2580 - val_loss: 0.1865
Epoch 3/8
5/5 [==============================] - 0s 75ms/step - loss: 0.2138 - val_loss: 0.1293
Epoch 4/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1913 - val_loss: 0.1482
Epoch 5/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1944 - val_loss: 0.1318
Epoch 6/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1877 - val_loss: 0.1312
Epoch 7/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1874 - val_loss: 0.1232
Epoch 8/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1818 - val_loss: 0.1266
Epoch 1/8
5/5 [==============================] - 4s 335ms/step - loss: 0.4827 - val_loss: 0.3860
Epoch 2/8
5/5 [==============================] - 0s 74ms/step - loss: 0.3317 - val_loss: 0.2813
Epoch 3/8
5/5 [==============================] - 0s 84ms/step - loss: 0.2618 - val_loss: 0.1749
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 0s 82ms/step - loss: 0.2005 - val_loss: 0.1354
Epoch 5/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1917 - val_loss: 0.1450
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1853 - val_loss: 0.1288
Epoch 7/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1808 - val_loss: 0.1299
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1808 - val_loss: 0.1286
Epoch 1/8
5/5 [==============================] - 4s 237ms/step - loss: 0.4630 - val_loss: 0.2591
Epoch 2/8
5/5 [==============================] - 0s 67ms/step - loss: 0.2357 - val_loss: 0.1823
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.2134 - val_loss: 0.1352
Epoch 4/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1881 - val_loss: 0.1349
Epoch 5/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1885 - val_loss: 0.1388
Epoch 6/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1846 - val_loss: 0.1279
Epoch 7/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1799 - val_loss: 0.1246
Epoch 8/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1800 - val_loss: 0.1269
Epoch 1/8
5/5 [==============================] - 4s 255ms/step - loss: 0.5214 - val_loss: 0.4254
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.3736 - val_loss: 0.3013
Epoch 3/8
5/5 [==============================] - 0s 72ms/step - loss: 0.2641 - val_loss: 0.1910
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.2119 - val_loss: 0.1532
Epoch 5/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1940 - val_loss: 0.1392
Epoch 6/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1952 - val_loss: 0.1437
Epoch 7/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1850 - val_loss: 0.1410
Epoch 8/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1844 - val_loss: 0.1304
Epoch 1/8
5/5 [==============================] - 3s 226ms/step - loss: 0.4843 - val_loss: 0.2980
Epoch 2/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2650 - val_loss: 0.1744
Epoch 3/8
5/5 [==============================] - 0s 70ms/step - loss: 0.2016 - val_loss: 0.1664
Epoch 4/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1910 - val_loss: 0.1399
Epoch 5/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1891 - val_loss: 0.1403
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1834 - val_loss: 0.1377
Epoch 7/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1808 - val_loss: 0.1242
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1789 - val_loss: 0.1285
Epoch 1/8
5/5 [==============================] - 5s 350ms/step - loss: 0.5210 - val_loss: 0.3093
Epoch 2/8
5/5 [==============================] - 0s 103ms/step - loss: 0.3043 - val_loss: 0.2095
Epoch 3/8
5/5 [==============================] - 0s 108ms/step - loss: 0.2177 - val_loss: 0.1496
Epoch 4/8
5/5 [==============================] - 0s 91ms/step - loss: 0.1954 - val_loss: 0.1648
Epoch 5/8
5/5 [==============================] - 0s 92ms/step - loss: 0.1959 - val_loss: 0.1374
Epoch 6/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1871 - val_loss: 0.1316
Epoch 7/8
5/5 [==============================] - 0s 93ms/step - loss: 0.1873 - val_loss: 0.1380
Epoch 8/8
5/5 [==============================] - 1s 114ms/step - loss: 0.1873 - val_loss: 0.1343
Epoch 1/8
5/5 [==============================] - 4s 246ms/step - loss: 0.5340 - val_loss: 0.3565
Epoch 2/8
5/5 [==============================] - 0s 70ms/step - loss: 0.3171 - val_loss: 0.1720
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.2071 - val_loss: 0.1418
Epoch 4/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1883 - val_loss: 0.1413
Epoch 5/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1892 - val_loss: 0.1318
Epoch 6/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1862 - val_loss: 0.1300
Epoch 7/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1799 - val_loss: 0.1243
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1772 - val_loss: 0.1259
Epoch 1/8
5/5 [==============================] - 4s 263ms/step - loss: 0.5136 - val_loss: 0.3324
Epoch 2/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2833 - val_loss: 0.1539
Epoch 3/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1945 - val_loss: 0.1585
Epoch 4/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1955 - val_loss: 0.1433
Epoch 5/8
5/5 [==============================] - 0s 80ms/step - loss: 0.1866 - val_loss: 0.1302
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1874 - val_loss: 0.1364
Epoch 7/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1828 - val_loss: 0.1287
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1807 - val_loss: 0.1332
Epoch 1/8
5/5 [==============================] - 3s 244ms/step - loss: 0.5185 - val_loss: 0.3172
Epoch 2/8
5/5 [==============================] - 0s 77ms/step - loss: 0.2466 - val_loss: 0.2061
Epoch 3/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1986 - val_loss: 0.1545
Epoch 4/8
5/5 [==============================] - 0s 81ms/step - loss: 0.1940 - val_loss: 0.1388
Epoch 5/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1885 - val_loss: 0.1329
Epoch 6/8
5/5 [==============================] - 0s 78ms/step - loss: 0.1835 - val_loss: 0.1288
Epoch 7/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1822 - val_loss: 0.1259
Epoch 8/8
5/5 [==============================] - 0s 79ms/step - loss: 0.1796 - val_loss: 0.1345
Epoch 1/8
5/5 [==============================] - 3s 229ms/step - loss: 0.5872 - val_loss: 0.3900
Epoch 2/8
5/5 [==============================] - 0s 75ms/step - loss: 0.3193 - val_loss: 0.2960
Epoch 3/8
5/5 [==============================] - 0s 79ms/step - loss: 0.2569 - val_loss: 0.2088
Epoch 4/8
5/5 [==============================] - 0s 82ms/step - loss: 0.2314 - val_loss: 0.1755
Epoch 5/8
5/5 [==============================] - 0s 83ms/step - loss: 0.1940 - val_loss: 0.1636
Epoch 6/8
5/5 [==============================] - 1s 121ms/step - loss: 0.1943 - val_loss: 0.1457
Epoch 7/8
5/5 [==============================] - 0s 98ms/step - loss: 0.1837 - val_loss: 0.1427
Epoch 8/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1839 - val_loss: 0.1304
Epoch 1/8
5/5 [==============================] - 4s 242ms/step - loss: 0.4376 - val_loss: 0.3040
Epoch 2/8
5/5 [==============================] - 0s 68ms/step - loss: 0.2356 - val_loss: 0.1868
Epoch 3/8
5/5 [==============================] - 0s 69ms/step - loss: 0.2124 - val_loss: 0.1483
Epoch 4/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1983 - val_loss: 0.1436
Epoch 5/8
5/5 [==============================] - 0s 71ms/step - loss: 0.1882 - val_loss: 0.1330
Epoch 6/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1837 - val_loss: 0.1369
Epoch 7/8
5/5 [==============================] - 0s 72ms/step - loss: 0.1831 - val_loss: 0.1230
Epoch 8/8
5/5 [==============================] - 0s 70ms/step - loss: 0.1795 - val_loss: 0.1289
Epoch 1/8
5/5 [==============================] - 4s 227ms/step - loss: 0.4241 - val_loss: 0.2990
```

(continues on next page)

(continued from previous page)

```
Epoch 2/8
5/5 [==============================] - 0s 78ms/step - loss: 0.2557 - val_loss: 0.1962
Epoch 3/8
5/5 [==============================] - 0s 76ms/step - loss: 0.2040 - val_loss: 0.1549
Epoch 4/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1984 - val_loss: 0.1368
Epoch 5/8
5/5 [==============================] - 0s 82ms/step - loss: 0.1985 - val_loss: 0.1474
Epoch 6/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1891 - val_loss: 0.1443
Epoch 7/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1866 - val_loss: 0.1277
Epoch 8/8
5/5 [==============================] - 0s 77ms/step - loss: 0.1803 - val_loss: 0.1315
Epoch 1/8
5/5 [==============================] - 4s 227ms/step - loss: 0.6345 - val_loss: 0.5964
Epoch 2/8
5/5 [==============================] - 0s 63ms/step - loss: 0.5611 - val_loss: 0.5045
Epoch 3/8
5/5 [==============================] - 0s 68ms/step - loss: 0.4514 - val_loss: 0.3171
Epoch 4/8
5/5 [==============================] - 0s 70ms/step - loss: 0.2691 - val_loss: 0.1828
Epoch 5/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2009 - val_loss: 0.1522
Epoch 6/8
5/5 [==============================] - 0s 62ms/step - loss: 0.1918 - val_loss: 0.1298
Epoch 7/8
5/5 [==============================] - 0s 69ms/step - loss: 0.1816 - val_loss: 0.1314
Epoch 8/8
5/5 [==============================] - 0s 75ms/step - loss: 0.1831 - val_loss: 0.1387
Epoch 1/8
5/5 [==============================] - 4s 227ms/step - loss: 0.5176 - val_loss: 0.3632
Epoch 2/8
5/5 [==============================] - 0s 76ms/step - loss: 0.2917 - val_loss: 0.1954
Epoch 3/8
5/5 [==============================] - 0s 74ms/step - loss: 0.2084 - val_loss: 0.1636
Epoch 4/8
5/5 [==============================] - 0s 73ms/step - loss: 0.2035 - val_loss: 0.1425
Epoch 5/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1906 - val_loss: 0.1327
Epoch 6/8
5/5 [==============================] - 0s 74ms/step - loss: 0.1843 - val_loss: 0.1407
Epoch 7/8
5/5 [==============================] - 0s 76ms/step - loss: 0.1824 - val_loss: 0.1326
Epoch 8/8
5/5 [==============================] - 0s 73ms/step - loss: 0.1805 - val_loss: 0.1276
Epoch 1/8
5/5 [==============================] - 9s 619ms/step - loss: 0.5797 - val_loss: 0.3233
Epoch 2/8
5/5 [==============================] - 1s 115ms/step - loss: 0.2873 - val_loss: 0.2037
Epoch 3/8
5/5 [==============================] - 1s 110ms/step - loss: 0.2205 - val_loss: 0.1515
```

(continues on next page)

(continued from previous page)

```
Epoch 4/8
5/5 [==============================] - 1s 111ms/step - loss: 0.1900 - val_loss: 0.1438
Epoch 5/8
5/5 [==============================] - 1s 112ms/step - loss: 0.1948 - val_loss: 0.1343
Epoch 6/8
5/5 [==============================] - 1s 108ms/step - loss: 0.1818 - val_loss: 0.1380
Epoch 7/8
5/5 [==============================] - 1s 138ms/step - loss: 0.1807 - val_loss: 0.1310
Epoch 8/8
5/5 [==============================] - 1s 104ms/step - loss: 0.1865 - val_loss: 0.1359
Epoch 1/8
5/5 [==============================] - 6s 347ms/step - loss: 0.4531 - val_loss: 0.2934
Epoch 2/8
5/5 [==============================] - 0s 86ms/step - loss: 0.2282 - val_loss: 0.1922
Epoch 3/8
5/5 [==============================] - 0s 97ms/step - loss: 0.2197 - val_loss: 0.1455
Epoch 4/8
5/5 [==============================] - 1s 136ms/step - loss: 0.1874 - val_loss: 0.1386
Epoch 5/8
5/5 [==============================] - 1s 107ms/step - loss: 0.1883 - val_loss: 0.1387
Epoch 6/8
5/5 [==============================] - 0s 94ms/step - loss: 0.1869 - val_loss: 0.1407
Epoch 7/8
5/5 [==============================] - 0s 99ms/step - loss: 0.1847 - val_loss: 0.1297
Epoch 8/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1815 - val_loss: 0.1287
Epoch 1/8
5/5 [==============================] - 5s 328ms/step - loss: 0.5409 - val_loss: 0.3259
Epoch 2/8
5/5 [==============================] - 0s 74ms/step - loss: 0.2661 - val_loss: 0.1302
Epoch 3/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1925 - val_loss: 0.1536
Epoch 4/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1915 - val_loss: 0.1486
Epoch 5/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1858 - val_loss: 0.1414
Epoch 6/8
5/5 [==============================] - 0s 91ms/step - loss: 0.1827 - val_loss: 0.1381
Epoch 7/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1850 - val_loss: 0.1300
Epoch 8/8
5/5 [==============================] - 0s 84ms/step - loss: 0.1866 - val_loss: 0.1332
Epoch 1/8
5/5 [==============================] - 5s 294ms/step - loss: 0.5476 - val_loss: 0.2092
Epoch 2/8
5/5 [==============================] - 0s 75ms/step - loss: 0.2396 - val_loss: 0.1394
Epoch 3/8
5/5 [==============================] - 0s 83ms/step - loss: 0.2041 - val_loss: 0.1674
Epoch 4/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1932 - val_loss: 0.1419
Epoch 5/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1873 - val_loss: 0.1302
```

(continues on next page)

(continued from previous page)

```
Epoch 6/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1809 - val_loss: 0.1300
Epoch 7/8
5/5 [==============================] - 0s 86ms/step - loss: 0.1819 - val_loss: 0.1334
Epoch 8/8
5/5 [==============================] - 0s 85ms/step - loss: 0.1775 - val_loss: 0.1294
Epoch 1/8
5/5 [==============================] - 5s 392ms/step - loss: 0.5127 - val_loss: 0.1769
Epoch 2/8
5/5 [==============================] - 0s 76ms/step - loss: 0.2590 - val_loss: 0.1552
Epoch 3/8
5/5 [==============================] - 0s 89ms/step - loss: 0.2232 - val_loss: 0.1886
Epoch 4/8
5/5 [==============================] - 0s 89ms/step - loss: 0.2017 - val_loss: 0.1344
Epoch 5/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1878 - val_loss: 0.1282
Epoch 6/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1829 - val_loss: 0.1369
Epoch 7/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1868 - val_loss: 0.1273
Epoch 8/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1797 - val_loss: 0.1324
Epoch 1/8
5/5 [==============================] - 5s 340ms/step - loss: 0.4706 - val_loss: 0.2935
Epoch 2/8
5/5 [==============================] - 0s 78ms/step - loss: 0.2333 - val_loss: 0.1686
Epoch 3/8
5/5 [==============================] - 0s 87ms/step - loss: 0.2142 - val_loss: 0.1574
Epoch 4/8
5/5 [==============================] - 0s 91ms/step - loss: 0.1905 - val_loss: 0.1354
Epoch 5/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1826 - val_loss: 0.1339
Epoch 6/8
5/5 [==============================] - 0s 93ms/step - loss: 0.1837 - val_loss: 0.1357
Epoch 7/8
5/5 [==============================] - 0s 87ms/step - loss: 0.1836 - val_loss: 0.1285
Epoch 8/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1812 - val_loss: 0.1267
Epoch 1/8
5/5 [==============================] - 5s 330ms/step - loss: 0.4341 - val_loss: 0.2047
Epoch 2/8
5/5 [==============================] - 0s 75ms/step - loss: 0.2194 - val_loss: 0.1852
Epoch 3/8
5/5 [==============================] - 0s 84ms/step - loss: 0.2155 - val_loss: 0.1368
Epoch 4/8
5/5 [==============================] - 0s 93ms/step - loss: 0.1913 - val_loss: 0.1391
Epoch 5/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1832 - val_loss: 0.1303
Epoch 6/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1830 - val_loss: 0.1276
Epoch 7/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1808 - val_loss: 0.1262
```

(continues on next page)

(continued from previous page)

```
Epoch 8/8
5/5 [==============================] - 0s 89ms/step - loss: 0.1813 - val_loss: 0.1277
Epoch 1/8
5/5 [==============================] - 5s 314ms/step - loss: 0.4232 - val_loss: 0.2485
Epoch 2/8
5/5 [==============================] - 0s 80ms/step - loss: 0.2193 - val_loss: 0.1619
Epoch 3/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1991 - val_loss: 0.1393
Epoch 4/8
5/5 [==============================] - 0s 90ms/step - loss: 0.1880 - val_loss: 0.1284
Epoch 5/8
5/5 [==============================] - 0s 95ms/step - loss: 0.1798 - val_loss: 0.1299
Epoch 6/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1817 - val_loss: 0.1267
Epoch 7/8
5/5 [==============================] - 0s 88ms/step - loss: 0.1809 - val_loss: 0.1282
Epoch 8/8
5/5 [==============================] - 0s 92ms/step - loss: 0.1810 - val_loss: 0.1328
```

[27]:
```
results = export_model_summaries(fdict,determine_best_by='LevelTestSetMAPE')
results[['Series','ModelNickname','LevelTestSetMAPE']].sort_values(['Series',
↪'LevelTestSetMAPE']).head(25)
```

[27]:

	Series	ModelNickname	LevelTestSetMAPE
0	Hou-Dom	prophet_cv_uv	0.024967
1	Hou-Dom	arima_tune_uv	0.031463
2	Hou-Dom	arima_cv_uv	0.031463
3	Hou-Dom	hwes_tune_uv	0.040938
4	Hou-Dom	hwes_cv_uv	0.040938
5	Hou-Dom	auto_arima_anom	0.041796
6	Hou-Dom	prophet_tune_uv	0.046722
7	Hou-Dom	sgd_cv_uv	0.054584
8	Hou-Dom	sgd_tune_uv	0.056646
9	Hou-Dom	stacking_mv	0.062217
10	Hou-Dom	auto_arima	0.071545
11	Hou-Dom	mlr_tune_uv	0.072219
12	Hou-Dom	mlr_cv_uv	0.072219
13	Hou-Dom	elasticnet_cv_uv	0.072386
14	Hou-Dom	weighted	0.072712
15	Hou-Dom	elasticnet_tune_uv	0.073144
16	Hou-Dom	mlp_cv_uv	0.075766
17	Hou-Dom	mlp_tune_uv	0.079230
18	Hou-Dom	mlp_tune_mv	0.080000
19	Hou-Dom	avg	0.081842
20	Hou-Dom	lightgbm_cv_uv	0.084266
21	Hou-Dom	sgd_cv_mv	0.086127
22	Hou-Dom	lightgbm_tune_mv	0.086545
23	Hou-Dom	sgd_tune_mv	0.086680
24	Hou-Dom	mlp_cv_mv	0.087113

[28]:
```
test = pd.read_csv(
    'data/Future Passengers.csv',
```

(continues on next page)

(continued from previous page)

```
    parse_dates=['DATE'],
    index_col=0,
)
test.head()
```

[28]:

DATE	HOU-Int	Hou-Dom	IAH-DOM	IAH-Int
2021-10-01	27153	498237	1375815	285498
2021-11-01	34242	495800	1356136	336156
2021-12-01	39101	477752	1331893	417561
2022-01-01	29084	387446	1036051	287998

9.7 Export Results

[29]:
```
final_results = results.copy()
for l, f in fdict.items():
    df = pd.DataFrame()
    fcsts = f.export('lvl_fcsts').set_index('DATE')
    for c in fcsts:
        df.loc[c,'FcstMAPE' + l] = metrics.mape(test[l],fcsts[c])
    df['Series'] = l
    final_results = final_results.merge(
        df.reset_index(),
        how='left',
        left_on=['ModelNickname','Series'],
        right_on=['index','Series'],
    )
final_results['FcstMAPE'] = final_results[
    [
        c for c in final_results if c.startswith('FcstMAPE')
    ]
].apply(
    lambda x: (
        x[0]
        if not np.isnan(x[0])
        else x[1] if not np.isnan(x[1])
        else x[2] if not np.isnan(x[2])
        else x[3]
    ),
    axis=1,
)
final_results.drop(
    [
        c for c in final_results if (
            c.startswith('FcstMAPE') and c != 'FcstMAPE'
        ) or c.startswith('index')
    ],
    axis=1,
    inplace=True,
)
```

(continues on next page)

(continued from previous page)

```
final_results['ObjectEstimator'] = final_results['ModelNickname'].apply(
    lambda x: 'MVForecaster' if '_mv' in x else 'Forecaster'
)
final_results['Used Anomalies'] = final_results['Xvars'].apply(
    lambda x: False if x is None else final_anom_selected[0] in x
)
```

[30]:
```
final_results[['Series','ModelNickname','LevelTestSetMAPE','FcstMAPE']].sort_values([
↪'Series','FcstMAPE']).head(10)
```

[30]:

	Series	ModelNickname	LevelTestSetMAPE	FcstMAPE
7	Hou-Dom	sgd_cv_uv	0.054584	0.027731
8	Hou-Dom	sgd_tune_uv	0.056646	0.034350
23	Hou-Dom	sgd_tune_mv	0.086680	0.049739
36	Hou-Dom	knn_tune_mv	0.106951	0.052050
21	Hou-Dom	sgd_cv_mv	0.086127	0.054210
15	Hou-Dom	elasticnet_tune_uv	0.073144	0.058379
0	Hou-Dom	prophet_cv_uv	0.024967	0.060044
6	Hou-Dom	prophet_tune_uv	0.046722	0.060630
25	Hou-Dom	lightgbm_tune_uv	0.087297	0.060856
14	Hou-Dom	weighted	0.072712	0.061285

[31]:
```
final_results.to_csv('final_results.csv',index=False)
```

[35]:
```
pd.pivot_table(
    final_results,
    index='Estimator',
    values='FcstMAPE',
    aggfunc=np.mean,
).sort_values('FcstMAPE')
```

[35]:

Estimator	FcstMAPE
sgd	0.059227
hwes	0.070423
combo	0.071353
prophet	0.076977
arima	0.078932
mlr	0.082993
lightgbm	0.086944
stacking	0.093409
silverkite	0.099222
knn	0.100048
mlp	0.108065
elasticnet	0.110694
rnn	0.124109
xgboost	0.126917
theta	0.153371

[33]:
```
pd.pivot_table(
    final_results,
    index='ObjectEstimator',
```

(continues on next page)

(continued from previous page)

```
    values='FcstMAPE',
    aggfunc=np.mean,
).sort_values('FcstMAPE')
```

[33]:
```
                 FcstMAPE
ObjectEstimator
Forecaster       0.091701
MVForecaster     0.101191
```

[34]:
```
with open('fdict.pckl', 'wb') as fl:
    pickle.dump(fdict,fl)
```

[ ]:
```

```

CHAPTER

TEN

HOLT-WINTERS EXPONENTIAL SMOOTHING

This notebook demonstrates running an HWES model in scalecast.

- See the documentation.

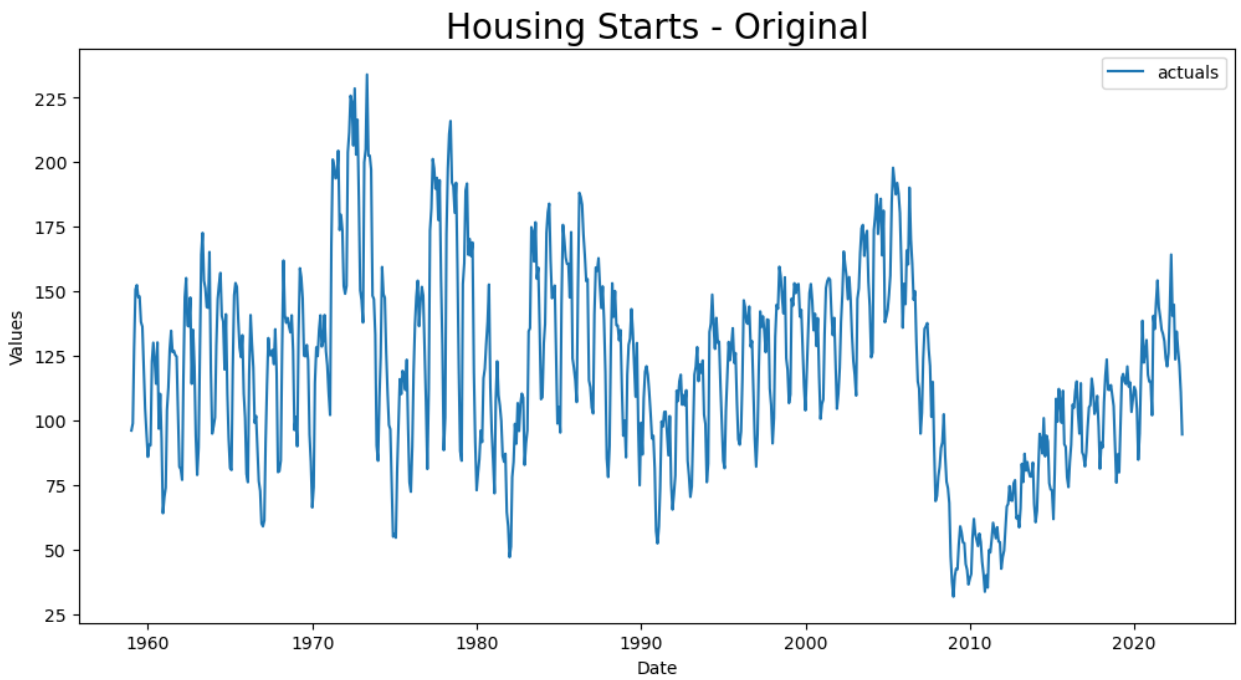
[1]:
```
import pandas as pd
import numpy as np
import pandas_datareader as pdr
import seaborn as sns
import matplotlib
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt
from scalecast.Forecaster import Forecaster
```

Download data from FRED (https://fred.stlouisfed.org/series/HOUSTNSA). This data is interesting due to its strong seasonality and irregular cycles. It measures monthly housing starts in the USA since 1959. Since exponential smoothing generally doesn't need as much data to work effectively, we start the series in 2010, where it has a clear trend and seasonality.

[2]:
```
df = pdr.get_data_fred('HOUSTNSA',start='1959-01-01',end='2022-12-31')
f = Forecaster(
    y=df['HOUSTNSA'],
    current_dates=df.index,
    future_dates = 24,
    test_length = 24,
    cis = True,
)
f
```

[2]:
```
Forecaster(
    DateStartActuals=1959-01-01T00:00:00.000000000
    DateEndActuals=2022-12-01T00:00:00.000000000
    Freq=MS
    N_actuals=768
    ForecastLength=24
    Xvars=[]
    TestLength=24
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[3]:
```
f.plot()
plt.title('Housing Starts - Original',size=20)
plt.show()
```



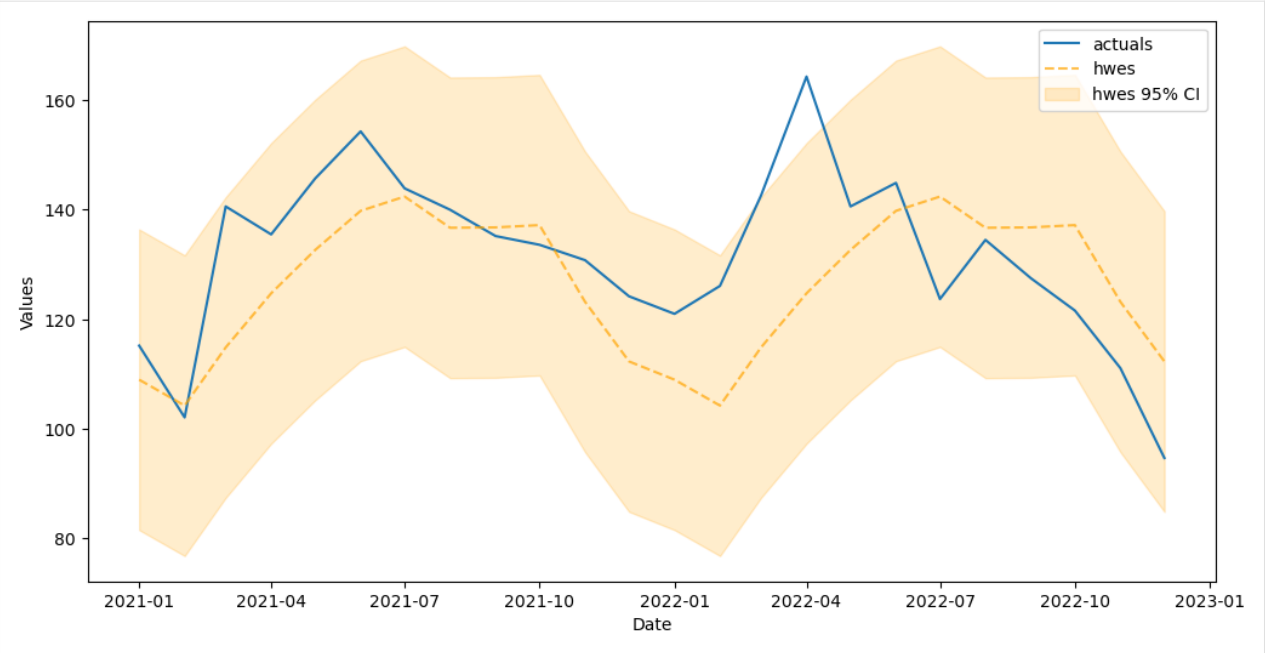
10.1 Forecast

[4]:
```
hwes_grid = {
    'trend':['add','mul',None],
    'seasonal':['add','mul',None],
    'use_boxcox':[True,False],
}
```

[5]:
```
f.set_estimator('hwes')
f.ingest_grid(hwes_grid)
f.cross_validate(k=10) # 10-fold cv to find optimal hyperparams
f.auto_forecast()
f.best_params
```

[5]:
```
{'trend': None, 'seasonal': 'add', 'use_boxcox': False}
```

[6]:
```
f.plot_test_set(ci=True,include_train=False);
```



10.2 Model Summary

[7]:
```
f.regr.summary()
```

[7]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                       ExponentialSmoothing Model Results
==============================================================================
Dep. Variable:                  HOUSTNSA   No. Observations:                  768
Model:             ExponentialSmoothing   SSE                          91014.035
Optimized:                         True   AIC                           3695.184
Trend:                             None   BIC                           3760.197
Seasonal:                      Additive   AICC                          3695.908
Seasonal Periods:                    12   Date:                 Mon, 10 Apr 2023
Box-Cox:                          False   Time:                         20:10:11
Box-Cox Coeff.:                    None
==============================================================================
                         coeff                 code              optimized
------------------------------------------------------------------------------
smoothing_level             0.6667527                alpha                 True
smoothing_seasonal          0.1106022                gamma                 True
initial_level               137.88929                  l.0                 True
initial_seasons.0          -42.836737                  s.0                 True
initial_seasons.1          -36.976933                  s.1                 True
initial_seasons.2          -3.3594145                  s.2                 True
initial_seasons.3           22.918534                  s.3                 True
initial_seasons.4           30.817362                  s.4                 True
initial_seasons.5           26.588908                  s.5                 True
initial_seasons.6           20.272506                  s.6                 True
initial_seasons.7           21.220218                  s.7                 True
```

(continues on next page)

(continued from previous page)

```
initial_seasons.8              8.3880755                    s.8                  True
initial_seasons.9              11.692122                    s.9                  True
initial_seasons.10            -12.917838                   s.10                  True
initial_seasons.11            -36.760941                   s.11                  True
--------------------------------------------------------------------------------
"""
```

10.3 Save Summary Stats

[8]:
```
f.save_summary_stats()
f.export_summary_stats('hwes')
```

[8]:

	coeff	code	optimized
smoothing_level	0.666753	alpha	True
smoothing_seasonal	0.110602	gamma	True
initial_level	137.889290	l.0	True
initial_seasons.0	-42.836737	s.0	True
initial_seasons.1	-36.976933	s.1	True
initial_seasons.2	-3.359415	s.2	True
initial_seasons.3	22.918534	s.3	True
initial_seasons.4	30.817362	s.4	True
initial_seasons.5	26.588908	s.5	True
initial_seasons.6	20.272506	s.6	True
initial_seasons.7	21.220218	s.7	True
initial_seasons.8	8.388075	s.8	True
initial_seasons.9	11.692122	s.9	True
initial_seasons.10	-12.917838	s.10	True
initial_seasons.11	-36.760941	s.11	True

[ ]:

CHAPTER

ELEVEN

LINKEDIN SILVERKITE

Silverkite is a model from the greykite package, developed by LinkedIn. It is considered an automated modeling procedure, like Facebook Prophet. It has its own database of holidays it forecasts with and attempts to tune its parameters automatically. The model itself is a type of linear regression with changepoints and regularization.

In this notebook, the following concepts are covered:

1. Loading Forecaster objects with external regressors
2. Forecasting with default silverkite model
3. Modifying changepoints in silverkite model
4. Tuning silverkite model
5. Forecasting on differenced data
6. Model summaries

- install greykite: `pip install greykite`
- read more about grekite: https://engineering.linkedin.com/blog/2021/greykite–a-flexible–intuitive–and-fast-forecasting-library
- see the combo notebook for an overview of this dataset with EDA

[1]:
```
import pandas as pd
import pandas_datareader as pdr
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from dateutil.relativedelta import relativedelta
from scalecast.Forecaster import Forecaster
from scalecast.util import find_optimal_transformation
from scalecast.Pipeline import Pipeline
```

[2]:
```
df = pdr.get_data_fred(['HOUSTNSA','JHDUSRGDPBR'],start='1900-01-01',end='2021-12-31')
df['JHDUSRGDPBR'] = df['JHDUSRGDPBR'].fillna(method='ffill').fillna(0)
f = Forecaster(y=df['HOUSTNSA'],current_dates=df.index)
f
```

[2]:
```
Forecaster(
    DateStartActuals=1959-01-01T00:00:00.000000000
    DateEndActuals=2021-12-01T00:00:00.000000000
    Freq=MS
    N_actuals=756
    ForecastLength=0
    Xvars=[]
```

(continues on next page)

(continued from previous page)

```
    TestLength=0
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

11.1 Prepare Forecast

11.1.1 Prepare the Recession Indicator

- use as an exogenous regressor

[3]:
```
# prepare recession indicator for future
# assume no future recessions in next two years
recessions = df.reset_index()[['DATE','JHDUSRGDPBR']]
fut_recessions = pd.DataFrame({
    'DATE':pd.date_range(
        start=recessions['DATE'].max(),
        periods=25,
        freq='MS'
    ).values[1:],
    'JHDUSRGDPBR':[0]*24}
)
recessions = pd.concat([recessions,fut_recessions])
recessions.tail()
```

[3]:
```
         DATE  JHDUSRGDPBR
19 2023-08-01          0.0
20 2023-09-01          0.0
21 2023-10-01          0.0
22 2023-11-01          0.0
23 2023-12-01          0.0
```

11.1.2 Load Object with Parameters and Regressors

- Forecast length: 24 periods (two years)
- Test length: 24 periods
- External recession indicator
- Conformal confidence intervals

[4]:
```
f.generate_future_dates(24)
f.set_test_length(24)
f.ingest_Xvars_df(recessions,date_col="DATE") # let the model consider recesssions
f.add_ar_terms(range(24,37)) # we can use direct autoregressive forecasting with␣
↪silverkite
```

(continues on next page)

(continued from previous page)

```
f.eval_cis()
f
```

[4]:
```
Forecaster(
    DateStartActuals=1959-01-01T00:00:00.000000000
    DateEndActuals=2021-12-01T00:00:00.000000000
    Freq=MS
    N_actuals=756
    ForecastLength=24
    Xvars=['JHDUSRGDPBR', 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30', 'AR31',
↪ 'AR32', 'AR33', 'AR34', 'AR35', 'AR36']
    TestLength=24
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

11.1.3 Find the optimal set of transformations

[5]:
```
transformer, reverter = find_optimal_transformation(
    f,
    estimator='silverkite',
    Xvars = 'all',
    verbose=True,
)
```

```
Using silverkite model to find the best transformation set on 1 test sets, each 24 in␣
↪length.
Last transformer tried:
[]
Score (rmse): 25.001200396713674
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (rmse): 49.79434396750028
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (rmse): 29.65330614730279
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (rmse): 51.360030782675594
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 12, 'model': 'add'})]
Score (rmse): 27.629940902996385
--------------------------------------------------
Last transformer tried:
```

(continues on next page)

(continued from previous page)

```
[('Transform', <function find_optimal_transformation.<locals>.boxcox_tr at␣
↪0x7fc8f5ba7670>, {'lmbda': -0.5})]
Score (rmse): 34.23912395743547
--------------------------------------------------
Last transformer tried:
[('Transform', <function find_optimal_transformation.<locals>.boxcox_tr at␣
↪0x7fc8f5ba7670>, {'lmbda': 0})]
Score (rmse): 30.86898988573646
--------------------------------------------------
Last transformer tried:
[('Transform', <function find_optimal_transformation.<locals>.boxcox_tr at␣
↪0x7fc8f5ba7670>, {'lmbda': 0.5})]
Score (rmse): 27.770186050360685
--------------------------------------------------
Last transformer tried:
[('DiffTransform', 1)]
Score (rmse): 27.461269567182537
--------------------------------------------------
Last transformer tried:
[('DiffTransform', 12)]
Score (rmse): 24.20622290670972
--------------------------------------------------
Last transformer tried:
[('DiffTransform', 12), ('ScaleTransform',)]
Score (rmse): 24.20622290670972
--------------------------------------------------
Last transformer tried:
[('DiffTransform', 12), ('MinMaxTransform',)]
Score (rmse): 24.20622290670973
--------------------------------------------------
Last transformer tried:
[('DiffTransform', 12), ('RobustScaleTransform',)]
Score (rmse): 24.20622290670972
--------------------------------------------------
Final Selection:
[('DiffTransform', 12)]
```

One seasonal difference chosen.

11.2 Apply the silverkite model

[6]:
```
silverkite_grid = {
    'changepoints':[2,None],
    'Xvars':[None,'all']
}
```

[7]:
```
def forecaster(f):
    f.set_estimator('silverkite')
    f.ingest_grid(silverkite_grid)
    f.cross_validate(
```

(continues on next page)

(continued from previous page)

```
        k=3,
        verbose=True,
        test_length=24,
    )
    f.auto_forecast()
    matplotlib.use("nbAgg")
    %matplotlib inline
```

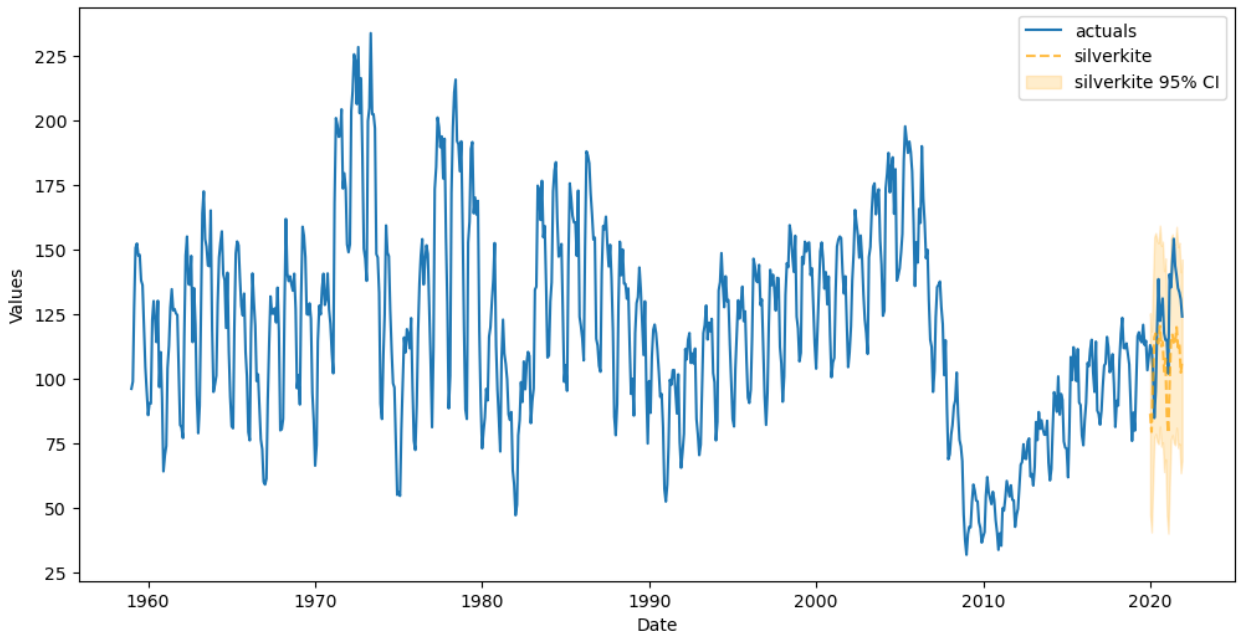
[8]:
```
pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter),
    ]
)
```

[9]:
```
f = pipeline.fit_predict(f)
```

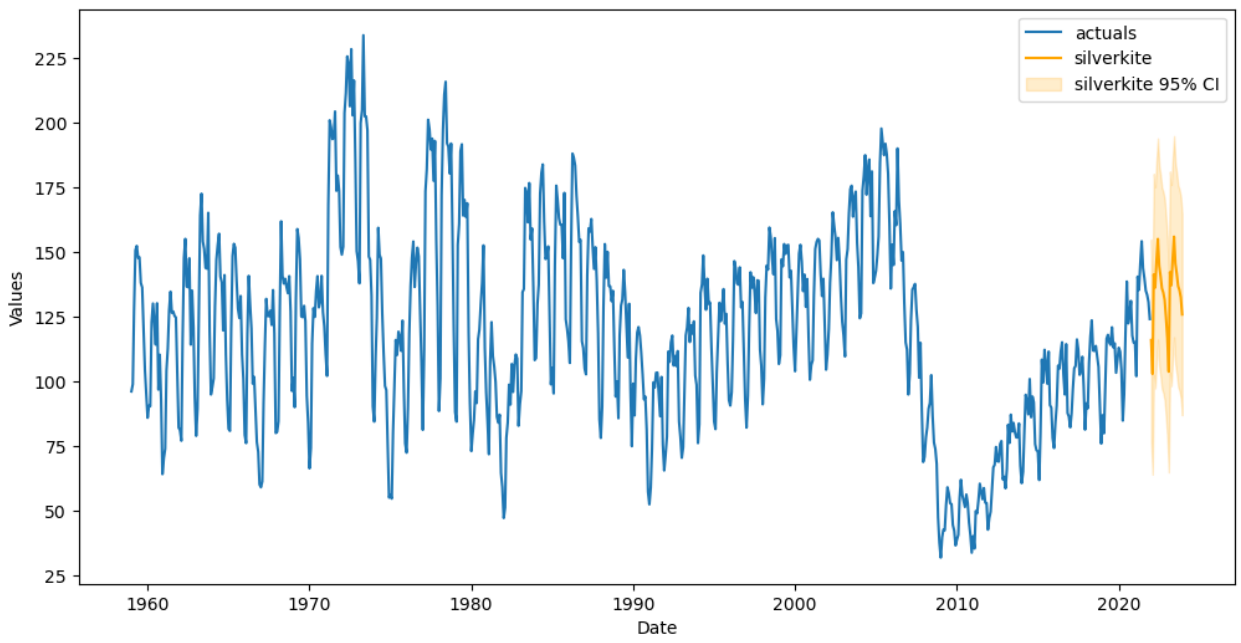
```
Num hyperparams to try for the silverkite model: 4.
Fold 0: Train size: 696 (1960-01-01 00:00:00 - 2017-12-01 00:00:00). Test Size: 24 (2018-
↪01-01 00:00:00 - 2019-12-01 00:00:00).
Fold 1: Train size: 672 (1960-01-01 00:00:00 - 2015-12-01 00:00:00). Test Size: 24 (2016-
↪01-01 00:00:00 - 2017-12-01 00:00:00).
Fold 2: Train size: 648 (1960-01-01 00:00:00 - 2013-12-01 00:00:00). Test Size: 24 (2014-
↪01-01 00:00:00 - 2015-12-01 00:00:00).
Chosen paramaters: {'changepoints': 2, 'Xvars': None}.
```

Now we can see the results plotted.

[10]:
```
f.plot_test_set(ci=True)
plt.show()
```



[11]:
```
f.plot(ci=True)
plt.show()
```



Scalecast model summary:

[12]:
```
f.export('model_summaries')
```

[12]:
```
  ModelNickname   Estimator Xvars          HyperParams  Observations  \
0    silverkite  silverkite    []  {'changepoints': 2}           744

   DynamicallyTested  TestSetLength  CILevel ValidationMetric  \
0               True             24     0.95             rmse

   ValidationMetricValue  ...  weights  best_model  InSampleRMSE  \
0              11.124548  ...      NaN        True     35.682039

   InSampleMAPE  InSampleMAE  InSampleR2  TestSetRMSE  TestSetMAPE  \
0      0.292353    27.853908    0.124307    24.206223     0.177581

   TestSetMAE  TestSetR2
0   21.884643  -1.097076

[1 rows x 21 columns]
```

Greykite model summary:

[13]:
```
f.save_summary_stats()
f.export_summary_stats('silverkite')
```

[13]:
```
                                                 Estimate  Std. Err  \
Pred_col
Intercept                                       -0.627656  1.621083
C(Q('events_Chinese New Year'), levels=['', 'ev...  0.000442  0.000428
```

(continues on next page)

(continued from previous page)

```
C(Q('events_Chinese New Year_minus_1'), levels=...  0.000344  0.000542
C(Q('events_Chinese New Year_minus_2'), levels=... -0.000170  0.000442
C(Q('events_Chinese New Year_plus_1'), levels=[...  0.000464  0.000415
C(Q('events_Chinese New Year_plus_2'), levels=[... -0.000005  0.000319
C(Q('events_Christmas Day'), levels=['', 'event...  0.000000  0.000000
C(Q('events_Christmas Day_minus_1'), levels=['''...  0.000000  0.000000
C(Q('events_Christmas Day_minus_2'), levels=['''...  0.000000  0.000000
C(Q('events_Christmas Day_plus_1'), levels=['',...  0.000000  0.000000
C(Q('events_Christmas Day_plus_2'), levels=['',...  0.000000  0.000000
C(Q('events_Good Friday'), levels=['', 'event']...  0.000652  0.000523
C(Q('events_Good Friday_minus_1'), levels=['', ...  0.000681  0.000538
C(Q('events_Good Friday_minus_2'), levels=['', ...  0.000129  0.000125
C(Q('events_Good Friday_plus_1'), levels=['', '...  0.000012  0.000141
C(Q('events_Good Friday_plus_2'), levels=['', '...  0.000115  0.000117
C(Q('events_Independence Day'), levels=['', 'ev...  0.000000  0.000000
C(Q('events_Independence Day_minus_1'), levels=...  0.000000  0.000000
C(Q('events_Independence Day_minus_2'), levels=...  0.000000  0.000000
C(Q('events_Independence Day_plus_1'), levels=[...  0.000000  0.000000
C(Q('events_Independence Day_plus_2'), levels=[...  0.000000  0.000000
C(Q('events_Labor Day'), levels=['', 'event'])[... -0.000107  0.002290
C(Q('events_Labor Day_minus_1'), levels=['', 'e...  0.000182  0.000642
C(Q('events_Labor Day_minus_2'), levels=['', 'e... -0.001285  0.000824
C(Q('events_Labor Day_plus_1'), levels=['', 'ev...  0.000000  0.000000
C(Q('events_Labor Day_plus_2'), levels=['', 'ev...  0.000000  0.000000
C(Q('events_Memorial Day'), levels=['', 'event'...  0.000000  0.000000
C(Q('events_Memorial Day_minus_1'), levels=['',...  0.000000  0.000000
C(Q('events_Memorial Day_minus_2'), levels=['',...  0.000000  0.000000
C(Q('events_Memorial Day_plus_1'), levels=['', ...  0.001146  0.000853
C(Q('events_Memorial Day_plus_2'), levels=['', ...  0.001383  0.000981
C(Q('events_New Years Day'), levels=['', 'event...  0.000124  0.001557
C(Q('events_New Years Day_minus_1'), levels=['''...  0.000000  0.000000
C(Q('events_New Years Day_minus_2'), levels=['''...  0.000000  0.000000
C(Q('events_New Years Day_plus_1'), levels=['',...  0.000000  0.000000
C(Q('events_New Years Day_plus_2'), levels=['',...  0.000000  0.000000
C(Q('events_Other'), levels=['', 'event'])[T.ev...  0.001696  0.003285
C(Q('events_Other_minus_1'), levels=['', 'event...  0.003931  0.003047
C(Q('events_Other_minus_2'), levels=['', 'event...  0.001698  0.002451
C(Q('events_Other_plus_1'), levels=['', 'event'...  0.001474  0.002845
C(Q('events_Other_plus_2'), levels=['', 'event'...  0.000897  0.002365
C(Q('events_Thanksgiving'), levels=['', 'event'...  0.000000  0.000000
C(Q('events_Thanksgiving_minus_1'), levels=['',...  0.000000  0.000000
C(Q('events_Thanksgiving_minus_2'), levels=['',...  0.000000  0.000000
C(Q('events_Thanksgiving_plus_1'), levels=['', ...  0.000000  0.000000
C(Q('events_Thanksgiving_plus_2'), levels=['', ...  0.000000  0.000000
C(Q('events_Veterans Day'), levels=['', 'event'...  0.000000  0.000000
C(Q('events_Veterans Day_minus_1'), levels=['',...  0.000000  0.000000
C(Q('events_Veterans Day_minus_2'), levels=['',...  0.000000  0.000000
C(Q('events_Veterans Day_plus_1'), levels=['', ...  0.000000  0.000000
C(Q('events_Veterans Day_plus_2'), levels=['', ...  0.000000  0.000000
ct1                                                 0.023843  0.033266

                                                    Pr(>)_boot sig. code  \
```

(continues on next page)

(continued from previous page)

```
Pred_col
Intercept                                                0.702
C(Q('events_Chinese New Year'), levels=['', 'ev...       0.300
C(Q('events_Chinese New Year_minus_1'), levels=...       0.534
C(Q('events_Chinese New Year_minus_2'), levels=...       0.698
C(Q('events_Chinese New Year_plus_1'), levels=[...       0.268
C(Q('events_Chinese New Year_plus_2'), levels=[...       0.988
C(Q('events_Christmas Day'), levels=['', 'event...       1.000
C(Q('events_Christmas Day_minus_1'), levels=['']...      1.000
C(Q('events_Christmas Day_minus_2'), levels=['']...      1.000
C(Q('events_Christmas Day_plus_1'), levels=['',...       1.000
C(Q('events_Christmas Day_plus_2'), levels=['',...       1.000
C(Q('events_Good Friday'), levels=['', 'event']...       0.200
C(Q('events_Good Friday_minus_1'), levels=['', ...       0.248
C(Q('events_Good Friday_minus_2'), levels=['', ...       0.162
C(Q('events_Good Friday_plus_1'), levels=['', '...       0.956
C(Q('events_Good Friday_plus_2'), levels=['', '...       0.536
C(Q('events_Independence Day'), levels=['', 'ev...       1.000
C(Q('events_Independence Day_minus_1'), levels=...       1.000
C(Q('events_Independence Day_minus_2'), levels=...       1.000
C(Q('events_Independence Day_plus_1'), levels=[...       1.000
C(Q('events_Independence Day_plus_2'), levels=[...       1.000
C(Q('events_Labor Day'), levels=['', 'event'])[...       0.966
C(Q('events_Labor Day_minus_1'), levels=['', 'e...       0.768
C(Q('events_Labor Day_minus_2'), levels=['', 'e...       0.108
C(Q('events_Labor Day_plus_1'), levels=['', 'ev...       1.000
C(Q('events_Labor Day_plus_2'), levels=['', 'ev...       1.000
C(Q('events_Memorial Day'), levels=['', 'event'...       1.000
C(Q('events_Memorial Day_minus_1'), levels=['',...       1.000
C(Q('events_Memorial Day_minus_2'), levels=['',...       1.000
C(Q('events_Memorial Day_plus_1'), levels=['', ...       0.178
C(Q('events_Memorial Day_plus_2'), levels=['', ...       0.152
C(Q('events_New Years Day'), levels=['', 'event...       0.934
C(Q('events_New Years Day_minus_1'), levels=['']...      1.000
C(Q('events_New Years Day_minus_2'), levels=['']...      1.000
C(Q('events_New Years Day_plus_1'), levels=['',...       1.000
C(Q('events_New Years Day_plus_2'), levels=['',...       1.000
C(Q('events_Other'), levels=['', 'event'])[T.ev...       0.616
C(Q('events_Other_minus_1'), levels=['', 'event...       0.190
C(Q('events_Other_minus_2'), levels=['', 'event...       0.486
C(Q('events_Other_plus_1'), levels=['', 'event'...       0.594
C(Q('events_Other_plus_2'), levels=['', 'event'...       0.682
C(Q('events_Thanksgiving'), levels=['', 'event'...       1.000
C(Q('events_Thanksgiving_minus_1'), levels=['',...       1.000
C(Q('events_Thanksgiving_minus_2'), levels=['',...       1.000
C(Q('events_Thanksgiving_plus_1'), levels=['', ...       1.000
C(Q('events_Thanksgiving_plus_2'), levels=['', ...       1.000
C(Q('events_Veterans Day'), levels=['', 'event'...       1.000
C(Q('events_Veterans Day_minus_1'), levels=['',...       1.000
C(Q('events_Veterans Day_minus_2'), levels=['',...       1.000
C(Q('events_Veterans Day_plus_1'), levels=['', ...       1.000
C(Q('events_Veterans Day_plus_2'), levels=['', ...       1.000
```

(continues on next page)

(continued from previous page)

```
ct1                                                    0.470

                                                                                                ␣
↪      95%CI
Pred_col
Intercept                                                    [-3.8531285086971647, 2.
↪356499683299088]
C(Q('events_Chinese New Year'), levels=['', 'ev...   [-0.0002607304049092221, 0.
↪0013420810713082817]
C(Q('events_Chinese New Year_minus_1'), levels=...    [-0.0006032882997487039, 0.
↪0015184635187620 43]
C(Q('events_Chinese New Year_minus_2'), levels=...   [-0.0009151696081354368, 0.
↪0007430782634949848]
C(Q('events_Chinese New Year_plus_1'), levels=[...   [-0.0001755691414453123, 0.
↪0013656991823609355]
C(Q('events_Chinese New Year_plus_2'), levels=[...   [-0.0005955917978176905, 0.
↪0006458427330378518]
C(Q('events_Christmas Day'), levels=['', 'event...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Christmas Day_minus_1'), levels=['''...                                             ␣
↪ [0.0, 0.0]
C(Q('events_Christmas Day_minus_2'), levels=['''...                                             ␣
↪ [0.0, 0.0]
C(Q('events_Christmas Day_plus_1'), levels=['',...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Christmas Day_plus_2'), levels=['',...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Good Friday'), levels=['', 'event']...  [-0.0001197341292146880 2, 0.
↪0017832190805516328]
C(Q('events_Good Friday_minus_1'), levels=['', ...   [-6.180472787720763e-06, 0.
↪0019063632891964072]
C(Q('events_Good Friday_minus_2'), levels=['', ...                     [0.0, 0.
↪0004179882658409 5656]
C(Q('events_Good Friday_plus_1'), levels=['', '...  [-0.0002533924169128 0985, 0.
↪0003210267888415958]
C(Q('events_Good Friday_plus_2'), levels=['', '...                     [0.0, 0.
↪0003772430928390 2117]
C(Q('events_Independence Day'), levels=['', 'ev...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Independence Day_minus_1'), levels=...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Independence Day_minus_2'), levels=...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Independence Day_plus_1'), levels=[...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Independence Day_plus_2'), levels=[...                                              ␣
↪ [0.0, 0.0]
C(Q('events_Labor Day'), levels=['', 'event'])[...     [-0.004791044889046614, 0.
↪004335130028251806]
C(Q('events_Labor Day_minus_1'), levels=['', 'e...    [-0.001172023060659738, 0.
↪0013385029209143616]
C(Q('events_Labor Day_minus_2'), levels=['', 'e...  [-0.0031060576549147042, 0.
```

(continues on next page)

(continued from previous page)

```
↪00012062622545997192]
C(Q('events_Labor Day_plus_1'), levels=['', 'ev...                                ␣
↪ [0.0, 0.0]
C(Q('events_Labor Day_plus_2'), levels=['', 'ev...                                ␣
↪ [0.0, 0.0]
C(Q('events_Memorial Day'), levels=['', 'event'...                                ␣
↪ [0.0, 0.0]
C(Q('events_Memorial Day_minus_1'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_Memorial Day_minus_2'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_Memorial Day_plus_1'), levels=['', ...   [-0.00023374375090142146, 0.
↪00295211730207247]
C(Q('events_Memorial Day_plus_2'), levels=['', ...   [-0.0003991890982334486, 0.
↪0033048549227503946]
C(Q('events_New Years Day'), levels=['', 'event...     [-0.002761111121212117, 0.
↪00332850157173629]
C(Q('events_New Years Day_minus_1'), levels=['''...                               ␣
↪ [0.0, 0.0]
C(Q('events_New Years Day_minus_2'), levels=['''...                               ␣
↪ [0.0, 0.0]
C(Q('events_New Years Day_plus_1'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_New Years Day_plus_2'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_Other'), levels=['', 'event'])[T.ev...    [-0.0047755062438510675, 0.
↪00780044259533481 5]
C(Q('events_Other_minus_1'), levels=['', 'event...    [-0.0019763326484872395, 0.
↪009885947626548024]
C(Q('events_Other_minus_2'), levels=['', 'event...     [-0.003617400314516922, 0.
↪00624240211827150 5]
C(Q('events_Other_plus_1'), levels=['', 'event'...     [-0.004060351507498453, 0.
↪00693851940237547 7]
C(Q('events_Other_plus_2'), levels=['', 'event'...    [-0.0039291870020548856, 0.
↪005265469190849324]
C(Q('events_Thanksgiving'), levels=['', 'event'...                                ␣
↪ [0.0, 0.0]
C(Q('events_Thanksgiving_minus_1'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_Thanksgiving_minus_2'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_Thanksgiving_plus_1'), levels=['', ...                                ␣
↪ [0.0, 0.0]
C(Q('events_Thanksgiving_plus_2'), levels=['', ...                                ␣
↪ [0.0, 0.0]
C(Q('events_Veterans Day'), levels=['', 'event'...                                ␣
↪ [0.0, 0.0]
C(Q('events_Veterans Day_minus_1'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_Veterans Day_minus_2'), levels=['',...                                ␣
↪ [0.0, 0.0]
C(Q('events_Veterans Day_plus_1'), levels=['', ...                                ␣
```

(continues on next page)

(continued from previous page)

```
↪ [0.0, 0.0]
C(Q('events_Veterans Day_plus_2'), levels=['', ...                                ␣
↪ [0.0, 0.0]
ct1                                                [-0.037338129675015985, 0.
↪09046991435719089]
```

```
[ ]:
```

CHAPTER

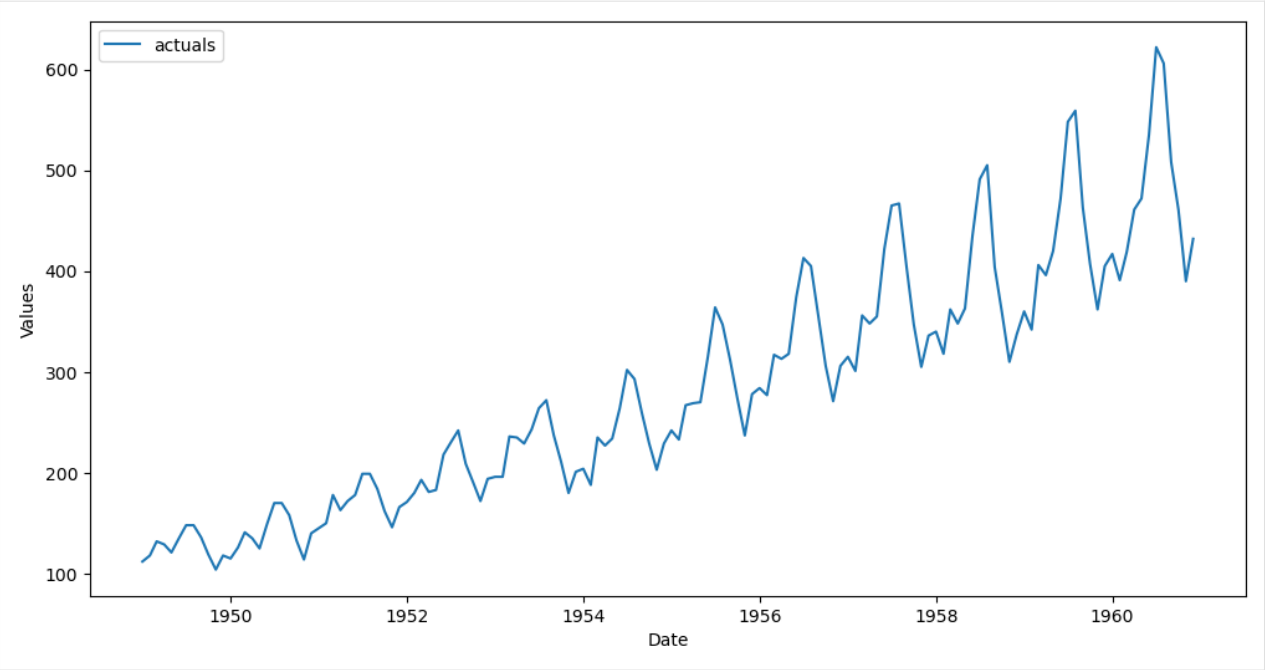
TWELVE

LSTM

12.1 Problem 1 - Univariate Forecasting

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scalecast.Forecaster import Forecaster
from scalecast.Pipeline import Transformer, Reverter, Pipeline
from scalecast.util import (
    find_optimal_transformation,
    gen_rnn_grid,
    backtest_for_resid_matrix,
    get_backtest_resid_matrix,
    overwrite_forecast_intervals,
    infer_apply_Xvar_selection,
)
from scalecast import GridGenerator
from tensorflow.keras.callbacks import EarlyStopping
import pandas_datareader as pdr
```

[2]:
```
data = pd.read_csv('AirPassengers.csv',parse_dates=['Month'])

f = Forecaster(
    y=data['#Passengers'],
    current_dates=data['Month'],
    future_dates = 24,
)

f.plot()
plt.show()
```



[3]:
```
def forecaster(f):
    f.set_estimator('rnn')
    f.manual_forecast(
        lags = 18,
        layers_struct = [
            ('LSTM',{'units':36,'activation':'tanh'}),
        ],
        epochs=200,
        call_me = 'lstm',
    )

transformer = Transformer(
    transformers = [
        ('DetrendTransform',{'poly_order':2}),
        'DeseasonTransform',
    ],
)

reverter = Reverter(
    reverters = [
        'DeseasonRevert',
        'DetrendRevert',
    ],
    base_transformer = transformer,
)

pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter),
```

(continues on next page)

(continued from previous page)

```
    ]
)

f = pipeline.fit_predict(f)
```

```
Epoch 1/200
4/4 [==============================] - 1s 6ms/step - loss: 0.3688
Epoch 2/200
4/4 [==============================] - 0s 6ms/step - loss: 0.3499
Epoch 3/200
4/4 [==============================] - 0s 6ms/step - loss: 0.3314
Epoch 4/200
4/4 [==============================] - 0s 5ms/step - loss: 0.3114
Epoch 5/200
4/4 [==============================] - 0s 6ms/step - loss: 0.2872
Epoch 6/200
4/4 [==============================] - 0s 5ms/step - loss: 0.2577
Epoch 7/200
4/4 [==============================] - 0s 6ms/step - loss: 0.2261
Epoch 8/200
4/4 [==============================] - 0s 6ms/step - loss: 0.2044
Epoch 9/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1807
Epoch 10/200
4/4 [==============================] - 0s 5ms/step - loss: 0.1608
Epoch 11/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1520
Epoch 12/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1460
Epoch 13/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1423
Epoch 14/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1425
Epoch 15/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1395
Epoch 16/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1379
Epoch 17/200
4/4 [==============================] - 0s 5ms/step - loss: 0.1375
Epoch 18/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1373
Epoch 19/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1365
Epoch 20/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1362
Epoch 21/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1359
Epoch 22/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1356
Epoch 23/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1351
Epoch 24/200
```

(continues on next page)

(continued from previous page)

```
4/4 [==============================] - 0s 5ms/step - loss: 0.1348
Epoch 25/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1343
Epoch 26/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1340
Epoch 27/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1339
Epoch 28/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1336
Epoch 29/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1334
Epoch 30/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1330
Epoch 31/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1332
Epoch 32/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1336
Epoch 33/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1328
Epoch 34/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1324
Epoch 35/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1322
Epoch 36/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1320
Epoch 37/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1318
Epoch 38/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1313
Epoch 39/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1310
Epoch 40/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1313
Epoch 41/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1307
Epoch 42/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1305
Epoch 43/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1302
Epoch 44/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1300
Epoch 45/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1300
Epoch 46/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1300
Epoch 47/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1298
Epoch 48/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1294
Epoch 49/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1292
Epoch 50/200
```

(continues on next page)

(continued from previous page)

```
4/4 [==============================] - 0s 8ms/step - loss: 0.1292
Epoch 51/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1288
Epoch 52/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1288
Epoch 53/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1286
Epoch 54/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1281
Epoch 55/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1282
Epoch 56/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1276
Epoch 57/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1271
Epoch 58/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1271
Epoch 59/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1268
Epoch 60/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1263
Epoch 61/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1257
Epoch 62/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1259
Epoch 63/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1253
Epoch 64/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1258
Epoch 65/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1260
Epoch 66/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1245
Epoch 67/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1241
Epoch 68/200
4/4 [==============================] - 0s 9ms/step - loss: 0.1250
Epoch 69/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1254
Epoch 70/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1240
Epoch 71/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1236
Epoch 72/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1242
Epoch 73/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1225
Epoch 74/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1221
Epoch 75/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1225
Epoch 76/200
```

(continues on next page)

(continued from previous page)

```
4/4 [==============================] - 0s 7ms/step - loss: 0.1221
Epoch 77/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1217
Epoch 78/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1216
Epoch 79/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1221
Epoch 80/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1207
Epoch 81/200
4/4 [==============================] - 0s 9ms/step - loss: 0.1213
Epoch 82/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1204
Epoch 83/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1201
Epoch 84/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1216
Epoch 85/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1197
Epoch 86/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1202
Epoch 87/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1205
Epoch 88/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1190
Epoch 89/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1191
Epoch 90/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1194
Epoch 91/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1197
Epoch 92/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1200
Epoch 93/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1192
Epoch 94/200
4/4 [==============================] - 0s 9ms/step - loss: 0.1188
Epoch 95/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1185
Epoch 96/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1177
Epoch 97/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1177
Epoch 98/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1172
Epoch 99/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1176
Epoch 100/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1168
Epoch 101/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1168
Epoch 102/200
```

(continues on next page)

(continued from previous page)

```
4/4 [==============================] - 0s 6ms/step - loss: 0.1166
Epoch 103/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1170
Epoch 104/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1176
Epoch 105/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1198
Epoch 106/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1195
Epoch 107/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1180
Epoch 108/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1170
Epoch 109/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1172
Epoch 110/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1154
Epoch 111/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1152
Epoch 112/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1150
Epoch 113/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1148
Epoch 114/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1146
Epoch 115/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1142
Epoch 116/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1137
Epoch 117/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1137
Epoch 118/200
4/4 [==============================] - 0s 6ms/step - loss: 0.1133
Epoch 119/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1130
Epoch 120/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1127
Epoch 121/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1129
Epoch 122/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1134
Epoch 123/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1141
Epoch 124/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1134
Epoch 125/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1120
Epoch 126/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1139
Epoch 127/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1128
Epoch 128/200
```

(continues on next page)

(continued from previous page)

```
4/4 [==============================] - 0s 7ms/step - loss: 0.1133
Epoch 129/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1116
Epoch 130/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1119
Epoch 131/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1130
Epoch 132/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1113
Epoch 133/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1106
Epoch 134/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1106
Epoch 135/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1099
Epoch 136/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1095
Epoch 137/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1101
Epoch 138/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1098
Epoch 139/200
4/4 [==============================] - 0s 9ms/step - loss: 0.1094
Epoch 140/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1092
Epoch 141/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1093
Epoch 142/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1102
Epoch 143/200
4/4 [==============================] - 0s 7ms/step - loss: 0.1091
Epoch 144/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1092
Epoch 145/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1083
Epoch 146/200
4/4 [==============================] - 0s 8ms/step - loss: 0.1084
Epoch 147/200
4/4 [==============================] - 0s 9ms/step - loss: 0.1076
Epoch 148/200
4/4 [==============================] - 0s 10ms/step - loss: 0.1088
Epoch 149/200
4/4 [==============================] - 0s 9ms/step - loss: 0.1078
Epoch 150/200
4/4 [==============================] - 0s 9ms/step - loss: 0.1072
Epoch 151/200
4/4 [==============================] - 0s 10ms/step - loss: 0.1071
Epoch 152/200
4/4 [==============================] - 0s 10ms/step - loss: 0.1067
Epoch 153/200
4/4 [==============================] - 0s 11ms/step - loss: 0.1073
Epoch 154/200
```

(continues on next page)

(continued from previous page)

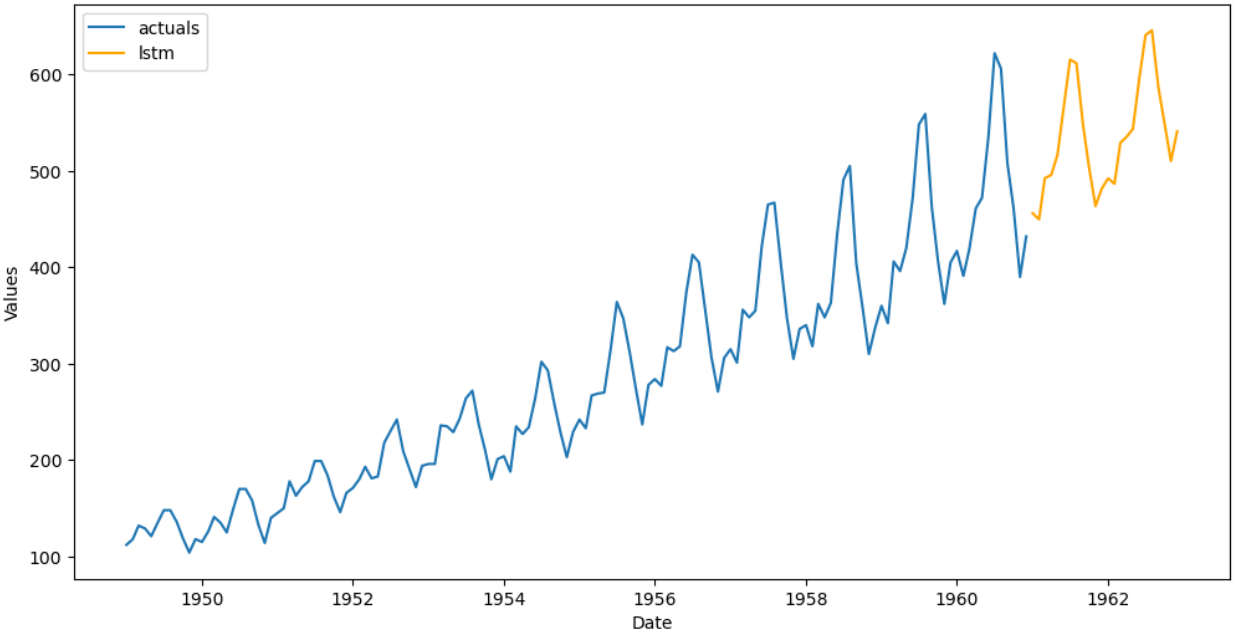
```
4/4 [==============================] - 0s 12ms/step - loss: 0.1066
Epoch 155/200
4/4 [==============================] - 0s 11ms/step - loss: 0.1065
Epoch 156/200
4/4 [==============================] - 0s 11ms/step - loss: 0.1057
Epoch 157/200
4/4 [==============================] - 0s 11ms/step - loss: 0.1060
Epoch 158/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1054
Epoch 159/200
4/4 [==============================] - 0s 11ms/step - loss: 0.1059
Epoch 160/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1052
Epoch 161/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1049
Epoch 162/200
4/4 [==============================] - 0s 11ms/step - loss: 0.1050
Epoch 163/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1040
Epoch 164/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1041
Epoch 165/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1041
Epoch 166/200
4/4 [==============================] - 0s 13ms/step - loss: 0.1038
Epoch 167/200
4/4 [==============================] - 0s 14ms/step - loss: 0.1042
Epoch 168/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1051
Epoch 169/200
4/4 [==============================] - 0s 13ms/step - loss: 0.1043
Epoch 170/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1034
Epoch 171/200
4/4 [==============================] - 0s 14ms/step - loss: 0.1035
Epoch 172/200
4/4 [==============================] - 0s 12ms/step - loss: 0.1034
Epoch 173/200
4/4 [==============================] - 0s 14ms/step - loss: 0.1029
Epoch 174/200
4/4 [==============================] - 0s 11ms/step - loss: 0.1044
Epoch 175/200
4/4 [==============================] - 0s 13ms/step - loss: 0.1037
Epoch 176/200
4/4 [==============================] - 0s 14ms/step - loss: 0.1032
Epoch 177/200
4/4 [==============================] - 0s 16ms/step - loss: 0.1033
Epoch 178/200
4/4 [==============================] - 0s 18ms/step - loss: 0.1029
Epoch 179/200
4/4 [==============================] - 0s 17ms/step - loss: 0.1024
Epoch 180/200
```

(continues on next page)

(continued from previous page)

```
4/4 [==============================] - 0s 16ms/step - loss: 0.1026
Epoch 181/200
4/4 [==============================] - 0s 16ms/step - loss: 0.1024
Epoch 182/200
4/4 [==============================] - 0s 16ms/step - loss: 0.1025
Epoch 183/200
4/4 [==============================] - 0s 15ms/step - loss: 0.1032
Epoch 184/200
4/4 [==============================] - 0s 14ms/step - loss: 0.1028
Epoch 185/200
4/4 [==============================] - 0s 16ms/step - loss: 0.1024
Epoch 186/200
4/4 [==============================] - 0s 16ms/step - loss: 0.1024
Epoch 187/200
4/4 [==============================] - 0s 18ms/step - loss: 0.1020
Epoch 188/200
4/4 [==============================] - 0s 17ms/step - loss: 0.1032
Epoch 189/200
4/4 [==============================] - 0s 17ms/step - loss: 0.1027
Epoch 190/200
4/4 [==============================] - 0s 17ms/step - loss: 0.1015
Epoch 191/200
4/4 [==============================] - 0s 25ms/step - loss: 0.1027
Epoch 192/200
4/4 [==============================] - 0s 20ms/step - loss: 0.1025
Epoch 193/200
4/4 [==============================] - 0s 20ms/step - loss: 0.1014
Epoch 194/200
4/4 [==============================] - 0s 19ms/step - loss: 0.1017
Epoch 195/200
4/4 [==============================] - 0s 15ms/step - loss: 0.1019
Epoch 196/200
4/4 [==============================] - 0s 17ms/step - loss: 0.1016
Epoch 197/200
4/4 [==============================] - 0s 20ms/step - loss: 0.1008
Epoch 198/200
4/4 [==============================] - 0s 20ms/step - loss: 0.1009
Epoch 199/200
4/4 [==============================] - 0s 20ms/step - loss: 0.1017
Epoch 200/200
4/4 [==============================] - 0s 19ms/step - loss: 0.1017
1/1 [==============================] - 1s 1s/step
4/4 [==============================] - 0s 9ms/step
```

[4]:
```
f.plot()
plt.savefig('LSTM Univariate.png')
plt.show()
```

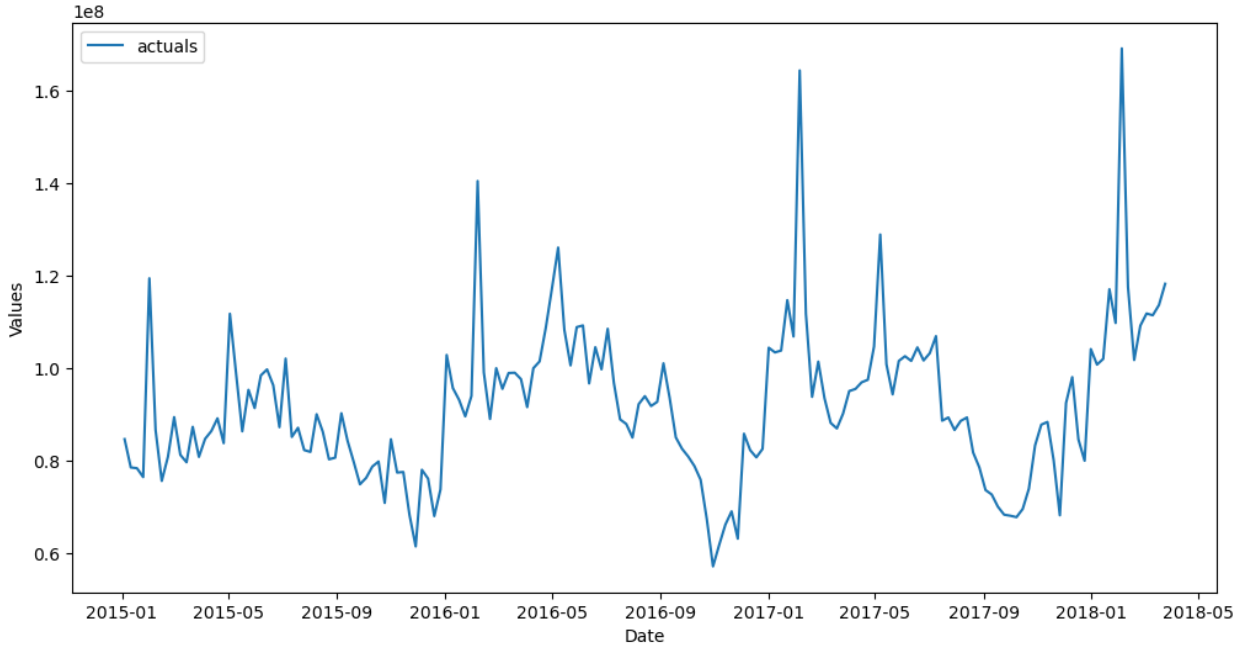


12.2 Problem 2 - Multivariate Forecasting

[5]:
```
data = pd.read_csv('avocado.csv')
```

[6]:
```
# demand
vol = data.groupby('Date')['Total Volume'].sum()
# price
price = data.groupby('Date')['AveragePrice'].sum()

fvol = Forecaster(
    y = vol,
    current_dates = vol.index,
    test_length = 13,
    validation_length = 13,
    future_dates = 13,
    metrics = ['rmse','r2'],
)
fprice = Forecaster(
    y = price,
    current_dates = price.index,
    future_dates = 13,
)
```

[7]:
```
fvol.plot()
plt.show()
```



[8]:
```
transformer, reverter = find_optimal_transformation(
    fvol,
    set_aside_test_set=True, # prevents leakage so we can benchmark the resulting models␣
↪fairly
    return_train_only = True, # prevents leakage so we can benchmark the resulting␣
↪models fairly
    verbose=True,
    detrend_kwargs=[
        {'loess':True},
        {'poly_order':1},
        {'ln_trend':True},
    ],
    m = 52, # what makes one seasonal cycle?
    test_length = 4,
)
```

```
Using mlr model to find the best transformation set on 1 test sets, each 4 in length.
All transformation tries will be evaluated with 52 lags.
Last transformer tried:
[]
Score (rmse): 10788435.478499832
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (rmse): 17367936.06969349
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (rmse): 12269036.46992314
--------------------------------------------------
Last transformer tried:
```

(continues on next page)

(continued from previous page)

```
[('DetrendTransform', {'ln_trend': True})]
Score (rmse): 12093617.597284064
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'})]
Score (rmse): 9289548.042079216
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', BoxcoxTransform, {'lmbda
↪': -0.5})]
Score (rmse): 10446492.57435158
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', BoxcoxTransform, {'lmbda
↪': 0})]
Score (rmse): 10242650.677245248
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', BoxcoxTransform, {'lmbda
↪': 0.5})]
Score (rmse): 9836892.333276885
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 1)]
Score (rmse): 9702659.462897006
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 52)]
Score (rmse): 36977467.903368585
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('ScaleTransform',)]
Score (rmse): 9289548.04207921
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('MinMaxTransform',)]
Score (rmse): 9289548.042079207
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 52, 'model': 'add'}), ('RobustScaleTransform',)]
Score (rmse): 9289548.042079205
--------------------------------------------------
Final Selection:
[('DeseasonTransform', {'m': 52, 'model': 'add', 'train_only': True}), (
↪'RobustScaleTransform', {'train_only': True})]
```

[9]:
```
fprice = transformer.fit_transform(fprice)
fvol = transformer.fit_transform(fvol)
```

[10]:
```
rnn_grid = gen_rnn_grid(
    layer_tries = 10,
```

(continues on next page)

(continued from previous page)

```
    min_layer_size = 3,
    max_layer_size = 5,
    units_pool = [100],
    epochs = [100],
    dropout_pool = [0,0.05],
    validation_split=.2,
    callbacks=EarlyStopping(
      monitor='val_loss',
      patience=3,
    ),
    random_seed = 20,
) # creates a grid of hyperparameter values to tune the LSTM model
```

[11]:
```
def forecaster(fvol,fprice):
    # naive forecast for benchmarking
    fvol.set_estimator('naive')
    fvol.manual_forecast()
    # univariate lstm model
    fvol.add_ar_terms(13) # the model will use 13 series lags
    fvol.set_estimator('rnn')
    fvol.ingest_grid(rnn_grid)
    fvol.tune()
    fvol.auto_forecast(call_me='lstm_univariate')
    # multivariate lstm model
    fvol.add_series(fprice.y,called='price')
    fvol.add_lagged_terms('price',lags=13,drop=True)
    fvol.ingest_grid(rnn_grid)
    fvol.tune()
    fvol.auto_forecast(call_me='lstm_multivariate')
```

[12]:
```
forecaster(fvol=fvol,fprice=fprice)
```

```
1/1 [==============================] - 0s 442ms/step
1/1 [==============================] - 0s 208ms/step
1/1 [==============================] - 0s 361ms/step
WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_predict_function.
↪<locals>.predict_function at 0x000002415F29CF70> triggered tf.function retracing.␣
↪Tracing is expensive and the excessive number of tracings could be due to (1) creating␣
↪@tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)␣
↪passing Python objects instead of tensors. For (1), please define your @tf.function␣
↪outside of the loop. For (2), @tf.function has reduce_retracing=True option that can␣
↪avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/
↪function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/
↪function for  more details.
1/1 [==============================] - 0s 155ms/step
WARNING:tensorflow:6 out of the last 10 calls to <function Model.make_predict_function.
↪<locals>.predict_function at 0x0000024158D988B0> triggered tf.function retracing.␣
↪Tracing is expensive and the excessive number of tracings could be due to (1) creating␣
↪@tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)␣
↪passing Python objects instead of tensors. For (1), please define your @tf.function␣
↪outside of the loop. For (2), @tf.function has reduce_retracing=True option that can␣
↪avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/
```

(continues on next page)

(continued from previous page)

```
↪function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/
↪function for  more details.
1/1 [==============================] - 0s 424ms/step
1/1 [==============================] - 0s 471ms/step
1/1 [==============================] - 0s 313ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 293ms/step
1/1 [==============================] - 0s 288ms/step
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
Keras model archive saving:
File Name                                              Modified             Size
config.json                                      2023-09-20 09:13:34         2234
metadata.json                                    2023-09-20 09:13:34           64
variables.h5                                     2023-09-20 09:13:34       524080
Keras model archive loading:
File Name                                              Modified             Size
config.json                                      2023-09-20 09:13:34         2234
metadata.json                                    2023-09-20 09:13:34           64
variables.h5                                     2023-09-20 09:13:34       524080
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\dense
...vars
...0
```

(continues on next page)

(continued from previous page)

```
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 202ms/step
5/5 [==============================] - 0s 5ms/step
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
```

(continues on next page)

(continued from previous page)

```
...4
...5
...6
...7
...8
...9
...vars
Keras model archive saving:
File Name                                             Modified             Size
config.json                                     2023-09-20 09:13:41         2234
metadata.json                                   2023-09-20 09:13:41           64
variables.h5                                    2023-09-20 09:13:41       524080
Keras model archive loading:
File Name                                             Modified             Size
config.json                                     2023-09-20 09:13:40         2234
metadata.json                                   2023-09-20 09:13:40           64
variables.h5                                    2023-09-20 09:13:40       524080
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\dense
...vars
...0
...1
```

(continues on next page)

(continued from previous page)

```
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...optimizer
...vars
...0
...vars
Keras model archive saving:
File Name                                             Modified             Size
config.json                                     2023-09-20 09:13:41         2234
metadata.json                                   2023-09-20 09:13:41           64
variables.h5                                    2023-09-20 09:13:41       180720
Keras model archive loading:
File Name                                             Modified             Size
config.json                                     2023-09-20 09:13:40         2234
metadata.json                                   2023-09-20 09:13:40           64
variables.h5                                    2023-09-20 09:13:40       180720
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...optimizer
...vars
...0
...vars
1/1 [==============================] - 0s 286ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 1s 550ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 310ms/step
1/1 [==============================] - 0s 407ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 364ms/step
1/1 [==============================] - 0s 336ms/step
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\dense
...vars
...0
...1
```

(continues on next page)

(continued from previous page)

```
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
Keras model archive saving:
File Name                                             Modified             Size
config.json                                      2023-09-20 09:14:20         2234
metadata.json                                    2023-09-20 09:14:20           64
variables.h5                                     2023-09-20 09:14:21       524080
Keras model archive loading:
File Name                                             Modified             Size
config.json                                      2023-09-20 09:14:20         2234
metadata.json                                    2023-09-20 09:14:20           64
variables.h5                                     2023-09-20 09:14:20       524080
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
```

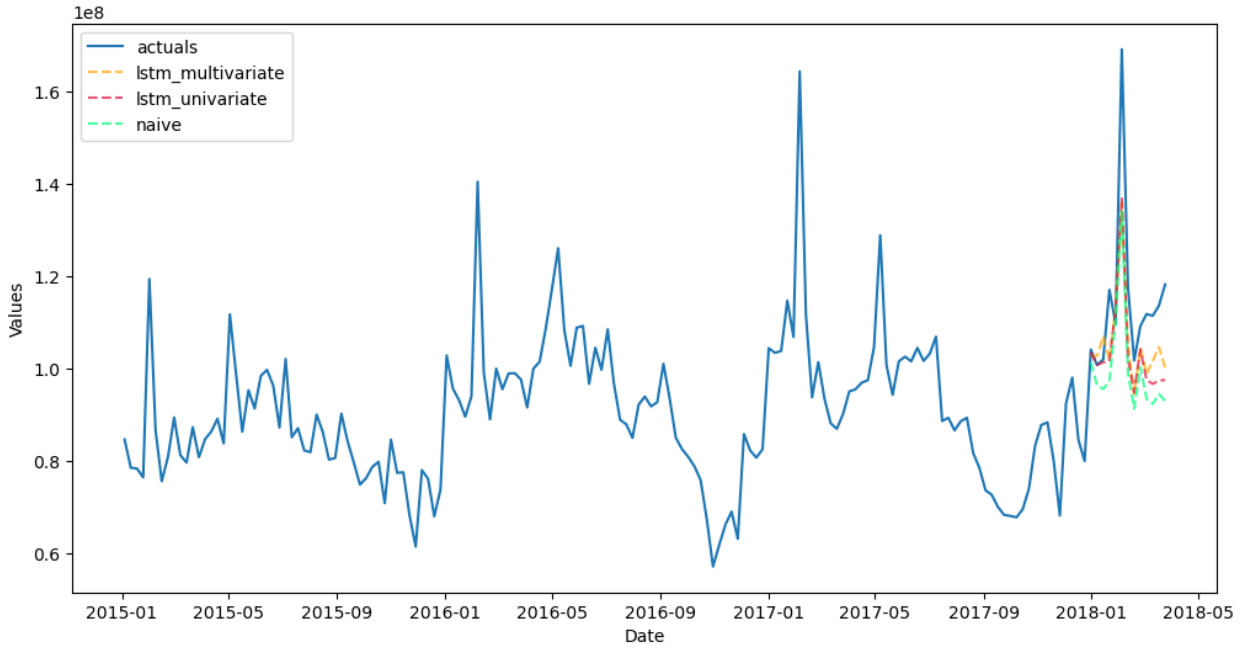
(continues on next page)

(continued from previous page)

```
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
1/1 [==============================] - 0s 470ms/step
1/1 [==============================] - 0s 303ms/step
5/5 [==============================] - 0s 8ms/step
```

[13]:
```
fvol = reverter.fit_transform(fvol)
```

[14]:
```
fvol.plot_test_set(order_by='TestSetRMSE')
plt.savefig('LSTM MV test results.png')
plt.show()
```



[15]:
```
pd.options.display.float_format = '{:,.4f}'.format
summ = fvol.export('model_summaries',determine_best_by='TestSetRMSE')
summ[['ModelNickname','TestSetRMSE','TestSetR2']]
```

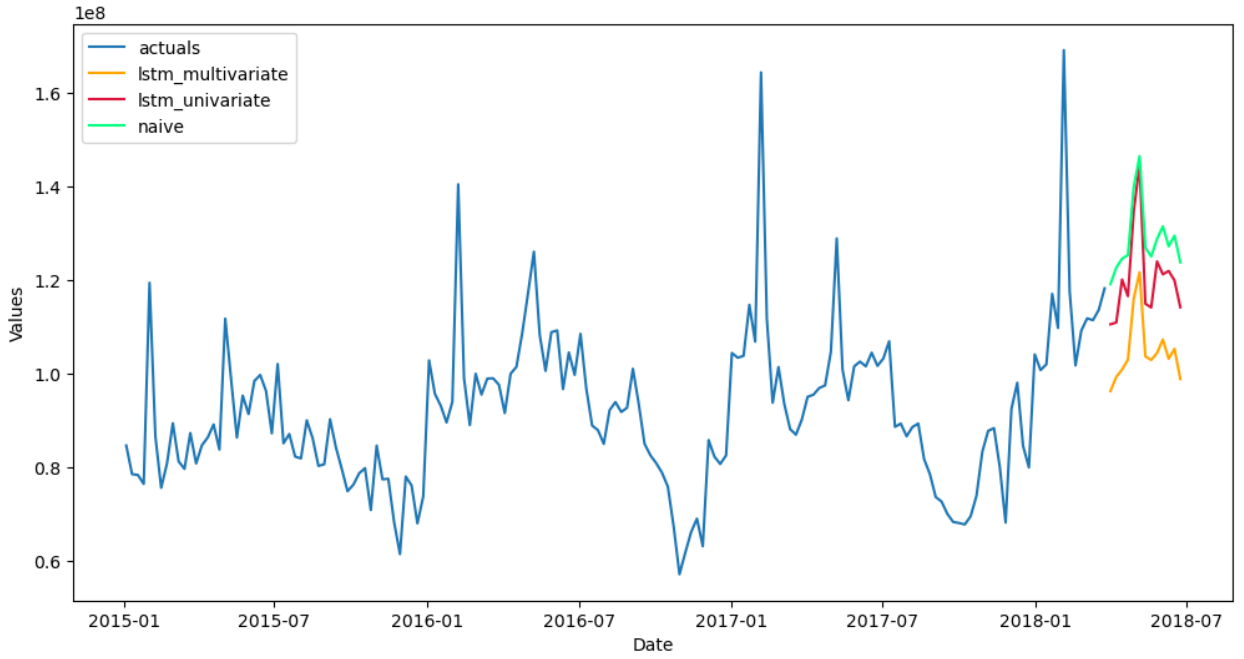
[15]:

	ModelNickname	TestSetRMSE	TestSetR2
0	lstm_multivariate	13,241,317.9599	0.3837
1	lstm_univariate	14,475,008.6416	0.2635
2	naive	17,403,456.6059	-0.0646

[16]:
```
summ[['ModelNickname','HyperParams']].style.set_properties(height = 5)
```

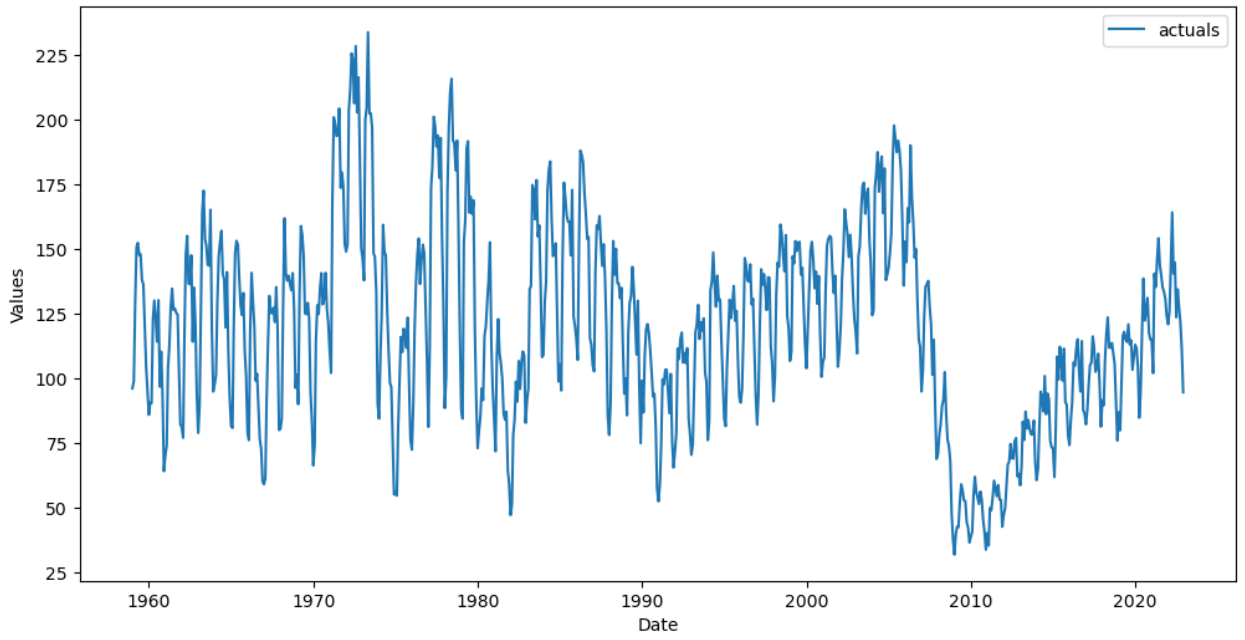
[16]:
```
<pandas.io.formats.style.Styler at 0x2416db9f640>
```

[17]:
```
fvol.plot(order_by='TestSetRMSE')
plt.show()
```



12.3 Problem 3 - Probabilistic Forecasting

[2]:
```
df = pdr.get_data_fred(
    'HOUSTNSA',
    start = '1959-01-01',
    end = '2022-12-31',
)

f = Forecaster(
    y = df['HOUSTNSA'],
    current_dates = df.index,
    future_dates = 24, # 2-year forecast horizon
    test_length = .1, # 10% test length
    cis = True,
    cilevel = .9, # 90% intervals
)

f.plot()
plt.show()
```



[3]:
```
transformer, reverter = find_optimal_transformation(
    f,
    estimator = 'lstm',
    epochs = 10,
    set_aside_test_set=True, # prevents leakage so we can benchmark the resulting models␣
↪fairly
    return_train_only = True, # prevents leakage so we can benchmark the resulting␣
↪models fairly
    verbose=True,
    m = 52, # what makes one seasonal cycle?
    test_length = 24,
    num_test_sets = 3,
    space_between_sets = 12,
    detrend_kwargs=[
        {'loess':True},
        {'poly_order':1},
        {'ln_trend':True},
    ],
)
```

```
Using lstm model to find the best transformation set on 3 test sets, each 24 in length.
Epoch 1/10
17/17 [==============================] - 2s 3ms/step - loss: 0.4265
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4051
Epoch 3/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3817
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3553
Epoch 5/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3252
```

(continues on next page)

(continued from previous page)

```
Epoch 6/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2918
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2571
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2244
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1972
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1771
1/1 [==============================] - 0s 485ms/step
Epoch 1/10
21/21 [==============================] - 3s 3ms/step - loss: 0.4330
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.4043
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3748
Epoch 4/10
21/21 [==============================] - 0s 4ms/step - loss: 0.3421
Epoch 5/10
21/21 [==============================] - 0s 4ms/step - loss: 0.3050
Epoch 6/10
21/21 [==============================] - 0s 5ms/step - loss: 0.2644
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2241
Epoch 8/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1913
Epoch 9/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1686
Epoch 10/10
21/21 [==============================] - 0s 3ms/step - loss: 0.1552
1/1 [==============================] - 1s 506ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 2ms/step - loss: 0.4244
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4041
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3819
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3571
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3290
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2980
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2655
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2344
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2068
Epoch 10/10
```

(continues on next page)

(continued from previous page)

```
16/16 [==============================] - 0s 4ms/step - loss: 0.1852
1/1 [==============================] - 0s 495ms/step
Epoch 1/10
20/20 [==============================] - 3s 4ms/step - loss: 0.4423
Epoch 2/10
20/20 [==============================] - 0s 4ms/step - loss: 0.4167
Epoch 3/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3912
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3639
Epoch 5/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3334
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2996
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2618
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2233
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1895
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1654
1/1 [==============================] - 1s 508ms/step
20/20 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 3ms/step - loss: 0.4192
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3979
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3747
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3487
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3197
Epoch 6/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2880
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2558
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2256
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2003
Epoch 10/10
16/16 [==============================] - 0s 3ms/step - loss: 0.1809
1/1 [==============================] - 0s 387ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4383
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4104
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3798
Epoch 4/10
```

(continues on next page)

(continued from previous page)

```
20/20 [==============================] - 0s 3ms/step - loss: 0.3454
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3066
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2656
Epoch 7/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2263
Epoch 8/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1930
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1697
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1561
1/1 [==============================] - 0s 403ms/step
20/20 [==============================] - 0s 1ms/step
Last transformer tried:
[]
Score (rmse): 22.1277468319124
--------------------------------------------------
Epoch 1/10
17/17 [==============================] - 2s 3ms/step - loss: 0.4547
Epoch 2/10
17/17 [==============================] - 0s 4ms/step - loss: 0.4325
Epoch 3/10
17/17 [==============================] - 0s 4ms/step - loss: 0.4079
Epoch 4/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3797
Epoch 5/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3473
Epoch 6/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3112
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2736
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2379
Epoch 9/10
17/17 [==============================] - 0s 5ms/step - loss: 0.2081
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1858
1/1 [==============================] - 0s 414ms/step
Epoch 1/10
21/21 [==============================] - 2s 3ms/step - loss: 0.4415
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.4120
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3786
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3401
Epoch 5/10
21/21 [==============================] - 0s 3ms/step - loss: 0.2974
Epoch 6/10
21/21 [==============================] - 0s 3ms/step - loss: 0.2528
```

(continues on next page)

(continued from previous page)

```
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2124
Epoch 8/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1820
Epoch 9/10
21/21 [==============================] - 0s 3ms/step - loss: 0.1634
Epoch 10/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1531
1/1 [==============================] - 1s 687ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 1s 2ms/step - loss: 0.4513
Epoch 2/10
16/16 [==============================] - 0s 2ms/step - loss: 0.4314
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4094
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3842
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3553
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3226
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2874
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2524
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2208
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.1952
1/1 [==============================] - 0s 417ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4442
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4171
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3871
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3526
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3131
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2709
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2303
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1968
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1739
Epoch 10/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1603
1/1 [==============================] - 0s 424ms/step
```

(continues on next page)

(continued from previous page)

```
20/20 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 1s 3ms/step - loss: 0.4491
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4290
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4077
Epoch 4/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3837
Epoch 5/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3562
Epoch 6/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3251
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2912
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2570
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2255
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.1994
1/1 [==============================] - 0s 421ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4478
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4203
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3918
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3610
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3272
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2906
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2522
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2163
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1863
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1665
1/1 [==============================] - 0s 405ms/step
20/20 [==============================] - 0s 2ms/step
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (rmse): 23.107732581389552
--------------------------------------------------
Epoch 1/10
17/17 [==============================] - 2s 3ms/step - loss: 0.4457
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4262
```

(continues on next page)

(continued from previous page)

```
Epoch 3/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4053
Epoch 4/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3813
Epoch 5/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3531
Epoch 6/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3205
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2847
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2489
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2166
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1919
1/1 [==============================] - 0s 409ms/step
Epoch 1/10
21/21 [==============================] - 1s 2ms/step - loss: 0.4143
Epoch 2/10
21/21 [==============================] - 0s 2ms/step - loss: 0.3870
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3567
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3228
Epoch 5/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2863
Epoch 6/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2502
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2177
Epoch 8/10
21/21 [==============================] - 0s 3ms/step - loss: 0.1920
Epoch 9/10
21/21 [==============================] - 0s 3ms/step - loss: 0.1749
Epoch 10/10
21/21 [==============================] - 0s 3ms/step - loss: 0.1662
1/1 [==============================] - 0s 338ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 1s 3ms/step - loss: 0.4417
Epoch 2/10
16/16 [==============================] - 0s 2ms/step - loss: 0.4218
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4010
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3777
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3513
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3216
Epoch 7/10
```

(continues on next page)

(continued from previous page)

```
16/16 [==============================] - 0s 3ms/step - loss: 0.2898
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2578
Epoch 9/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2279
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2026
1/1 [==============================] - 0s 445ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4191
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3921
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3652
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3374
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3080
Epoch 6/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2777
Epoch 7/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2479
Epoch 8/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2202
Epoch 9/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1968
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1789
1/1 [==============================] - 0s 350ms/step
20/20 [==============================] - 0s 1ms/step
Epoch 1/10
16/16 [==============================] - 1s 3ms/step - loss: 0.4330
Epoch 2/10
16/16 [==============================] - 0s 2ms/step - loss: 0.4124
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3905
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3662
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3389
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3088
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2774
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2466
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2191
Epoch 10/10
16/16 [==============================] - 0s 3ms/step - loss: 0.1961
1/1 [==============================] - 0s 338ms/step
Epoch 1/10
```

(continues on next page)

(continued from previous page)

```
20/20 [==============================] - 2s 3ms/step - loss: 0.4293
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4036
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3779
Epoch 4/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3501
Epoch 5/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3193
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2859
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2518
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2198
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1938
Epoch 10/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1750
1/1 [==============================] - 1s 628ms/step
20/20 [==============================] - 0s 2ms/step
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (rmse): 16.923530213289453
--------------------------------------------------
Epoch 1/10
17/17 [==============================] - 2s 4ms/step - loss: 0.4302
Epoch 2/10
17/17 [==============================] - 0s 4ms/step - loss: 0.4091
Epoch 3/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3866
Epoch 4/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3615
Epoch 5/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3325
Epoch 6/10
17/17 [==============================] - 0s 5ms/step - loss: 0.3002
Epoch 7/10
17/17 [==============================] - 0s 5ms/step - loss: 0.2659
Epoch 8/10
17/17 [==============================] - 0s 5ms/step - loss: 0.2329
Epoch 9/10
17/17 [==============================] - 0s 5ms/step - loss: 0.2040
Epoch 10/10
17/17 [==============================] - 0s 5ms/step - loss: 0.1821
1/1 [==============================] - 0s 478ms/step
Epoch 1/10
21/21 [==============================] - 2s 3ms/step - loss: 0.4169
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3858
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3518
```

(continues on next page)

(continued from previous page)

```
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3145
Epoch 5/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2749
Epoch 6/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2360
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2018
Epoch 8/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1766
Epoch 9/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1608
Epoch 10/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1527
1/1 [==============================] - 0s 415ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 3ms/step - loss: 0.4252
Epoch 2/10
16/16 [==============================] - 0s 2ms/step - loss: 0.4038
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3808
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3555
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3275
Epoch 6/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2970
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2659
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2360
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2099
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.1889
1/1 [==============================] - 0s 393ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4294
Epoch 2/10
20/20 [==============================] - 0s 4ms/step - loss: 0.4011
Epoch 3/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3713
Epoch 4/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3388
Epoch 5/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3035
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2660
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2304
Epoch 8/10
```

(continues on next page)

(continued from previous page)

```
20/20 [==============================] - 0s 4ms/step - loss: 0.1993
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1757
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1620
1/1 [==============================] - 0s 429ms/step
20/20 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 3ms/step - loss: 0.4228
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4024
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3806
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3567
Epoch 5/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3300
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3003
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2691
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2386
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2112
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.1891
1/1 [==============================] - 0s 387ms/step
Epoch 1/10
20/20 [==============================] - 2s 2ms/step - loss: 0.4390
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4129
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3859
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3563
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3238
Epoch 6/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2887
Epoch 7/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2530
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2185
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1898
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1685
1/1 [==============================] - 0s 384ms/step
20/20 [==============================] - 0s 2ms/step
Last transformer tried:
[('DetrendTransform', {'ln_trend': True})]
```

(continues on next page)

(continued from previous page)

```
Score (rmse): 21.321529285599414
--------------------------------------------------
Epoch 1/10
17/17 [==============================] - 1s 3ms/step - loss: 0.4634
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4418
Epoch 3/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4178
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3904
Epoch 5/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3589
Epoch 6/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3234
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2853
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2482
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2162
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1913
1/1 [==============================] - 0s 410ms/step
Epoch 1/10
21/21 [==============================] - 2s 3ms/step - loss: 0.4300
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.4009
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3692
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3337
Epoch 5/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2953
Epoch 6/10
21/21 [==============================] - 0s 3ms/step - loss: 0.2566
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2208
Epoch 8/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1928
Epoch 9/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1744
Epoch 10/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1642
1/1 [==============================] - 0s 436ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 3ms/step - loss: 0.4747
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4537
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4309
Epoch 4/10
```

(continues on next page)

(continued from previous page)

```
16/16 [==============================] - 0s 3ms/step - loss: 0.4049
Epoch 5/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3754
Epoch 6/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3425
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3067
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2703
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2366
Epoch 10/10
16/16 [==============================] - 0s 5ms/step - loss: 0.2086
1/1 [==============================] - 1s 566ms/step
Epoch 1/10
20/20 [==============================] - 2s 2ms/step - loss: 0.4486
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4211
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3916
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3589
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3233
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2853
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2481
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2154
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1896
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1736
1/1 [==============================] - 0s 365ms/step
20/20 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 2ms/step - loss: 0.4813
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4603
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4375
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4115
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3820
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3483
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3113
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2730
```

(continues on next page)

(continued from previous page)

```
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2375
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2079
1/1 [==============================] - 0s 473ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4680
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4410
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4114
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3776
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3399
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2994
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2593
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2243
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1969
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1783
1/1 [==============================] - 0s 386ms/step
20/20 [==============================] - 0s 2ms/step
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪)]
Score (rmse): 14.610087883063033
------------------------------------------------
Epoch 1/10
17/17 [==============================] - 2s 2ms/step - loss: 0.8790
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.8552
Epoch 3/10
17/17 [==============================] - 0s 3ms/step - loss: 0.8282
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: 0.7962
Epoch 5/10
17/17 [==============================] - 0s 3ms/step - loss: 0.7575
Epoch 6/10
17/17 [==============================] - 0s 4ms/step - loss: 0.7107
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.6548
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.5886
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.5119
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.4244
```

(continues on next page)

(continued from previous page)

```
1/1 [==============================] - 0s 387ms/step
Epoch 1/10
17/17 [==============================] - 1s 2ms/step - loss: nan
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: nan
Epoch 3/10
17/17 [==============================] - 0s 3ms/step - loss: nan
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: nan
Epoch 5/10
17/17 [==============================] - 0s 4ms/step - loss: nan
Epoch 6/10
17/17 [==============================] - 0s 4ms/step - loss: nan
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: nan
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: nan
Epoch 9/10
17/17 [==============================] - 0s 5ms/step - loss: nan
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: nan
1/1 [==============================] - 0s 439ms/step
Epoch 1/10
17/17 [==============================] - 2s 2ms/step - loss: 0.2927
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.2914
Epoch 3/10
17/17 [==============================] - 0s 3ms/step - loss: 0.2904
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: 0.2894
Epoch 5/10
17/17 [==============================] - 0s 3ms/step - loss: 0.2881
Epoch 6/10
17/17 [==============================] - 0s 3ms/step - loss: 0.2869
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2857
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2842
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2825
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2807
1/1 [==============================] - 0s 366ms/step
Epoch 1/10
17/17 [==============================] - 2s 3ms/step - loss: 0.4306
Epoch 2/10
17/17 [==============================] - 0s 4ms/step - loss: 0.4097
Epoch 3/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3865
Epoch 4/10
17/17 [==============================] - 0s 4ms/step - loss: 0.3601
Epoch 5/10
```

(continues on next page)

(continued from previous page)

```
17/17 [==============================] - 0s 4ms/step - loss: 0.3297
Epoch 6/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2947
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2555
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2148
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1765
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1473
WARNING:tensorflow:5 out of the last 25 calls to <function Model.make_predict_function.
↪<locals>.predict_function at 0x000001C6543F2550> triggered tf.function retracing.␣
↪Tracing is expensive and the excessive number of tracings could be due to (1) creating␣
↪@tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)␣
↪passing Python objects instead of tensors. For (1), please define your @tf.function␣
↪outside of the loop. For (2), @tf.function has reduce_retracing=True option that can␣
↪avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/
↪function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/
↪function for  more details.
1/1 [==============================] - 1s 538ms/step
Epoch 1/10
21/21 [==============================] - 2s 3ms/step - loss: 0.4202
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3913
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3587
Epoch 4/10
21/21 [==============================] - 0s 4ms/step - loss: 0.3206
Epoch 5/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2760
Epoch 6/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2266
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1787
Epoch 8/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1436
Epoch 9/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1254
Epoch 10/10
21/21 [==============================] - 0s 5ms/step - loss: 0.1189
WARNING:tensorflow:6 out of the last 26 calls to <function Model.make_predict_function.
↪<locals>.predict_function at 0x000001C658A3AAF0> triggered tf.function retracing.␣
↪Tracing is expensive and the excessive number of tracings could be due to (1) creating␣
↪@tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)␣
↪passing Python objects instead of tensors. For (1), please define your @tf.function␣
↪outside of the loop. For (2), @tf.function has reduce_retracing=True option that can␣
↪avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/
↪function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/
↪function for  more details.
1/1 [==============================] - 0s 479ms/step
21/21 [==============================] - 0s 2ms/step
```

(continues on next page)

(continued from previous page)

```
Epoch 1/10
16/16 [==============================] - 2s 3ms/step - loss: 0.4244
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4036
Epoch 3/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3812
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3561
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3272
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2941
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2574
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2192
Epoch 9/10
16/16 [==============================] - 0s 3ms/step - loss: 0.1831
Epoch 10/10
16/16 [==============================] - 0s 3ms/step - loss: 0.1544
1/1 [==============================] - 0s 409ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4247
Epoch 2/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3992
Epoch 3/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3715
Epoch 4/10
20/20 [==============================] - 0s 5ms/step - loss: 0.3399
Epoch 5/10
20/20 [==============================] - 0s 5ms/step - loss: 0.3033
Epoch 6/10
20/20 [==============================] - 0s 5ms/step - loss: 0.2621
Epoch 7/10
20/20 [==============================] - 0s 5ms/step - loss: 0.2186
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1782
Epoch 9/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1475
Epoch 10/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1294
1/1 [==============================] - 0s 483ms/step
20/20 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 3ms/step - loss: 0.4294
Epoch 2/10
16/16 [==============================] - 0s 4ms/step - loss: 0.4092
Epoch 3/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3866
Epoch 4/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3608
Epoch 5/10
```

(continues on next page)

(continued from previous page)

```
16/16 [==============================] - 0s 4ms/step - loss: 0.3309
Epoch 6/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2965
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2584
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2194
Epoch 9/10
16/16 [==============================] - 0s 5ms/step - loss: 0.1829
Epoch 10/10
16/16 [==============================] - 0s 5ms/step - loss: 0.1545
1/1 [==============================] - 0s 473ms/step
Epoch 1/10
20/20 [==============================] - 2s 2ms/step - loss: 0.4205
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3943
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3642
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3280
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2846
Epoch 6/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2359
Epoch 7/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1880
Epoch 8/10
20/20 [==============================] - 0s 2ms/step - loss: 0.1506
Epoch 9/10
20/20 [==============================] - 0s 2ms/step - loss: 0.1309
Epoch 10/10
20/20 [==============================] - 0s 2ms/step - loss: 0.1235
1/1 [==============================] - 0s 322ms/step
20/20 [==============================] - 0s 1ms/step
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 1)]
Score (rmse): 81.22989950780023
--------------------------------------------------
Epoch 1/10
15/15 [==============================] - 1s 2ms/step - loss: 0.4980
Epoch 2/10
15/15 [==============================] - 0s 2ms/step - loss: 0.4787
Epoch 3/10
15/15 [==============================] - 0s 3ms/step - loss: 0.4585
Epoch 4/10
15/15 [==============================] - 0s 2ms/step - loss: 0.4365
Epoch 5/10
15/15 [==============================] - 0s 2ms/step - loss: 0.4117
Epoch 6/10
15/15 [==============================] - 0s 3ms/step - loss: 0.3837
Epoch 7/10
```

(continues on next page)

(continued from previous page)

```
15/15 [==============================] - 0s 2ms/step - loss: 0.3525
Epoch 8/10
15/15 [==============================] - 0s 3ms/step - loss: 0.3188
Epoch 9/10
15/15 [==============================] - 0s 3ms/step - loss: 0.2845
Epoch 10/10
15/15 [==============================] - 0s 3ms/step - loss: 0.2511
1/1 [==============================] - 0s 322ms/step
Epoch 1/10
19/19 [==============================] - 1s 2ms/step - loss: 0.4822
Epoch 2/10
19/19 [==============================] - 0s 2ms/step - loss: 0.4549
Epoch 3/10
19/19 [==============================] - 0s 2ms/step - loss: 0.4261
Epoch 4/10
19/19 [==============================] - 0s 3ms/step - loss: 0.3941
Epoch 5/10
19/19 [==============================] - 0s 2ms/step - loss: 0.3577
Epoch 6/10
19/19 [==============================] - 0s 3ms/step - loss: 0.3172
Epoch 7/10
19/19 [==============================] - 0s 3ms/step - loss: 0.2753
Epoch 8/10
19/19 [==============================] - 0s 2ms/step - loss: 0.2348
Epoch 9/10
19/19 [==============================] - 0s 2ms/step - loss: 0.2016
Epoch 10/10
19/19 [==============================] - 0s 2ms/step - loss: 0.1775
1/1 [==============================] - 0s 300ms/step
19/19 [==============================] - 0s 1ms/step
Epoch 1/10
15/15 [==============================] - 2s 3ms/step - loss: 0.4967
Epoch 2/10
15/15 [==============================] - 0s 3ms/step - loss: 0.4761
Epoch 3/10
15/15 [==============================] - 0s 3ms/step - loss: 0.4546
Epoch 4/10
15/15 [==============================] - 0s 3ms/step - loss: 0.4310
Epoch 5/10
15/15 [==============================] - 0s 3ms/step - loss: 0.4044
Epoch 6/10
15/15 [==============================] - 0s 3ms/step - loss: 0.3750
Epoch 7/10
15/15 [==============================] - 0s 3ms/step - loss: 0.3430
Epoch 8/10
15/15 [==============================] - 0s 3ms/step - loss: 0.3096
Epoch 9/10
15/15 [==============================] - 0s 3ms/step - loss: 0.2761
Epoch 10/10
15/15 [==============================] - 0s 3ms/step - loss: 0.2452
1/1 [==============================] - 0s 431ms/step
Epoch 1/10
```

(continues on next page)

(continued from previous page)

```
19/19 [==============================] - 2s 2ms/step - loss: 0.4897
Epoch 2/10
19/19 [==============================] - 0s 2ms/step - loss: 0.4642
Epoch 3/10
19/19 [==============================] - 0s 2ms/step - loss: 0.4369
Epoch 4/10
19/19 [==============================] - 0s 2ms/step - loss: 0.4063
Epoch 5/10
19/19 [==============================] - 0s 2ms/step - loss: 0.3709
Epoch 6/10
19/19 [==============================] - 0s 3ms/step - loss: 0.3309
Epoch 7/10
19/19 [==============================] - 0s 3ms/step - loss: 0.2880
Epoch 8/10
19/19 [==============================] - 0s 3ms/step - loss: 0.2460
Epoch 9/10
19/19 [==============================] - 0s 3ms/step - loss: 0.2090
Epoch 10/10
19/19 [==============================] - 0s 3ms/step - loss: 0.1816
1/1 [==============================] - 0s 386ms/step
19/19 [==============================] - 0s 1ms/step
Epoch 1/10
14/14 [==============================] - 2s 2ms/step - loss: 0.4934
Epoch 2/10
14/14 [==============================] - 0s 2ms/step - loss: 0.4759
Epoch 3/10
14/14 [==============================] - 0s 2ms/step - loss: 0.4566
Epoch 4/10
14/14 [==============================] - 0s 3ms/step - loss: 0.4353
Epoch 5/10
14/14 [==============================] - 0s 3ms/step - loss: 0.4112
Epoch 6/10
14/14 [==============================] - 0s 3ms/step - loss: 0.3844
Epoch 7/10
14/14 [==============================] - 0s 3ms/step - loss: 0.3549
Epoch 8/10
14/14 [==============================] - 0s 4ms/step - loss: 0.3236
Epoch 9/10
14/14 [==============================] - 0s 4ms/step - loss: 0.2918
Epoch 10/10
14/14 [==============================] - 0s 4ms/step - loss: 0.2607
1/1 [==============================] - 0s 464ms/step
Epoch 1/10
18/18 [==============================] - 2s 2ms/step - loss: 0.4966
Epoch 2/10
18/18 [==============================] - 0s 3ms/step - loss: 0.4727
Epoch 3/10
18/18 [==============================] - 0s 3ms/step - loss: 0.4488
Epoch 4/10
18/18 [==============================] - 0s 3ms/step - loss: 0.4236
Epoch 5/10
18/18 [==============================] - 0s 3ms/step - loss: 0.3965
```

(continues on next page)

(continued from previous page)

```
Epoch 6/10
18/18 [==============================] - 0s 3ms/step - loss: 0.3666
Epoch 7/10
18/18 [==============================] - 0s 4ms/step - loss: 0.3342
Epoch 8/10
18/18 [==============================] - 0s 4ms/step - loss: 0.3004
Epoch 9/10
18/18 [==============================] - 0s 6ms/step - loss: 0.2665
Epoch 10/10
18/18 [==============================] - 0s 5ms/step - loss: 0.2346
1/1 [==============================] - 1s 559ms/step
18/18 [==============================] - 0s 2ms/step
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('DiffTransform', 52)]
Score (rmse): 72.21666118044968
--------------------------------------------------
Epoch 1/10
17/17 [==============================] - 2s 3ms/step - loss: 0.4649
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4440
Epoch 3/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4212
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3954
Epoch 5/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3659
Epoch 6/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3322
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2952
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2578
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2239
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1971
1/1 [==============================] - 1s 530ms/step
Epoch 1/10
21/21 [==============================] - 2s 2ms/step - loss: 0.4294
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.4007
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3691
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3335
Epoch 5/10
21/21 [==============================] - 0s 3ms/step - loss: 0.2945
Epoch 6/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2549
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2198
```

(continues on next page)

(continued from previous page)

```
Epoch 8/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1919
Epoch 9/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1736
Epoch 10/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1639
1/1 [==============================] - 0s 421ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 1s 2ms/step - loss: 0.4752
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4558
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4344
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4100
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3819
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3497
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3138
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2760
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2402
Epoch 10/10
16/16 [==============================] - 0s 5ms/step - loss: 0.2099
1/1 [==============================] - 1s 530ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4517
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4265
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3990
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3672
Epoch 5/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3308
Epoch 6/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2905
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2500
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2143
Epoch 9/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1876
Epoch 10/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1714
1/1 [==============================] - 0s 390ms/step
20/20 [==============================] - 0s 2ms/step
Epoch 1/10
```

(continues on next page)

(continued from previous page)

```
16/16 [==============================] - 2s 3ms/step - loss: 0.4811
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4598
Epoch 3/10
16/16 [==============================] - 0s 4ms/step - loss: 0.4369
Epoch 4/10
16/16 [==============================] - 0s 4ms/step - loss: 0.4110
Epoch 5/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3816
Epoch 6/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3484
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3117
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2739
Epoch 9/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2385
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2085
1/1 [==============================] - 0s 437ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4640
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4362
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4051
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3699
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3309
Epoch 6/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2894
Epoch 7/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2496
Epoch 8/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2149
Epoch 9/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1888
Epoch 10/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1723
1/1 [==============================] - 0s 364ms/step
20/20 [==============================] - 0s 2ms/step
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('ScaleTransform',)]
Score (rmse): 14.208245879919753
--------------------------------------------------
Epoch 1/10
17/17 [==============================] - 2s 2ms/step - loss: 0.4655
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4443
Epoch 3/10
```

(continues on next page)

(continued from previous page)

```
17/17 [==============================] - 0s 3ms/step - loss: 0.4211
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3944
Epoch 5/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3636
Epoch 6/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3284
Epoch 7/10
17/17 [==============================] - 0s 3ms/step - loss: 0.2903
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2529
Epoch 9/10
17/17 [==============================] - 0s 3ms/step - loss: 0.2194
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1938
1/1 [==============================] - 0s 479ms/step
Epoch 1/10
21/21 [==============================] - 2s 3ms/step - loss: 0.4372
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.4097
Epoch 3/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3802
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3472
Epoch 5/10
21/21 [==============================] - 0s 4ms/step - loss: 0.3102
Epoch 6/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2712
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2332
Epoch 8/10
21/21 [==============================] - 0s 3ms/step - loss: 0.2013
Epoch 9/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1791
Epoch 10/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1666
1/1 [==============================] - 0s 480ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 1s 2ms/step - loss: 0.4697
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4469
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4213
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3925
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3604
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3252
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2884
```

(continues on next page)

(continued from previous page)

```
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2526
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2214
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.1970
1/1 [==============================] - 1s 543ms/step
Epoch 1/10
20/20 [==============================] - 2s 2ms/step - loss: 0.4489
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4207
Epoch 3/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3894
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3545
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3164
Epoch 6/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2773
Epoch 7/10
20/20 [==============================] - 0s 3ms/step - loss: 0.2407
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2099
Epoch 9/10
20/20 [==============================] - 0s 3ms/step - loss: 0.1869
Epoch 10/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1723
1/1 [==============================] - 1s 737ms/step
20/20 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 2s 2ms/step - loss: 0.4819
Epoch 2/10
16/16 [==============================] - 0s 2ms/step - loss: 0.4620
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4413
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4183
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3924
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3630
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3296
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2930
Epoch 9/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2561
Epoch 10/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2224
1/1 [==============================] - 0s 422ms/step
Epoch 1/10
20/20 [==============================] - 2s 3ms/step - loss: 0.4605
```

(continues on next page)

(continued from previous page)

```
Epoch 2/10
20/20 [==============================] - 0s 3ms/step - loss: 0.4320
Epoch 3/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3999
Epoch 4/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3634
Epoch 5/10
20/20 [==============================] - 0s 3ms/step - loss: 0.3232
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2817
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2422
Epoch 8/10
20/20 [==============================] - 0s 5ms/step - loss: 0.2099
Epoch 9/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1870
Epoch 10/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1726
1/1 [==============================] - 1s 556ms/step
20/20 [==============================] - 0s 3ms/step
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
↪), ('MinMaxTransform',)]
Score (rmse): 14.445980804969805
--------------------------------------------------
Epoch 1/10
17/17 [==============================] - 2s 3ms/step - loss: 0.4660
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4441
Epoch 3/10
17/17 [==============================] - 0s 3ms/step - loss: 0.4213
Epoch 4/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3959
Epoch 5/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3669
Epoch 6/10
17/17 [==============================] - 0s 3ms/step - loss: 0.3336
Epoch 7/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2968
Epoch 8/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2593
Epoch 9/10
17/17 [==============================] - 0s 4ms/step - loss: 0.2255
Epoch 10/10
17/17 [==============================] - 0s 4ms/step - loss: 0.1983
1/1 [==============================] - 1s 594ms/step
Epoch 1/10
21/21 [==============================] - 2s 3ms/step - loss: 0.4268
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.3978
Epoch 3/10
21/21 [==============================] - 0s 4ms/step - loss: 0.3656
```

(continues on next page)

(continued from previous page)

```
Epoch 4/10
21/21 [==============================] - 0s 4ms/step - loss: 0.3287
Epoch 5/10
21/21 [==============================] - 0s 3ms/step - loss: 0.2884
Epoch 6/10
21/21 [==============================] - 0s 3ms/step - loss: 0.2479
Epoch 7/10
21/21 [==============================] - 0s 4ms/step - loss: 0.2122
Epoch 8/10
21/21 [==============================] - 0s 3ms/step - loss: 0.1856
Epoch 9/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1686
Epoch 10/10
21/21 [==============================] - 0s 4ms/step - loss: 0.1603
1/1 [==============================] - 1s 600ms/step
21/21 [==============================] - 0s 2ms/step
Epoch 1/10
16/16 [==============================] - 3s 2ms/step - loss: 0.4761
Epoch 2/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4564
Epoch 3/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4355
Epoch 4/10
16/16 [==============================] - 0s 3ms/step - loss: 0.4118
Epoch 5/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3844
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3529
Epoch 7/10
16/16 [==============================] - 0s 3ms/step - loss: 0.3176
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2803
Epoch 9/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2444
Epoch 10/10
16/16 [==============================] - 0s 3ms/step - loss: 0.2132
1/1 [==============================] - 0s 442ms/step
Epoch 1/10
20/20 [==============================] - 3s 6ms/step - loss: 0.4506
Epoch 2/10
20/20 [==============================] - 0s 6ms/step - loss: 0.4247
Epoch 3/10
20/20 [==============================] - 0s 5ms/step - loss: 0.3967
Epoch 4/10
20/20 [==============================] - 0s 5ms/step - loss: 0.3651
Epoch 5/10
20/20 [==============================] - 0s 5ms/step - loss: 0.3291
Epoch 6/10
20/20 [==============================] - 0s 6ms/step - loss: 0.2899
Epoch 7/10
20/20 [==============================] - 0s 5ms/step - loss: 0.2510
Epoch 8/10
```

(continues on next page)

(continued from previous page)

```
20/20 [==============================] - 0s 6ms/step - loss: 0.2171
Epoch 9/10
20/20 [==============================] - 0s 7ms/step - loss: 0.1909
Epoch 10/10
20/20 [==============================] - 0s 7ms/step - loss: 0.1748
1/1 [==============================] - 1s 1s/step
20/20 [==============================] - 0s 3ms/step
Epoch 1/10
16/16 [==============================] - 2s 4ms/step - loss: 0.4811
Epoch 2/10
16/16 [==============================] - 0s 4ms/step - loss: 0.4618
Epoch 3/10
16/16 [==============================] - 0s 5ms/step - loss: 0.4408
Epoch 4/10
16/16 [==============================] - 0s 6ms/step - loss: 0.4169
Epoch 5/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3891
Epoch 6/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3571
Epoch 7/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3209
Epoch 8/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2824
Epoch 9/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2453
Epoch 10/10
16/16 [==============================] - 0s 4ms/step - loss: 0.2130
1/1 [==============================] - 1s 528ms/step
Epoch 1/10
20/20 [==============================] - 2s 2ms/step - loss: 0.4618
Epoch 2/10
20/20 [==============================] - 0s 2ms/step - loss: 0.4342
Epoch 3/10
20/20 [==============================] - 0s 4ms/step - loss: 0.4030
Epoch 4/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3672
Epoch 5/10
20/20 [==============================] - 0s 4ms/step - loss: 0.3268
Epoch 6/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2841
Epoch 7/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2437
Epoch 8/10
20/20 [==============================] - 0s 4ms/step - loss: 0.2101
Epoch 9/10
20/20 [==============================] - 0s 4ms/step - loss: 0.1850
Epoch 10/10
20/20 [==============================] - 0s 5ms/step - loss: 0.1700
1/1 [==============================] - 0s 344ms/step
20/20 [==============================] - 0s 1ms/step
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 52, 'model': 'add'}
```

(continues on next page)

(continued from previous page)

```
↪), ('RobustScaleTransform',)]
Score (rmse): 15.184932604377607
--------------------------------------------------
Final Selection:
[('DetrendTransform', {'poly_order': 1, 'train_only': True}), ('DeseasonTransform', {'m':
↪ 52, 'model': 'add', 'train_only': True}), ('ScaleTransform', {'train_only': True})]
```

[4]:
```
rnn_grid = gen_rnn_grid(
    layer_tries = 100,
    min_layer_size = 1,
    max_layer_size = 5,
    units_pool = [100],
    epochs = [100],
    dropout_pool = [0,0.05],
    validation_split=.2,
    callbacks=EarlyStopping(
      monitor='val_loss',
      patience=3,
    ),
    random_seed = 20,
) # make a really big grid and limit it manually
```

[5]:
```
def forecaster(f,grid):
    f.auto_Xvar_select(
        try_trend=False,
        try_seasonalities=False,
        max_ar=100
    )
    f.set_estimator('rnn')
    f.ingest_grid(grid)
    f.limit_grid_size(10) # randomly reduce the big grid to 10
    f.cross_validate(k=3,test_length=24) # three-fold cross-validation
    f.auto_forecast()
```

[6]:
```
pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter),
    ]
)

f = pipeline.fit_predict(f,grid=rnn_grid)
```

```
1/1 [==============================] - 0s 317ms/step
1/1 [==============================] - 0s 427ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 309ms/step
1/1 [==============================] - 0s 420ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 313ms/step
1/1 [==============================] - 0s 307ms/step
```

(continues on next page)

(continued from previous page)

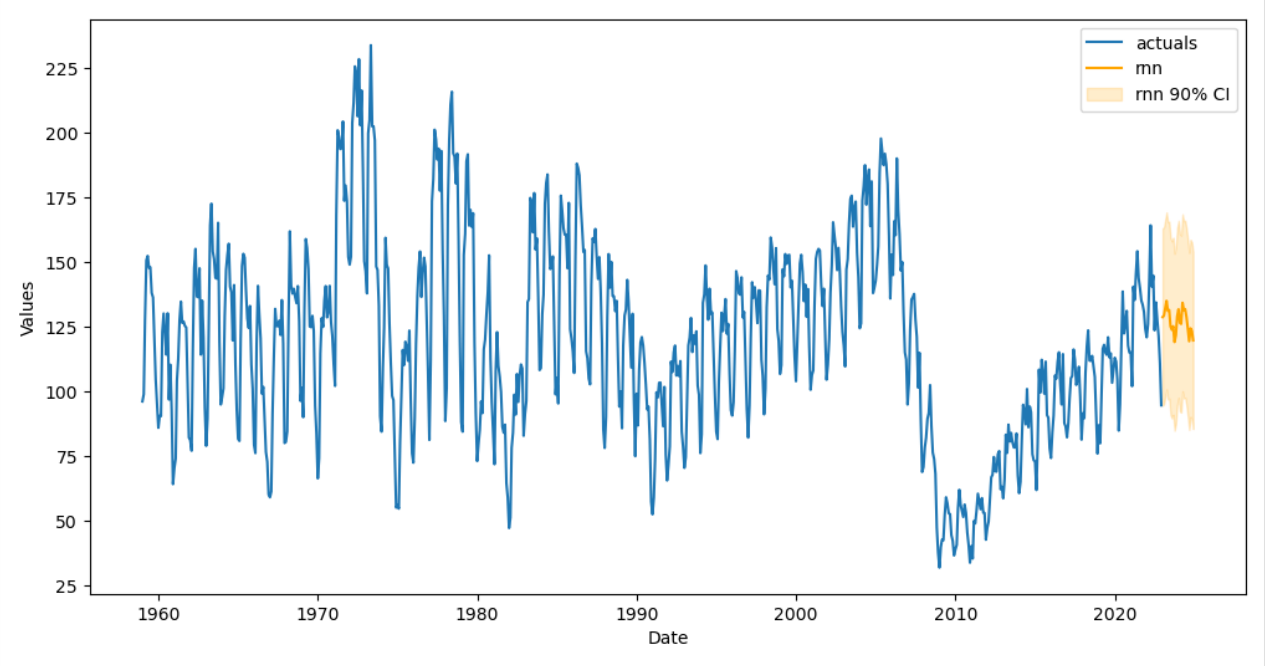
```
1/1 [==============================] - 0s 292ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 358ms/step
1/1 [==============================] - 0s 454ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 479ms/step
1/1 [==============================] - 0s 312ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 360ms/step
1/1 [==============================] - 0s 340ms/step
1/1 [==============================] - 0s 359ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 361ms/step
1/1 [==============================] - 0s 486ms/step
1/1 [==============================] - 0s 211ms/step
1/1 [==============================] - 1s 638ms/step
1/1 [==============================] - 1s 829ms/step
1/1 [==============================] - 0s 458ms/step
1/1 [==============================] - 0s 405ms/step
1/1 [==============================] - 0s 472ms/step
1/1 [==============================] - 0s 373ms/step
1/1 [==============================] - 0s 167ms/step
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
```

(continues on next page)

(continued from previous page)

```
Keras model archive saving:
File Name                                             Modified             Size
config.json                                     2023-09-20 11:49:23         2235
metadata.json                                   2023-09-20 11:49:23           64
variables.h5                                    2023-09-20 11:49:23       537280
Keras model archive loading:
File Name                                             Modified             Size
config.json                                     2023-09-20 11:49:22         2235
metadata.json                                   2023-09-20 11:49:22           64
variables.h5                                    2023-09-20 11:49:22       537280
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
1/1 [==============================] - 0s 371ms/step
1/1 [==============================] - 0s 400ms/step
23/23 [==============================] - 0s 5ms/step
```

[7]:
```
f.plot(ci=True)
plt.savefig('Probabilistic LSTM.png')
plt.show()
```



12.4 Problem 4 - Dynamic Probabilistic Forecasting

[8]:
```
params = f.best_params
num_chosen_lags = len(f.get_regressor_names())
```

[9]:
```
def forecaster(f,params):
    f.set_estimator('rnn')
    f.manual_forecast(**params,test_again=False,lags=num_chosen_lags)
```

[10]:
```
pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter),
    ]
)
```

[11]:
```
f = pipeline.fit_predict(f,params = params)
```

```
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
```

(continues on next page)

(continued from previous page)

```
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
...2
...3
...4
...5
...6
...7
...8
...9
...vars
Keras model archive saving:
File Name                                             Modified             Size
config.json                                      2023-09-20 11:49:34         2235
metadata.json                                    2023-09-20 11:49:34           64
variables.h5                                     2023-09-20 11:49:34       537280
Keras model archive loading:
File Name                                             Modified             Size
config.json                                      2023-09-20 11:49:34         2235
metadata.json                                    2023-09-20 11:49:34           64
variables.h5                                     2023-09-20 11:49:34       537280
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...metrics\mean
...vars
...0
...1
...optimizer
...vars
...0
...1
...10
```

(continues on next page)

(continued from previous page)

```
...2
...3
...4
...5
...6
...7
...8
...9
...vars
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...optimizer
...vars
...0
...vars
Keras model archive saving:
File Name                                              Modified             Size
config.json                                      2023-09-20 11:49:35         2235
metadata.json                                    2023-09-20 11:49:35           64
variables.h5                                     2023-09-20 11:49:35       185120
Keras model archive loading:
File Name                                              Modified             Size
config.json                                      2023-09-20 11:49:34         2235
metadata.json                                    2023-09-20 11:49:34           64
variables.h5                                     2023-09-20 11:49:34       185120
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\dense
...vars
...0
...1
...layers\lstm
...vars
...layers\lstm\cell
...vars
...0
...1
...2
...optimizer
...vars
...0
...vars
1/1 [==============================] - 0s 361ms/step
```

(continues on next page)

(continued from previous page)

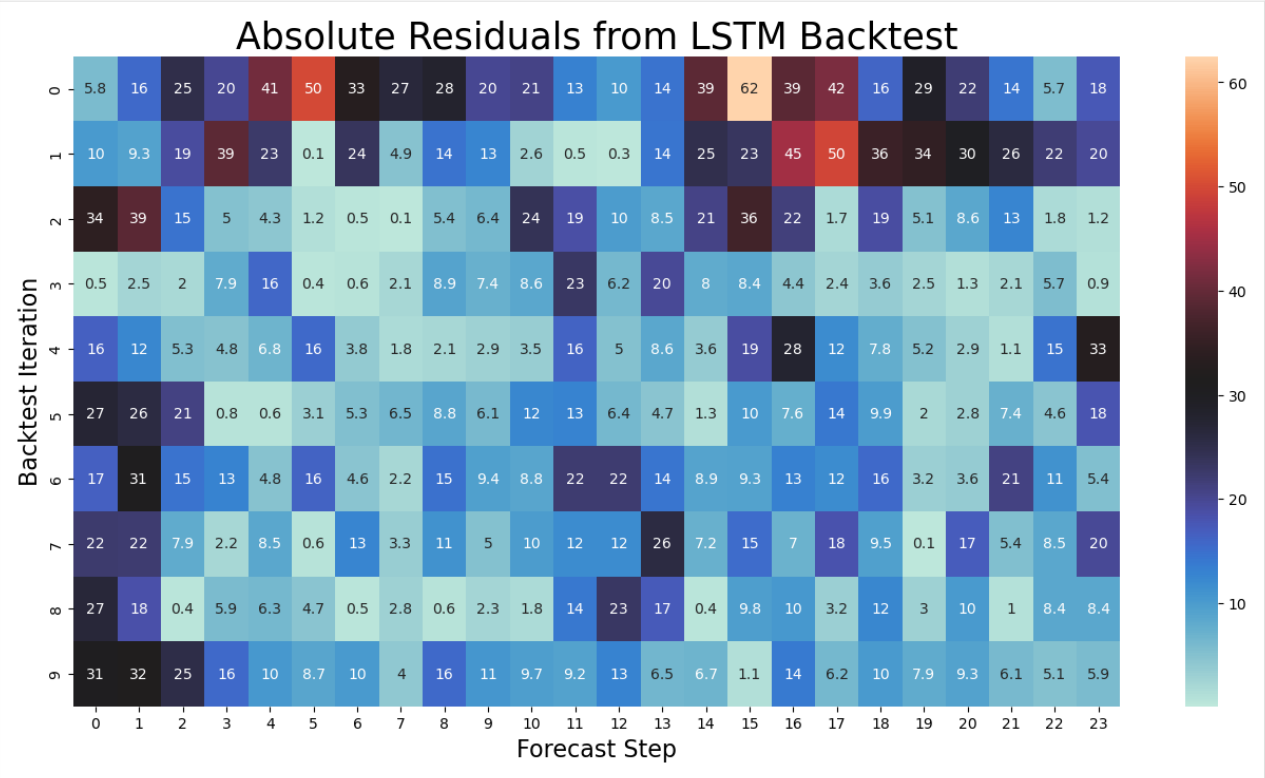
```
23/23 [==============================] - 0s 5ms/step
```

[12]:
```
backtest_results = backtest_for_resid_matrix(
    f,
    pipeline=pipeline,
    alpha = .1,
    jump_back = 12,
    params = f.best_params,
)
```

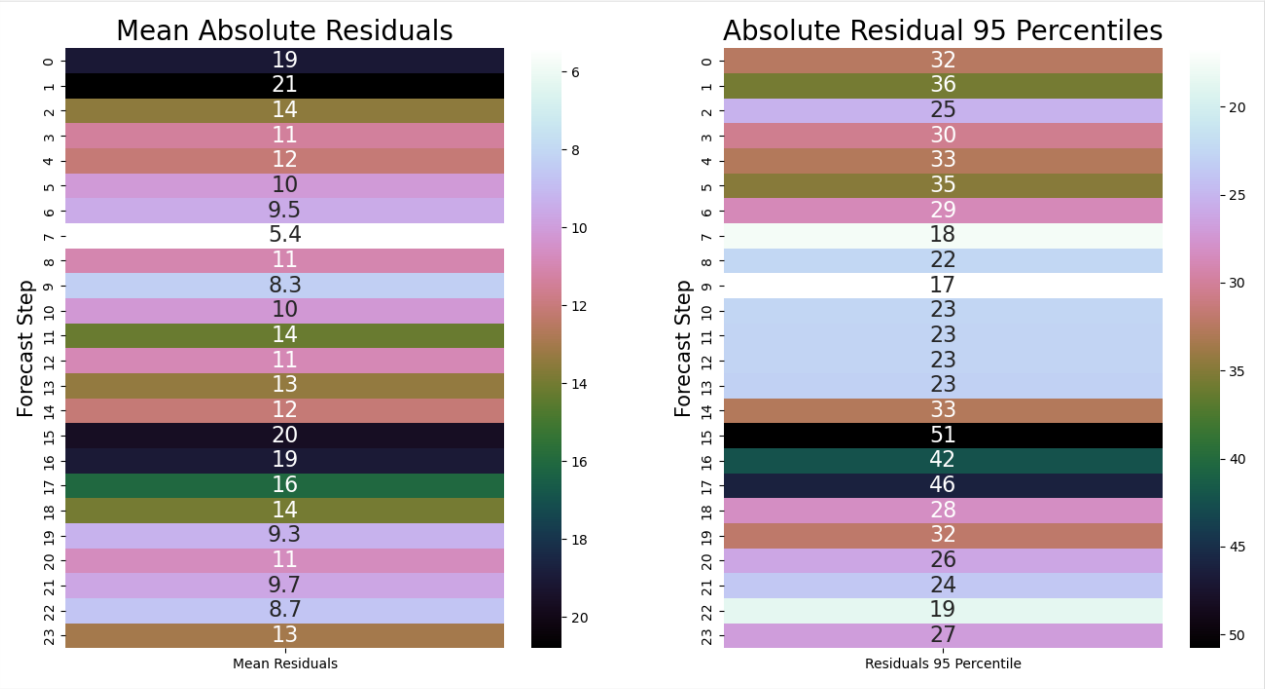
```
1/1 [==============================] - 0s 342ms/step
23/23 [==============================] - 0s 4ms/step
1/1 [==============================] - 0s 499ms/step
22/22 [==============================] - 0s 6ms/step
1/1 [==============================] - 0s 492ms/step
22/22 [==============================] - 0s 5ms/step
1/1 [==============================] - 0s 457ms/step
22/22 [==============================] - 0s 6ms/step
1/1 [==============================] - 0s 438ms/step
21/21 [==============================] - 0s 5ms/step
1/1 [==============================] - 1s 604ms/step
21/21 [==============================] - 0s 7ms/step
1/1 [==============================] - 0s 381ms/step
20/20 [==============================] - 0s 5ms/step
1/1 [==============================] - 1s 514ms/step
20/20 [==============================] - 0s 6ms/step
1/1 [==============================] - 0s 351ms/step
20/20 [==============================] - 0s 4ms/step
1/1 [==============================] - 0s 442ms/step
19/19 [==============================] - 0s 4ms/step
```

[13]:
```
backtest_resid_matrix = get_backtest_resid_matrix(backtest_results)
```

[14]:
```
pd.options.display.max_columns = None
fig, ax = plt.subplots(figsize=(16,8))
mat = pd.DataFrame(np.abs(backtest_resid_matrix[0]['rnn']))
sns.heatmap(
    mat.round(1),
    annot = True,
    ax = ax,
    cmap = sns.color_palette("icefire", as_cmap=True),
)
plt.ylabel('Backtest Iteration',size=16)
plt.xlabel('Forecast Step',size = 16)
plt.title('Absolute Residuals from LSTM Backtest',size=25)
plt.savefig('LSTM Resid Matrix.png')
plt.show()
```

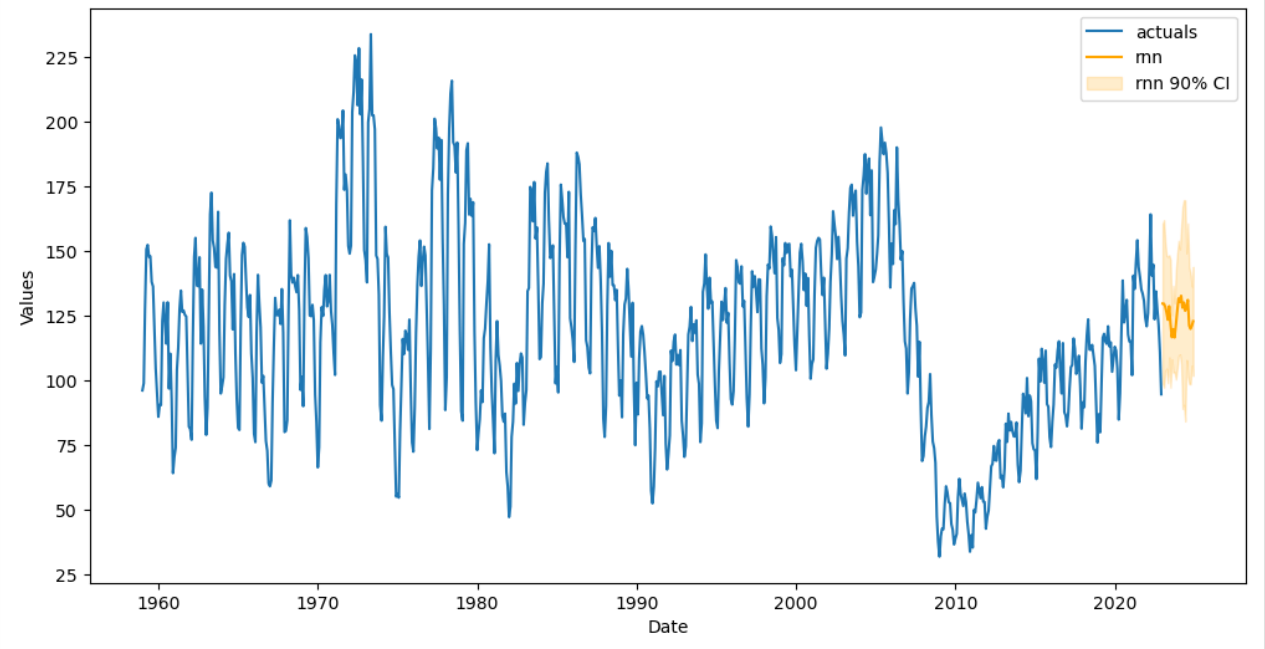


[15]:
```
fig, ax = plt.subplots(1,2,figsize=(16,8))
sns.heatmap(
    pd.DataFrame({'Mean Residuals':mat.mean().round(1)}),
    annot = True,
    cmap = 'cubehelix_r',
    ax = ax[0],
    annot_kws={"fontsize": 16},
)
cbar = ax[0].collections[0].colorbar
cbar.ax.invert_yaxis()
ax[0].set_title('Mean Absolute Residuals',size=20)
ax[0].set_ylabel('Forecast Step',size=15)
ax[0].set_xlabel('')
sns.heatmap(
    pd.DataFrame({'Residuals 95 Percentile':np.percentile(mat, q=95, axis = 0)}),
    annot = True,
    cmap = 'cubehelix_r',
    ax = ax[1],
    annot_kws={"fontsize": 16},
)
cbar = ax[1].collections[0].colorbar
cbar.ax.invert_yaxis()
ax[1].set_title('Absolute Residual 95 Percentiles',size=20)
ax[1].set_ylabel('Forecast Step',size=15)
ax[1].set_xlabel('')
plt.show()
```



[16]:
```
overwrite_forecast_intervals(f,backtest_resid_matrix=backtest_resid_matrix,alpha=.1)
```

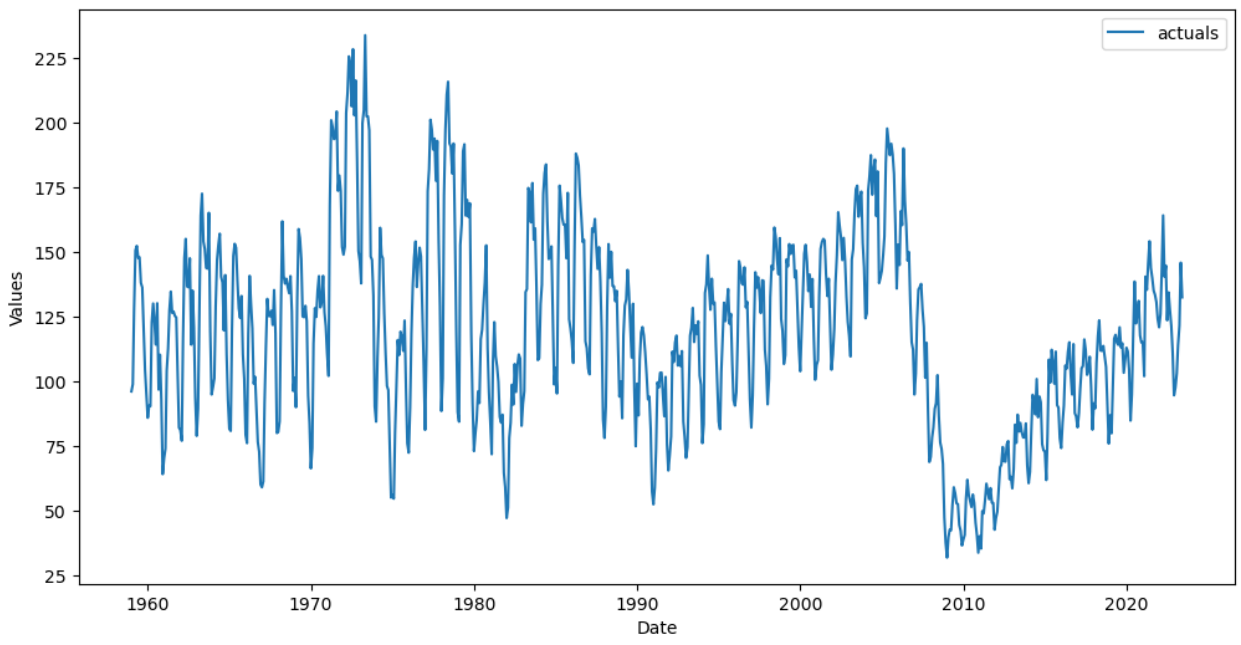
[17]:
```
f.plot(ci=True)
plt.savefig('LSTM dynamic intervals.png')
plt.show()
```



12.5 Problem 5 - Transfer Learning

12.5.1 Scenario 1: New data from the same series

[49]:
```
df = pdr.get_data_fred(
    'HOUSTNSA',
    start = '1959-01-01',
    end = '2023-06-30',
)

f_new = Forecaster(
    y = df.iloc[:,0],
    current_dates = df.index,
    future_dates = 24, # 2-year forecast horizon
)

f_new.plot()
plt.show()
```

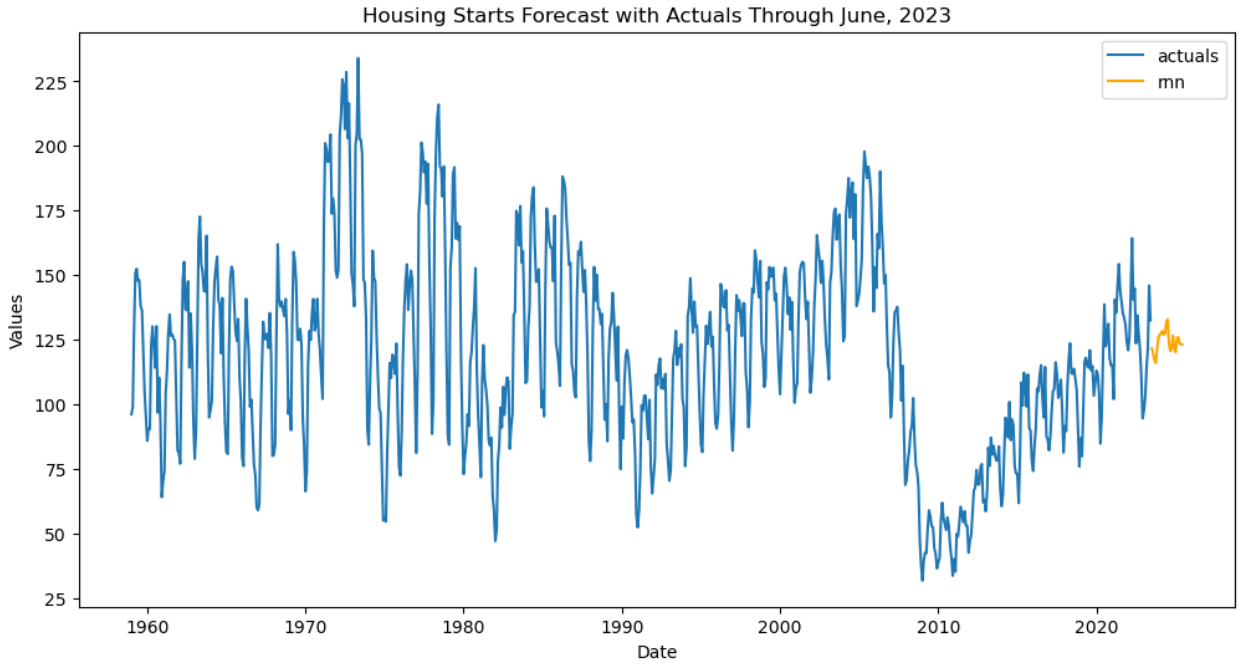


[50]:
```
def transfer_forecast(f_new,transfer_from):
    f_new = infer_apply_Xvar_selection(
        infer_from=transfer_from,
        apply_to=f_new
    )
    f_new.transfer_predict(
        transfer_from=transfer_from,
        model='rnn',
        model_type='tf'
    )
```

[51]:
```
pipeline_can = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Transfer Forecast',transfer_forecast),
        ('Revert',reverter),
    ]
)

f_new = pipeline_can.fit_predict(f_new,transfer_from=f)
```

```
1/1 [==============================] - 0s 20ms/step
24/24 [==============================] - 0s 3ms/step
```

[53]:
```
f_new.plot()
plt.title('Housing Starts Forecast with Actuals Through June, 2023')
plt.savefig('RNN transferred same series.png')
plt.show()
```



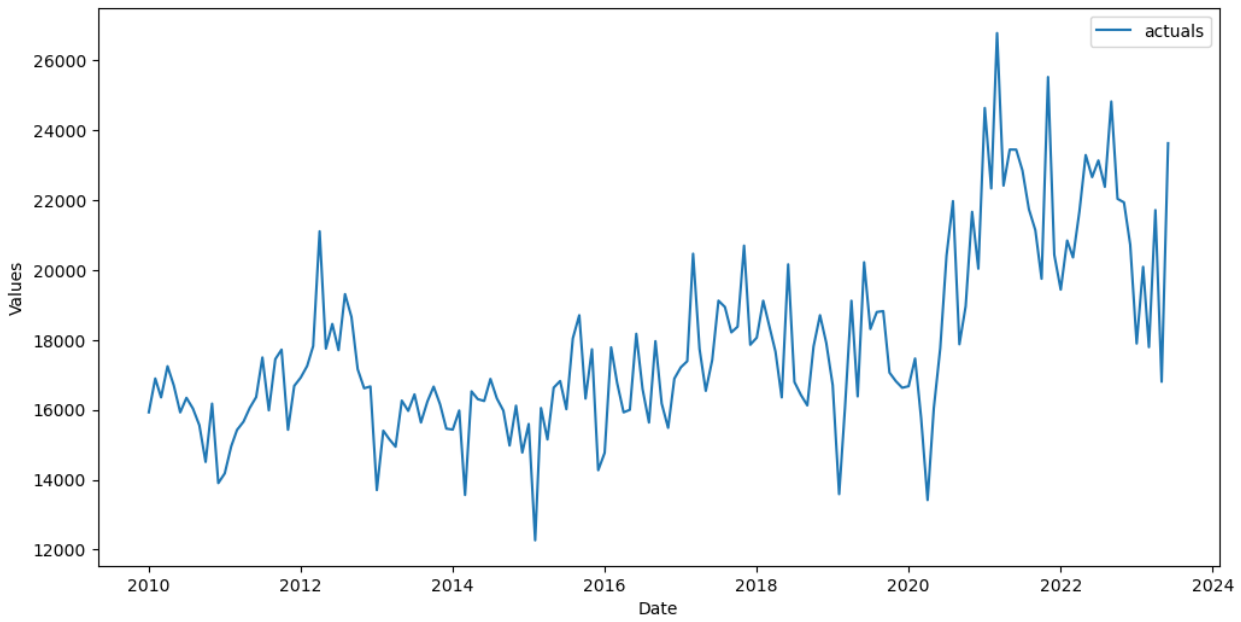
12.5.2 Scenario 2: A new time series with similar characteristics

[54]:
```
df = pdr.get_data_fred(
    'CANWSCNDW01STSAM',
    start = '2010-01-01',
    end = '2023-06-30',
)

f_new = Forecaster(
    y = df.iloc[:,0],
    current_dates = df.index,
```

(continues on next page)

(continued from previous page)

```
    future_dates = 24, # 2-year forecast horizon
)

f_new.plot()
plt.show()
```



[55]:
```
def transfer_forecast(f_new,transfer_from):
    f_new = infer_apply_Xvar_selection(
        infer_from=transfer_from,
        apply_to=f_new
    )
    f_new.transfer_predict(
        transfer_from=transfer_from,
        model='rnn',
        model_type='tf'
    )
```

[56]:
```
pipeline_can = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Transfer Forecast',transfer_forecast),
        ('Revert',reverter),
    ]
)

f_new = pipeline_can.fit_predict(f_new,transfer_from=f)
```

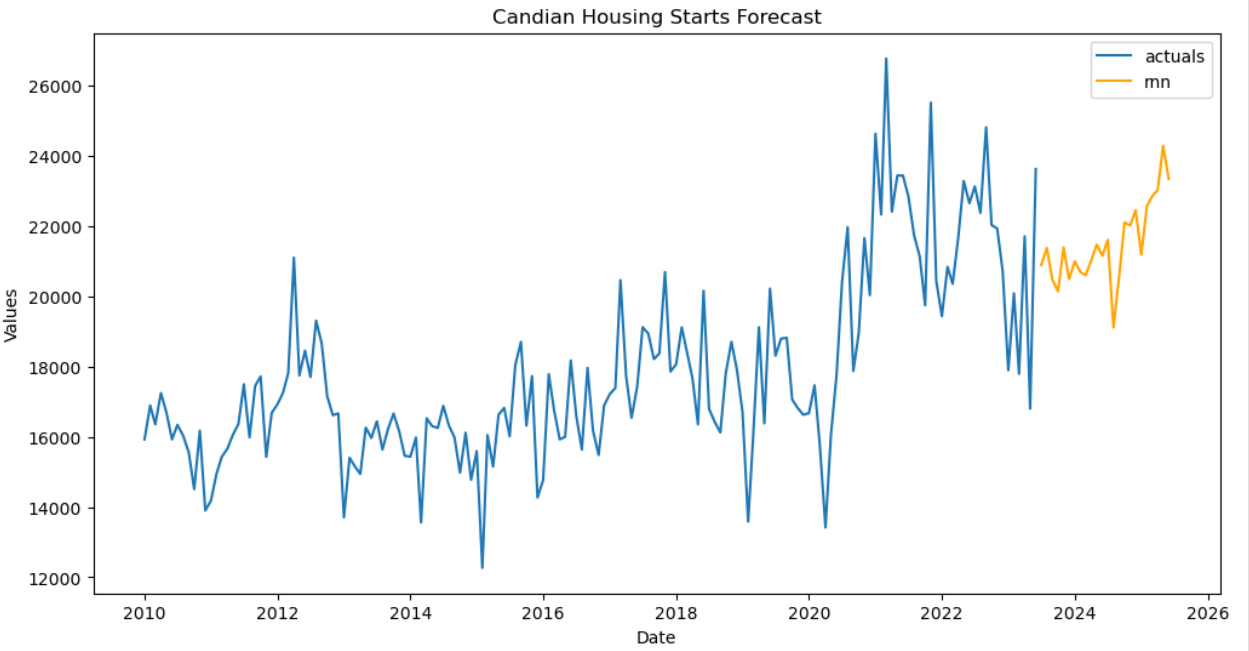
```
1/1 [==============================] - 0s 22ms/step
4/4 [==============================] - 0s 4ms/step
```

[57]:
```
f_new.plot()
```

(continues on next page)

(continued from previous page)

```
plt.title('Candian Housing Starts Forecast')
plt.savefig('Transferred RNN Canada')
plt.show()
```




```
[ ]:
```

CHAPTER

THIRTEEN

META PROPHET

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.It is designed to se automatically find a good set of hyperparameters for the model in an effort to make forrecast for time series with seasonality and trends.

The Prophet model is not autoregressive, like ARIMA, exponential smoothing, and the other methods we study in a typical time series course (including my own).

The 3 components are:

1. The trend g(t) which can be either linear or logistic.
2. The seasonality s(t), modeled using a Fourier series.
3. The holiday component h(t), which is essentially a one-hot vector "dotted" with a vector of weights, each representing the contribution from their respective holiday.

In this notebook, following concepts are covered:
- Loading time-series
- EDA
- Preparing the data for modeling
- Implementing prophet model
- Evaluating the results

Data: https://www.kaggle.com/datasets/bobnau/daily-website-visitors
Install prophet: `pip install prophet`

Special thanks to Zohoor Nezhad Halafi for contributing this notebook! Connect with her on LinkedIn.

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scalecast.Forecaster import Forecaster

sns.set(rc={'figure.figsize':(16,8)})
```

13.1 Loading time series

[2]:
```
data = pd.read_csv('daily-website-visitors.csv',parse_dates=['Date'])
```

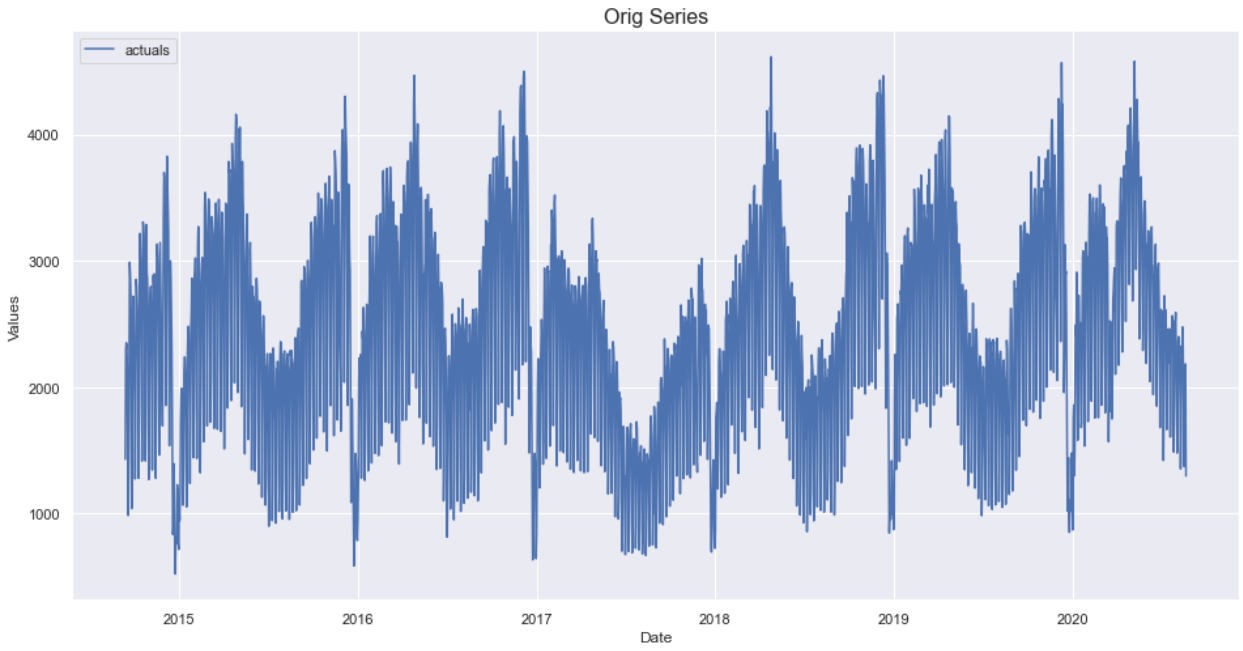
[3]:
```
data=data[['First.Time.Visits','Date']]
```

[4]:
```
data.head()
```

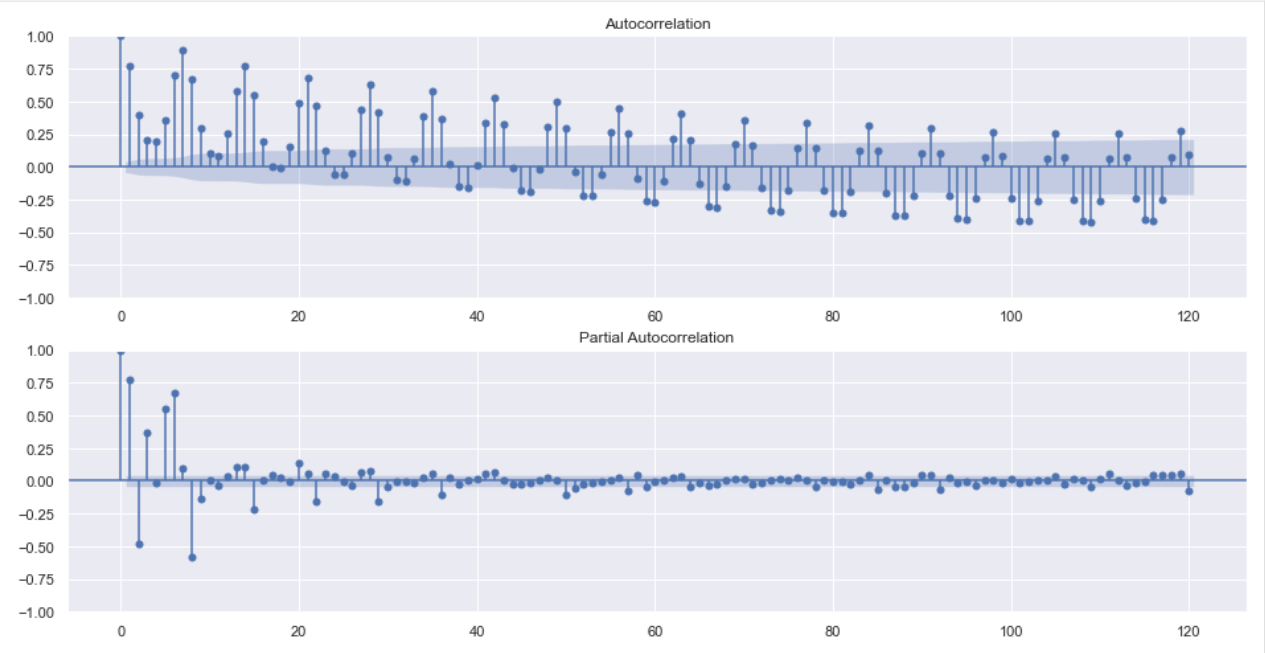
[4]:
```
   First.Time.Visits       Date
0               1430 2014-09-14
1               2297 2014-09-15
2               2352 2014-09-16
3               2327 2014-09-17
4               2130 2014-09-18
```

13.2 EDA

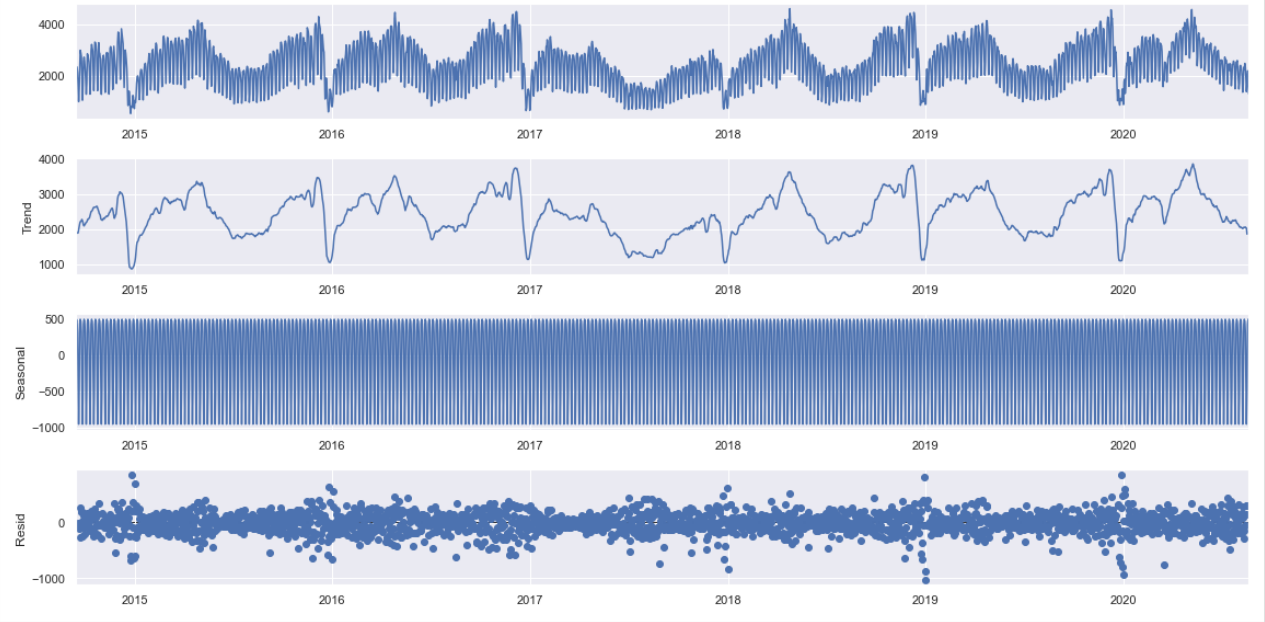
[5]:
```
f=Forecaster(y=data['First.Time.Visits'],current_dates=data['Date'])
f.plot()
plt.title('Orig Series',size=16)
plt.show()
```



[7]:
```
figs, axs = plt.subplots(2, 1)
f.plot_acf(ax=axs[0],lags=120)
f.plot_pacf(ax=axs[1],lags=120)
plt.show()
```



[8]:
```
f.seasonal_decompose().plot()
plt.show()
```



ADF Test

[9]:
```
critical_pval = 0.05
print('-'*100)
print('Augmented Dickey-Fuller results:')
stat, pval, _, _, _, _ = f.adf_test(full_res=True)
print('the test-stat value is: {:.2f}'.format(stat))
print('the p-value is {:.4f}'.format(pval))
print('the series is {}'.format('stationary' if pval < critical_pval else 'not stationary
```

(continues on next page)

(continued from previous page)

```
↪'))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
↪-----------
Augmented Dickey-Fuller results:
the test-stat value is: -4.48
the p-value is 0.0002
the series is stationary
----------------------------------------------------------------------------------------------------
↪-----------
```

13.3 Preparing the data for modeling

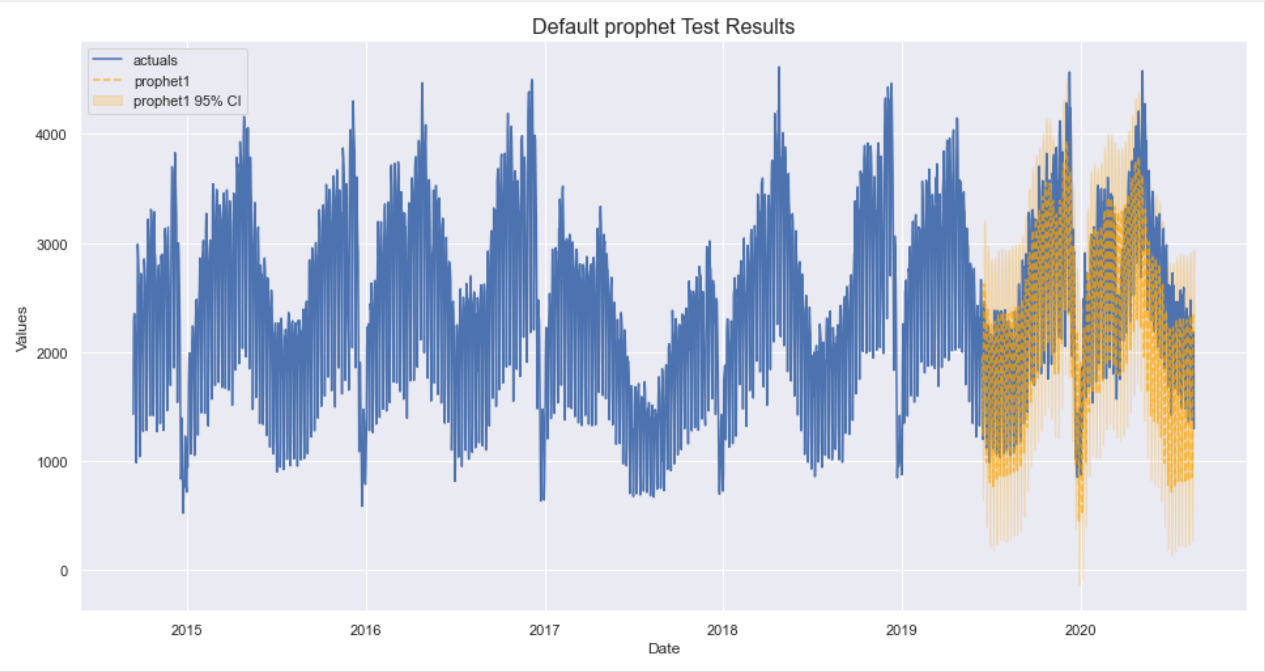
[10]:
```
f.generate_future_dates(60)
f.set_test_length(.2)
f.set_estimator('prophet')
f
```

[10]:
```
Forecaster(
    DateStartActuals=2014-09-14T00:00:00.000000000
    DateEndActuals=2020-08-19T00:00:00.000000000
    Freq=D
    N_actuals=2167
    ForecastLength=60
    Xvars=[]
    Differenced=0
    TestLength=433
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    BootstrapSamples=100
    CurrentEstimator=prophet
)
```

13.4 Forecasting

[11]:
```
f.manual_forecast(call_me='prophet1')
```

[12]:
```
f.plot_test_set(ci=True,models='prophet1')
plt.title('Default prophet Test Results',size=16)
plt.show()
```

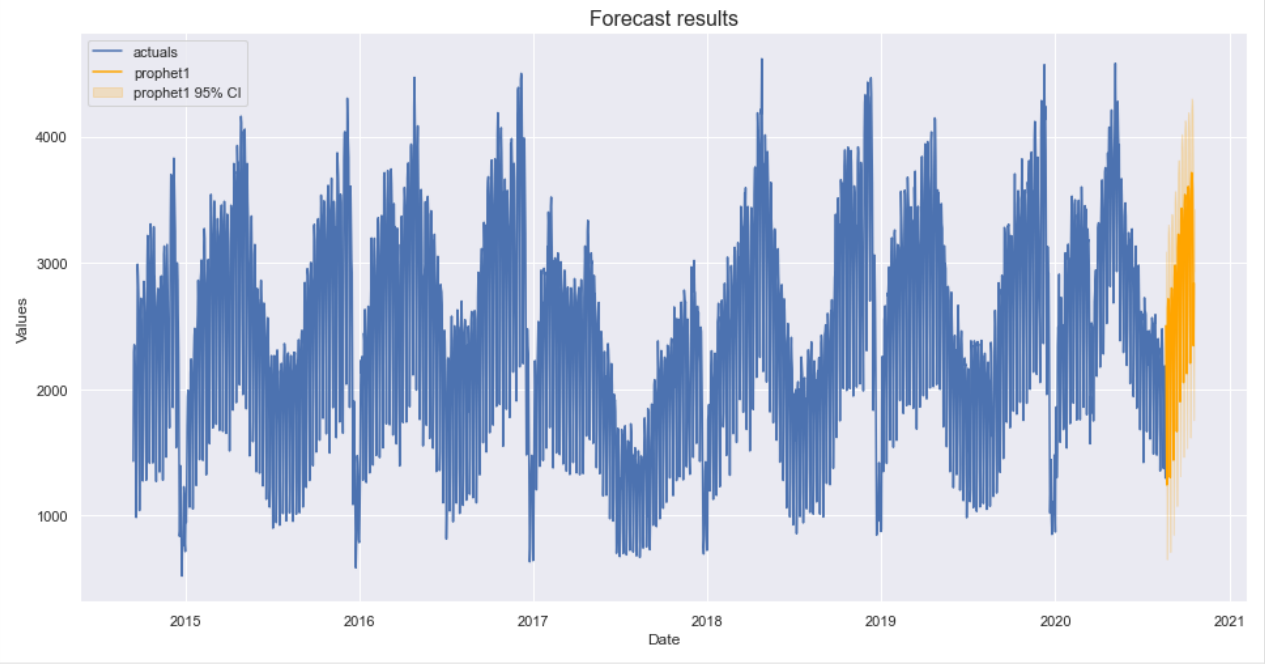


[15]:
```
results = f.export('model_summaries')
results[['TestSetRMSE','InSampleRMSE','TestSetMAPE','InSampleMAPE']]
```

[15]:

	TestSetRMSE	InSampleRMSE	TestSetMAPE	InSampleMAPE
0	403.968233	283.145403	0.136388	0.104243

[14]:
```
f.plot(ci=True,models='prophet1')
plt.title('Forecast results',size=16)
plt.show()
```



```
[ ]:
```

CHAPTER

FOURTEEN

MULTIVARIATE FORECASTING

This notebook outlines an example of scalecast using multiple series to forecast one another using scikit-learn models.

The data is available on Kaggle: https://www.kaggle.com/datasets/neuromusic/avocado-prices
See the blog post:
https://towardsdatascience.com/multiple-series-forecast-them-together-with-any-sklearn-model-96319d46269

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt
from scalecast.Forecaster import Forecaster
from scalecast.MVForecaster import MVForecaster
from scalecast.multiseries import export_model_summaries
from scalecast import GridGenerator
```

[2]:
```
# read data
data = pd.read_csv('avocado.csv',parse_dates=['Date']).sort_values(['Date'])
# sort appropriately (not doing this could cause issues)
data = data.sort_values(['region','type','Date'])
```

We will be forecasting the organic and conventional avocado sales from California only.

[3]:
```
data_cali = data.loc[data['region'] == 'California']
data_cali_org = data_cali.loc[data_cali['type'] == 'organic']
data_cali_con = data_cali.loc[data_cali['type'] == 'conventional']
```

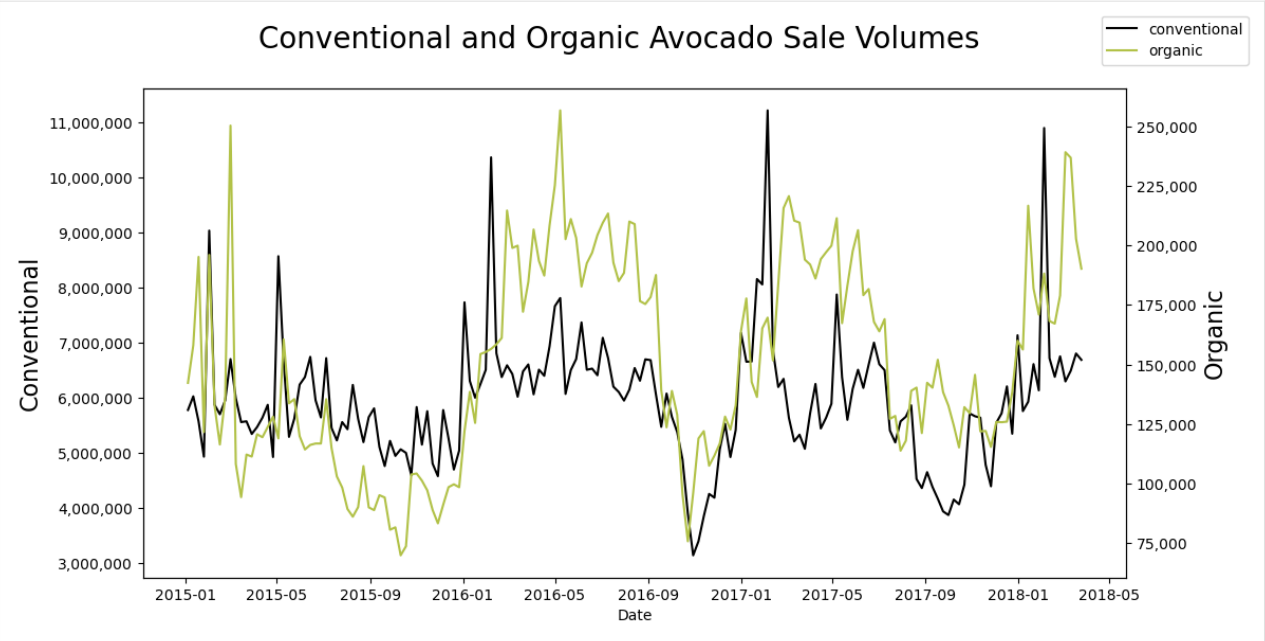
14.1 Choose Models and Import Validation Grids

[4]:
```
# download template validation grids (will not overwrite existing Grids.py file by␣
↪default)
models = ('mlr','elasticnet','knn','rf','gbt','xgboost','mlp')
GridGenerator.get_example_grids()
GridGenerator.get_mv_grids()
```

14.2 EDA

14.2.1 Plot

[5]:
```
fig, ax = plt.subplots(figsize=(12,6))
sns.lineplot(
    x='Date',
    y='Total Volume',
    data=data_cali_con,
    label='conventional',
    ax=ax,
    color='black',
    legend=False
)
plt.ylabel('Conventional',size=16)
ax2 = ax.twinx()
sns.lineplot(
    x='Date',
    y='Total Volume',
    data=data_cali_org,
    label='organic',
    ax=ax2,
    color='#B2C248',
    legend=False
)
ax.figure.legend()
plt.ylabel('Organic',size=16)
ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))
ax2.yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))
plt.suptitle('Conventional and Organic Avocado Sale Volumes',size=20)
plt.show()
```



14.2.2 Examine Correlation between the series

[6]:
```
corr = np.corrcoef(data_cali_org['Total Volume'].values,data_cali_con['Total Volume'].
↪values)[0,1]
print('{:.2%}'.format(corr))
```

```
48.02%
```

14.2.3 Load into Forecaster from scalecast

[7]:
```
fcon = Forecaster(
    y=data_cali_con['Total Volume'],
    current_dates = data_cali_con['Date'],
    test_length = .2,
    future_dates = 52,
    validation_length = 4,
    metrics = ['rmse','r2'],
    cis = True,
)
fcon
```

[7]:
```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=52
    Xvars=[]
    TestLength=33
    ValidationMetric=rmse
```

(continues on next page)

(continued from previous page)

```
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[8]:
```
forg = Forecaster(
    y=data_cali_org['Total Volume'],
    current_dates = data_cali_org['Date'],
    test_length = .2,
    future_dates = 52,
    validation_length = 4,
    metrics = ['rmse','r2'],
    cis = True,
)
forg
```

[8]:
```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=52
    Xvars=[]
    TestLength=33
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

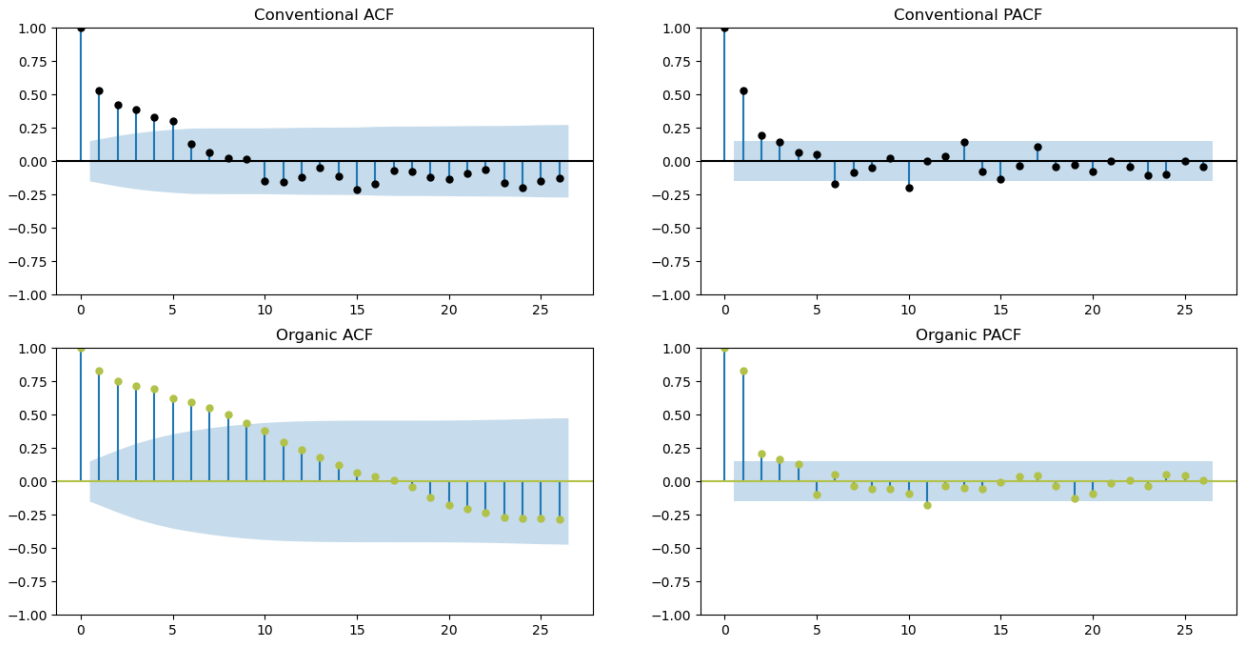
14.2.4 ACF and PACF Plots

[9]:
```
figs, axs = plt.subplots(2, 2,figsize=(16,8))
fcon.plot_acf(
    ax=axs[0,0],
    title='Conventional ACF',
    lags=26,
    color='black'
)
fcon.plot_pacf(
    ax=axs[0,1],
    title='Conventional PACF',
    lags=26,
    color='black',
    method='ywm'
)
forg.plot_acf(
    ax=axs[1,0],
    title='Organic ACF',
```

(continues on next page)

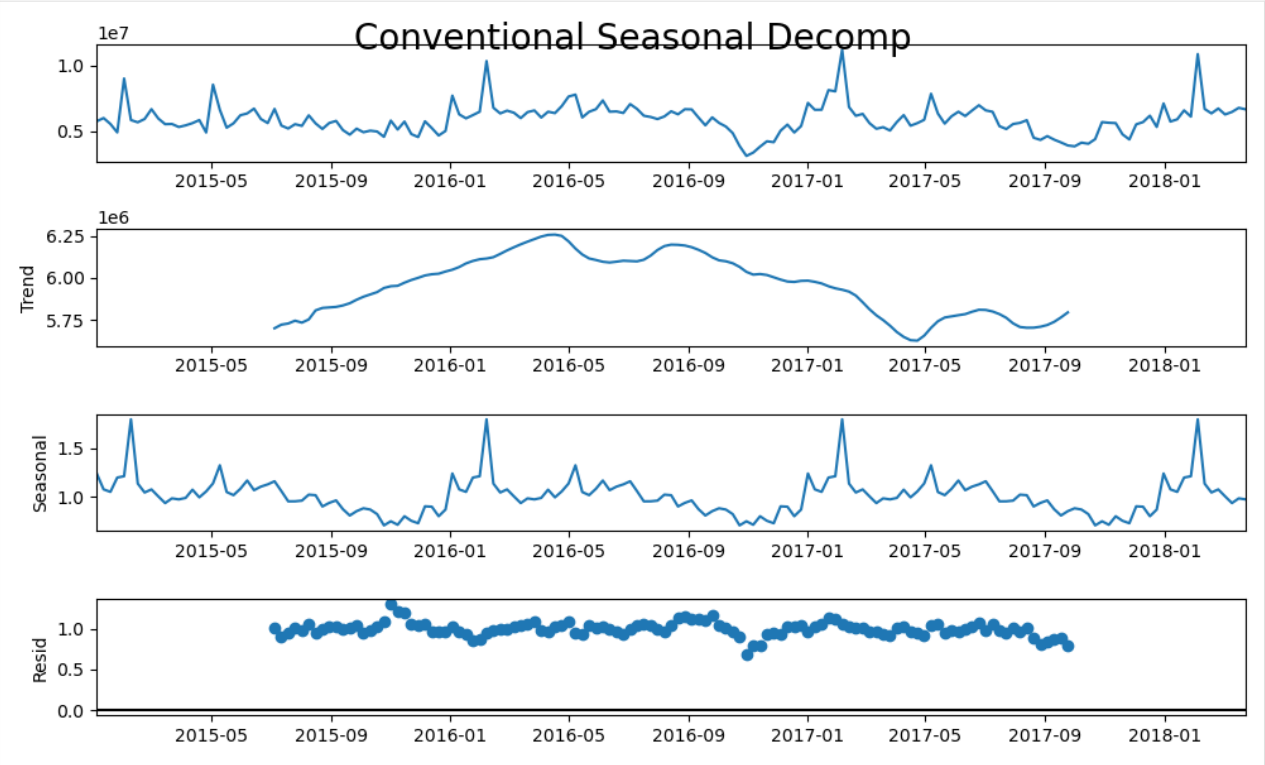
(continued from previous page)

```
    lags=26,
    color='#B2C248'
)
forg.plot_pacf(
    ax=axs[1,1],
    title='Organic PACF',
    lags=26,
    color='#B2C248',
    method='ywm'
)
plt.show()
```

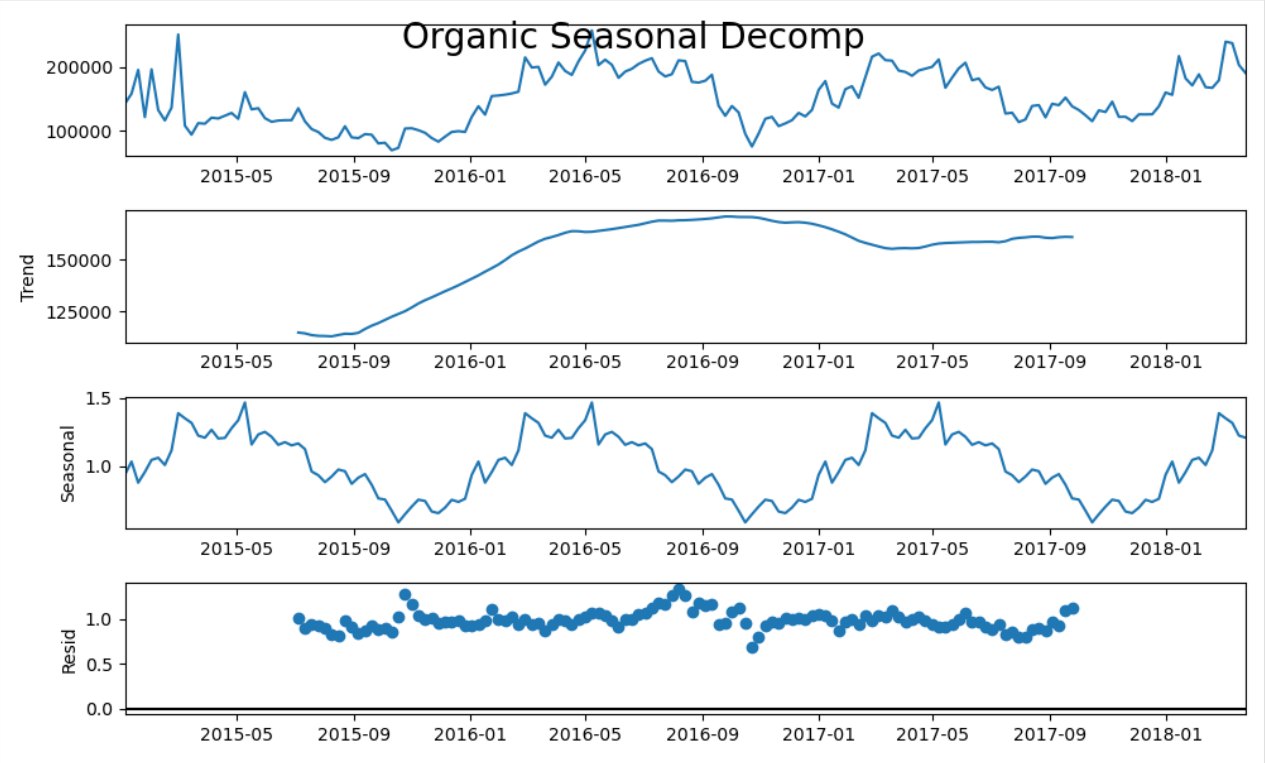


14.2.5 Seasonal Decomposition Plots

[10]:
```
plt.rc("figure",figsize=(10,6))
fcon.seasonal_decompose(model='mul').plot()
plt.suptitle('Conventional Seasonal Decomp',size=20)
plt.show()
```



[11]:
```
forg.seasonal_decompose(model='mul').plot()
plt.suptitle('Organic Seasonal Decomp',size=20)
plt.show()
```



14.2.6 Check Stationarity

[12]:
```
critical_pval = 0.05
print('-'*100)
print('Conventional Augmented Dickey-Fuller results:')
stat, pval, _, _, _, _ = fcon.adf_test(full_res=True)
print('the test-stat value is: {:.2f}'.format(stat))
print('the p-value is {:.4f}'.format(pval))
print('the series is {}'.format('stationary' if pval < critical_pval else 'not stationary
↪'))
print('-'*100)
print('Organic Augmented Dickey-Fuller results:')
stat, pval, _, _, _, _ = forg.adf_test(full_res=True)
print('the test-stat value is: {:.2f}'.format(stat))
print('the p-value is {:.4f}'.format(pval))
print('the series is {}'.format('stationary' if pval < critical_pval else 'not stationary
↪'))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
↪-----------
Conventional Augmented Dickey-Fuller results:
the test-stat value is: -3.35
the p-value is 0.0128
the series is stationary
----------------------------------------------------------------------------------------------------
↪-----------
Organic Augmented Dickey-Fuller results:
the test-stat value is: -2.94
the p-value is 0.0404
the series is stationary
----------------------------------------------------------------------------------------------------
↪-----------
```

14.3 Scalecast - Univariate

14.3.1 Load Objects with Xvars:

- Find best combination of trend, lags, and seasonality monitoring the validation set (4 observations before the test set)

[13]:
```
for f in (fcon,forg):
    f.auto_Xvar_select(
        estimator = 'elasticnet',
        monitor = 'ValidationMetricValue',
        irr_cycles = [26], # try irregular semi-annual seaosnality
        cross_validate=True,
        cvkwargs={
            'k':3,
            'test_length':13,
            'space_between_sets':4,
```

(continues on next page)

(continued from previous page)

```
        }
    )
    print(f)
```

```
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=52
    Xvars=['weeksin', 'weekcos', 'monthsin', 'monthcos', 'quartersin', 'quartercos', 'AR1
↪', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11', 'AR12',
↪'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21', 'AR22', 'AR23',
↪ 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30', 'AR31', 'AR32', 'AR33']
    TestLength=33
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
Forecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    ForecastLength=52
    Xvars=['lnt', 'AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10',
↪'AR11', 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21',
↪ 'AR22', 'AR23', 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30', 'AR31', 'AR32
↪', 'AR33']
    TestLength=33
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

14.3.2 Tune and Forecast with Selected Models

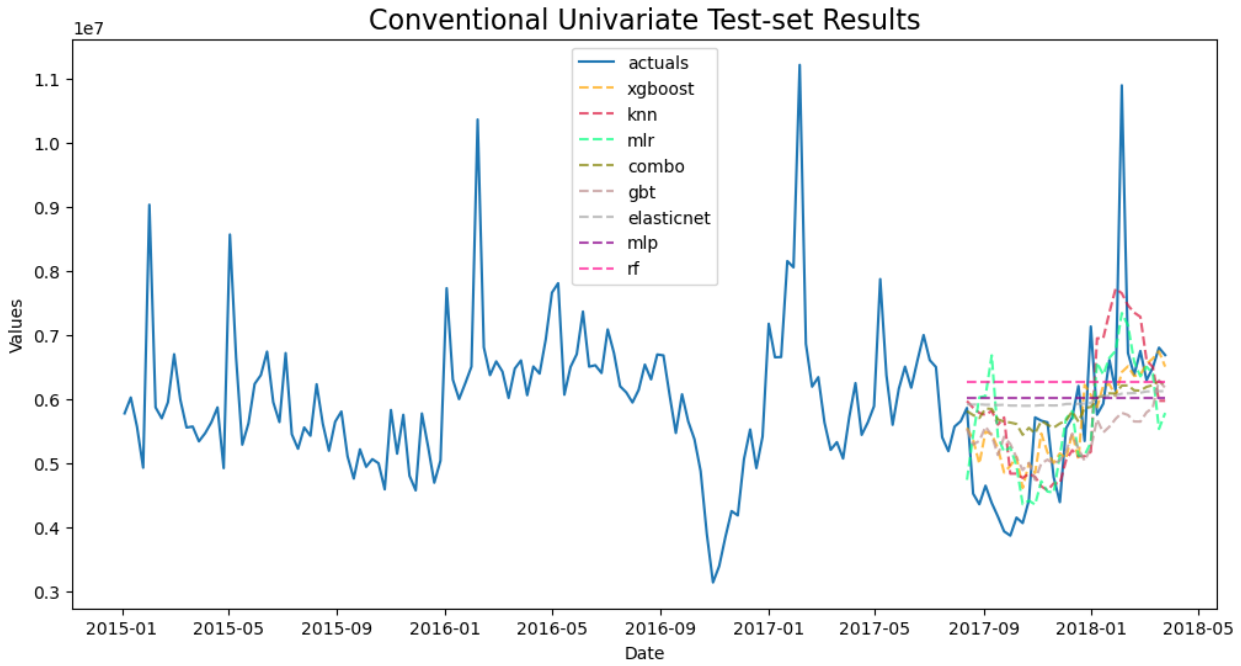
- Find optimal hyperparameters using the four out-of-sample observations then test results on test set

14.3.2.1 Conventional

[14]:
```
fcon.tune_test_forecast(models,feature_importance=True)
best_model_con_uni = fcon.order_fcsts()[0]
fcon.set_estimator('combo')
fcon.manual_forecast(how='weighted')
```

```
2023-04-11 15:32:19.815928: I tensorflow/core/platform/cpu_feature_guard.cc:193] This␣
↪TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use␣
↪the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler␣
↪flags.
```

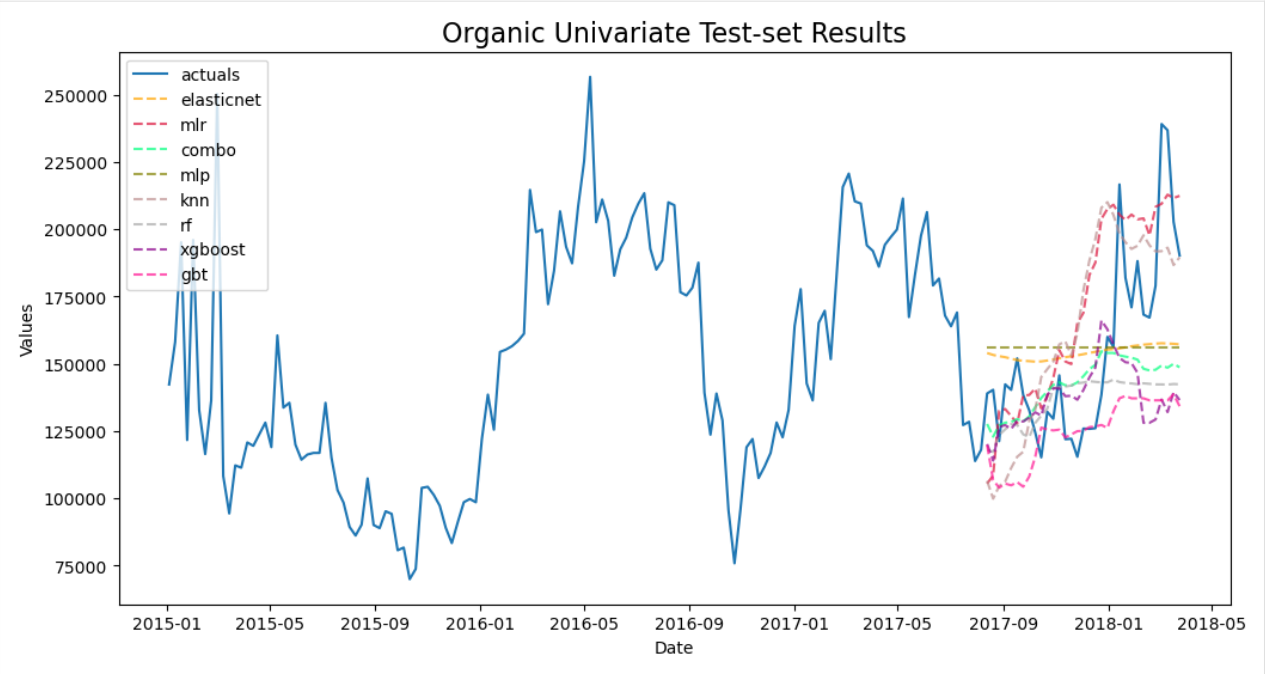
[15]:
```
fcon.plot_test_set(order_by='TestSetRMSE')
plt.title('Conventional Univariate Test-set Results',size=16)
plt.show()
```



14.3.2.2 Organic

[16]:
```
forg.tune_test_forecast(models,feature_importance=True)
best_model_org_uni = forg.order_fcsts()[0]
forg.set_estimator('combo')
forg.manual_forecast(how='weighted')
```

[17]:
```
forg.plot_test_set(order_by='TestSetRMSE')
plt.title('Organic Univariate Test-set Results',size=16)
plt.show()
```



14.3.3 Model Summaries

[22]:
```
pd.set_option('display.float_format',  '{:.4f}'.format)
ms = export_model_summaries({'Conventional':fcon,'Organic':forg},determine_best_by=
↪'TestSetRMSE')
ms[
    [
        'ModelNickname',
        'Series',
        'TestSetRMSE',
        'TestSetR2',
        'InSampleRMSE',
        'InSampleR2',
        'best_model'
    ]
]
```

[22]:

	ModelNickname	Series	TestSetRMSE	TestSetR2	InSampleRMSE	\
0	xgboost	Conventional	1006995.5599	0.4446	3.6627	
1	knn	Conventional	1131608.0344	0.2987	817774.1332	
2	mlr	Conventional	1149633.3896	0.2761	758866.6492	
3	combo	Conventional	1212096.9382	0.1953	688989.9601	
4	gbt	Conventional	1220551.0383	0.1841	194073.1900	
5	elasticnet	Conventional	1337617.6356	0.0201	1143622.0298	
6	mlp	Conventional	1405812.4388	-0.0824	1220380.8148	
7	rf	Conventional	1491918.2868	-0.2191	1223366.9295	
8	elasticnet	Organic	31734.3993	0.1053	37019.4163	
9	mlr	Organic	31826.3068	0.1001	15369.3197	
10	combo	Organic	32193.1479	0.0793	13693.4548	
11	mlp	Organic	33604.8672	-0.0032	42607.8305	

(continues on next page)

(continued from previous page)

```
12           knn       Organic   35461.8762    -0.1172    13801.7064
13            rf       Organic   35785.6551    -0.1377     9688.4987
14       xgboost       Organic   37617.5747    -0.2571        0.6991
15           gbt       Organic   40632.6313    -0.4667     7511.2968

    InSampleR2  best_model
0       1.0000        True
1       0.5510       False
2       0.6133       False
3       0.6813       False
4       0.9747       False
5       0.1218       False
6      -0.0000       False
7      -0.0049       False
8       0.2451        True
9       0.8699       False
10      0.8967       False
11     -0.0000       False
12      0.8951       False
13      0.9483       False
14      1.0000       False
15      0.9689       False
```

Because running so many models can cause overfitting on the test set, we can check the average test error between all models to get another metric of how effective our modeling process is.

[23]:
```
print('-'*100)
for series in ms['Series'].unique():
    print('univariate average test MAPE for {}: {:.4f}'.format(series,ms.loc[ms['Series
↪'] == series,'TestSetRMSE'].mean()))
    print('univariate average test R2 for {}: {:.2f}'.format(series,ms.loc[ms['Series']␣
↪== series,'TestSetR2'].mean()))
    print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
↪-----------
univariate average test MAPE for Conventional: 1244529.1652
univariate average test R2 for Conventional: 0.14
----------------------------------------------------------------------------------------------------
↪-----------
univariate average test MAPE for Organic: 34857.0573
univariate average test R2 for Organic: -0.09
----------------------------------------------------------------------------------------------------
↪-----------
```

These are interesting error metrics, but a lot of models, including XGBoost and GBT appear to be overfitting. Let's see if we can beat the results with a multivariate approach.

14.4 Scalecast - Multivariate

14.4.1 Set MV Parameters

- Forecast horizon already set
- Xvars already set
- Test size must be set: 20%
- Validation size must be set: 4 periods

[24]:
```
mvf = MVForecaster(
    fcon,forg,
    names=['Conventional','Organic'],
    test_length = .2,
    valiation_length = 4,
    cis = True,
    metrics = ['rmse','r2'],
) # init the mvf object
mvf
```

[24]:
```
MVForecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    N_series=2
    SeriesNames=['Conventional', 'Organic']
    ForecastLength=52
    Xvars=['weeksin', 'weekcos', 'monthsin', 'monthcos', 'quartersin', 'quartercos', 'lnt
↪']
    TestLength=33
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    OptimizeOn=mean
    GridsFile=MVGrids
)
```

The lags from each Forecaster object dropped but they will be added into the multivariate models differently.

14.4.2 View Series Correlation

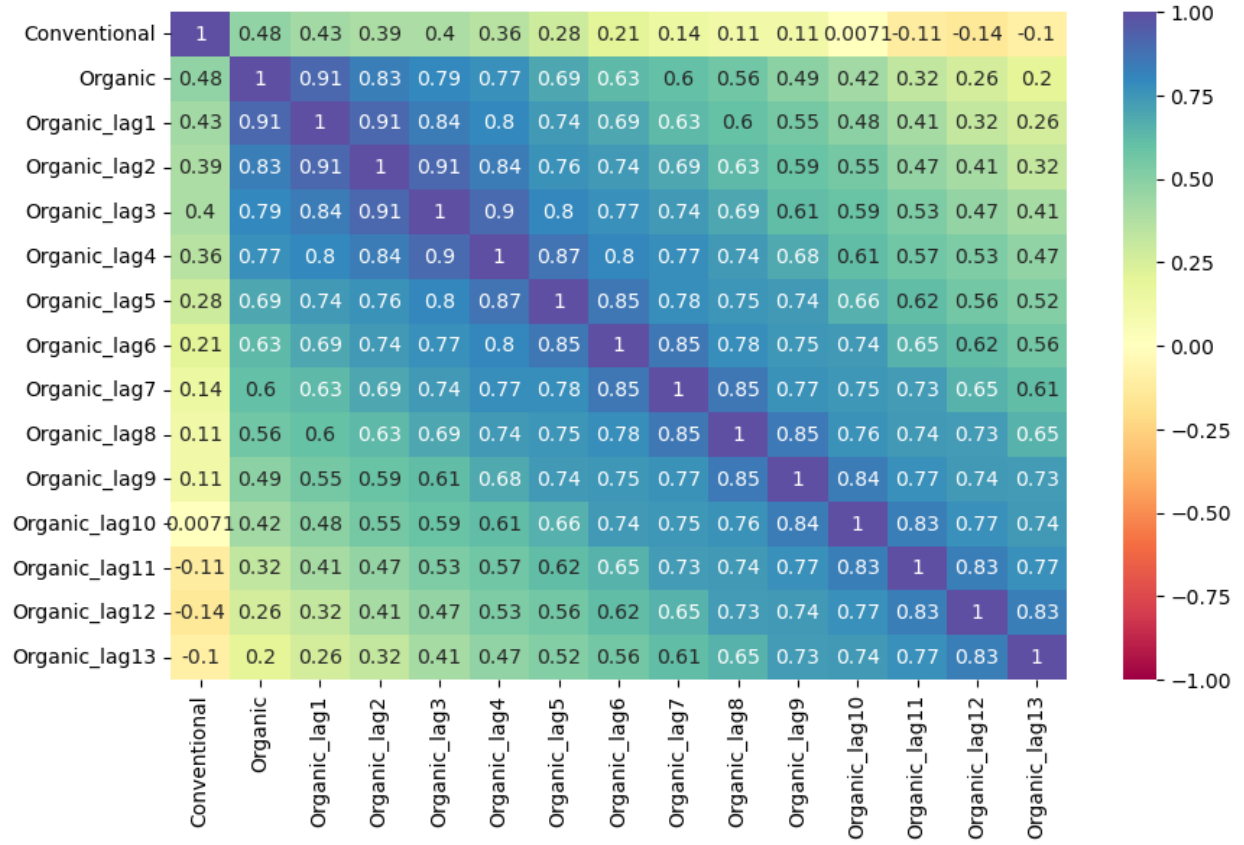
[25]:
```
mvf.corr()
```

[25]:

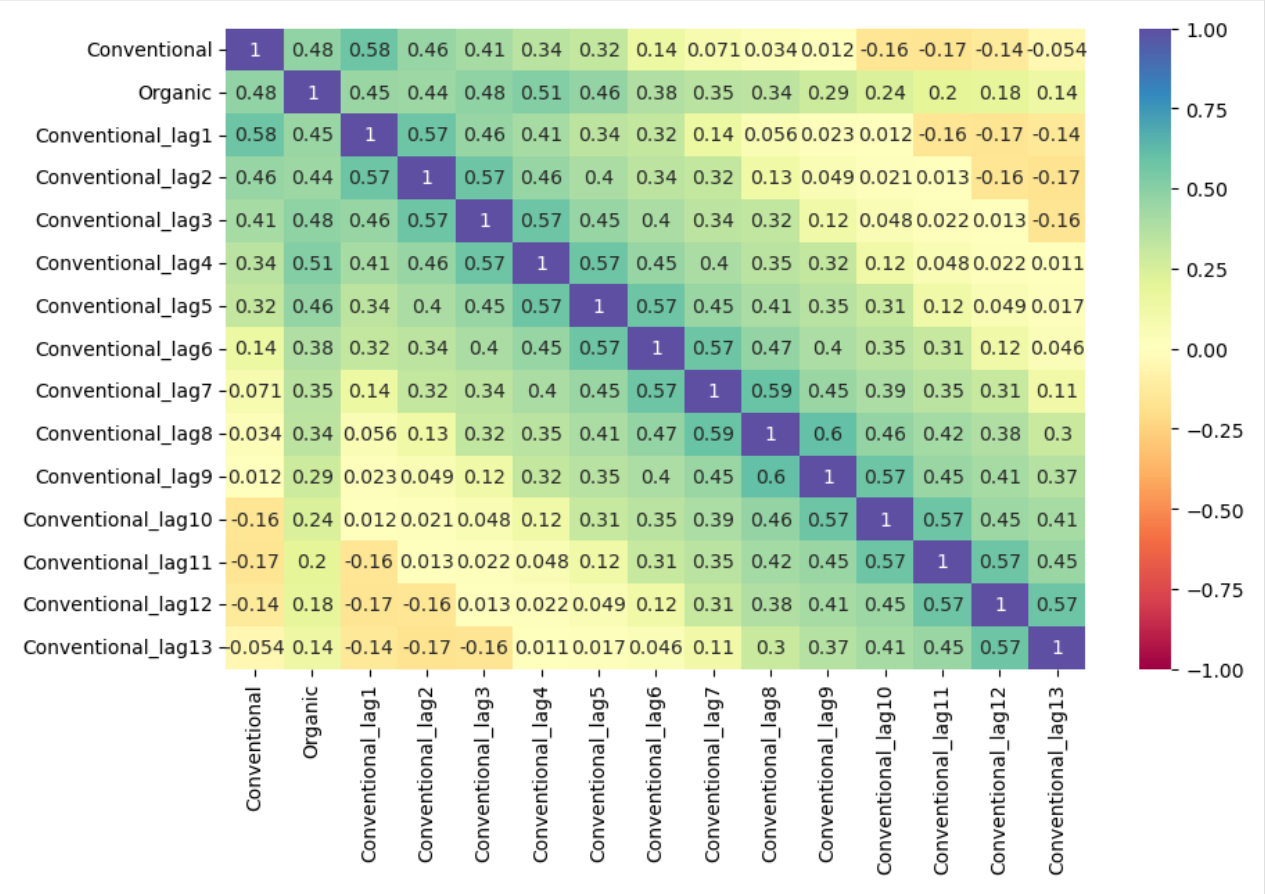
	Conventional	Organic
Conventional	1.0000	0.4802
Organic	0.4802	1.0000

14.4.3 View Series Correlation with each others' lags

[26]:
```
mvf.corr_lags(
    y='Conventional',
    x='Organic',
    lags=13,
    disp='heatmap',
    annot=True,
    vmin=-1,
    vmax=1,
    cmap = 'Spectral',
)
plt.show()
```



[27]:
```
mvf.corr_lags(
    y='Organic',
    x='Conventional',
    lags=13,
    disp='heatmap',
    annot=True,
    vmin=-1,
    vmax=1,
    cmap = 'Spectral',
)
plt.show()
```



14.4.4 Set Optimize On

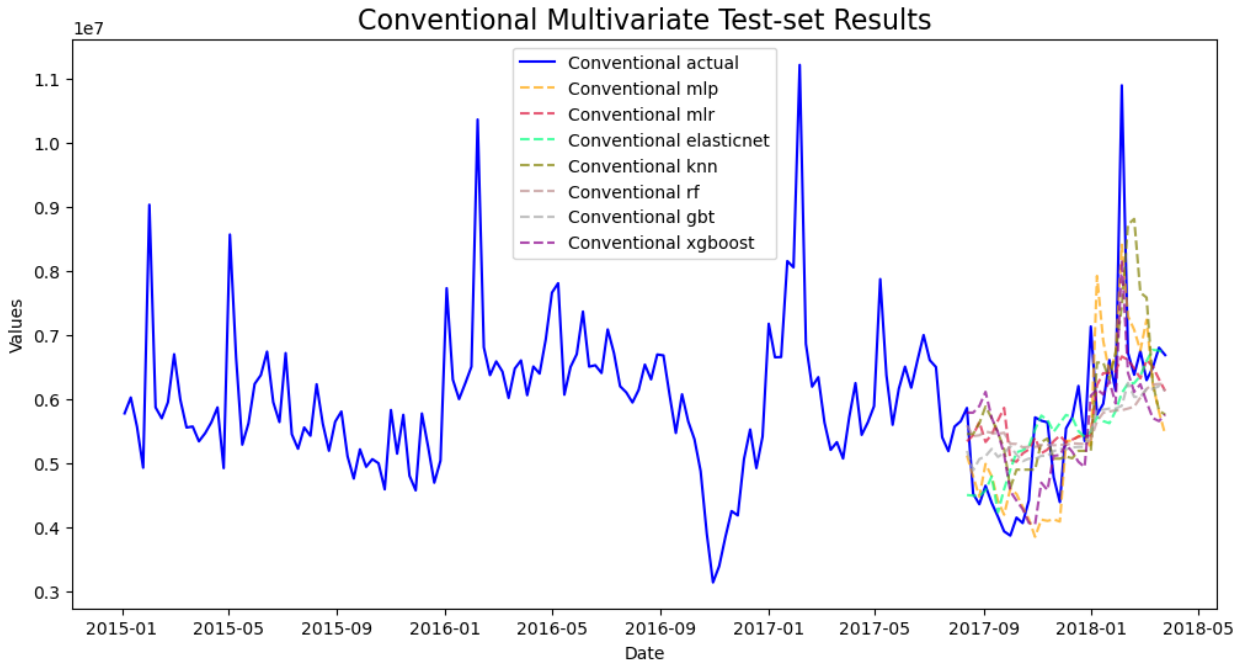
- If predicting one series is more important than predicting the other, you can use this code to let the code know to favor one over the other
- By default, it uses the mean error metrics between all series

[28]:
```
# how to optimize on one series
mvf.set_optimize_on('Organic')
# how to optimize using a weighted avarage
mvf.add_optimizer_func(lambda x: x[0]*.25 + x[1]*.75,'weighted')
mvf.set_optimize_on('weighted')
# how to optimize on the average of both/all series (default)
mvf.set_optimize_on('mean')
```

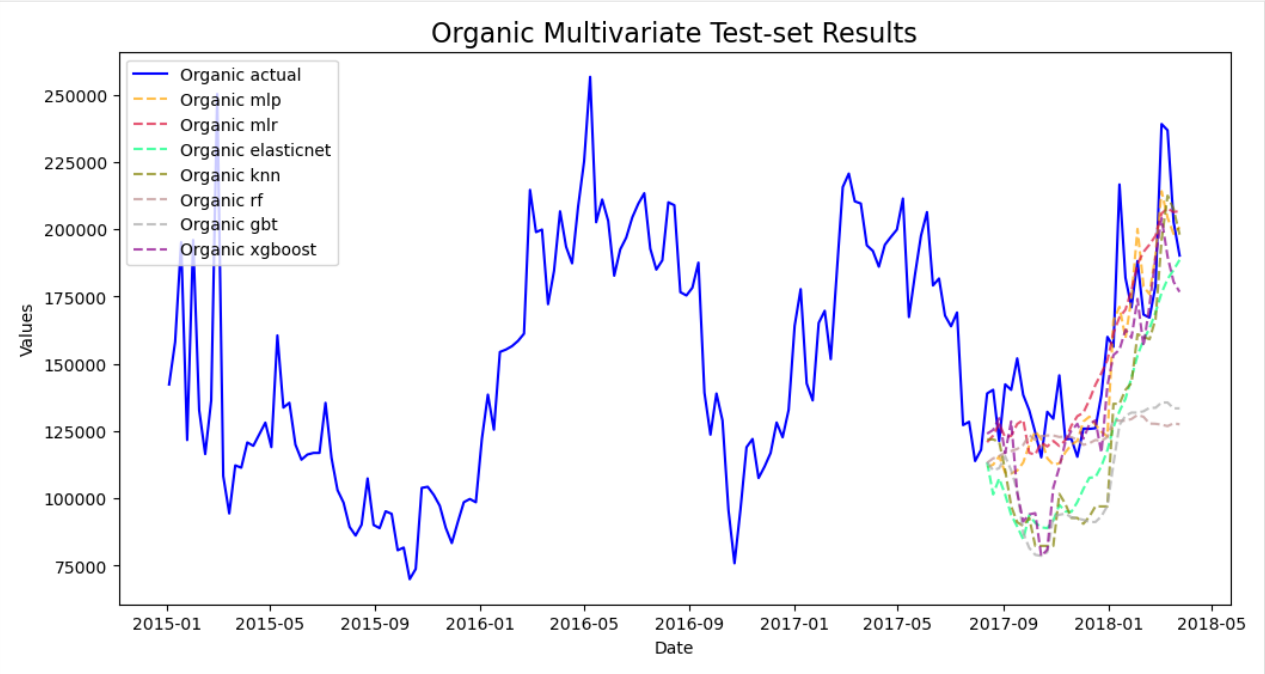
14.4.5 Tune and Test with Selected Models

[29]:
```
mvf.tune_test_forecast(models)
mvf.set_best_model(determine_best_by='TestSetRMSE')
```

[30]:
```
# not plotting both series at the same time because they have significantly different␣
↪scales
mvf.plot_test_set(series='Conventional',put_best_on_top=True)
plt.title('Conventional Multivariate Test-set Results',size=16)
plt.show()
```



[31]:
```
mvf.plot_test_set(series='Organic',put_best_on_top=True)
plt.title('Organic Multivariate Test-set Results',size=16)
plt.show()
```



14.4.6 Export Model Summaries

[32]:
```
pd.options.display.max_colwidth = 100
results = mvf.export('model_summaries')
results[
    [
        'ModelNickname',
        'Series',
        'HyperParams',
        'TestSetRMSE',
        'TestSetR2',
        'InSampleRMSE',
        'InSampleR2',
        'Lags'
    ]
]
```

[32]:
```
   ModelNickname        Series  \
0            mlp  Conventional
1            mlr  Conventional
2     elasticnet  Conventional
3            knn  Conventional
4             rf  Conventional
5            gbt  Conventional
6        xgboost  Conventional
7            mlp       Organic
8            mlr       Organic
9     elasticnet       Organic
10           knn       Organic
11            rf       Organic
```

(continues on next page)

(continued from previous page)

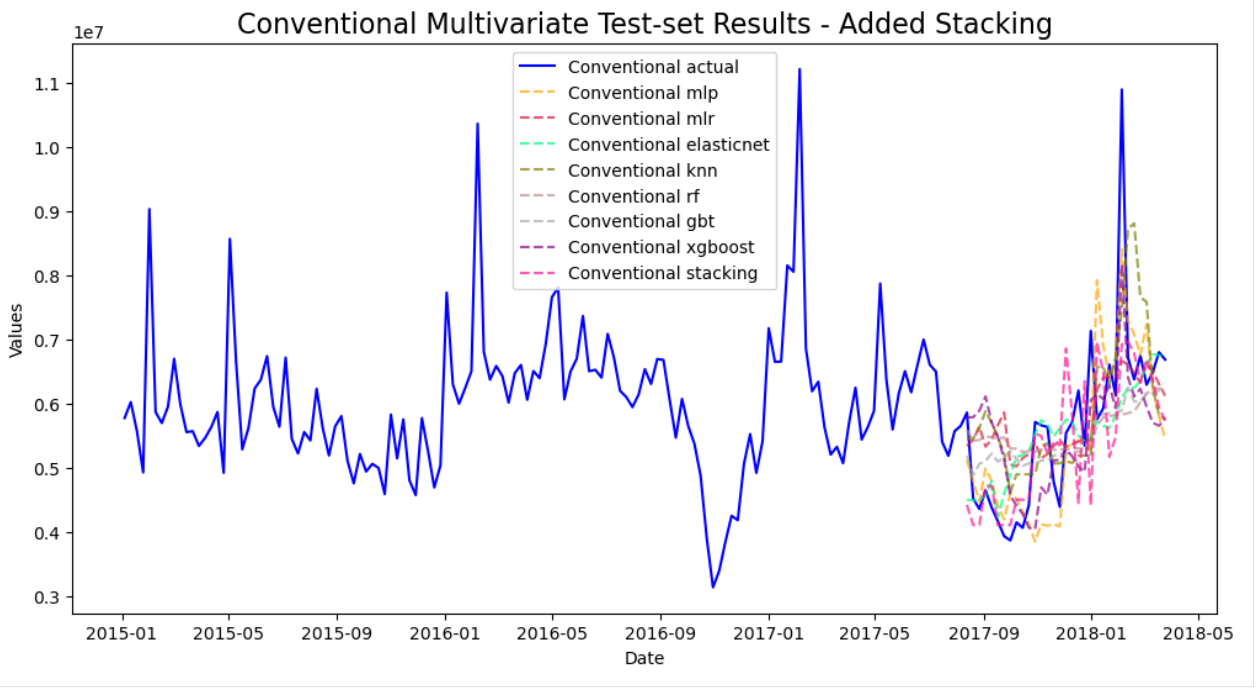
```
12           gbt       Organic
13       xgboost       Organic


                                                                                                                ␣
↪   HyperParams  \
0   {'activation': 'relu', 'hidden_layer_sizes': (25, 25), 'solver': 'lbfgs', 'normalizer
↪': 'minmax'...
1                                                                                                          {
↪'normalizer': None}
2                                                  {'alpha': 0.9, 'l1_ratio': 0.75,
↪'normalizer': None}
3                                                              {'n_neighbors': 4,
↪'weights': 'uniform'}
4                                                                 {'max_depth': 5, 'n_
↪estimators': 100}
5                                                                {'max_depth': 2, 'max_
↪features': None}
6                                                                                              {
↪'max_depth': 2}
7   {'activation': 'relu', 'hidden_layer_sizes': (25, 25), 'solver': 'lbfgs', 'normalizer
↪': 'minmax'...
8                                                                                          {
↪'normalizer': None}
9                                                  {'alpha': 0.9, 'l1_ratio': 0.75,
↪'normalizer': None}
10                                                             {'n_neighbors': 4,
↪'weights': 'uniform'}
11                                                                {'max_depth': 5, 'n_
↪estimators': 100}
12                                                               {'max_depth': 2, 'max_
↪features': None}
13                                                                                             {
↪'max_depth': 2}

    TestSetRMSE  TestSetR2  InSampleRMSE  InSampleR2  Lags
0   976640.7441     0.4776   726613.9497      0.6134     3
1  1090383.4159     0.3488   889221.9216      0.4209     3
2  1053128.2384     0.3926   752345.3345      0.5889    13
3  1178477.0280     0.2394   663362.5001      0.6804    13
4  1202344.6297     0.2082   343671.0745      0.9142    13
5  1124594.8764     0.3073   259436.1165      0.9511    13
6   974696.0984     0.4797   233374.4324      0.9597     1
7    20920.8153     0.6112    16252.8787      0.8544     3
8    17923.9250     0.7146    20380.0266      0.7710     3
9    37629.9902    -0.2580    14968.7668      0.8746    13
10   36796.6649    -0.2029    12551.0264      0.9118    13
11   43547.8368    -0.6848     8941.5674      0.9552    13
12   50265.9563    -1.2447     7325.5184      0.9700    13
13   26986.2883     0.3530     6488.8763      0.9767     1
```

14.4.7 Import a Foreign Sklearn Estimator for Ensemble Modeling

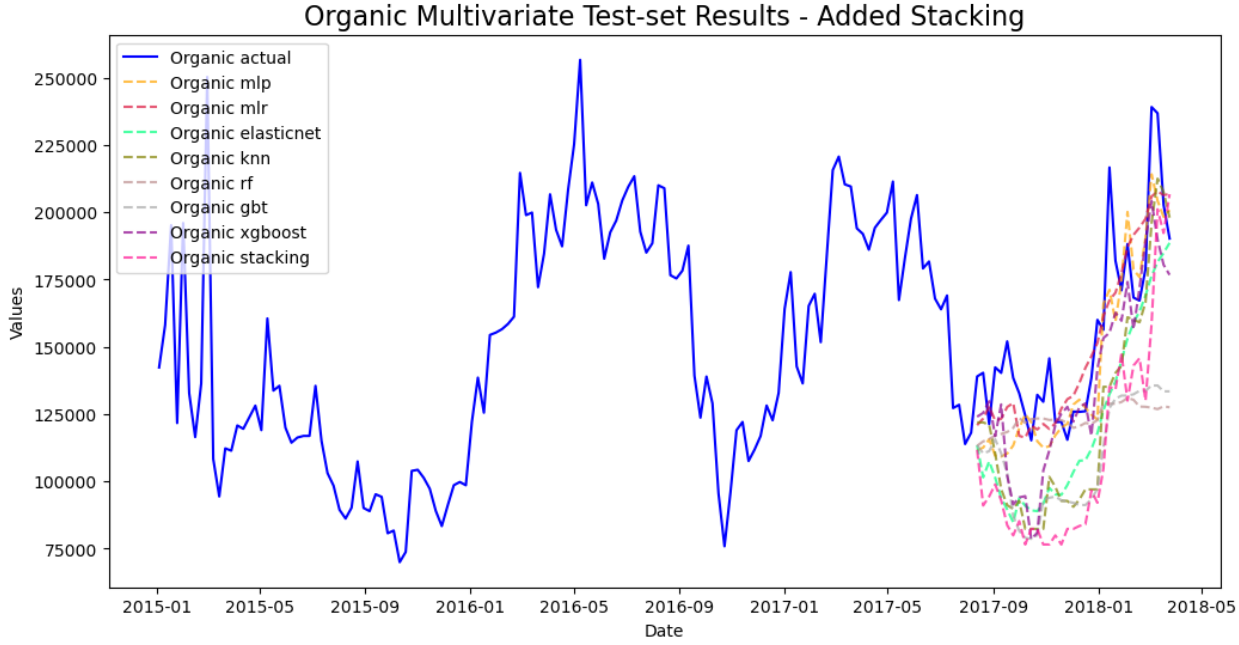
[33]:
```
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
estimators = [
    (
        'mlr',
        LinearRegression()
    ),
    (
        'elasticnet',
        ElasticNet(
            **{
                k:v for k,v in (
                    results.loc[
                        results['ModelNickname'] == 'elasticnet',
                        'HyperParams',
                    ].values[0]
                ).items() if k != 'normalizer'
              }
        )
    ),
    (
        'mlp',
        MLPRegressor(
            **{
                k:v for k,v in (
                    results.loc[
                        results['ModelNickname'] == 'mlp',
                        'HyperParams',
                    ].values[0]
                ).items() if k != 'normalizer'
              }
        )
    )
]

final_estimator = KNeighborsRegressor(
    **{
        k:v for k,v in (
            results.loc[
                results['ModelNickname'] == 'knn',
                'HyperParams',
            ].values[0]
        ).items() if k != 'normalizer'
      }
)
```

[34]:
```
mvf.add_sklearn_estimator(StackingRegressor,'stacking')
mvf.set_estimator('stacking')
mvf.manual_forecast(estimators=estimators,final_estimator=final_estimator,lags=13)
```

[35]:
```
mvf.plot_test_set(series='Conventional',put_best_on_top=True)
plt.title('Conventional Multivariate Test-set Results - Added Stacking',size=16)
plt.show()
```



[36]:
```
mvf.plot_test_set(series='Organic',put_best_on_top=True)
plt.title('Organic Multivariate Test-set Results - Added Stacking',size=16)
plt.show()
```



[37]:
```
mvf.set_best_model(determine_best_by='TestSetRMSE')
results2 = mvf.export('model_summaries')
results2 = results2[
    [
        'ModelNickname',
        'Series',
        'TestSetRMSE',
        'TestSetR2',
        'InSampleRMSE',
        'InSampleR2',
        'Lags',
        'best_model'
    ]
]

results2
```

[37]:

	ModelNickname	Series	TestSetRMSE	TestSetR2	InSampleRMSE	\
0	mlp	Conventional	976640.7441	0.4776	726613.9497	
1	mlr	Conventional	1090383.4159	0.3488	889221.9216	
2	elasticnet	Conventional	1053128.2384	0.3926	752345.3345	
3	knn	Conventional	1178477.0280	0.2394	663362.5001	
4	rf	Conventional	1202344.6297	0.2082	343671.0745	
5	gbt	Conventional	1124594.8764	0.3073	259436.1165	
6	xgboost	Conventional	974696.0984	0.4797	233374.4324	
7	stacking	Conventional	1098442.4396	0.3392	838807.3770	
8	mlp	Organic	20920.8153	0.6112	16252.8787	
9	mlr	Organic	17923.9250	0.7146	20380.0266	
10	elasticnet	Organic	37629.9902	-0.2580	14968.7668	
11	knn	Organic	36796.6649	-0.2029	12551.0264	
12	rf	Organic	43547.8368	-0.6848	8941.5674	

(continues on next page)

(continued from previous page)

```
13           gbt       Organic   50265.9563    -1.2447     7325.5184
14       xgboost       Organic   26986.2883     0.3530     6488.8763
15      stacking       Organic   47928.2496    -1.0407    17511.0377

    InSampleR2  Lags  best_model
0       0.6134     3        True
1       0.4209     3       False
2       0.5889    13       False
3       0.6804    13       False
4       0.9142    13       False
5       0.9511    13       False
6       0.9597     1       False
7       0.4890    13       False
8       0.8544     3        True
9       0.7710     3       False
10      0.8746    13       False
11      0.9118    13       False
12      0.9552    13       False
13      0.9700    13       False
14      0.9767     1       False
15      0.8284    13       False
```

[38]:
```
print('-'*100)
for series in results2['Series'].unique():
    print('multivariate average test MAPE for {}: {:.4f}'.format(series,results2.
↪loc[results2['Series'] == series,'TestSetRMSE'].mean()))
    print('multivariate average test R2 for {}: {:.2f}'.format(series,results2.
↪loc[results2['Series'] == series,'TestSetR2'].mean()))
    print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
↪-----------
multivariate average test MAPE for Conventional: 1087338.4338
multivariate average test R2 for Conventional: 0.35
----------------------------------------------------------------------------------------------------
↪-----------
multivariate average test MAPE for Organic: 35249.9658
multivariate average test R2 for Organic: -0.22
----------------------------------------------------------------------------------------------------
↪-----------
```

14.4.8 Plot Final Forecasts

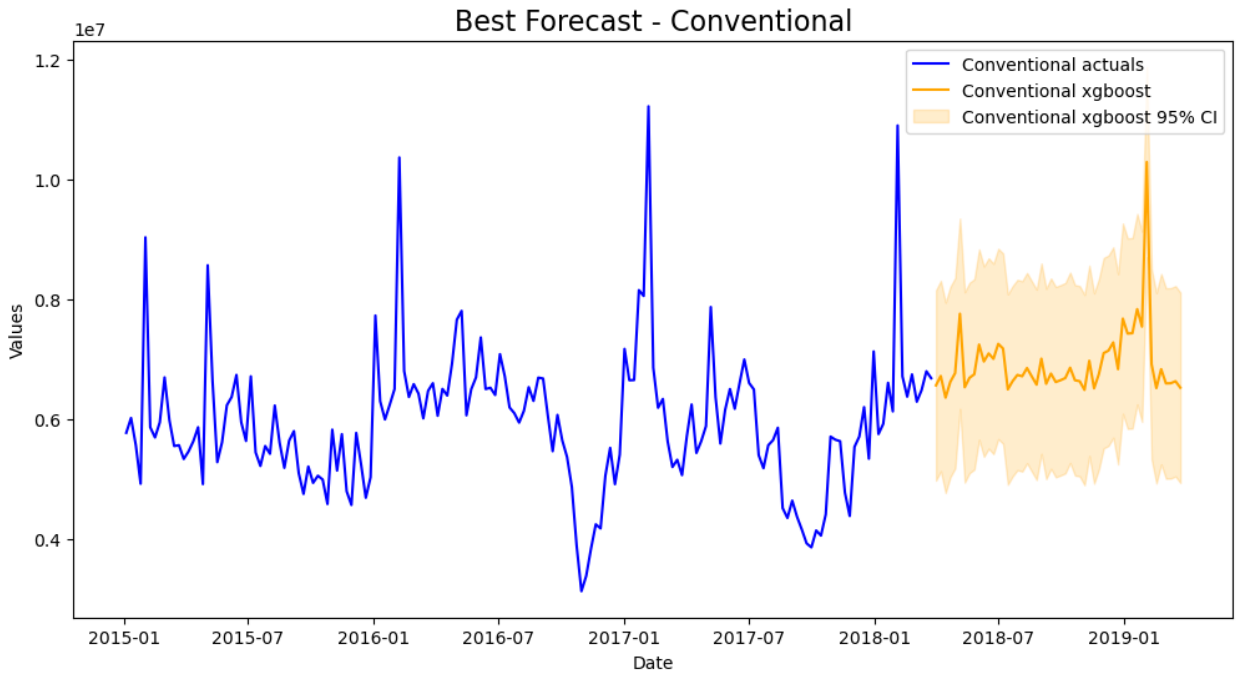
[39]:
```
best_model_con = (
    results2.loc[results2['Series'] == 'Conventional']
    .sort_values('TestSetR2',ascending=False)
    .iloc[0,0]
)
```

[40]:
```
mvf.plot(series='Conventional',models=best_model_con,ci=True)
```

(continues on next page)

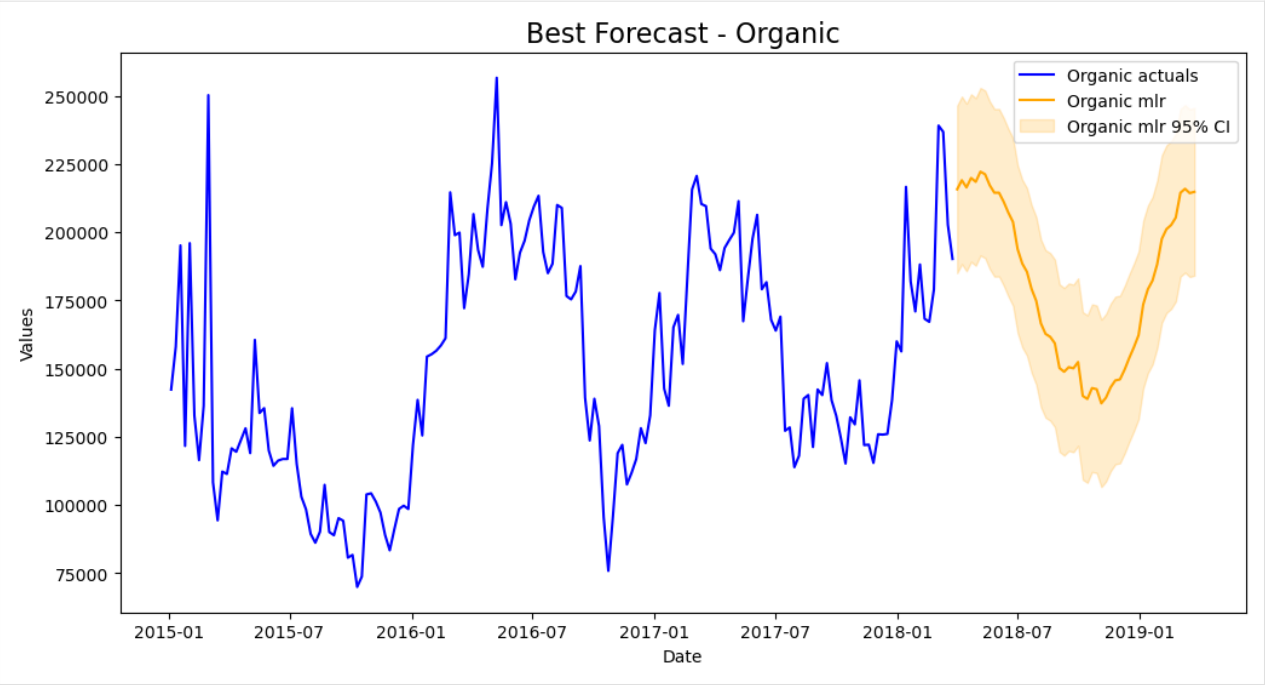
(continued from previous page)

```
plt.title('Best Forecast - Conventional',size=16)
plt.show()
```



[41]:
```
best_model_org = (
    results2.loc[results2['Series'] == 'Organic']
    .sort_values('TestSetR2',ascending=False)
    .iloc[0,0]
)
```

[42]:
```
mvf.plot(series='Organic',models=best_model_org,ci=True)
plt.title('Best Forecast - Organic',size=16)
plt.show()
```



All-in-all, better results on the test-set were obtained by using the multivariate approach.

14.5 Multivariate Backtest

Just like in univariate modeling, we can evaluate the results of this modeling approach using backtesting.

[46]:
```
from scalecast.Pipeline import Transformer, Reverter, MVPipeline
from scalecast.util import backtest_metrics

def mvforecaster(mvf):
    mvf.add_seasonal_regressors('month','quarter',raw=False,sincos=True)
    mvf.tune_test_forecast(
        models=[best_model_con,best_model_org],
    )

pipeline = MVPipeline(
    steps = [('Forecast',mvforecaster)],
    validation_length = 4,
)
```

[47]:
```
backtest_results = pipeline.backtest(
    fcon,
    forg,
    cis=False,
    test_length = 0,
    fcst_length = 52,
    jump_back = 4,
    n_iter = 3,
)
```

[48]:
```
backtest_metrics(
    backtest_results,
    mets=['rmse','r2'],
    names=['Conventional','Organic'],
)
```

[48]:

Series	Model	Metric	Iter0	Iter1	Iter2	Average
Conventional	xgboost	rmse	1147549.7983	1406113.2605	1291947.8525	1281870.3037
		r2	0.0323	-0.4901	-0.2085	-0.2221
	mlr	rmse	1182695.7635	1091062.5392	1004623.8802	1092794.0610
		r2	-0.0278	0.1028	0.2693	0.1148
Organic	xgboost	rmse	38107.4049	48419.8474	50356.9359	45628.0627
		r2	-0.3062	-1.3092	-1.3834	-0.9996
	mlr	rmse	34053.1727	29246.3029	21894.3284	28397.9347
		r2	-0.0430	0.1575	0.5495	0.2213

[ ]:

CHAPTER

FIFTEEN

MULTIVARIATE - BEYOND THE BASICS

This notebook shows multivariate forecasting procedures with scalecast. It requires `0.18.2`. It uses the Avocados dataset.

We will treat this like a demand forecasting problem. We want to know how many total Avocados will be in demand in the next quarter. But since we know demand and price are intricately related, we will use the historical Avocado prices as a predictor of demand.

[1]:
```
import pandas as pd
import numpy as np
from scalecast.Forecaster import Forecaster
from scalecast.MVForecaster import MVForecaster
from scalecast.Pipeline import MVPipeline
from scalecast.util import (
    find_optimal_transformation,
    find_optimal_lag_order,
    break_mv_forecaster,
    backtest_metrics,
    backtest_for_resid_matrix,
    get_backtest_resid_matrix,
    overwrite_forecast_intervals,
)
from scalecast import GridGenerator
```

Read in hyperparameter grids for optimizing models.

[2]:
```
GridGenerator.get_example_grids()
GridGenerator.get_mv_grids()
```

[3]:
```
pd.options.display.max_colwidth = 100
pd.options.display.float_format = '{:,.2f}'.format
```

[4]:
```
# arguments to pass to every Forecaster/MVForecaster object we create
Forecaster_kws = dict(
    test_length = 13,
    validation_length = 13,
    metrics = ['rmse','r2'],
)
```

[5]:
```
# model summary columns to export everytime we check a model's performance
export_cols = ['ModelNickname','HyperParams','TestSetR2','TestSetRMSE']
```

[6]:
```
# read data
data = pd.read_csv('avocado.csv',parse_dates=['Date']).sort_values(['Date'])
# sort appropriately (not doing this could cause issues)
data = data.sort_values(['region','type','Date'])

data.head()
```

[6]:
```
    Unnamed: 0       Date  AveragePrice  Total Volume     4046      4225  \
51          51 2015-01-04          1.22     40,873.28 2,819.50 28,287.42
50          50 2015-01-11          1.24     41,195.08 1,002.85 31,640.34
49          49 2015-01-18          1.17     44,511.28   914.14 31,540.32
48          48 2015-01-25          1.06     45,147.50   941.38 33,196.16
47          47 2015-02-01          0.99     70,873.60 1,353.90 60,017.20

     4770  Total Bags  Small Bags  Large Bags  XLarge Bags          type  \
51  49.90    9,716.46    9,186.93      529.53         0.00  conventional
50 127.12    8,424.77    8,036.04      388.73         0.00  conventional
49 135.77   11,921.05   11,651.09      269.96         0.00  conventional
48 164.14   10,845.82   10,103.35      742.47         0.00  conventional
47 179.32    9,323.18    9,170.82      152.36         0.00  conventional

    year  region
51  2015  Albany
50  2015  Albany
49  2015  Albany
48  2015  Albany
47  2015  Albany
```

[7]:
```
# demand
vol = data.groupby('Date')['Total Volume'].sum()
```

[8]:
```
# price
price = data.groupby('Date')['AveragePrice'].sum()
```

[9]:
```
# one Forecaster object needed for each series we want to predict multivariately
# volume
fvol = Forecaster(
    y = vol,
    current_dates = vol.index,
    future_dates = 13,
    **Forecaster_kws,
)
```

[10]:
```
# price
fprice = Forecaster(
    y = price,
    current_dates = price.index,
    future_dates = 13,
    **Forecaster_kws,
)
```

[11]:
```
# combine Forecaster objects into one MVForecaster object
# all dates will line up and all models will recursively predict values for all series
mvf = MVForecaster(
    fvol,
    fprice,
    names=['volume','price'],
    **Forecaster_kws,
)
```

[12]:
```
mvf
```

[12]:
```
MVForecaster(
    DateStartActuals=2015-01-04T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=169
    N_series=2
    SeriesNames=['volume', 'price']
    ForecastLength=13
    Xvars=[]
    TestLength=13
    ValidationLength=13
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    OptimizeOn=mean
    GridsFile=MVGrids
)
```

15.1 1. Transformations

To make the forecasting task easier, we can transform the data in each Forecaster object before feeding them to the MVForecaster object. The below function will search through many transformations, using out-of-sample testing to score each one. We pass four possible seasonalities to the function (monthly, quarterly, bi-annually, annually) and the results are several seasonal adjustments get selected.

[13]:
```
transformers = []
reverters = []

for name, f in zip(('volume','price'),(fvol,fprice)):
    print(f'\nFinding best transformation for the {name} series.')
    transformer, reverter = find_optimal_transformation(
        f,
        m = [
            4,
            13,
            26,
            52,
        ],
        test_length = 13,
```

(continues on next page)

(continued from previous page)

```
        num_test_sets = 2,
        space_between_sets = 13,
        return_train_only = True,
        verbose = True,
    )
    transformers.append(transformer)
    reverters.append(reverter)
```

```
Finding best transformation for the volume series.
Using mlr model to find the best transformation set on 2 test sets, each 13 in length.
All transformation tries will be evaluated with 4 lags.
Last transformer tried:
[]
Score (rmse): 19481972.64636622
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (rmse): 22085767.144446835
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (rmse): 19630858.294620857
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (rmse): 22320325.279892858
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'})]
Score (rmse): 18763298.437913556
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'})]
Score (rmse): 18061445.02080934
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 26, 'model': 'add'})]
Score (rmse): 18351627.623842016
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'})]
Score (rmse): 15388459.611609437
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', <function␣
↪find_optimal_transformation.<locals>.boxcox_tr at 0x0000022788613D30>, {'lmbda': -0.5}
↪)]
```

(continues on next page)

(continued from previous page)

```
Score (rmse): 15776741.170206662
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', <function␣
↪find_optimal_transformation.<locals>.boxcox_tr at 0x0000022788613D30>, {'lmbda': 0})]
Score (rmse): 15640424.466095788
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('Transform', <function␣
↪find_optimal_transformation.<locals>.boxcox_tr at 0x0000022788613D30>, {'lmbda': 0.5})]
Score (rmse): 15512957.889126703
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 1)]
Score (rmse): 15929820.9564328
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 4)]
Score (rmse): 14324958.982509937
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 4), (
↪'DiffTransform', 13)]
Score (rmse): 18135344.27502767
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 4), (
↪'DiffTransform', 26)]
Score (rmse): 21861866.629635938
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 4), (
↪'DiffTransform', 52)]
Score (rmse): 20808840.990807127
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 4), (
↪'ScaleTransform',)]
Score (rmse): 14324958.982509933
--------------------------------------------------
Last transformer tried:
[('DeseasonTransform', {'m': 4, 'model': 'add'}), ('DeseasonTransform', {'m': 13, 'model
↪': 'add'}), ('DeseasonTransform', {'m': 52, 'model': 'add'}), ('DiffTransform', 4), (
↪'MinMaxTransform',)]
```

(continues on next page)

(continued from previous page)

```
Score (rmse): 14324958.98250994
--------------------------------------------------
Final Selection:
[('DeseasonTransform', {'m': 4, 'model': 'add', 'train_only': True}), ('DeseasonTransform
↪', {'m': 13, 'model': 'add', 'train_only': True}), ('DeseasonTransform', {'m': 52,
↪'model': 'add', 'train_only': True}), ('DiffTransform', 4), ('ScaleTransform', {'train_
↪only': True})]

Finding best transformation for the price series.
Using mlr model to find the best transformation set on 2 test sets, each 13 in length.
All transformation tries will be evaluated with 4 lags.
Last transformer tried:
[]
Score (rmse): 22.25551611050048
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (rmse): 25.65997061765327
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (rmse): 22.148856499520484
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (rmse): 32.75467733406476
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪)]
Score (rmse): 21.72760152488739
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'})]
Score (rmse): 20.055641074156764
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 26,
↪'model': 'add'})]
Score (rmse): 22.020127438895862
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
↪'model': 'add'})]
Score (rmse): 14.604251739058533
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
```

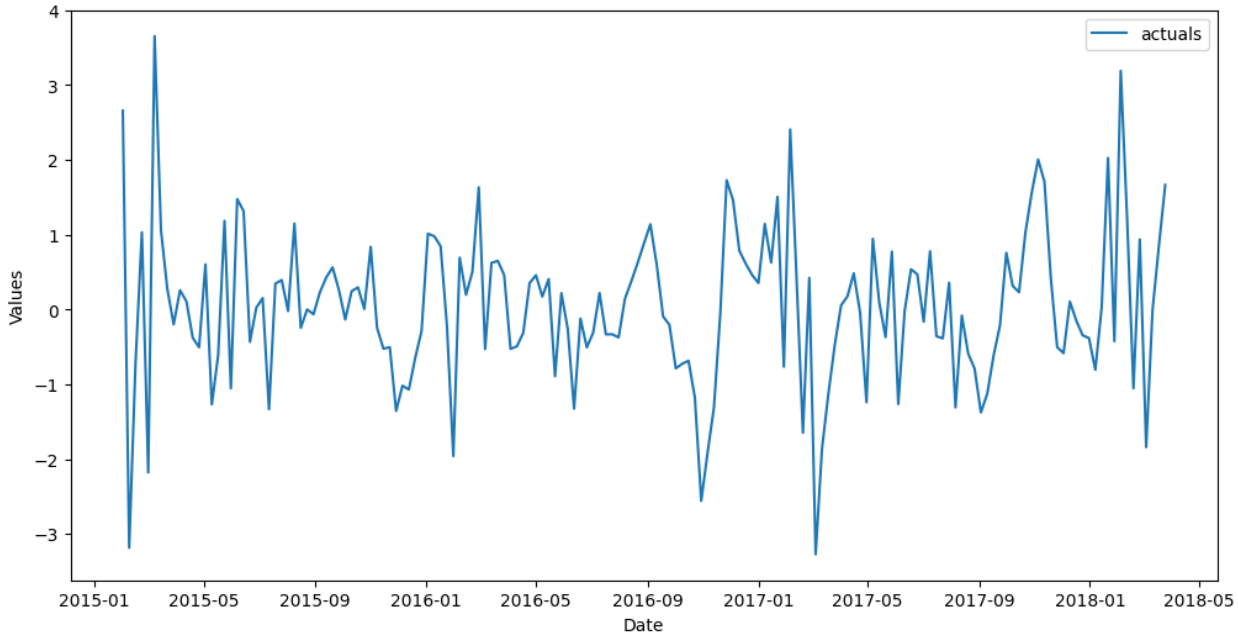
(continues on next page)

(continued from previous page)

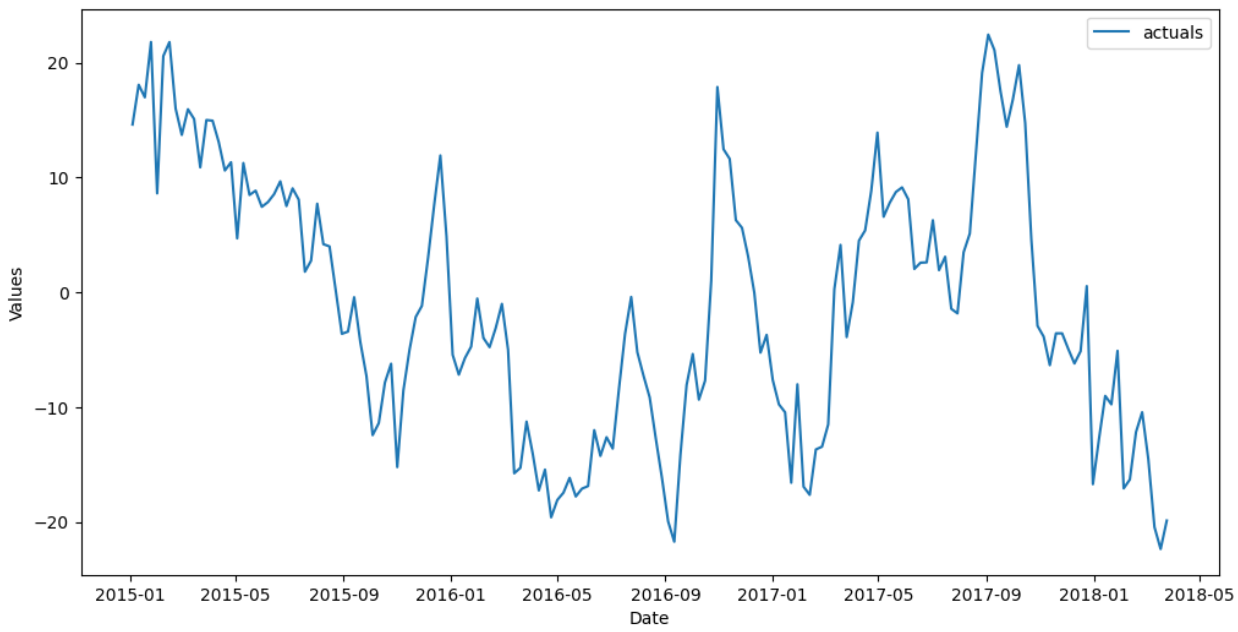
```
↪'model': 'add'}), ('DiffTransform', 1)]
Score (rmse): 18.183007629056675
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
↪'model': 'add'}), ('DiffTransform', 4)]
Score (rmse): 15.96916031713575
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
↪'model': 'add'}), ('DiffTransform', 13)]
Score (rmse): 18.4021660531495
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
↪'model': 'add'}), ('DiffTransform', 26)]
Score (rmse): 25.298723431620186
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
↪'model': 'add'}), ('DiffTransform', 52)]
Score (rmse): 19.452999810002588
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
↪'model': 'add'}), ('ScaleTransform',)]
Score (rmse): 14.604251739058554
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1}), ('DeseasonTransform', {'m': 4, 'model': 'add'}
↪), ('DeseasonTransform', {'m': 13, 'model': 'add'}), ('DeseasonTransform', {'m': 52,
↪'model': 'add'}), ('MinMaxTransform',)]
Score (rmse): 14.604251739058554
--------------------------------------------------
Final Selection:
[('DetrendTransform', {'poly_order': 1, 'train_only': True}), ('DeseasonTransform', {'m':
↪ 4, 'model': 'add', 'train_only': True}), ('DeseasonTransform', {'m': 13, 'model': 'add
↪', 'train_only': True}), ('DeseasonTransform', {'m': 52, 'model': 'add', 'train_only':␣
↪True})]
```

Plot the series after a transformation has been taken.

[14]:
```
fvol1 = transformers[0].fit_transform(fvol)
fvol1.plot();
```



[15]:
```
fprice1 = transformers[1].fit_transform(fprice)
fprice1.plot();
```



Now, combine into an `MVForecaster` object.

[16]:
```
mvf1 = MVForecaster(
    fvol1,
    fprice1,
    names = ['volume','price'],
    **Forecaster_kws,
)
```

15.2 2. Optimal Lag Selection

15.2.1 Method 1: Univariate out-of-sample testing

- The functions below choose the best lags based on what minimizes RMSE on an out-of-sample validation set.

[17]:
```
fvol1.auto_Xvar_select(try_trend=False,try_seasonalities=False)
fvol1.get_regressor_names()
```

[17]:
```
['AR1',
 'AR2',
 'AR3',
 'AR4',
 'AR5',
 'AR6',
 'AR7',
 'AR8',
 'AR9',
 'AR10',
 'AR11',
 'AR12',
 'AR13']
```

[18]:
```
fprice1.auto_Xvar_select(try_trend=False,try_seasonalities=False)
fprice1.get_regressor_names()
```

[18]:
```
['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11']
```

15.2.2 Method 2: Information Criteria Search with VAR

[19]:
```
lag_order_res = find_optimal_lag_order(mvf1,train_only=True)
lag_orders = pd.DataFrame({
    'aic':[lag_order_res.aic],
    'bic':[lag_order_res.bic],
})

lag_orders
```

[19]:
```
   aic  bic
0   12    4
```

15.2.3 Method 3: Multivariate Cross Validation with MLR

[20]:
```
lags = [
    1,
    2,
    3,
    4,
    9,
    10,
```

(continues on next page)

(continued from previous page)

```
    11,
    12,
    13,
    {'volume':13,'price':9},
    [4,9,12,13],
]
```

[21]:
```
grid = dict(
    lags = lags
)
```

[22]:
```
mvf1.set_optimize_on('volume')
```

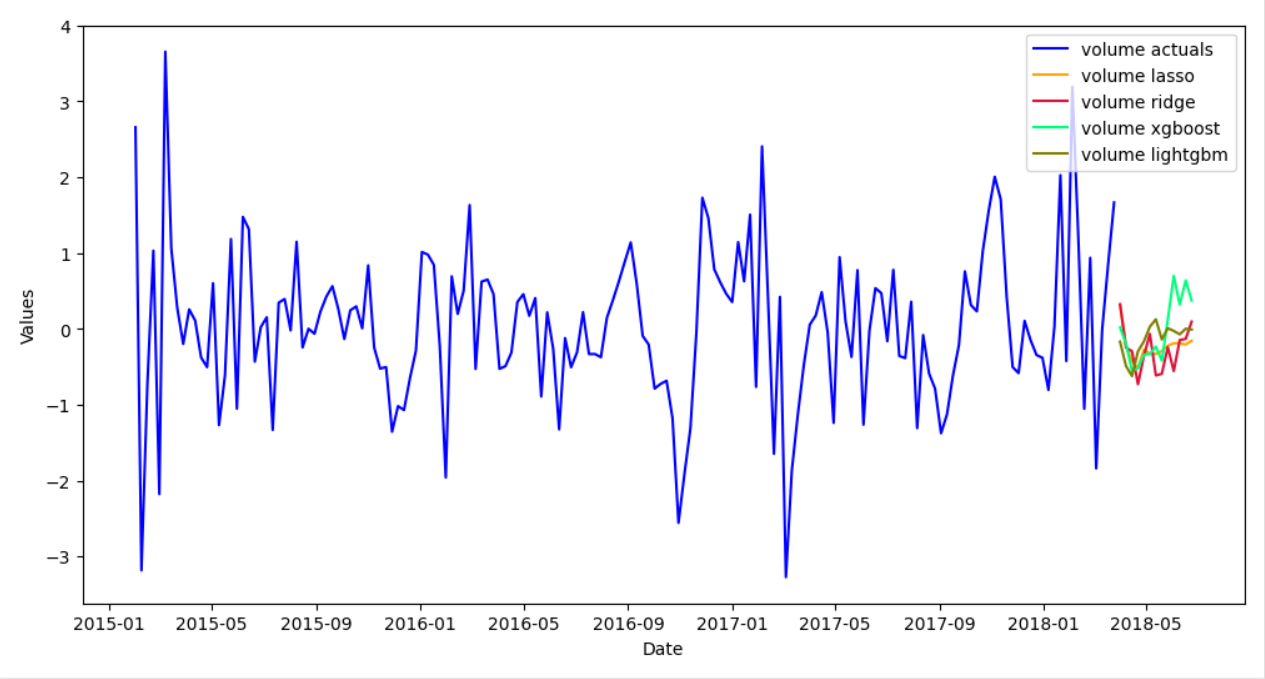
[23]:
```
mvf1.ingest_grid(grid)
mvf1.cross_validate(k=3,test_length=13,verbose = True,dynamic_tuning=True)
```

```
Num hyperparams to try for the mlr model: 11.
Fold 0: Train size: 139 (2015-02-01 00:00:00 - 2017-09-24 00:00:00). Test Size: 13 (2017-
↪10-01 00:00:00 - 2017-12-24 00:00:00).
Fold 1: Train size: 126 (2015-02-01 00:00:00 - 2017-06-25 00:00:00). Test Size: 13 (2017-
↪07-02 00:00:00 - 2017-09-24 00:00:00).
Fold 2: Train size: 113 (2015-02-01 00:00:00 - 2017-03-26 00:00:00). Test Size: 13 (2017-
↪04-02 00:00:00 - 2017-06-25 00:00:00).
Chosen paramaters: {'lags': 10}.
```

15.3 3. Model Optimization with Cross Validation

[24]:
```
def forecaster(mvf):
    mvf.tune_test_forecast(
        ['lasso','ridge','xgboost','lightgbm'],
        cross_validate = True,
        k = 3,
        test_length = 13,
        dynamic_tuning=True,
        limit_grid_size=.2,
        min_grid_size=4,
    )

forecaster(mvf1)
```

[25]:
```
mvf1.plot(series='volume');
```



[26]:
```
mvf1.export('model_summaries',series='volume')[['Series'] + export_cols + ['Lags']].
↪style.set_properties(height = 5)
```

[26]:
```
<pandas.io.formats.style.Styler at 0x22789064880>
```

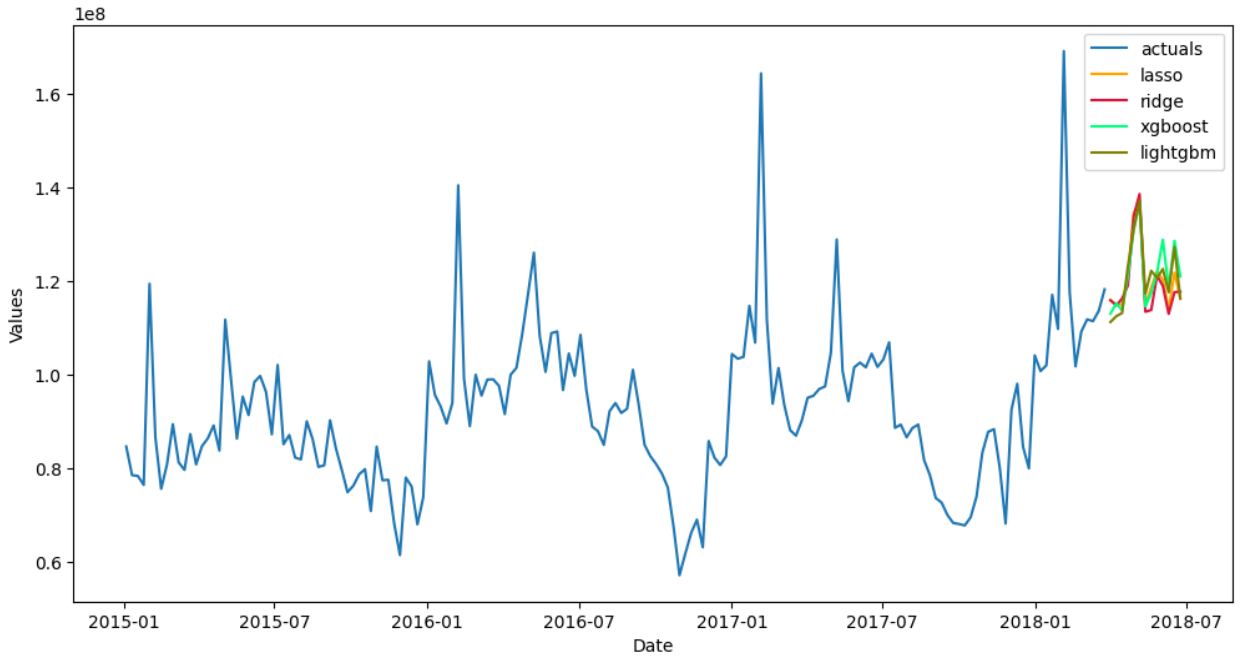
[27]:
```
mvf1
```

[27]:
```
MVForecaster(
    DateStartActuals=2015-02-01T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
    N_actuals=165
    N_series=2
    SeriesNames=['volume', 'price']
    ForecastLength=13
    Xvars=[]
    TestLength=13
    ValidationLength=13
    ValidationMetric=rmse
    ForecastsEvaluated=['lasso', 'ridge', 'xgboost', 'lightgbm']
    CILevel=None
    CurrentEstimator=lightgbm
    OptimizeOn=volume
    GridsFile=MVGrids
)
```

[28]:
```
fvol1, fprice1 = break_mv_forecaster(mvf1)
```

[29]:
```
reverter = reverters[0]
fvol1 = reverter.fit_transform(fvol1)
```

[30]:
```
fvol1.plot();
```



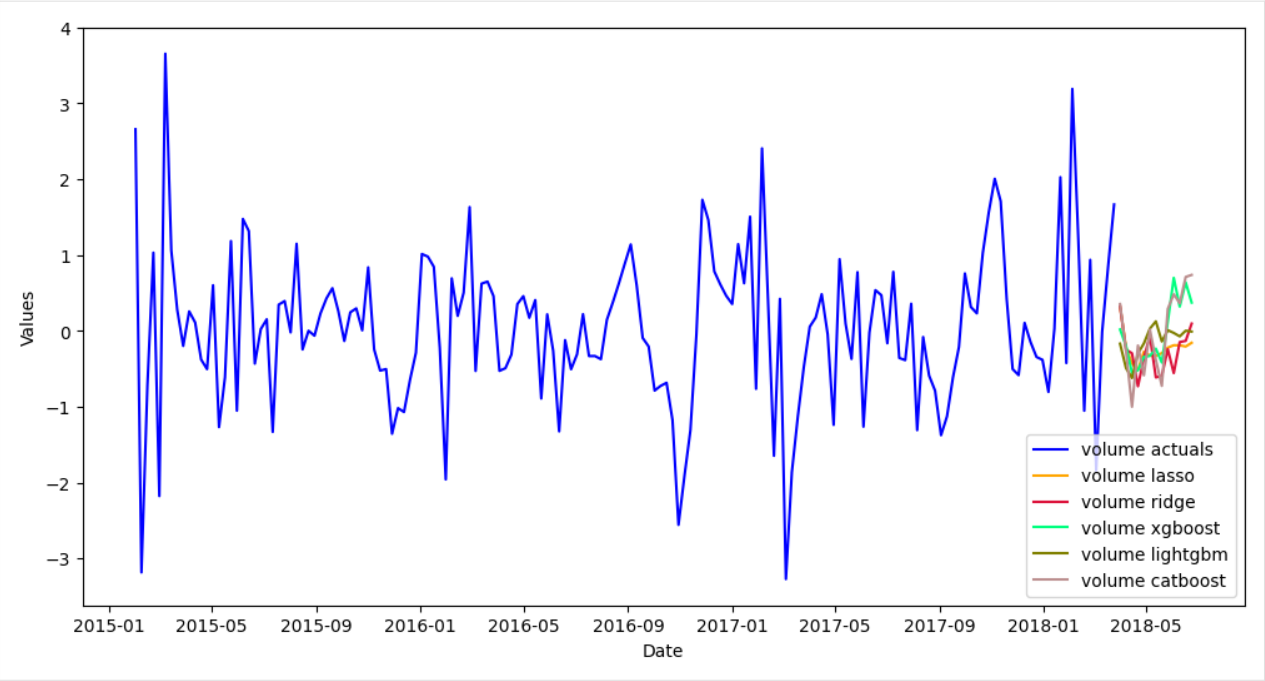
[31]:
```
fvol1.export('model_summaries')[export_cols].style.set_properties(height = 5)
```

[31]:
```
<pandas.io.formats.style.Styler at 0x2278a189670>
```

15.4 4. Model Stacking

[32]:
```
def model_stack(mvf,train_only=False):
    mvf.add_signals(['lasso','ridge','lightgbm','xgboost'],train_only=train_only)
    mvf.set_estimator('catboost')
    mvf.manual_forecast(
        lags = 13,
        verbose = False,
    )

model_stack(mvf1,train_only=True)
```

[33]:
```
mvf1.plot(series='volume');
```



[34]:
```
mvf1.export('model_summaries',series='volume')[['Series'] + export_cols + ['Lags']].
↪style.set_properties(height = 5)
```

[34]:
```
<pandas.io.formats.style.Styler at 0x2278903ed30>
```

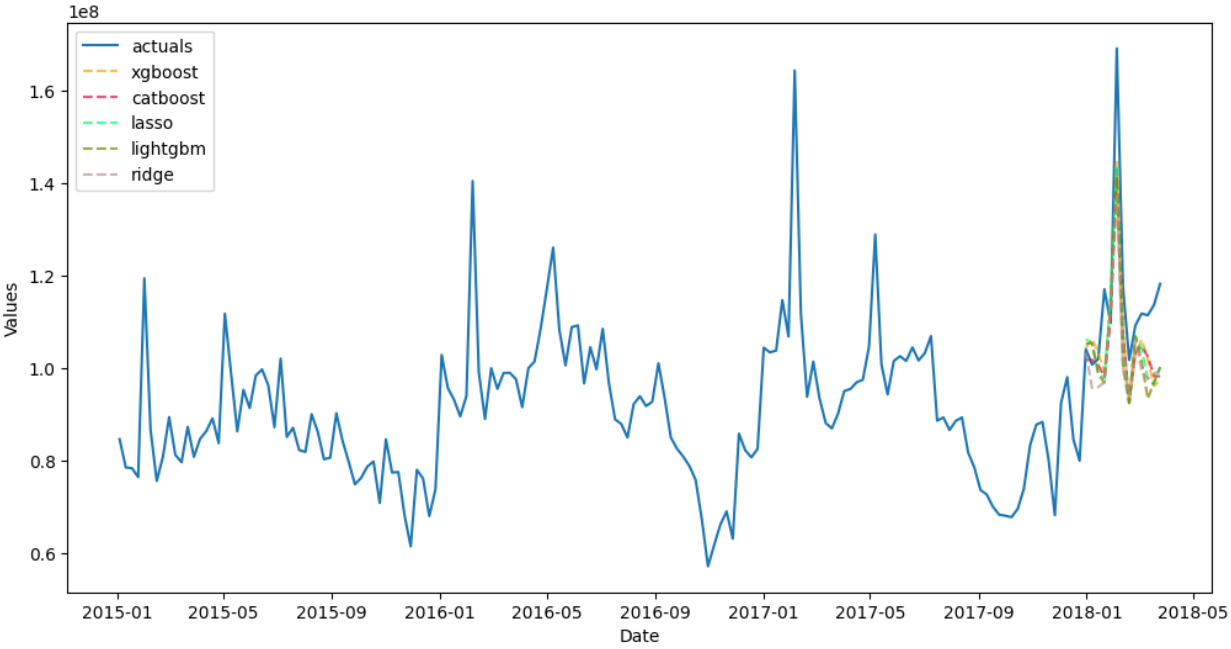
[35]:
```
fvol1, fprice1 = break_mv_forecaster(mvf1)
```

[36]:
```
fvol1 = reverter.fit_transform(fvol1)
```

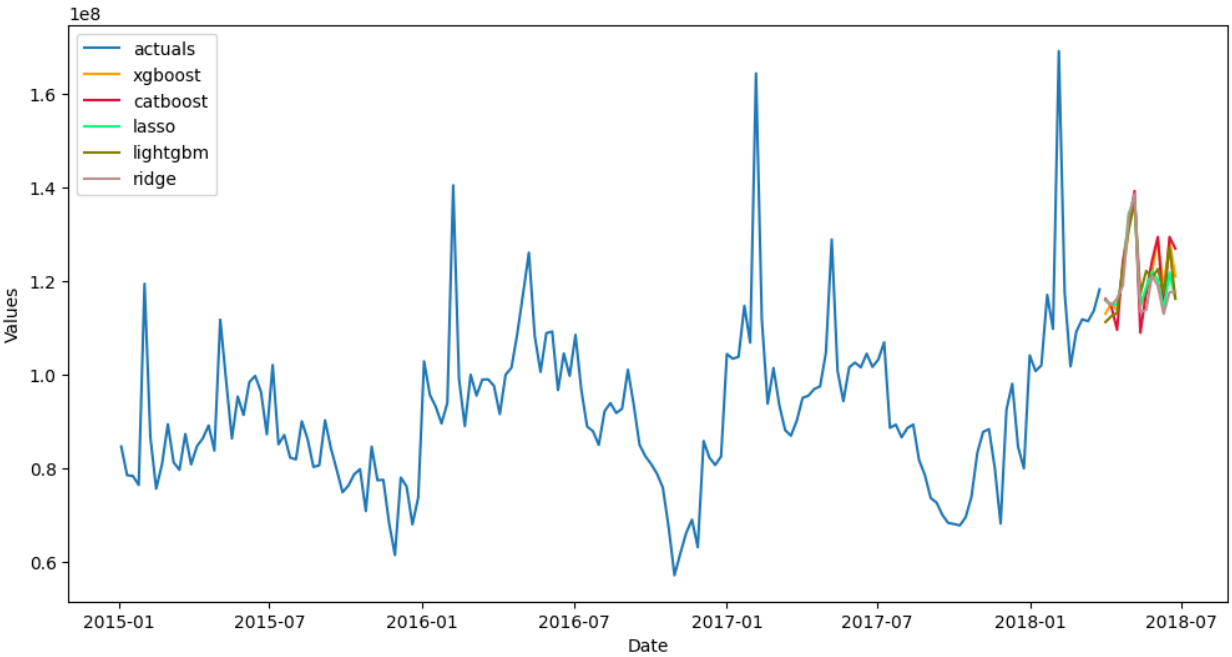
[37]:
```
fvol1.export('model_summaries',determine_best_by='TestSetRMSE')[export_cols].style.set_
↪properties(height = 5)
```

[37]:
```
<pandas.io.formats.style.Styler at 0x2278902d6a0>
```

[38]:
```
fvol1.plot_test_set(order_by='TestSetRMSE');
```



[39]:
```
fvol1.plot(order_by='TestSetRMSE');
```



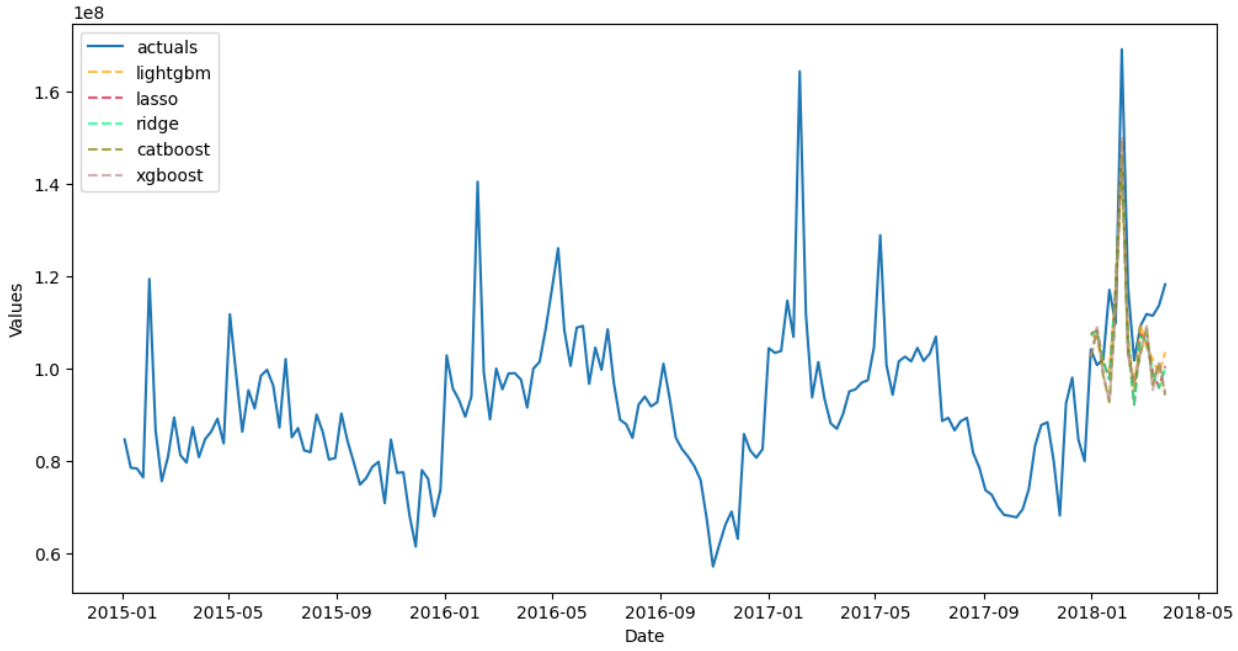
15.5 5. Multivariate Pipelines

[40]:
```
def mvforecaster(mvf,train_only=False):
    forecaster(mvf)
    model_stack(mvf,train_only=train_only)
```

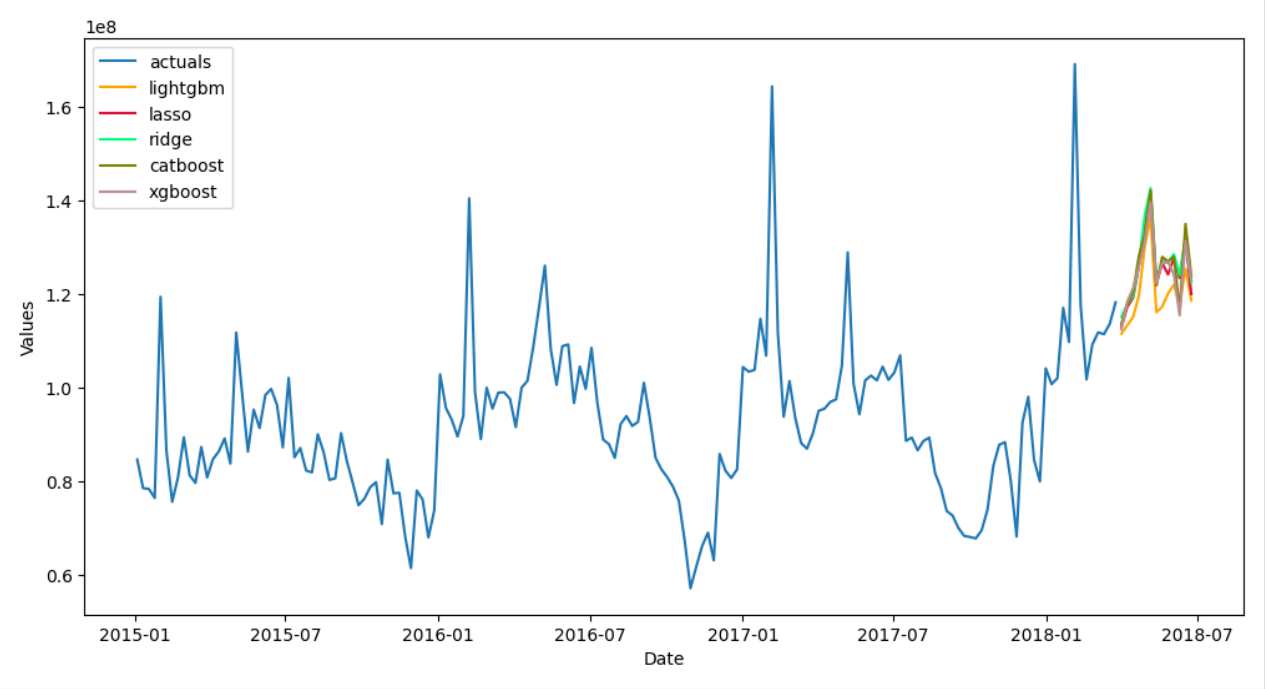
[41]:
```
pipeline = MVPipeline(
    steps = [
        ('Transform',transformers),
        ('Forecast',mvforecaster),
        ('Revert',reverters),
    ],
    **Forecaster_kws,
)
```

[42]:
```
fvol1, fprice1 = pipeline.fit_predict(fvol,fprice,train_only=True)
```

[43]:
```
fvol1.plot_test_set(order_by='TestSetRMSE');
```



[44]:
```
fvol1.plot(order_by='TestSetRMSE');
```



[45]:
```
fvol1.export('model_summaries',determine_best_by='TestSetRMSE')[export_cols].style.set_
↪properties(height = 5)
```

[45]:
```
<pandas.io.formats.style.Styler at 0x2278a1acd90>
```

15.6 6. Backtesting

[46]:
```
backtest_results = pipeline.backtest(
    fvol,
    fprice,
    n_iter = 4,
    fcst_length = 13,
    test_length = 0,
    jump_back = 13,
)
```

[47]:
```
backtest_metrics(
    backtest_results[:1], # volume only
    mets=['rmse','mae','r2','bias'],
    #names=['volume','price'],
)
```

[47]:

Model	Metric	Iter0	Iter1	Iter2	Iter3
lasso	rmse	12,676,035.50	19,334,820.60	12,408,199.56	6,865,555.60
	mae	10,622,259.32	16,818,913.78	10,196,006.95	4,890,701.26
	r2	0.44	-2.98	-0.17	0.34
	bias	-103,321,272.45	-216,135,991.11	106,453,214.29	7,242,071.64
ridge	rmse	13,245,785.08	19,757,231.61	12,581,587.51	8,092,421.06

(continues on next page)

(continued from previous page)

```
         mae      10,864,770.69   17,175,778.82  10,362,478.27   6,239,668.77
         r2                0.38           -3.16          -0.20           0.09
         bias   -119,334,285.17 -221,927,247.43 109,823,042.50  55,636,823.14
xgboost  rmse     19,261,511.73   15,233,136.06  15,781,395.08   6,583,385.75
         mae      15,981,374.97   13,767,893.53  13,479,663.34   5,216,355.57
         r2               -0.30           -1.47          -0.89           0.40
         bias   -103,418,980.41 -151,259,604.70 155,235,475.90 -16,515,829.46
lightgbm rmse     11,239,291.40   17,262,898.64  14,840,433.88   7,289,722.74
         mae       9,087,987.10   15,146,134.37  12,373,711.83   5,735,222.10
         r2                0.56           -2.17          -0.67           0.26
         bias    -86,731,196.07 -189,464,392.96 140,926,488.55  43,025,640.10
catboost rmse     17,455,804.71   14,955,271.67  16,116,336.26   6,315,491.61
         mae      14,805,029.65   13,567,739.76  13,603,593.10   5,036,532.77
         r2               -0.07           -1.38          -0.97           0.45
         bias   -108,026,362.31 -146,926,722.77 162,948,970.47  24,860,226.70

                        Average
Model    Metric
lasso    rmse     12,821,152.81
         mae      10,631,970.33
         r2               -0.59
         bias   -51,440,494.41
ridge    rmse     13,419,256.32
         mae      11,160,674.14
         r2               -0.72
         bias   -43,950,416.74
xgboost  rmse     14,214,857.16
         mae      12,111,321.85
         r2               -0.57
         bias   -28,989,734.67
lightgbm rmse     12,658,086.67
         mae      10,585,763.85
         r2               -0.51
         bias   -23,060,865.10
catboost rmse     13,710,726.06
         mae      11,753,223.82
         r2               -0.50
         bias   -16,785,971.98
```

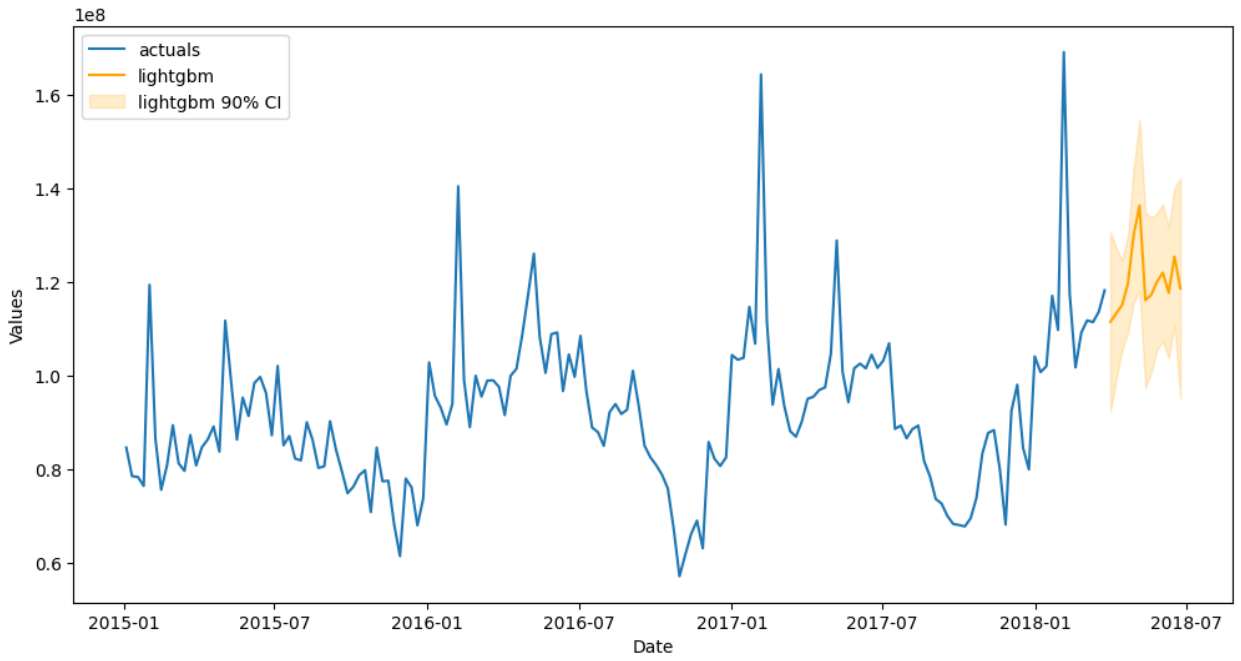
15.7 7. Dynamic Intervals

[48]:
```
backtest_results = backtest_for_resid_matrix(
    fvol,
    fprice,
    pipeline = pipeline,
    alpha = 0.1, # 90% intervals
)
```

[49]:
```
backtest_resid_matrix = get_backtest_resid_matrix(backtest_results)
```

[50]:
```
overwrite_forecast_intervals(
    fvol1,
    fprice1,
    backtest_resid_matrix=backtest_resid_matrix,
    alpha=0.1,
)
```

[51]:
```
fvol1.plot(models='top_1',order_by='TestSetRMSE',ci=True);
```



15.8 8. LSTM Modeling

[52]:
```
fvol1 = transformers[0].fit_transform(fvol1)
fprice1 = transformers[1].fit_transform(fprice1)
```

[53]:
```
fvol1.add_ar_terms(13)
```

[54]:
```
fvol1.set_estimator('rnn')
fvol1.tune()
fvol1.auto_forecast(call_me='lstm_uv')
```

[55]:
```
fvol1.add_series(fprice1.y,called='price')
fvol1.add_lagged_terms('price',lags=13,drop=True)
fvol1
```

[55]:
```
Forecaster(
    DateStartActuals=2015-02-01T00:00:00.000000000
    DateEndActuals=2018-03-25T00:00:00.000000000
    Freq=W-SUN
```

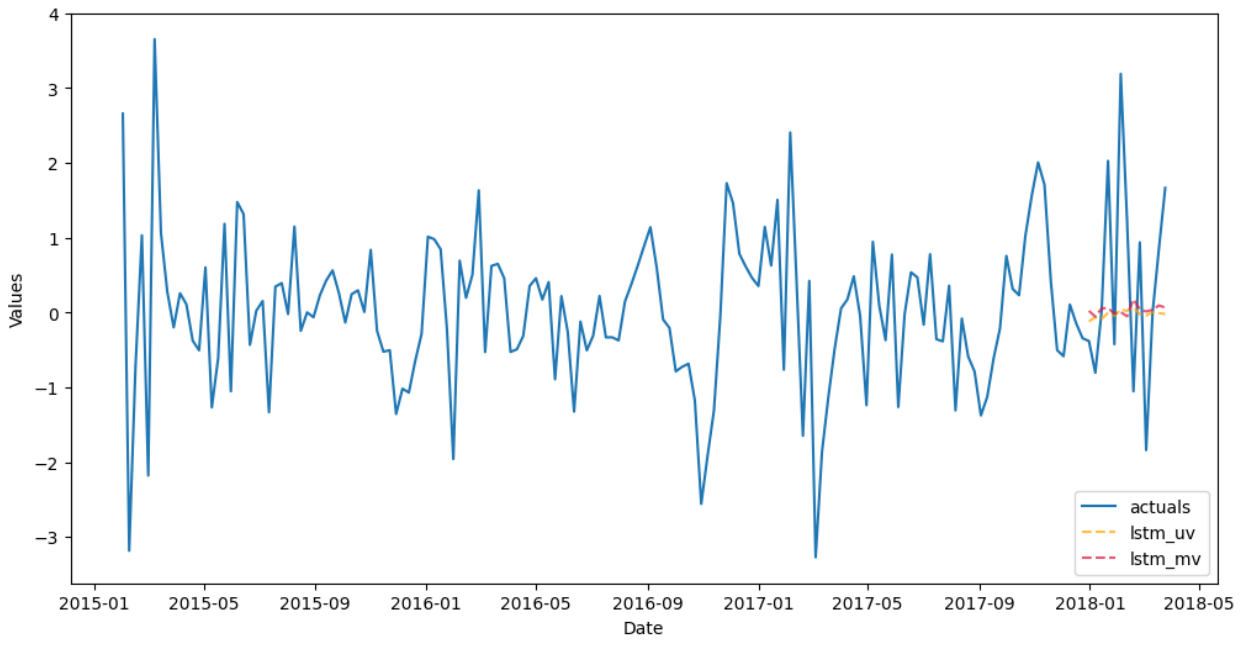
(continues on next page)

(continued from previous page)

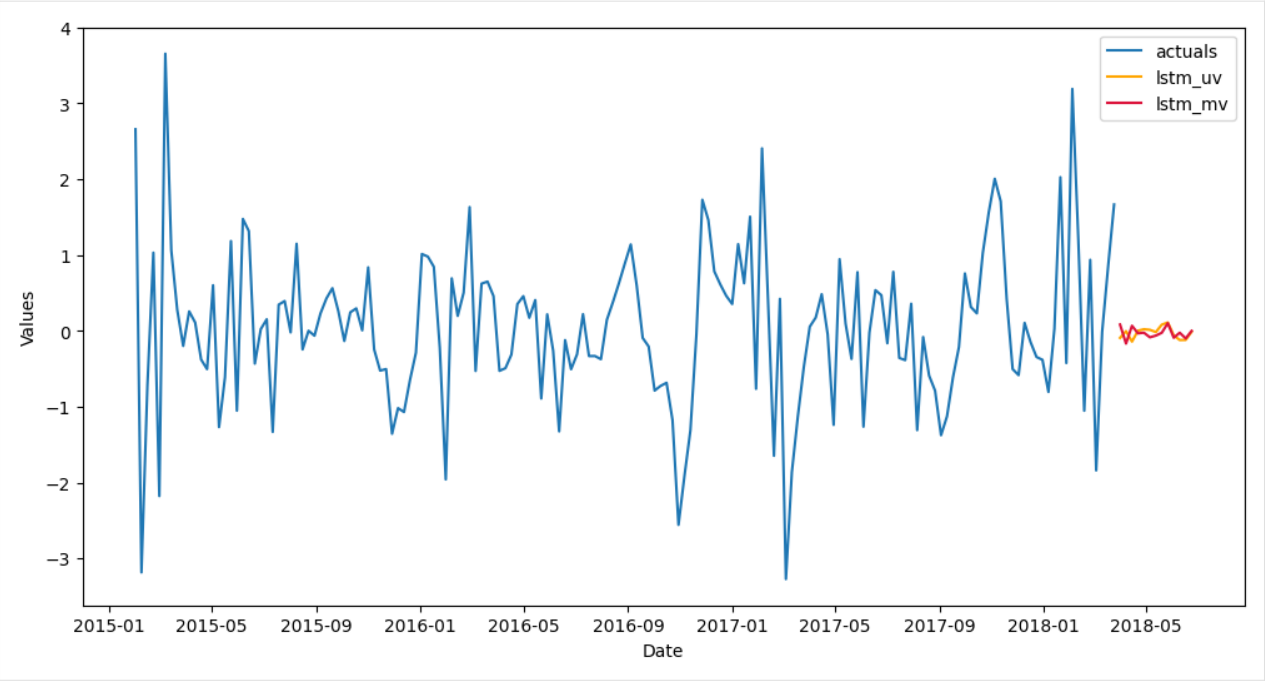
```
    N_actuals=165
    ForecastLength=13
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11',
↪ 'AR12', 'AR13', 'pricelag_1', 'pricelag_2', 'pricelag_3', 'pricelag_4', 'pricelag_5',
↪'pricelag_6', 'pricelag_7', 'pricelag_8', 'pricelag_9', 'pricelag_10', 'pricelag_11',
↪'pricelag_12', 'pricelag_13']
    TestLength=13
    ValidationMetric=rmse
    ForecastsEvaluated=['lasso', 'ridge', 'xgboost', 'lightgbm', 'catboost', 'lstm_uv']
    CILevel=None
    CurrentEstimator=rnn
    GridsFile=Grids
)
```

[56]:
```
fvol1.tune()
fvol1.auto_forecast(call_me='lstm_mv')
```

[57]:
```
fvol1.plot_test_set(models=['lstm_uv','lstm_mv']);
```



[58]:
```
fvol1.plot(models=['lstm_uv','lstm_mv']);
```

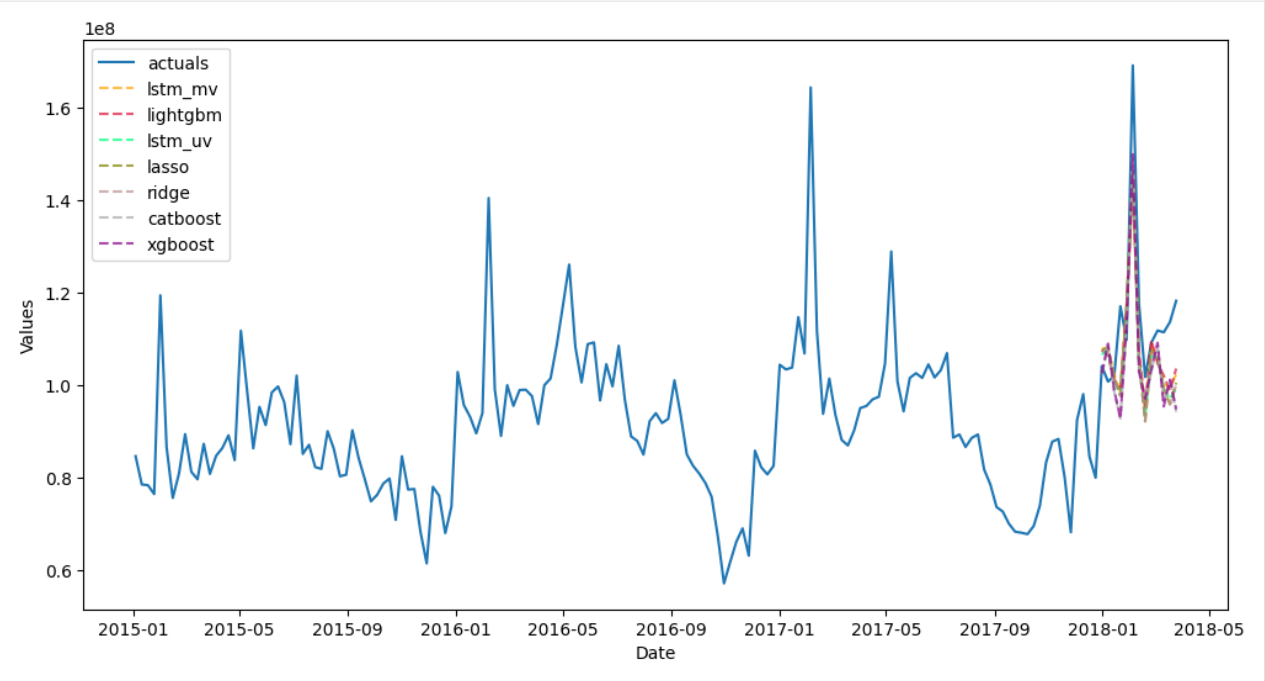


[59]:
```
fvol1 = reverters[0].fit_transform(fvol1,exclude_models=['lightgbm','lasso','ridge',
↪'xgboost','catboost'])
fprice1 = reverters[1].fit_transform(fprice1,exclude_models=['lightgbm','lasso','ridge',
↪'xgboost','catboost'])
```

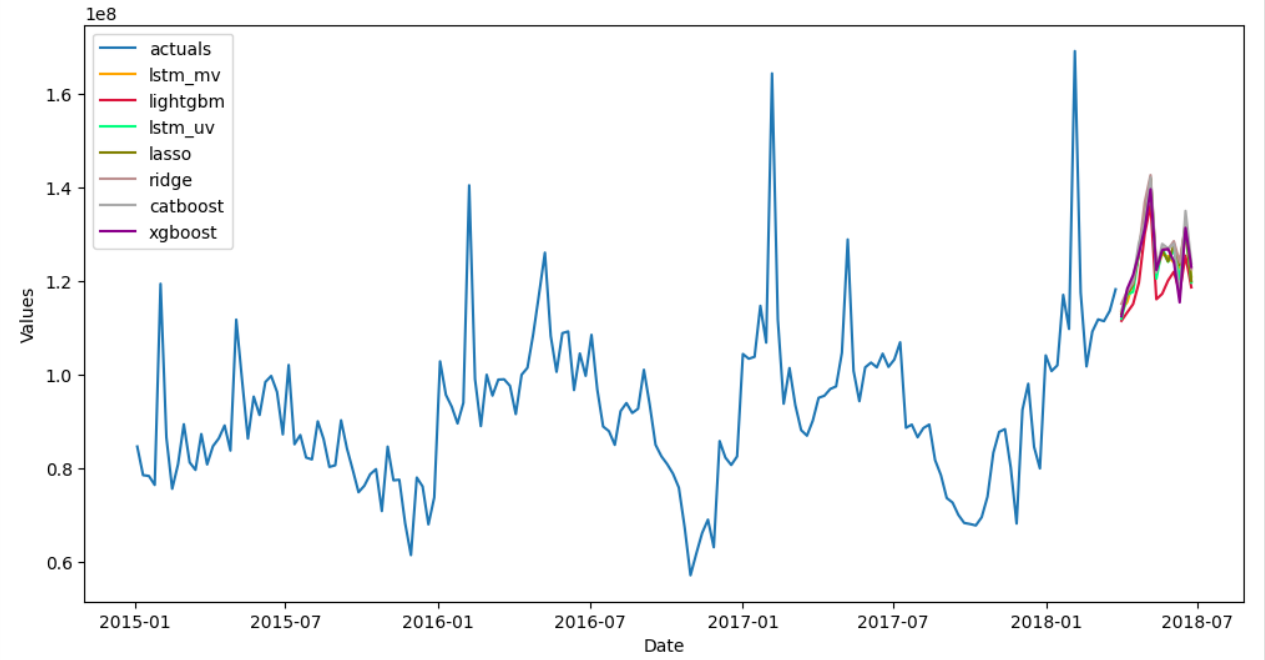
[60]:
```
ms = fvol1.export('model_summaries')
ms = ms[export_cols]
ms.style.set_properties(height = 5)
```

[60]:
```
<pandas.io.formats.style.Styler at 0x2279fc467f0>
```

[61]:
```
fvol1.plot_test_set(order_by = 'TestSetRMSE');
```



[62]:
```
fvol1.plot(order_by = 'TestSetRMSE');
```



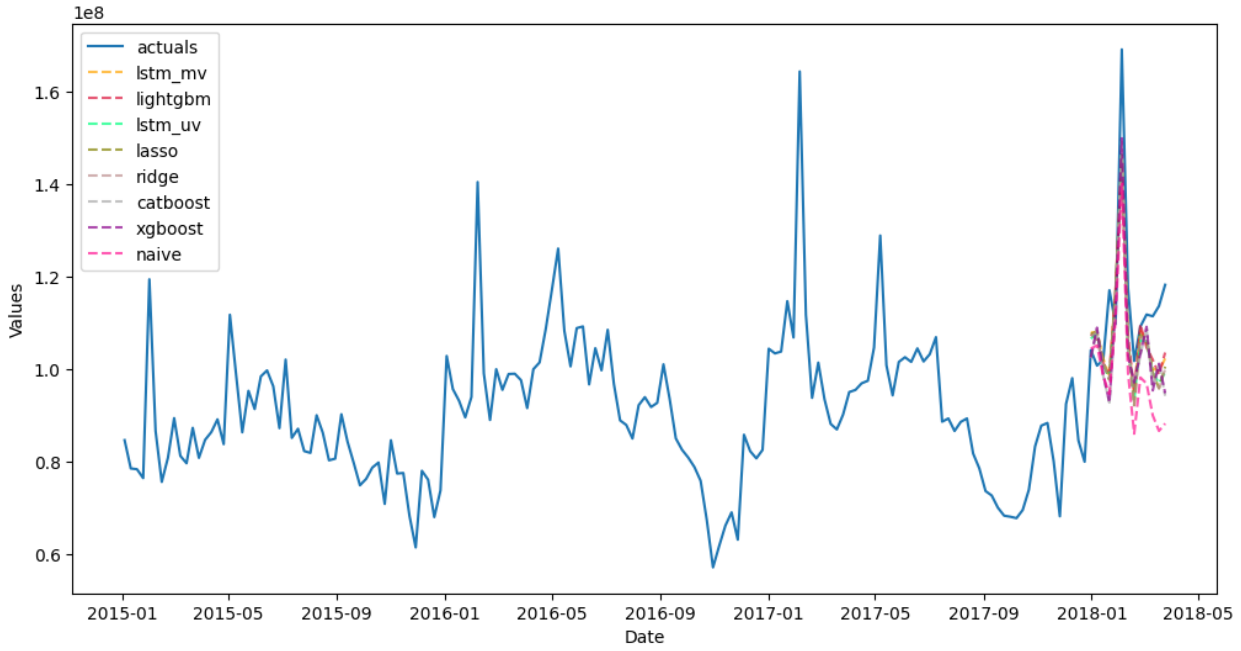
15.9 9. Benchmarking against Naive Model

[63]:
```
fvol1 = transformers[0].fit_transform(fvol1)
fvol1.set_estimator('naive')
fvol1.manual_forecast()
fvol1 = reverters[0].fit_transform(fvol1,exclude_models=['lightgbm','lasso','ridge',
↪'xgboost','catboost','lstm_uv','lstm_mv'])
```

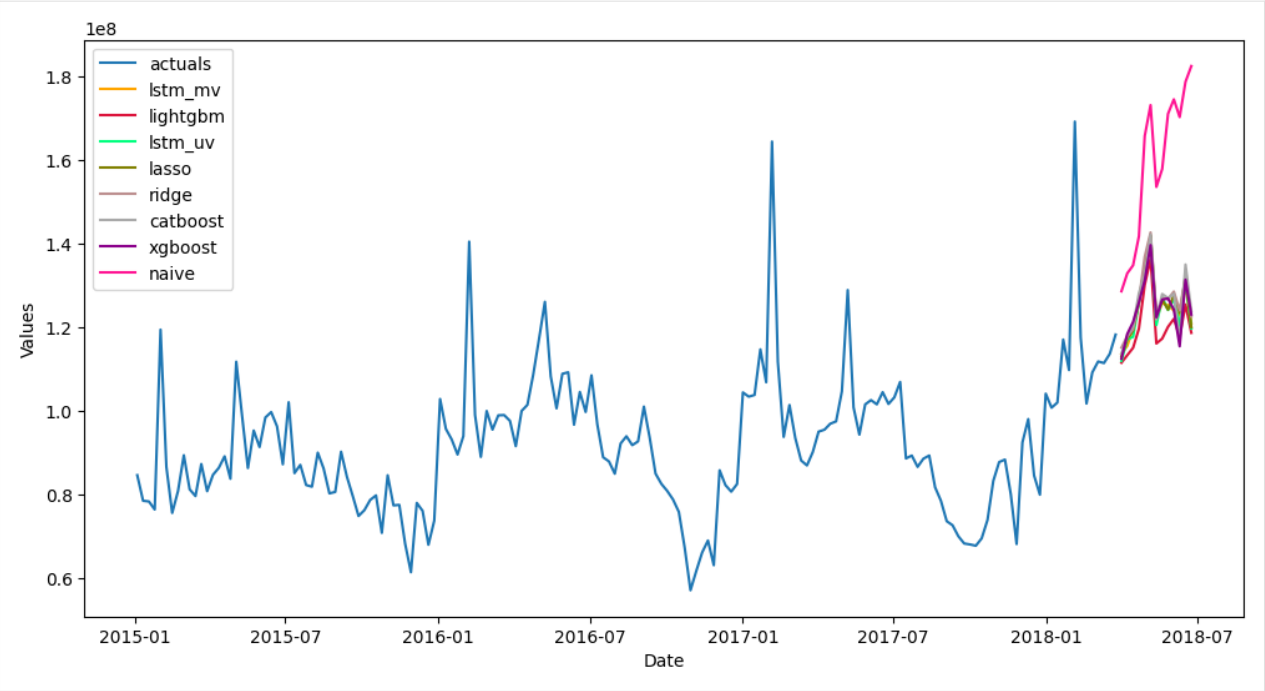
[67]:
```
ms = fvol1.export('model_summaries',determine_best_by='TestSetRMSE')
ms = ms[export_cols]
ms.style.set_properties(height = 5)
```

[67]:
```
<pandas.io.formats.style.Styler at 0x2278a239e20>
```

[65]:
```
fvol1.plot_test_set(order_by = 'TestSetRMSE');
```



[66]:
```
fvol1.plot(order_by = 'TestSetRMSE');
```


1e8
actuals
lstm_mv
lightgbm
lstm_uv
lasso
ridge
catboost
xgboost
naive
Values
1.8
1.6
1.4
1.2
1.0
0.8
0.6
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
Date

CHAPTER

SIXTEEN

RNN AND LSTM

This notebook demonstrates the capabilities of the RNN model in scalecast.

- Download data: https://www.kaggle.com/robervalt/sunspots
- Requirements for this notebook:
- `pip install tensorflow`
- `!pip install tqdm`
- `!pip install ipython`
- `!pip install ipywidgets`
- `!jupyter nbextension enable --py widgetsnbextension`
- if using Jupyter Lab: `!jupyter labextension install @jupyter-widgets/jupyterlab-manager`

[1]:
```
import pandas as pd
import numpy as np
import pandas_datareader as pdr
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm.notebook import tqdm
from dateutil.relativedelta import relativedelta
from scalecast.Forecaster import Forecaster
```

[2]:
```
df = pd.read_csv('Sunspots.csv',index_col=0,names=['Date','Target'],header=0)
f = Forecaster(
    y=df['Target'],
    current_dates=df['Date'],
    test_length = 240,
    future_dates = 240,
    cis = True,
)
f
```

[2]:
```
Forecaster(
    DateStartActuals=1749-01-31T00:00:00.000000000
    DateEndActuals=2021-01-31T00:00:00.000000000
    Freq=M
    N_actuals=3265
    ForecastLength=240
    Xvars=[]
```

(continues on next page)

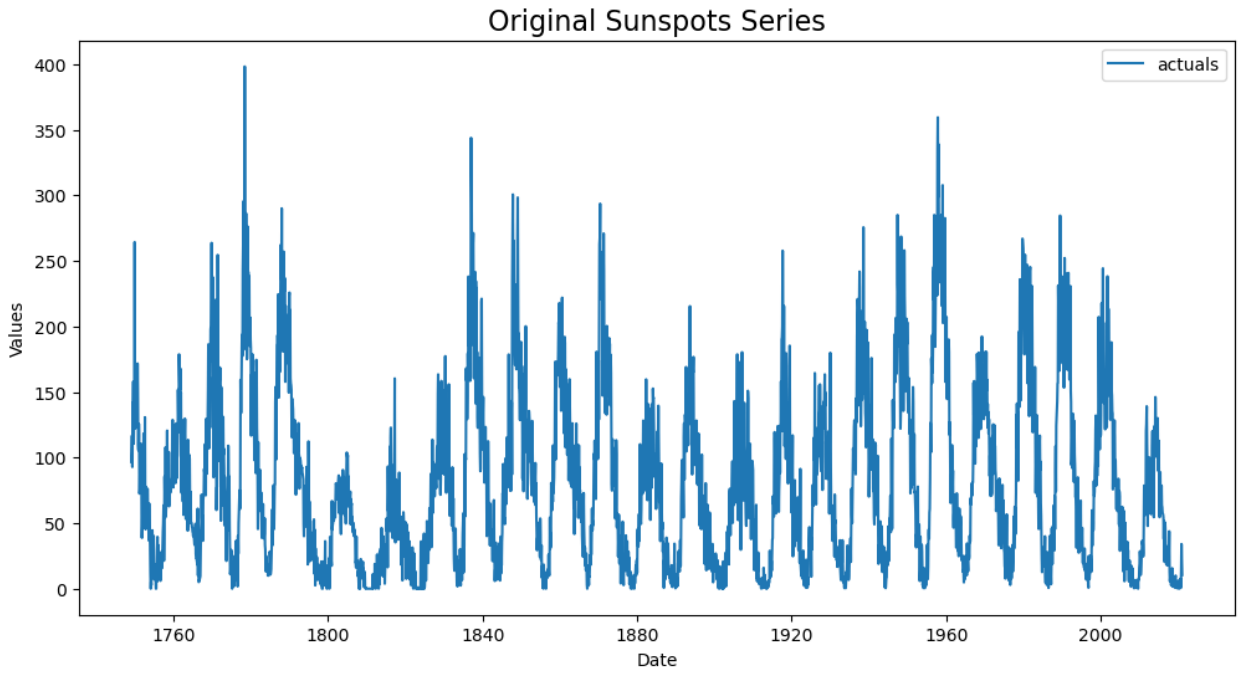
(continued from previous page)

```
    TestLength=240
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

16.1 EDA

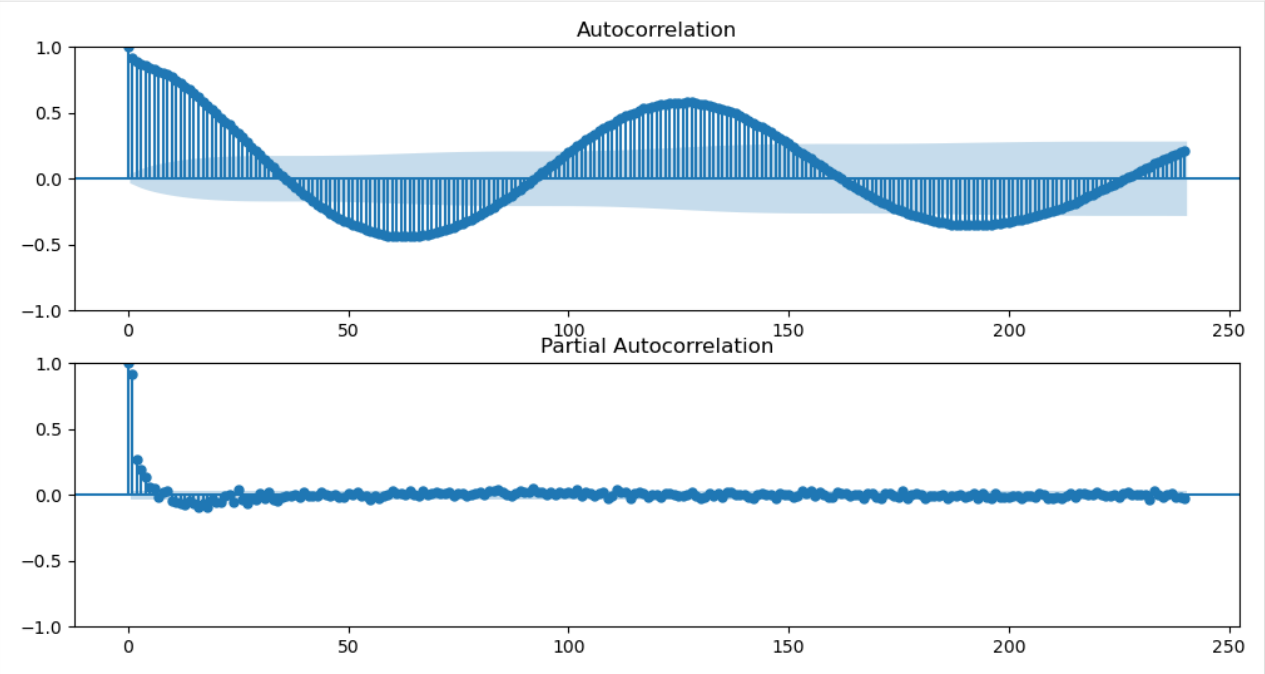
[3]:
```
f.plot()
plt.title('Original Sunspots Series',size=16)
```

[3]:
```
Text(0.5, 1.0, 'Original Sunspots Series')
```



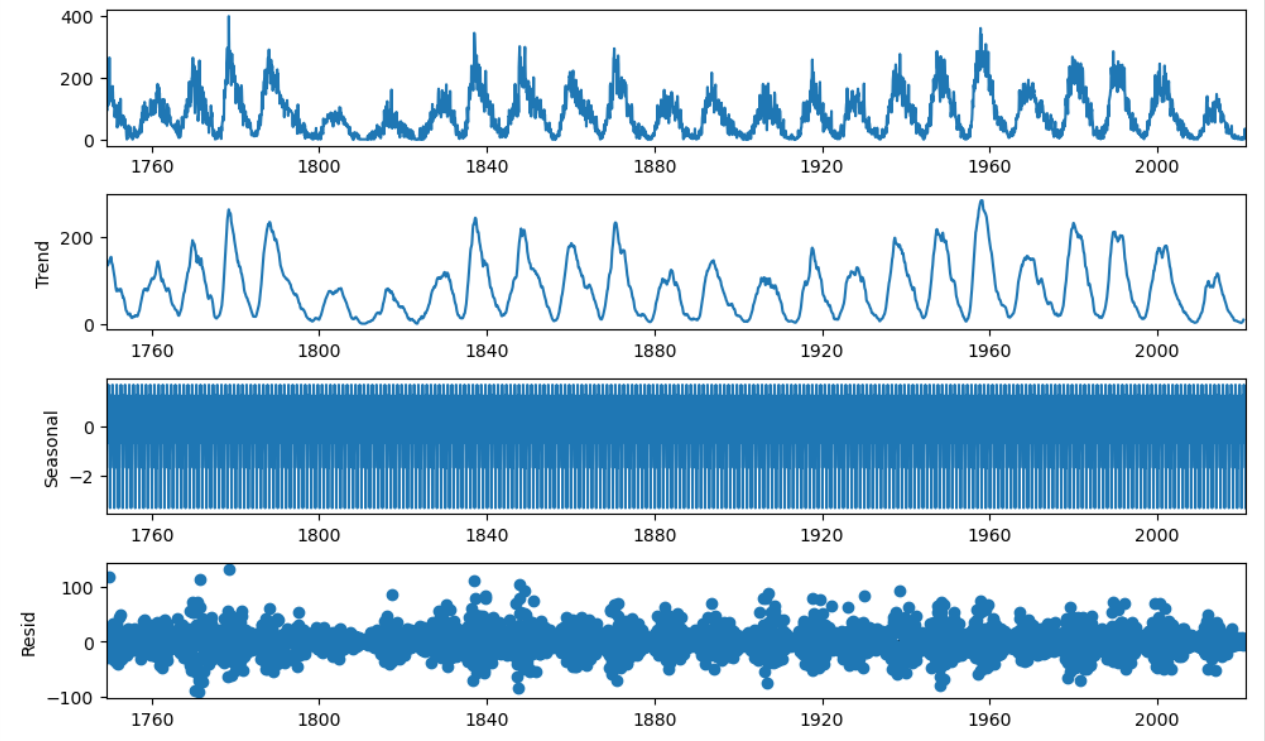
Let's see the ACF and PACF plots, allowing 240 lags to see this series' irregular 10-year cycle.

[4]:
```
figs, axs = plt.subplots(2, 1, figsize = (12,6))
f.plot_acf(ax=axs[0],lags=240)
f.plot_pacf(ax=axs[1],lags=240,method='ywm')
plt.show()
```



Let's view the series' seasonal decomposition.

[5]:
```
plt.rc("figure",figsize=(10,6))
f.seasonal_decompose().plot()
plt.show()
```



It is very difficult to make out anything useful from this graph.

16.2 Forecast RNN Model

- See Darts' GRU on the same series: https://unit8co.github.io/darts/examples/04-RNN-examples.html?highlight=sunspots

16.2.1 SimpleRNN

16.2.1.1 Unlayered model

- 1 hidden layer with 20% dropout
- 25 epochs
- 20% validcation split
- 32 batch size
- Tanh activation
- Adam optimizer
- MAE loss function

[6]:
```
f.auto_Xvar_select(
    try_trend = False,
    irr_cycles = [120,132,144],
    cross_validate = True,
    cvkwargs = {'k':3},
    dynamic_tuning = 240,
)
f.set_estimator('rnn')
f
```

[6]:
```
Forecaster(
    DateStartActuals=1749-01-31T00:00:00.000000000
    DateEndActuals=2021-01-31T00:00:00.000000000
    Freq=M
    N_actuals=3265
    ForecastLength=240
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11',
↪ 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21', 'AR22
↪', 'AR23', 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30', 'AR31', 'AR32',
↪'AR33', 'AR34', 'AR35', 'AR36', 'AR37', 'AR38', 'AR39', 'AR40', 'AR41', 'AR42', 'AR43',
↪ 'AR44', 'AR45', 'AR46', 'AR47', 'AR48', 'AR49', 'AR50', 'AR51', 'AR52', 'AR53', 'AR54
↪', 'AR55', 'AR56', 'AR57', 'AR58', 'AR59', 'AR60', 'AR61', 'AR62', 'AR63', 'AR64',
↪'AR65', 'AR66', 'AR67', 'AR68', 'AR69', 'AR70', 'AR71', 'AR72', 'AR73', 'AR74', 'AR75',
↪ 'AR76', 'AR77', 'AR78', 'AR79', 'AR80', 'AR81', 'AR82', 'AR83', 'AR84', 'AR85', 'AR86
↪', 'AR87', 'AR88', 'AR89', 'AR90', 'AR91', 'AR92', 'AR93', 'AR94', 'AR95', 'AR96',
↪'AR97', 'AR98', 'AR99', 'AR100', 'AR101', 'AR102', 'AR103', 'AR104', 'AR105', 'AR106',
↪'AR107', 'AR108', 'AR109', 'AR110', 'AR111', 'AR112', 'AR113', 'AR114', 'AR115', 'AR116
↪', 'AR117', 'AR118', 'AR119', 'AR120', 'AR121', 'AR122', 'AR123', 'AR124', 'AR125',
↪'AR126', 'AR127', 'AR128', 'AR129', 'AR130', 'AR131', 'AR132', 'AR133', 'AR134', 'AR135
↪', 'AR136', 'AR137', 'AR138', 'AR139', 'AR140', 'AR141', 'AR142', 'AR143', 'AR144',
↪'AR145', 'AR146', 'AR147', 'AR148', 'AR149', 'AR150', 'AR151', 'AR152', 'AR153', 'AR154
↪', 'AR155', 'AR156', 'AR157', 'AR158', 'AR159', 'AR160', 'AR161', 'AR162', 'AR163',
```

(continues on next page)

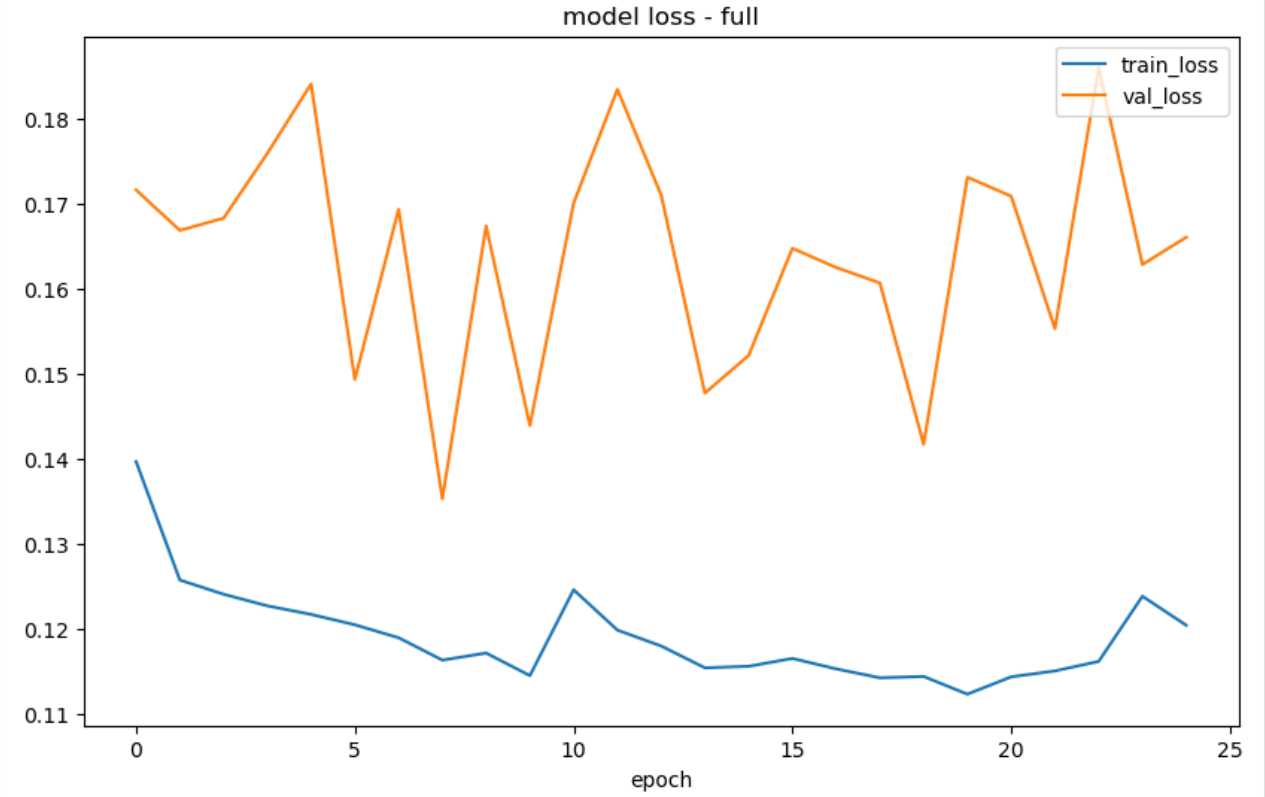
(continued from previous page)

```
↪'AR164', 'AR165', 'AR166', 'AR167', 'AR168', 'AR169', 'AR170', 'AR171', 'AR172', 'AR173
↪', 'AR174', 'AR175', 'AR176', 'AR177', 'AR178', 'AR179', 'AR180', 'AR181', 'AR182',
↪'AR183', 'AR184', 'AR185', 'AR186', 'AR187', 'AR188', 'AR189', 'AR190', 'AR191', 'AR192
↪', 'AR193', 'AR194', 'AR195', 'AR196', 'AR197', 'AR198', 'AR199', 'AR200', 'AR201',
↪'AR202', 'AR203', 'AR204', 'AR205', 'AR206', 'AR207', 'AR208', 'AR209', 'AR210', 'AR211
↪', 'AR212', 'AR213', 'AR214', 'AR215', 'AR216', 'AR217', 'AR218', 'AR219', 'AR220',
↪'AR221', 'AR222', 'AR223', 'AR224', 'AR225', 'AR226', 'AR227', 'AR228', 'AR229', 'AR230
↪', 'AR231', 'AR232', 'AR233', 'AR234']
    TestLength=240
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=rnn
    GridsFile=Grids
)
```

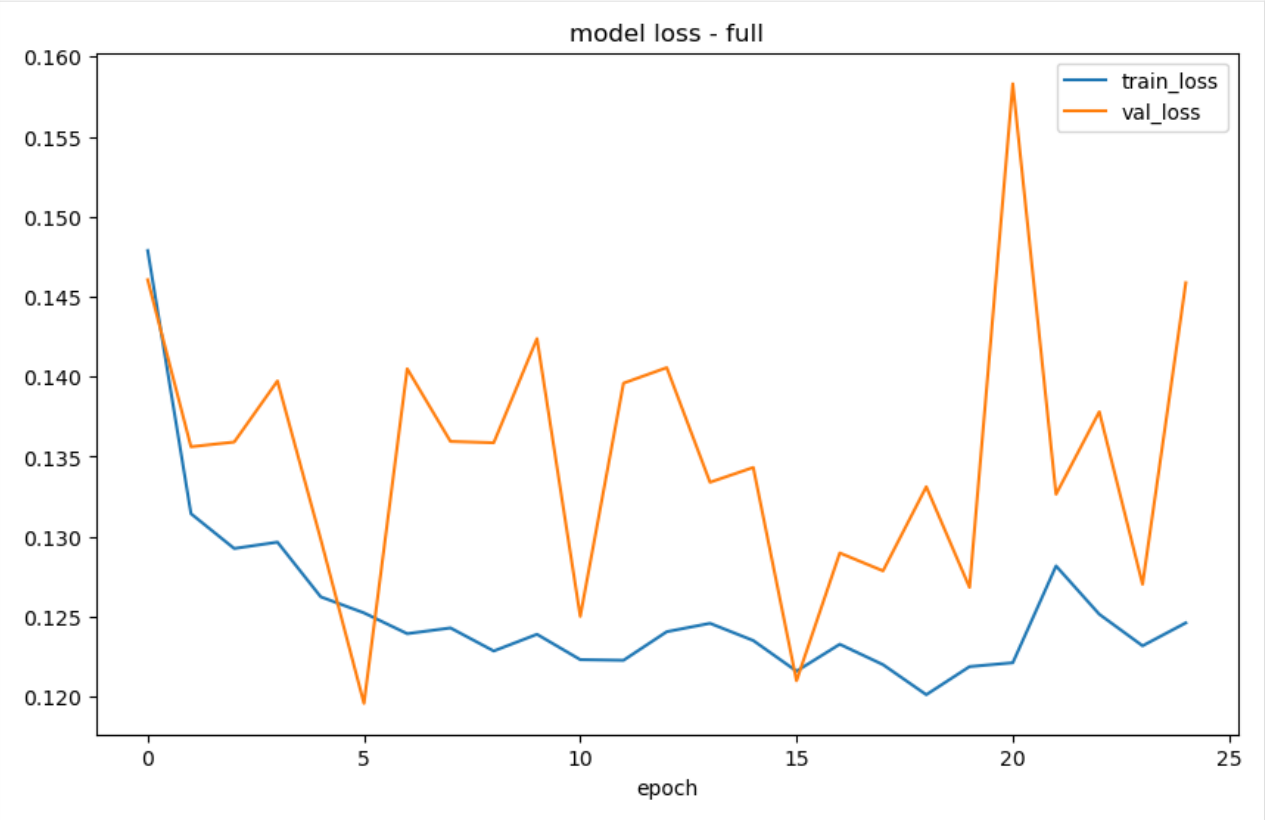
[7]:
```
f.manual_forecast(
    layers_struct=[('SimpleRNN',{'units':100,'dropout':0.2})],
    epochs=25,
    validation_split=0.2,
    plot_loss=True,
    call_me="rnn_1layer",
    verbose=0, # so it doesn't print each epoch and saves space in the notebook
)
```

```
2023-04-11 16:31:38.533751: I tensorflow/core/platform/cpu_feature_guard.cc:193] This␣
↪TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use␣
↪the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler␣
↪flags.
2023-04-11 16:31:40.033534: I tensorflow/core/platform/cpu_feature_guard.cc:193] This␣
↪TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use␣
↪the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler␣
↪flags.
```

```
1/1 [==============================] - 0s 108ms/step
```




```
1/1 [==============================] - 0s 91ms/step
88/88 [==============================] - 1s 10ms/step
```

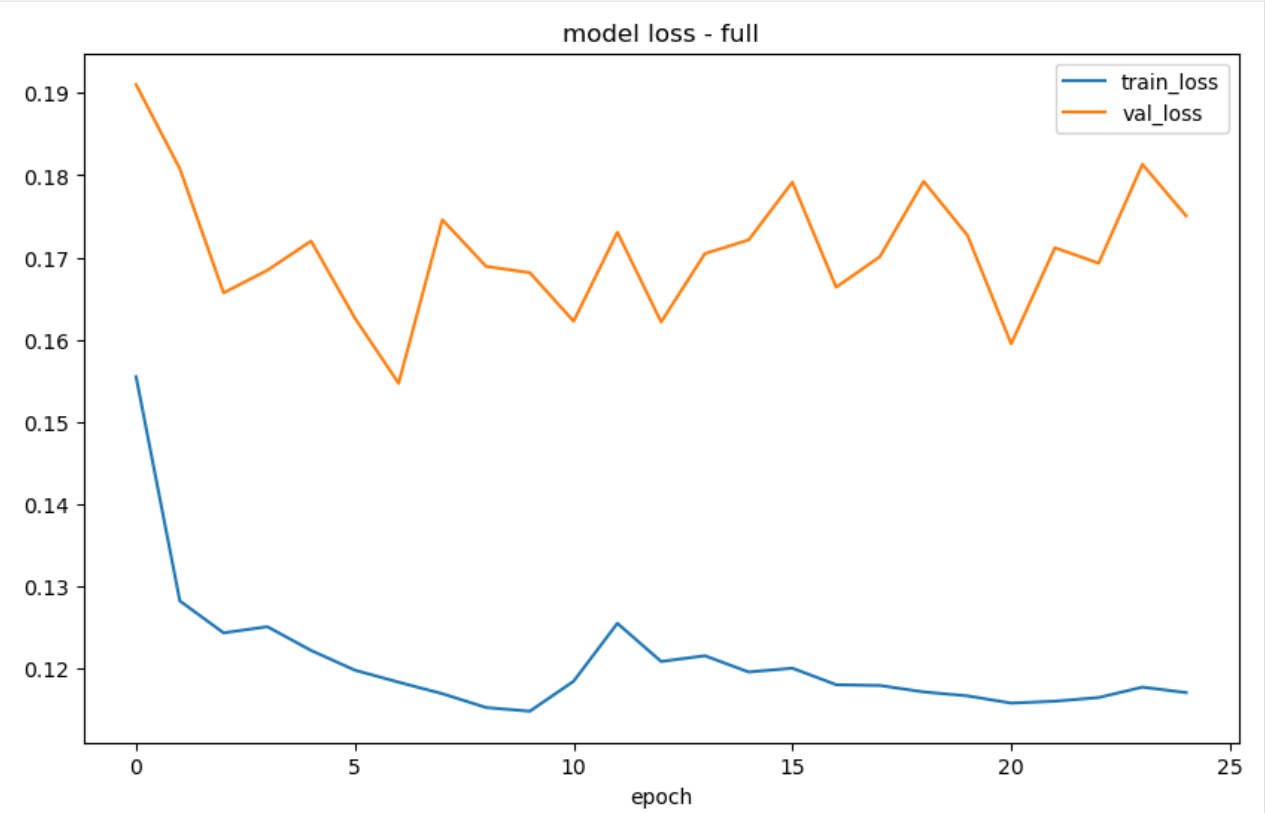


16.2.1.2 Layered Model

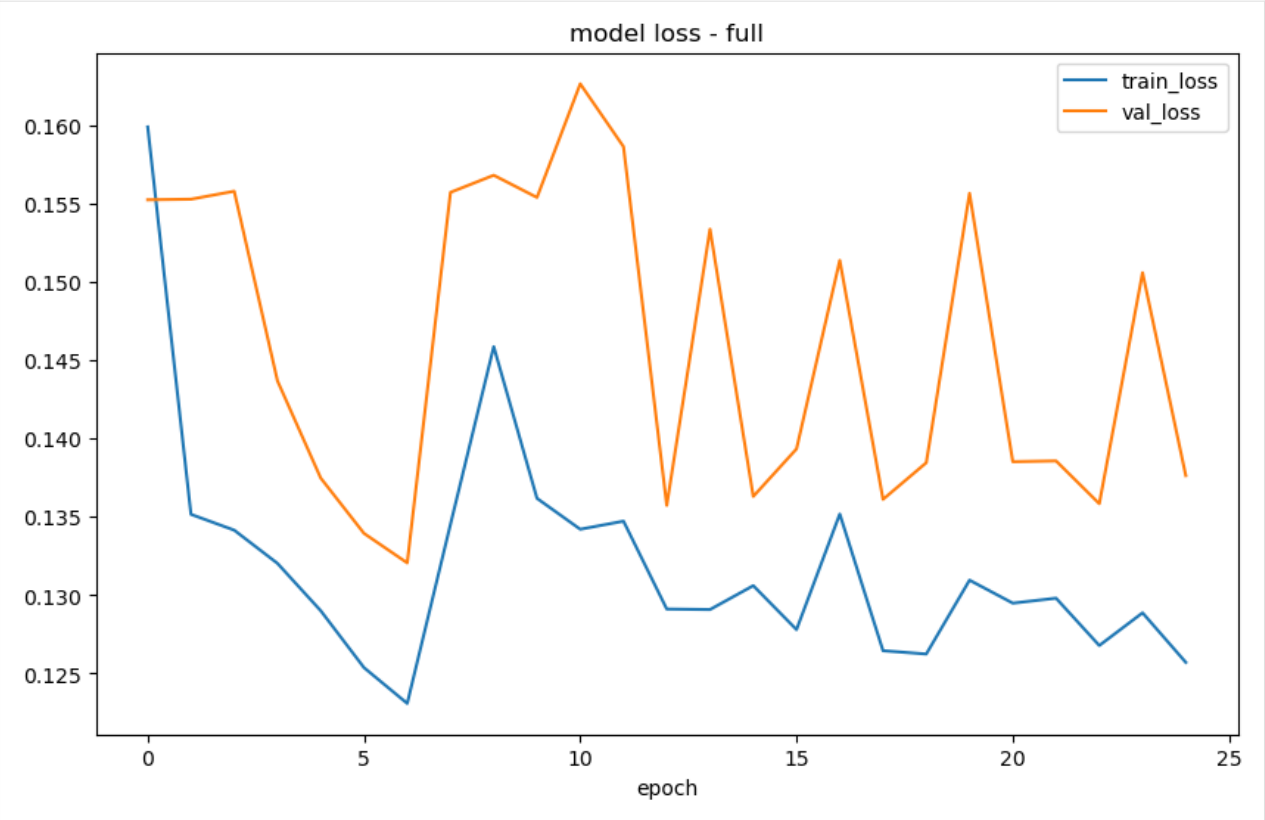
- 2 SimpleRNN layers, 100 units each
- 2 Dense layers, 10 units each
- No dropout
- Everything else the same

[8]:
```
f.manual_forecast(
    layers_struct=[
        ('SimpleRNN',{'units':100,'dropout':0})
    ] * 2 + [
        ('Dense',{'units':10})
    ]*2,
    epochs=25,
    random_seed=42,
    plot_loss=True,
    validation_split=0.2,
    call_me='rnn_layered',
    verbose=0
)
```

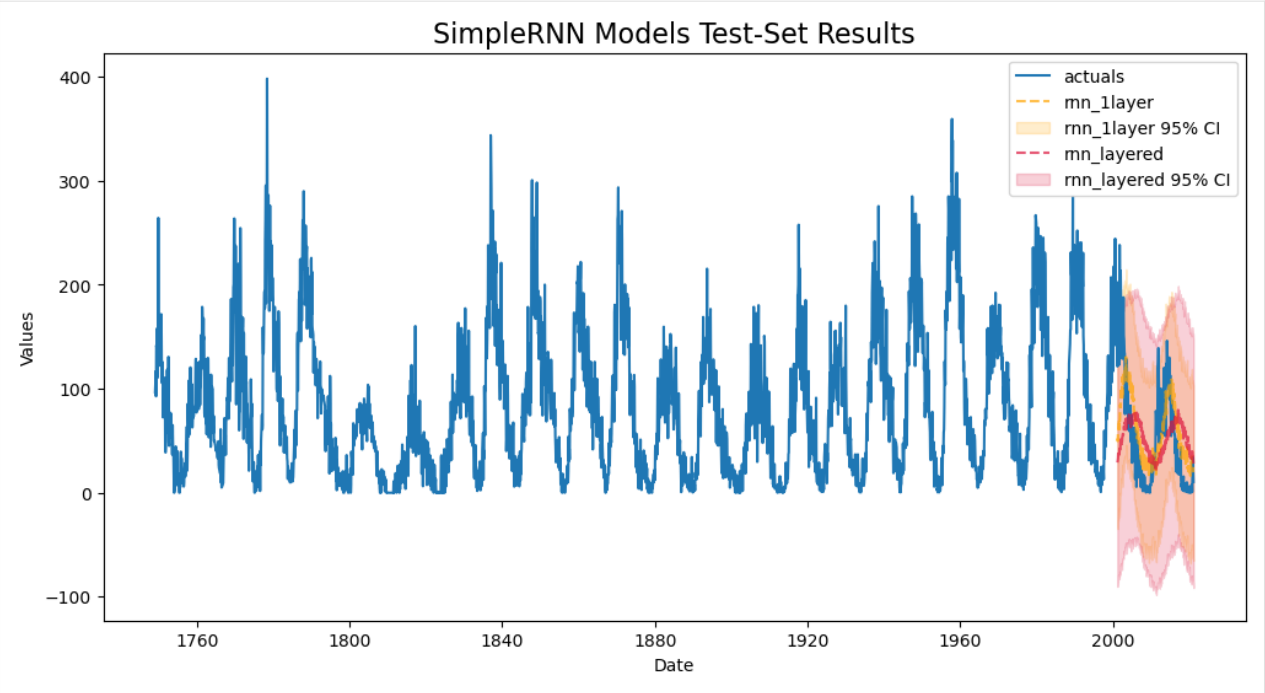
```
1/1 [==============================] - 0s 150ms/step
```




```
1/1 [==============================] - 0s 154ms/step
88/88 [==============================] - 2s 19ms/step
```



[9]:
```
f.plot_test_set(
    ci=True,
    models=['rnn_1layer','rnn_layered'],
    order_by='TestSetRMSE',
)
plt.title('SimpleRNN Models Test-Set Results',size=16)
plt.show()
```



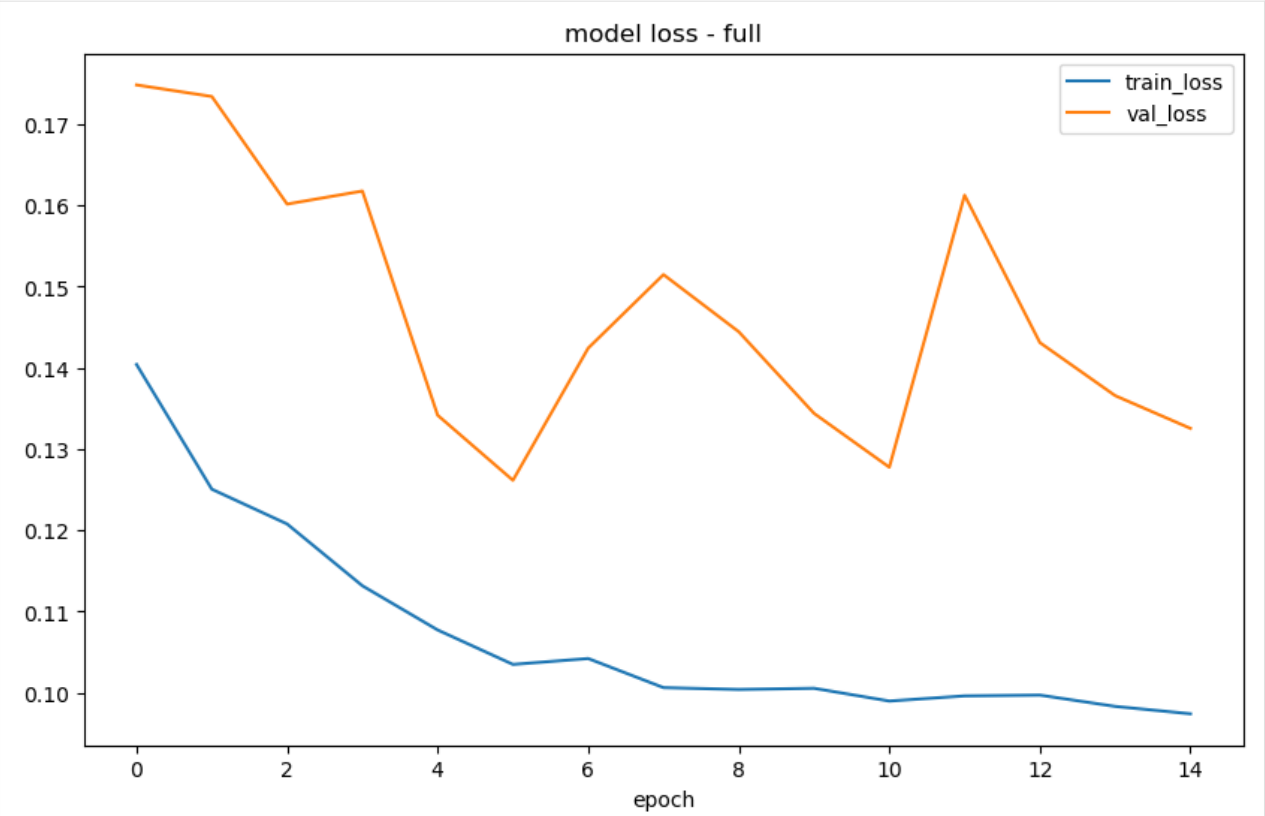
16.2.2 LSTM

16.2.2.1 Unlayered Model

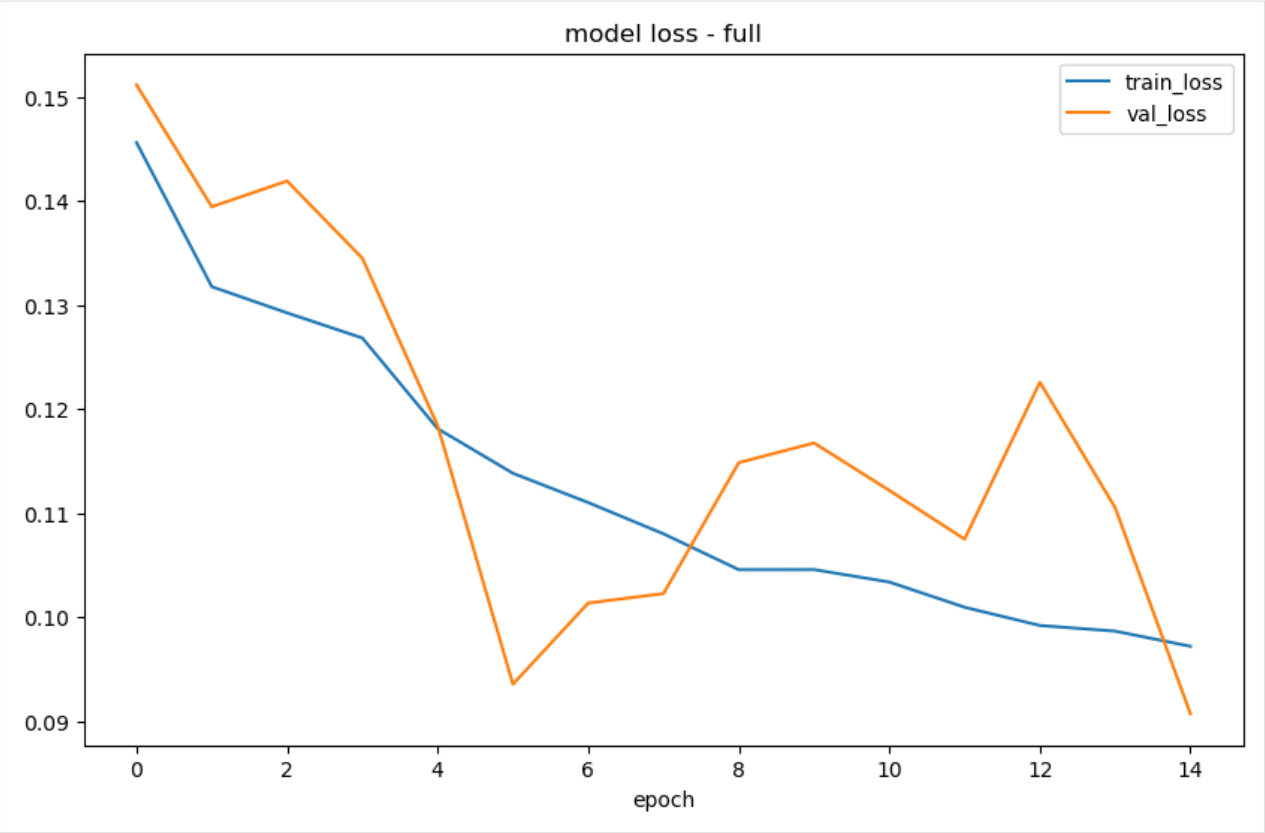
- One LSTM layer with 20% dropout
- 15 epochs
- Everything else the same

[10]:
```
f.manual_forecast(
    layers_struct=[('LSTM',{'units':100,'dropout':0.2})],
    epochs=15,
    validation_split=0.2,
    plot_loss=True,
    call_me="lstm_1layer",
    verbose=0, # so it doesn't print each epoch and saves space in the notebook
)
```

```
1/1 [==============================] - 0s 216ms/step
```




```
1/1 [==============================] - 0s 219ms/step
88/88 [==============================] - 3s 29ms/step
```

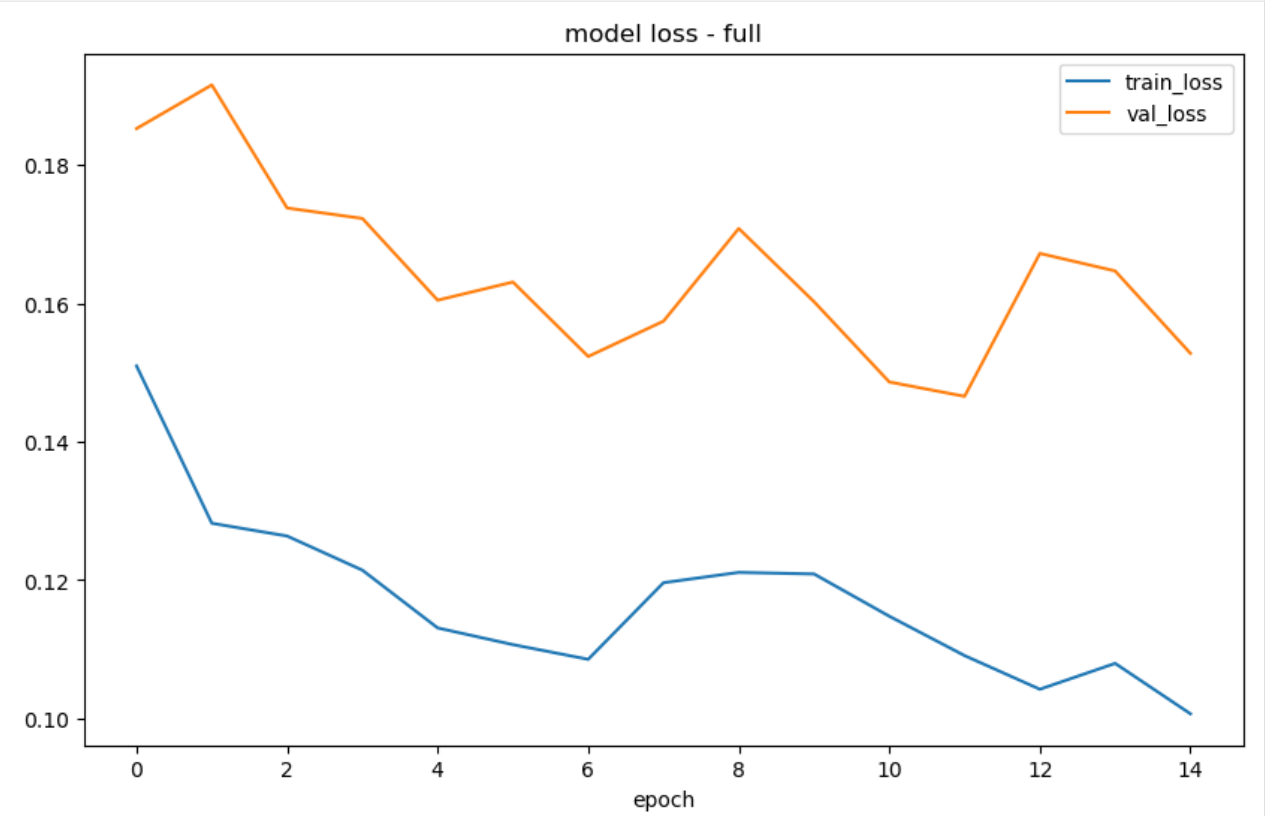


16.2.2.2 Layered Model

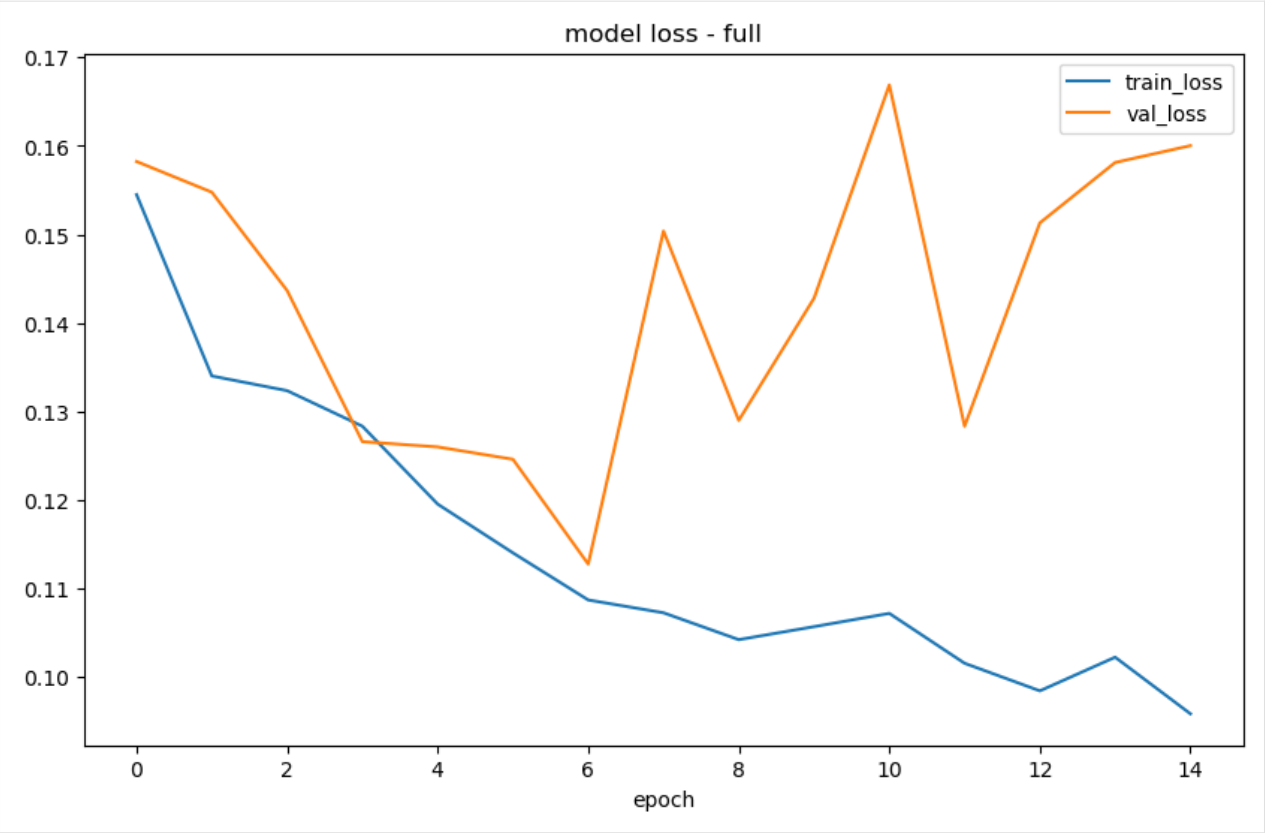
- 2 lstm layers and 2 dense layers
- No dropout

[11]:
```
f.manual_forecast(
    layers_struct=[
        ('LSTM',{'units':100,'dropout':0})
    ] * 2 + [
        ('Dense',{'units':10})
    ] * 2,
    epochs=15,
    random_seed=42,
    plot_loss=True,
    validation_split=0.2,
    call_me='lstm_layered',
    verbose=0
)
```

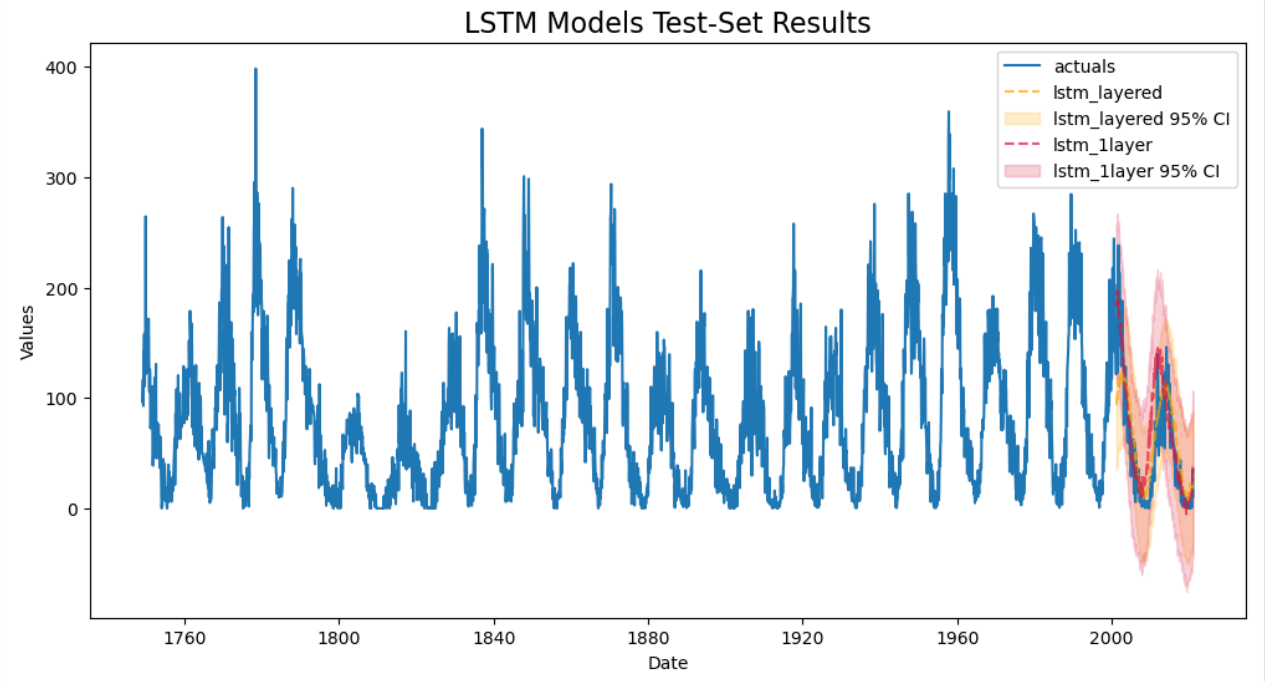
```
1/1 [==============================] - 0s 408ms/step
```




```
1/1 [==============================] - 0s 411ms/step
88/88 [==============================] - 5s 59ms/step
```



[12]:
```
f.plot_test_set(models=['lstm_1layer','lstm_layered'],ci=True,order_by='TestSetRMSE')
plt.title('LSTM Models Test-Set Results',size=16)
plt.show()
```



16.3 Prepare NNAR Model

- Let's try building a model you can read about here: https://otexts.com/fpp2/nnetar.html
- This is a time-series based dense neural network model, where the final predictions are the average of several models with randomly selected starting weights
- It is not a recurrent neural network and does not use the rnn estimator, but I thought it'd be fun to demonstrate here.
- It takes three input parameters:
- p (number of lags)
- P (number of seasonal lags)
- m (seasonal period)
- It also accepts exogenous variables. We add a time trend, monthly seasonality in wave functions, and a 120-period cycle
- Another parameter to consider is k: the size of the hidden layer. By default, this is the total number of inputs divided by 2, rounded up. We will keep the default
- The default number of models to average is 20 and we will keep that default as well
- See the model's documentation in R: https://www.rdocumentation.org/packages/forecast/versions/8.16/topics/nnetar

[13]:
```
p = 10 # non-seasonal lags
P = 6  # seasonal lags
m = 12 # seasonal period

f.drop_all_Xvars()
f.add_ar_terms(p)
f.add_AR_terms((P,m))

# in addition to the process described in the linked book, we can add monthly␣
↪seasonality in a fourier transformation
f.add_seasonal_regressors('month',raw=False,sincos=True)
# lastly, we add an irregular 10-year cycle that is idionsycratic to this dataset
f.add_cycle(120)
```

[14]:
```
f
```

[14]:
```
Forecaster(
    DateStartActuals=1749-01-31T00:00:00.000000000
    DateEndActuals=2021-01-31T00:00:00.000000000
    Freq=M
    N_actuals=3265
    ForecastLength=240
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR12',
↪ 'AR24', 'AR36', 'AR48', 'AR60', 'AR72', 'monthsin', 'monthcos', 'cycle120sin',
↪'cycle120cos']
    TestLength=240
    ValidationMetric=rmse
    ForecastsEvaluated=['rnn_1layer', 'rnn_layered', 'lstm_1layer', 'lstm_layered']
    CILevel=0.95
```

(continues on next page)

(continued from previous page)

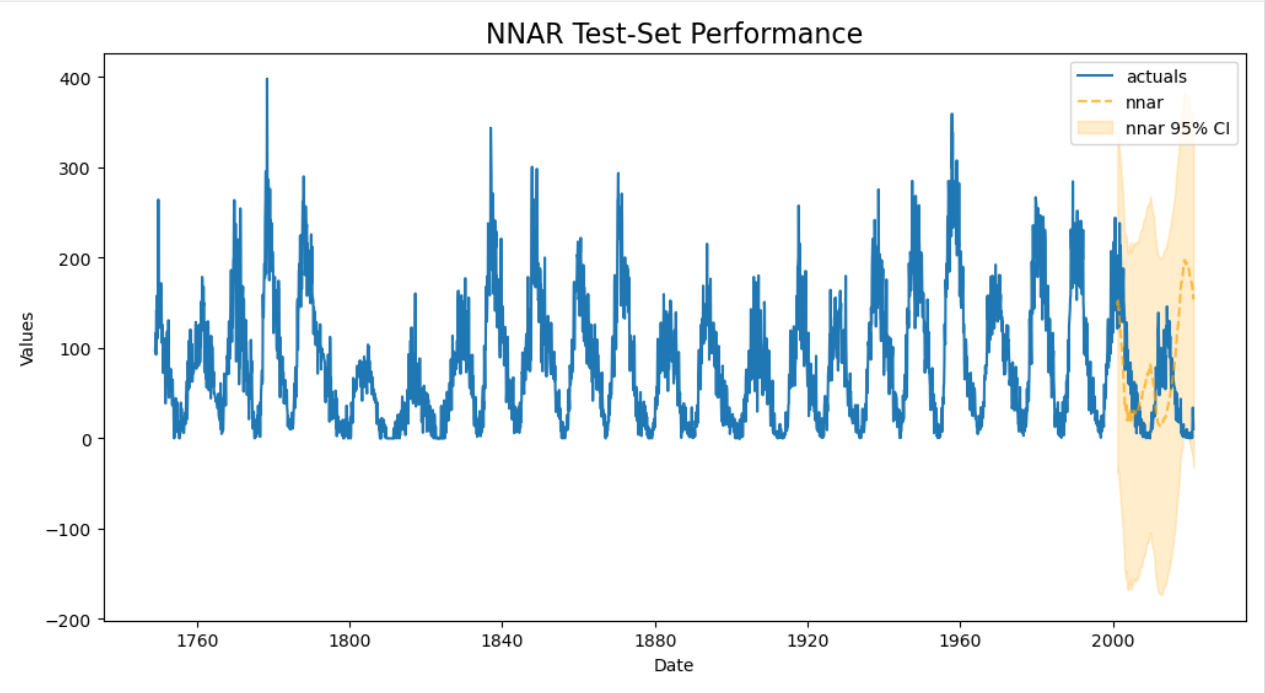
```
    CurrentEstimator=rnn
    GridsFile=Grids
)
```

16.4 Forecast NNAR Model

[15]:
```
# the default parameter used in the book is the total number of inputs divided by 2,␣
↪rounded up
k = int(np.ceil(len(f.get_regressor_names())/2))
repeats = 20 # default repeats number used in book
f.set_estimator('mlp')
for r in tqdm(range(repeats)): # repeats
    f.manual_forecast(
        hidden_layer_sizes=(k,),
        activation='relu',
        random_state=r,
        normalizer='scale',
        call_me=f'mlp_{r}',
    )
f.save_feature_importance()
# now we take the averages of all models to create the final NNAR
f.set_estimator('combo')
f.manual_forecast(how='simple',models=[f'mlp_{r}' for r in range(20)],call_me='nnar')
```

```
  0%|          | 0/20 [00:00<?, ?it/s]
```

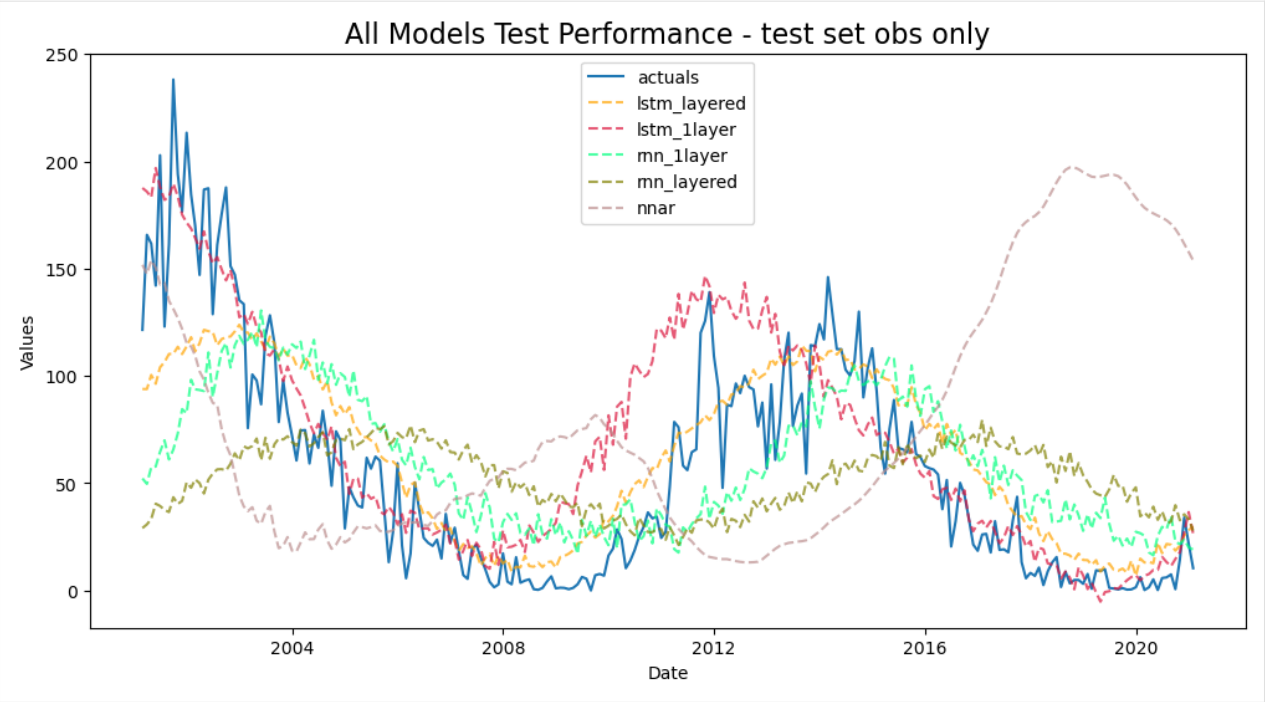
[16]:
```
f.plot_test_set(ci=True,models='nnar')
plt.title('NNAR Test-Set Performance',size=16)
plt.show()
```



This doesn't look as good as any of our evaluated RNN models.

One thing we notice with all these test-set graphs is that the precise deviations from the trend are hard to see because the series is so large. Let's call the function again with all evaluated models ordered by test RMSE and zoomed in to the test-set section of the series only by setting `include_train=False`.

[17]:
```
f.plot_test_set(
    models=['rnn_1layer','rnn_layered','lstm_1layer','lstm_layered','nnar'],
    order_by='TestSetRMSE',
    include_train=False,
)
plt.title('All Models Test Performance - test set obs only',size=16)
plt.show()
```



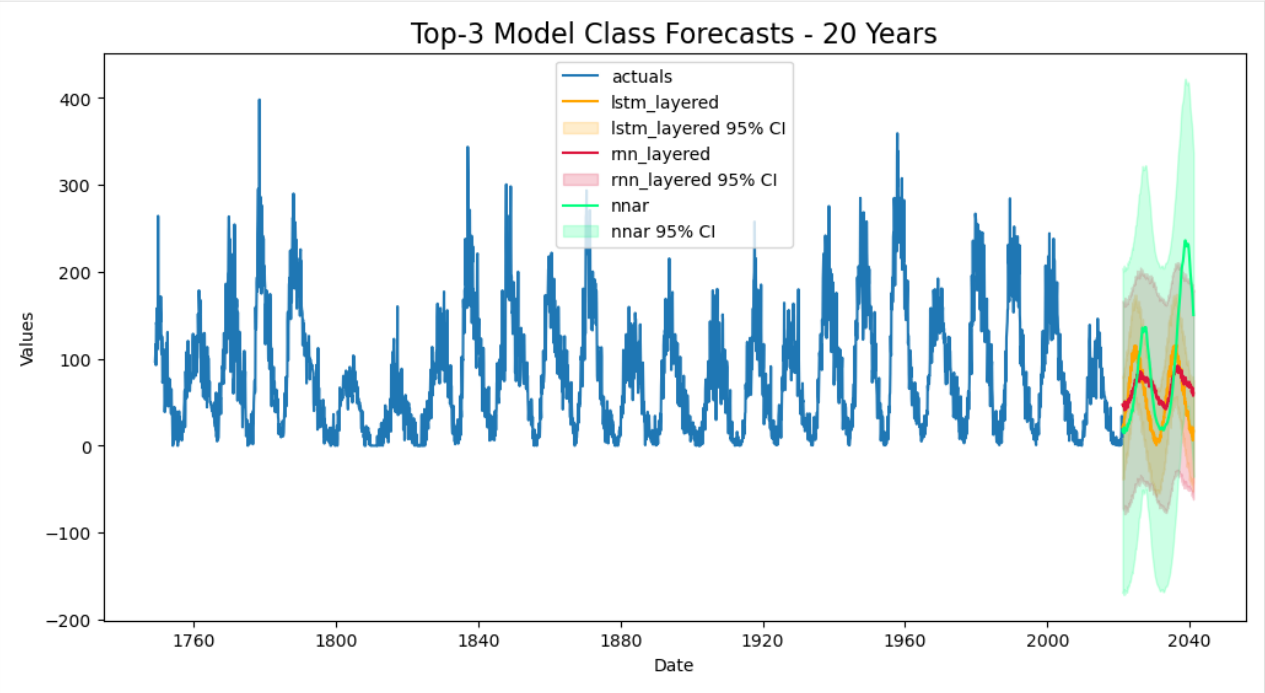
With this view, it is not clear that any model did as well as we might have thought from the other graphs.

16.5 Compare All Models

16.5.1 Forecast Plot

- best rnn model
- best lstm model
- nnar model

[18]:
```
f.plot(models=['rnn_layered','lstm_layered','nnar'],order_by='TestSetRMSE',ci=True)
plt.title('Top-3 Model Class Forecasts - 20 Years',size=16)
plt.show()
```



16.5.2 Export Model Summaries for all applied models

[19]:
```
pd.set_option('display.float_format',  '{:.4f}'.format)
f.export(
    'model_summaries',
    models=['rnn_1layer','rnn_layered','lstm_1layer','lstm_layered','nnar'],
    determine_best_by = 'TestSetRMSE',
)[
    [
        'ModelNickname',
        'TestSetRMSE',
        'TestSetR2',
        'InSampleRMSE',
        'InSampleR2',
        'best_model'
    ]
]
```

[19]:

	ModelNickname	TestSetRMSE	TestSetR2	InSampleRMSE	InSampleR2	best_model
0	lstm_layered	28.5508	0.7083	56.6811	0.3310	True
1	lstm_1layer	31.8674	0.6366	43.3984	0.6078	False
2	rnn_1layer	41.2311	0.3918	68.7316	0.0163	False
3	rnn_layered	55.6165	-0.1067	60.6494	0.2341	False
4	nnar	92.0142	-2.0293	28.4102	0.8266	False

16.5.3 Backtest Best Model

The LSTM with four layers appears to be our most accurate model out of the 5 we ran and showed. The NNAR model, in spite of being fun to set up, was the worst. With more time spent, even better results could be extracted. Although that model appears accurate on the test set we gave to it, an interesting question is how good it would have been if we had used it in production for the last 5 years. That's what backtesting allows us to evaluate. Backtesting can take a long time for more complex models.

[32]:
```
from scalecast.Pipeline import Pipeline
from scalecast.util import backtest_metrics

def forecaster(f):
    f.auto_Xvar_select(
        try_trend = False,
        irr_cycles = [120,132,144],
        cross_validate = True,
        cvkwargs = {'k':3},
        dynamic_tuning = 240,
        max_ar = 240,
    )
    f.set_estimator('rnn')
    f.manual_forecast(
        layers_struct=[
            ('LSTM',{'units':100,'dropout':0})
        ] * 2 + [
            ('Dense',{'units':10})
        ] * 2,
        epochs=15,
        random_seed=42,
        validation_split=0.2,
        call_me='lstm_layered',
        verbose=0
    )

pipeline = Pipeline(steps = [('Forecast',forecaster)])
```

[33]:
```
# default args below
backtest_results = pipeline.backtest(
    f,
    n_iter=5,
    fcst_length = 240,
    test_length = 0,
    jump_back = 12, # place a year between consecutive training sets
    cis = False,
)
bm = backtest_metrics(backtest_results,mets=['r2','rmse'])
bm
```

```
1/1 [==============================] - 0s 439ms/step
80/80 [==============================] - 5s 63ms/step
1/1 [==============================] - 0s 418ms/step
80/80 [==============================] - 5s 67ms/step
1/1 [==============================] - 0s 414ms/step
79/79 [==============================] - 5s 67ms/step
```

(continues on next page)

(continued from previous page)

```
1/1 [==============================] - 0s 435ms/step
79/79 [==============================] - 6s 70ms/step
1/1 [==============================] - 0s 390ms/step
80/80 [==============================] - 5s 58ms/step
```

[33]:

Model	Metric	Iter0	Iter1	Iter2	Iter3	Iter4	Average
lstm_layered	r2	0.7385	0.3077	0.3695	0.3589	0.2503	0.4050
	rmse	27.0322	48.1908	46.7773	45.9266	49.2748	43.4403

[34]:

```
rmse = bm.loc[('lstm_layered','rmse'),'Average']
r2 = bm.loc[('lstm_layered','r2'),'Average']

print(
    'This shows that if we had used this model on actual data in the previous 5 years,'
    f' it would have obtained an average RMSE of {rmse:.1f} and R2 of {r2:.1%}.'
)
```

```
This shows that if we had used this model on actual data in the previous 5 years, it␣
↪would have obtained an average RMSE of 43.4 and R2 of 40.5%.
```

[ ]:

CHAPTER

SEVENTEEN

SCIKIT-LEARN MODELS

This notebook showcases the scalecast wrapper around scikit-learn models.

The data is available for download here: https://www.kaggle.com/datasets/bobnau/daily-website-visitors
See the blog post:
https://medium.com/towards-data-science/expand-your-time-series-arsenal-with-these-models-10c807d37558

[1]:
```
import pandas as pd
import numpy as np
import pandas_datareader as pdr
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from dateutil.relativedelta import relativedelta
from scalecast.Forecaster import Forecaster
from scalecast import GridGenerator
from scalecast.auxmodels import mlp_stack
from tqdm.notebook import tqdm
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.neural_network import MLPRegressor

pd.set_option('max_colwidth',500)
pd.set_option('display.max_rows',500)
```

[2]:
```
def plot_test_export_summaries(f):
    """ exports the relevant statisitcal information and displays a plot of the test-set␣
↪results for the last model run
    """
    f.plot_test_set(models=f.estimator,ci=True)
    plt.title(f'{f.estimator} test-set results',size=16)
    plt.show()
    return f.export('model_summaries',determine_best_by='TestSetMAPE')[
        [
            'ModelNickname',
            'HyperParams',
            'TestSetMAPE',
            'TestSetR2',
```

(continues on next page)

(continued from previous page)

```
            'InSampleMAPE',
            'InSampleR2'
        ]
    ]
```

We will bring in the default grids from scalecast and tune some of our models this way. For others, we will create our own grids.

[3]:
```
GridGenerator.get_example_grids()
```

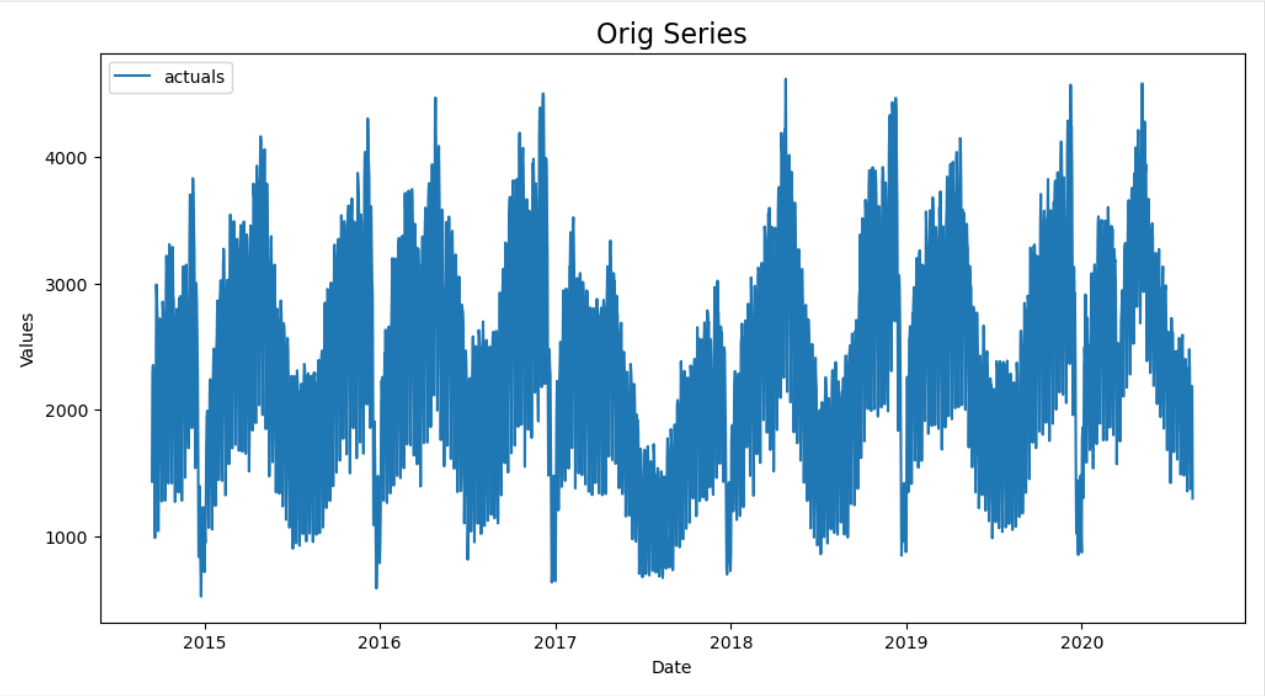
[4]:
```
data = pd.read_csv('daily-website-visitors.csv')
data.head()
```

[4]:
```
   Row        Day  Day.Of.Week       Date Page.Loads Unique.Visits  \
0    1     Sunday            1  9/14/2014      2,146         1,582
1    2     Monday            2  9/15/2014      3,621         2,528
2    3    Tuesday            3  9/16/2014      3,698         2,630
3    4  Wednesday            4  9/17/2014      3,667         2,614
4    5   Thursday            5  9/18/2014      3,316         2,366

   First.Time.Visits Returning.Visits
0               1430              152
1               2297              231
2               2352              278
3               2327              287
4               2130              236
```

[5]:
```
data.shape
```

[5]:
```
(2167, 8)
```

[6]:
```
f=Forecaster(y=data['First.Time.Visits'],current_dates=data['Date'])
f.plot()
plt.title('Orig Series',size=16)
plt.show()
```



Not all models we will showcase come standard in scalecast, but we can easily add them by using the code below:

[7]:
```
f.add_sklearn_estimator(BaggingRegressor,'bagging')
f.add_sklearn_estimator(StackingRegressor,'stacking')
```

For more EDA on this dataset, see the prophet example: https://scalecast-examples.readthedocs.io/en/latest/prophet/prophet.html#EDA

17.1 Prepare Forecast

- 60-day forecast horizon
- 20% test split
- Turn confidence intervals on
- Add time series regressors:
- 7 autoregressive terms
- 4 seasonal autoregressive terms spaced 7-periods apart
- month, quarter, week, and day of year with a fournier transformation
- dayofweek, leap year, and week as dummy variables
- year

[8]:
```
fcst_length = 60
f.generate_future_dates(fcst_length)
f.set_test_length(.2)
f.eval_cis() # tell the object to build confidence intervals for all models
f.add_ar_terms(7)
```

(continues on next page)

(continued from previous page)

```
f.add_AR_terms((4,7))
f.add_seasonal_regressors('month','quarter','week','dayofyear',raw=False,sincos=True)
f.add_seasonal_regressors('dayofweek','is_leap_year','week',raw=False,dummy=True,drop_
↪first=True)
f.add_seasonal_regressors('year')
f
```

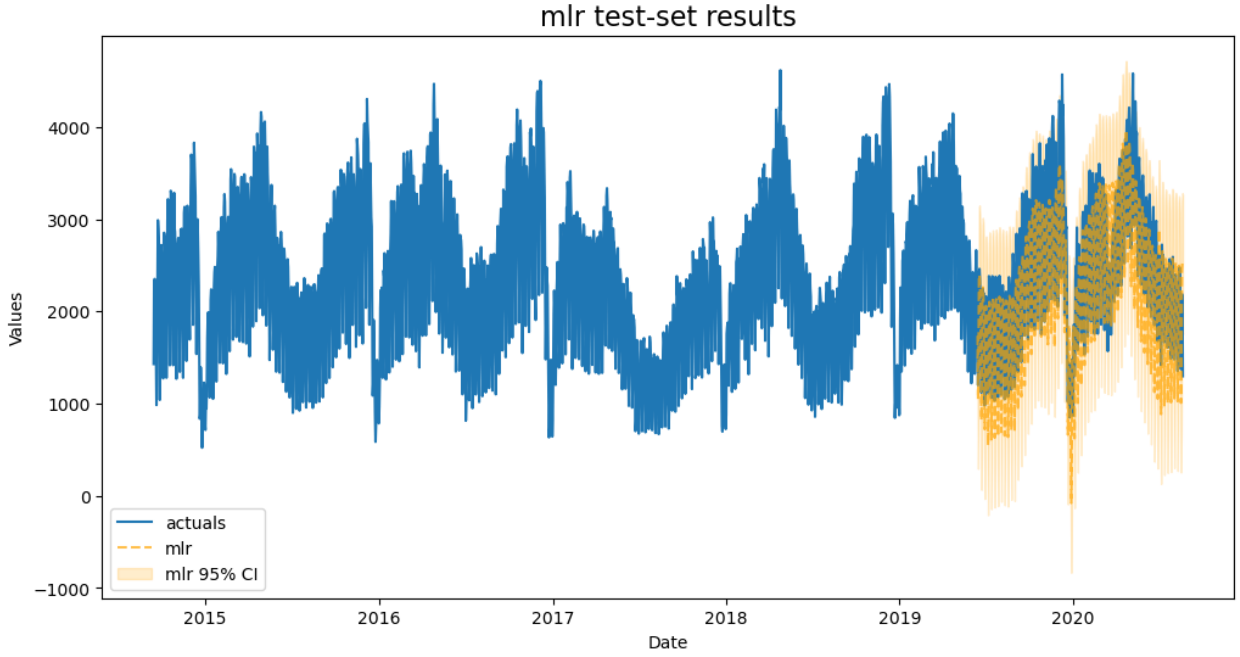
[8]:
```
Forecaster(
    DateStartActuals=2014-09-14T00:00:00.000000000
    DateEndActuals=2020-08-19T00:00:00.000000000
    Freq=D
    N_actuals=2167
    ForecastLength=60
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR14', 'AR21', 'AR28',
↪'monthsin', 'monthcos', 'quartersin', 'quartercos', 'weeksin', 'weekcos', 'dayofyearsin
↪', 'dayofyearcos', 'dayofweek_1', 'dayofweek_2', 'dayofweek_3', 'dayofweek_4',
↪'dayofweek_5', 'dayofweek_6', 'is_leap_year_True', 'week_10', 'week_11', 'week_12',
↪'week_13', 'week_14', 'week_15', 'week_16', 'week_17', 'week_18', 'week_19', 'week_2',
↪'week_20', 'week_21', 'week_22', 'week_23', 'week_24', 'week_25', 'week_26', 'week_27',
↪ 'week_28', 'week_29', 'week_3', 'week_30', 'week_31', 'week_32', 'week_33', 'week_34',
↪ 'week_35', 'week_36', 'week_37', 'week_38', 'week_39', 'week_4', 'week_40', 'week_41',
↪ 'week_42', 'week_43', 'week_44', 'week_45', 'week_46', 'week_47', 'week_48', 'week_49
↪', 'week_5', 'week_50', 'week_51', 'week_52', 'week_53', 'week_6', 'week_7', 'week_8',
↪'week_9', 'year']
    TestLength=433
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

17.2 MLR

- use default parameters (which means all input vars are scaled with a minmax scaler by default for all sklearn models)

[9]:
```
f.set_estimator('mlr')
f.manual_forecast()
```

[10]:
```
plot_test_export_summaries(f)
```



[10]:

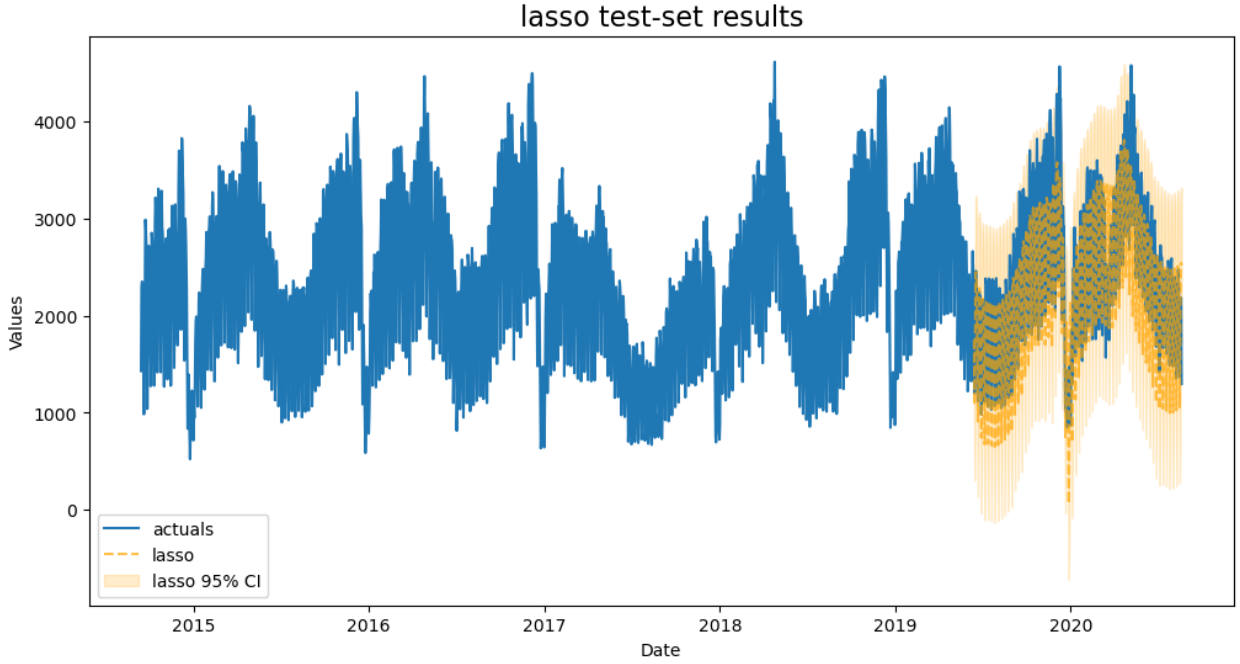
	ModelNickname	HyperParams	TestSetMAPE	TestSetR2	InSampleMAPE	InSampleR2
0	mlr	{}	0.14485	0.733553	0.064183	0.953482

17.3 Lasso

- tune alpha with 100 choices using 3-fold cross validation

[11]:
```
f.set_estimator('lasso')
lasso_grid = {'alpha':np.linspace(0,2,100)}
f.ingest_grid(lasso_grid)
f.cross_validate(k=3)
f.auto_forecast()
```

[12]:
```
plot_test_export_summaries(f)
```



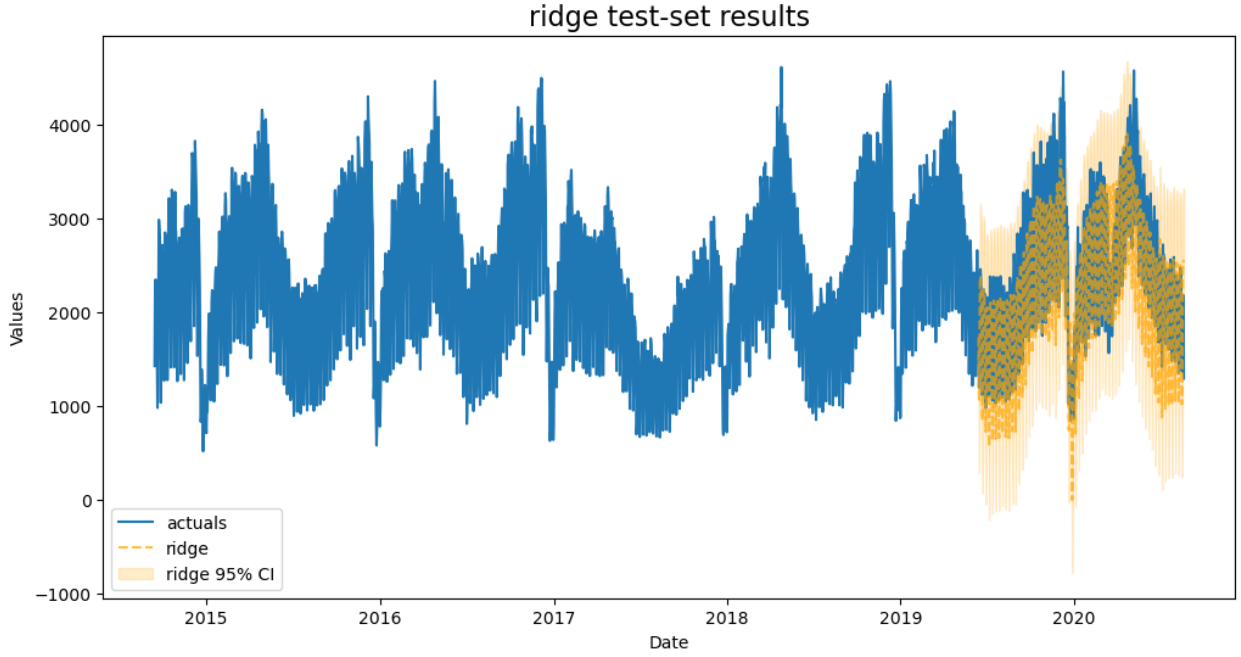
[12]:
```
  ModelNickname                       HyperParams  TestSetMAPE  TestSetR2  \
0         lasso  {'alpha': 0.32323232323232326}     0.139526   0.744279
1           mlr                                {}     0.144850   0.733553

   InSampleMAPE  InSampleR2
0      0.065458    0.952205
1      0.064183    0.953482
```

17.4 Ridge

- tune alpha with the same grid used for lasso

[13]:
```
f.set_estimator('ridge')
f.ingest_grid(lasso_grid)
f.cross_validate(k=3)
f.auto_forecast()
```

[14]:
```
plot_test_export_summaries(f)
```



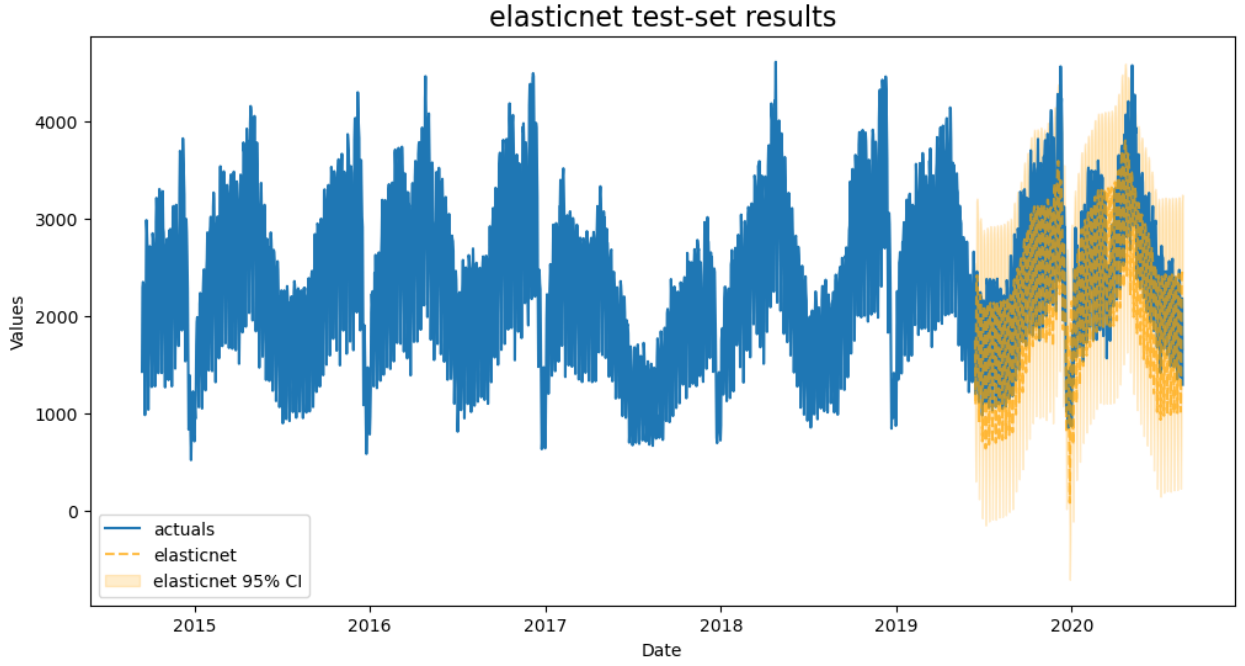
[14]:
```
  ModelNickname                    HyperParams  TestSetMAPE  TestSetR2  \
0         lasso  {'alpha': 0.32323232323232326}     0.139526   0.744279
1         ridge   {'alpha': 0.6060606060606061}     0.140560   0.743738
2           mlr                             {}     0.144850   0.733553

   InSampleMAPE  InSampleR2
0      0.065458    0.952205
1      0.064592    0.952812
2      0.064183    0.953482
```

17.5 Elasticnet

- this model mixes L1 and L2 regularization parameters
- its default grid is pretty good for finding the optimal alpha value and l1 ratio

[15]:
```
f.set_estimator('elasticnet')
f.cross_validate(k=3)
f.auto_forecast()
```

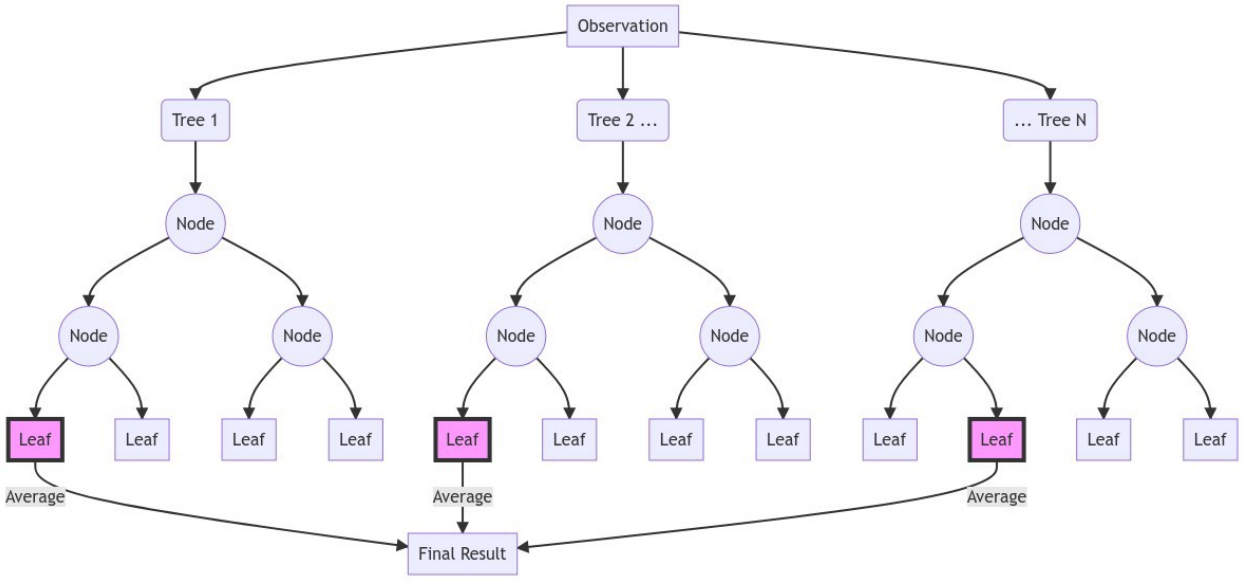
[16]:
```
plot_test_export_summaries(f)
```



[16]:

	ModelNickname	HyperParams	TestSetMAPE	TestSetR2	InSampleMAPE	InSampleR2
0	lasso	{'alpha': 0.32323232323232326}	0.139526	0.744279	0.065458	0.952205
1	ridge	{'alpha': 0.6060606060606061}	0.140560	0.743738	0.064592	0.952812
2	elasticnet	{'alpha': 2.0, 'l1_ratio': 1, 'normalizer': 'scale'}	0.141774	0.733125	0.065364	0.952024
3	mlr	{}	0.144850	0.733553	0.064183	0.953482

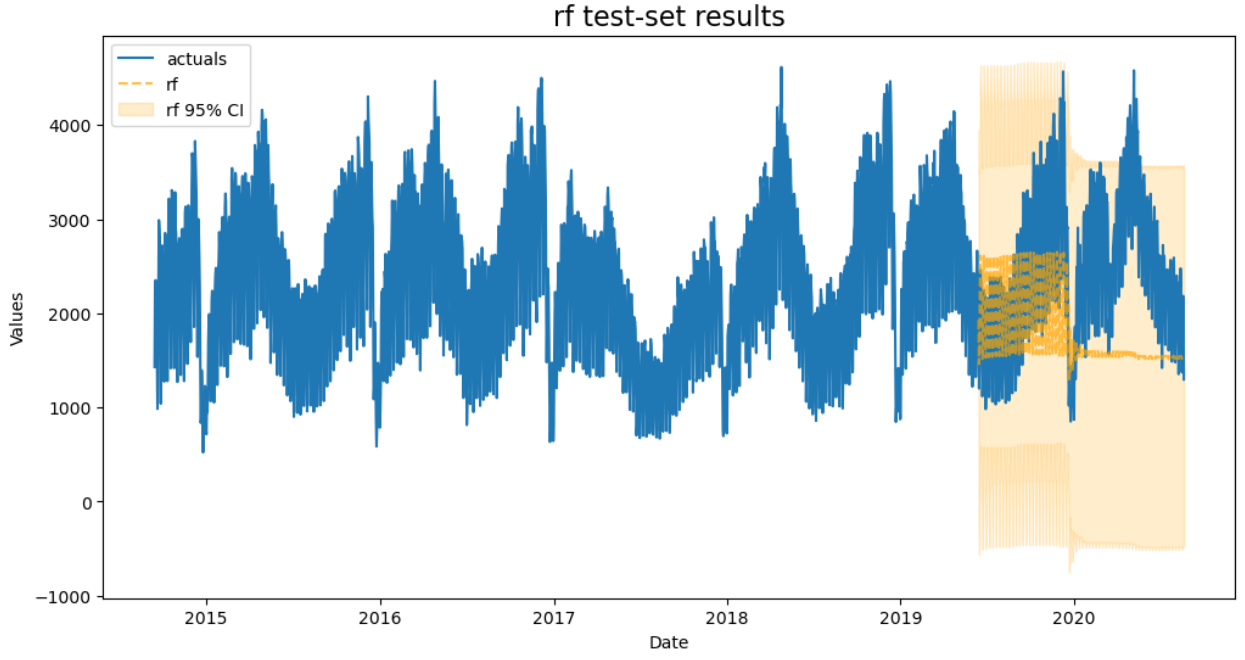
17.6 RF



create a grid to tune Random Forest with more options than what is available in the default grids

[17]:
```
f.set_estimator('rf')
rf_grid = {
    'max_depth':[2,3,4,5],
    'n_estimators':[100,200,500],
    'max_features':['auto','sqrt','log2'],
    'max_samples':[.75,.9,1],
}
f.ingest_grid(rf_grid)
f.cross_validate(k=3)
f.auto_forecast()
```

[18]:
```
plot_test_export_summaries(f)
```



[18]:

	ModelNickname	HyperParams	TestSetMAPE	TestSetR2	InSampleMAPE	InSampleR2
0	lasso	{'alpha': 0.32323232323232326}	0.139526	0.744279	0.065458	0.952205
1	ridge	{'alpha': 0.6060606060606061}	0.140560	0.743738	0.064592	0.952812
2	mlr	{}	0.144850	0.733553	0.064183	0.953482
3	elasticnet	{'alpha': 2.0, 'l1_ratio': 0, 'normalizer': None}	0.259322	0.152033	0.092402	0.904719
4	rf	{'max_depth': 5, 'n_estimators': 200, 'max_features': 'auto', 'max_samples': 0.75}	0.318096	-0.916329	0.083374	0.918095

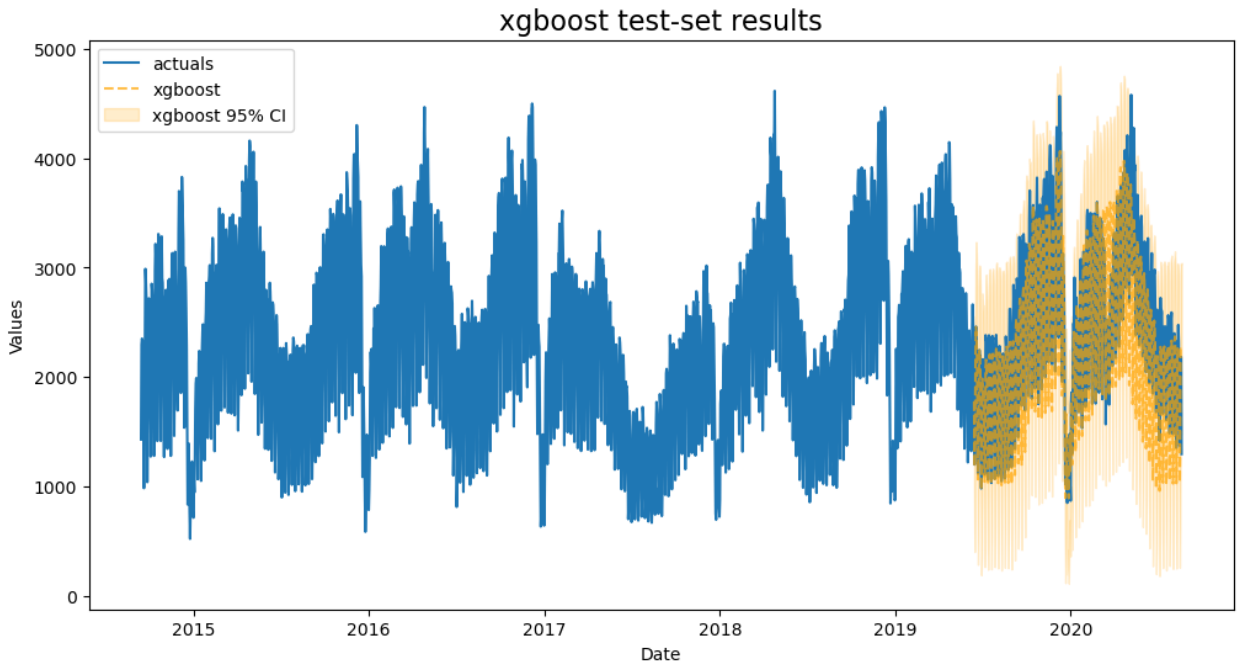
So far, both using a visual inspection and examing its test-set performance, Random Forest does worse than all others.

17.7 XGBoost

- Build a grid to tune XGBoost

[19]:
```
f.set_estimator('xgboost')
xgboost_grid = {
     'n_estimators':[150,200,250],
     'scale_pos_weight':[5,10],
     'learning_rate':[0.1,0.2],
     'gamma':[0,3,5],
     'subsample':[0.8,0.9]
}
f.ingest_grid(xgboost_grid)
f.cross_validate(k=3)
f.auto_forecast()
```

[20]:
```
plot_test_export_summaries(f)
```



[20]:
```
  ModelNickname  \
0       xgboost
1         lasso
2         ridge
3           mlr
4    elasticnet
5            rf


↪HyperParams  \
0  {'n_estimators': 150, 'scale_pos_weight': 5, 'learning_rate': 0.1, 'gamma': 5,
↪'subsample': 0.9}
1                                                                {'alpha': 0.
```

(continues on next page)

(continued from previous page)

```
↪32323232323232326}
2                                                                    {'alpha': 0.
↪6060606060606061}
3                                                                                ␣
↪        {}
4                                                {'alpha': 2.0, 'l1_ratio': 0,
↪'normalizer': None}
5                {'max_depth': 5, 'n_estimators': 200, 'max_features': 'auto', 'max_
↪samples': 0.75}

   TestSetMAPE  TestSetR2  InSampleMAPE  InSampleR2
0     0.115551   0.769377      0.018676    0.995667
1     0.139526   0.744279      0.065458    0.952205
2     0.140560   0.743738      0.064592    0.952812
3     0.144850   0.733553      0.064183    0.953482
4     0.259322   0.152033      0.092402    0.904719
5     0.318096  -0.916329      0.083374    0.918095
```

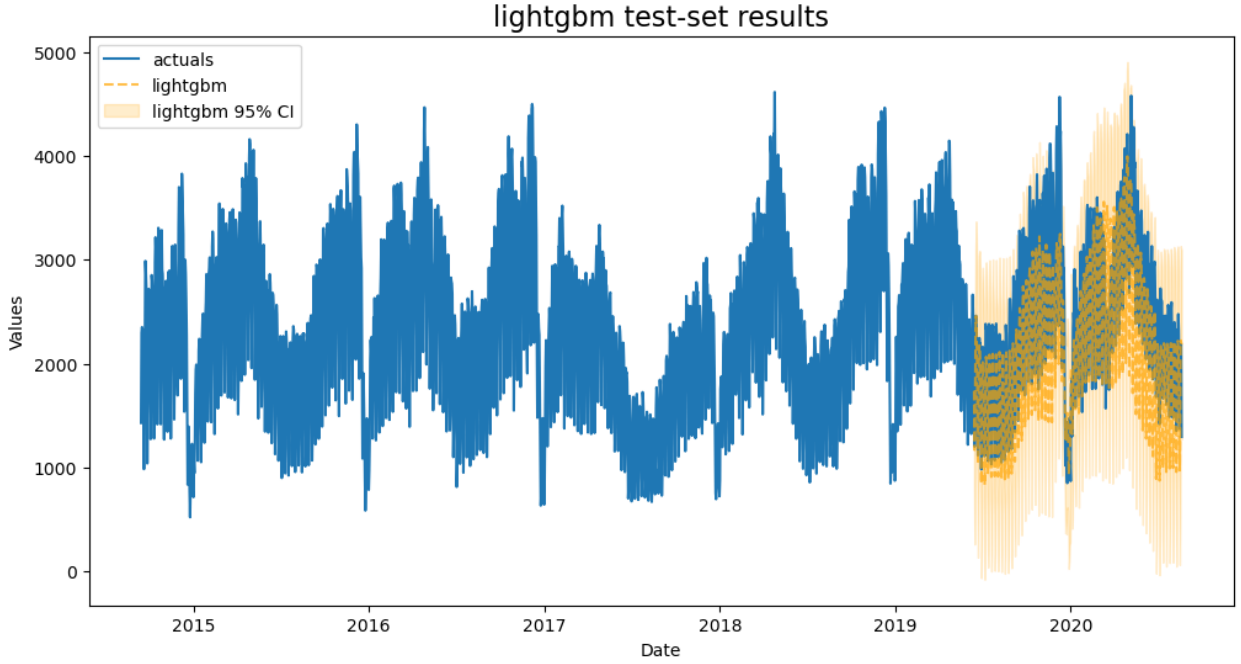
This is our best model so far using the test MAPE as the metric to determine that by. It appears to be also be highly overfit.

17.8 LightGBM

- Build a grid to tune LightGBM

[21]:
```
f.set_estimator('lightgbm')
lightgbm_grid = {
    'n_estimators':[150,200,250],
    'boosting_type':['gbdt','dart','goss'],
    'max_depth':[1,2,3],
    'learning_rate':[0.001,0.01,0.1],
    'reg_alpha':np.linspace(0,1,5),
    'reg_lambda':np.linspace(0,1,5),
    'num_leaves':np.arange(5,50,5),
}
f.ingest_grid(lightgbm_grid)
f.limit_grid_size(100,random_seed=2)
f.cross_validate(k=3)
f.auto_forecast()
```

[22]:
```
plot_test_export_summaries(f)
```



[22]:
```
  ModelNickname  \
0       xgboost
1         lasso
2         ridge
3           mlr
4      lightgbm
5    elasticnet
6            rf

                                                                                ␣
↪                                          HyperParams  \
0                                            {'n_estimators': 150, 'scale_pos_weight':␣
↪5, 'learning_rate': 0.1, 'gamma': 5, 'subsample': 0.9}
1                                                                               ␣
↪                         {'alpha': 0.32323232323232326}
2                                                                               ␣
↪                          {'alpha': 0.6060606060606061}
3                                                                               ␣
↪                                                    {}
4  {'n_estimators': 200, 'boosting_type': 'goss', 'max_depth': 3, 'learning_rate': 0.1,
↪'reg_alpha': 0.25, 'reg_lambda': 1.0, 'num_leaves': 35}
5                                                                               ␣
↪     {'alpha': 2.0, 'l1_ratio': 0, 'normalizer': None}
6                                                          {'max_depth': 5, 'n_
↪estimators': 200, 'max_features': 'auto', 'max_samples': 0.75}

   TestSetMAPE  TestSetR2  InSampleMAPE  InSampleR2
0     0.115551   0.769377      0.018676    0.995667
1     0.139526   0.744279      0.065458    0.952205
2     0.140560   0.743738      0.064592    0.952812
3     0.144850   0.733553      0.064183    0.953482
```

(continues on next page)

(continued from previous page)

```
4     0.166747   0.592318      0.050726    0.970159
5     0.259322   0.152033      0.092402    0.904719
6     0.318096  -0.916329      0.083374    0.918095
```

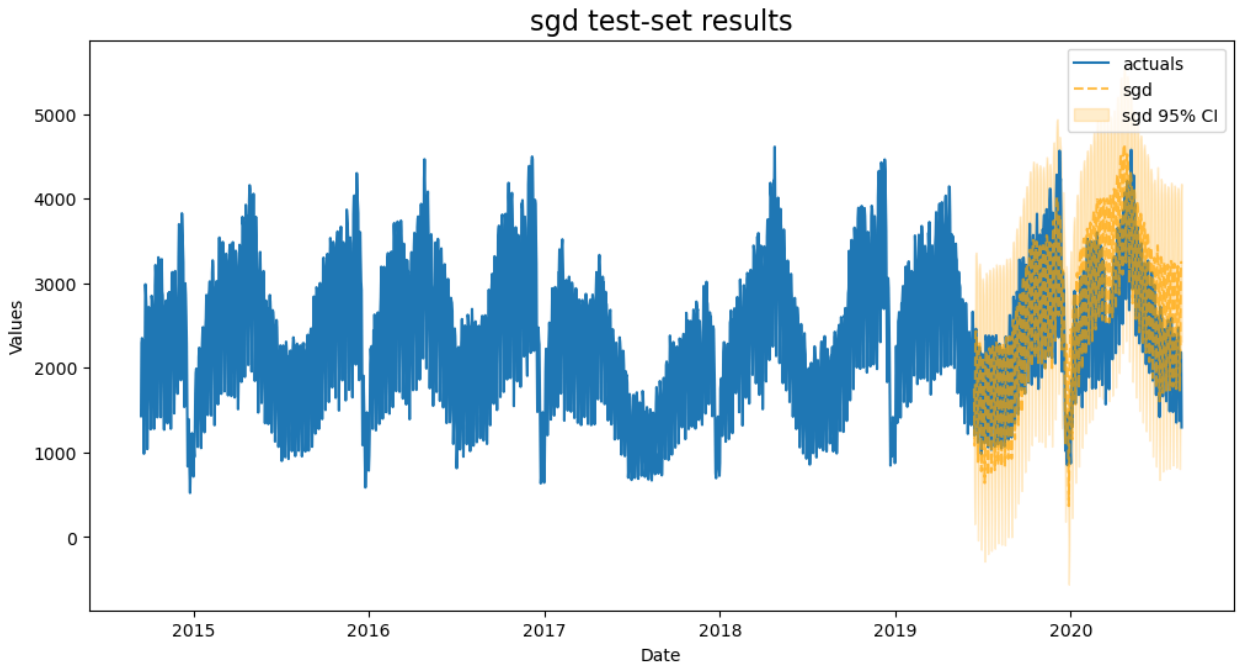
17.9 SGD

- Use the default Stochastic Gradient Descent grid

[23]:
```
f.set_estimator('sgd')
f.cross_validate(k=3)
f.auto_forecast()
```

```
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
```

[24]:
```
plot_test_export_summaries(f)
```



[24]:
```
  ModelNickname  \
0       xgboost
1         lasso
2         ridge
3           mlr
4           sgd
5      lightgbm
6    elasticnet
7            rf
```

(continues on next page)

(continued from previous page)

```
                                                                                ␣
↪                                          HyperParams  \
0                                            {'n_estimators': 150, 'scale_pos_weight':␣
↪5, 'learning_rate': 0.1, 'gamma': 5, 'subsample': 0.9}
1                                                                               ␣
↪                        {'alpha': 0.32323232323232326}
2                                                                               ␣
↪                         {'alpha': 0.6060606060606061}
3                                                                               ␣
↪                                                    {}
4                                                                    {'penalty':
↪'elasticnet', 'l1_ratio': 1, 'learning_rate': 'constant'}
5  {'n_estimators': 200, 'boosting_type': 'goss', 'max_depth': 3, 'learning_rate': 0.1,
↪'reg_alpha': 0.25, 'reg_lambda': 1.0, 'num_leaves': 35}
6                                                                               ␣
↪     {'alpha': 2.0, 'l1_ratio': 0, 'normalizer': None}
7                                                          {'max_depth': 5, 'n_
↪estimators': 200, 'max_features': 'auto', 'max_samples': 0.75}

   TestSetMAPE  TestSetR2  InSampleMAPE  InSampleR2
0     0.115551   0.769377      0.018676    0.995667
1     0.139526   0.744279      0.065458    0.952205
2     0.140560   0.743738      0.064592    0.952812
3     0.144850   0.733553      0.064183    0.953482
4     0.166129   0.609554      0.065153    0.951795
5     0.166747   0.592318      0.050726    0.970159
6     0.259322   0.152033      0.092402    0.904719
7     0.318096  -0.916329      0.083374    0.918095
```

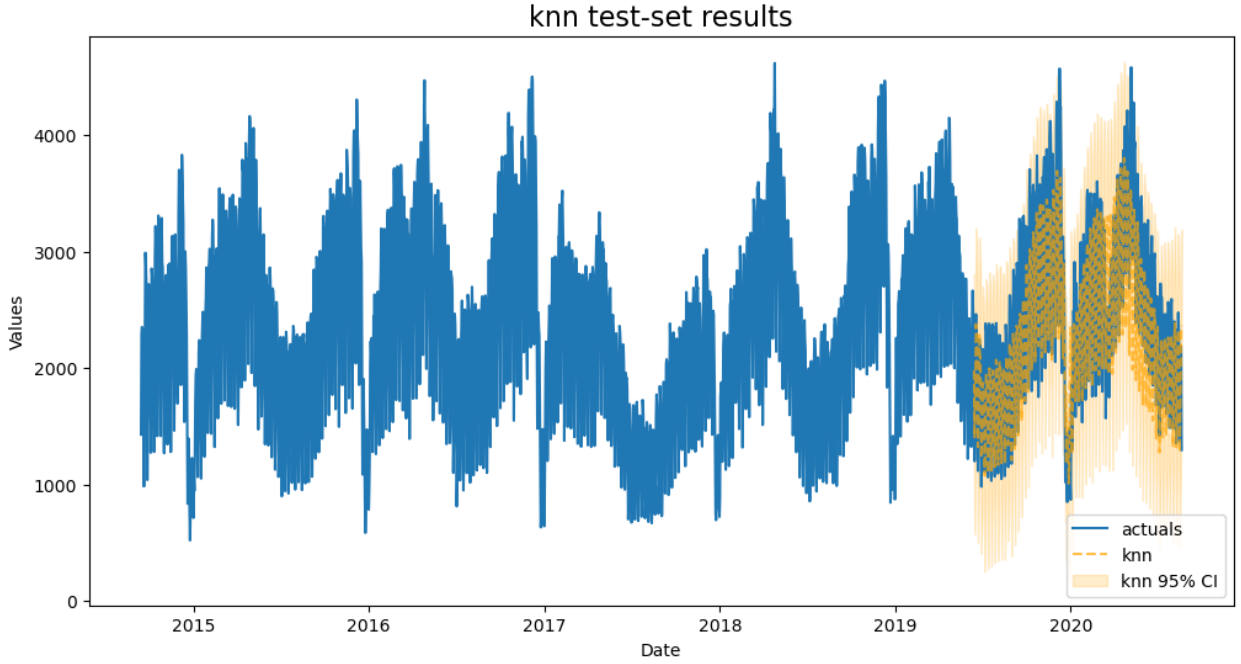
17.10 KNN

- Use the default K-nearest Neighbors grid

[25]:
```
f.set_estimator('knn')
f.cross_validate(k=3)
f.auto_forecast()
```

```
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
Finished loading model, total used 200 iterations
```

[26]:
```
plot_test_export_summaries(f)
```



[26]:
```
  ModelNickname  \
0       xgboost
1           knn
2         lasso
3         ridge
4           mlr
5           sgd
6      lightgbm
7    elasticnet
8            rf

                                                                                ␣
↪                                        HyperParams  \
0                                          {'n_estimators': 150, 'scale_pos_weight':␣
↪5, 'learning_rate': 0.1, 'gamma': 5, 'subsample': 0.9}
1                                                                               ␣
↪             {'n_neighbors': 20, 'weights': 'uniform'}
2                                                                               ␣
↪                        {'alpha': 0.32323232323232326}
3                                                                               ␣
↪                         {'alpha': 0.6060606060606061}
4                                                                               ␣
↪                                                    {}
5                                                                    {'penalty':
↪'elasticnet', 'l1_ratio': 1, 'learning_rate': 'constant'}
6  {'n_estimators': 200, 'boosting_type': 'goss', 'max_depth': 3, 'learning_rate': 0.1,
↪'reg_alpha': 0.25, 'reg_lambda': 1.0, 'num_leaves': 35}
7                                                                               ␣
↪     {'alpha': 2.0, 'l1_ratio': 0, 'normalizer': None}
8                                                          {'max_depth': 5, 'n_
↪estimators': 200, 'max_features': 'auto', 'max_samples': 0.75}
```

(continues on next page)

(continued from previous page)

```
   TestSetMAPE  TestSetR2  InSampleMAPE  InSampleR2
0     0.115551   0.769377      0.018676    0.995667
1     0.130121   0.713268      0.117198    0.873220
2     0.139526   0.744279      0.065458    0.952205
3     0.140560   0.743738      0.064592    0.952812
4     0.144850   0.733553      0.064183    0.953482
5     0.166129   0.609554      0.065153    0.951795
6     0.166747   0.592318      0.050726    0.970159
7     0.259322   0.152033      0.092402    0.904719
8     0.318096  -0.916329      0.083374    0.918095
```

KNN is our most overfit model so far, but it fits the test-set pretty well.

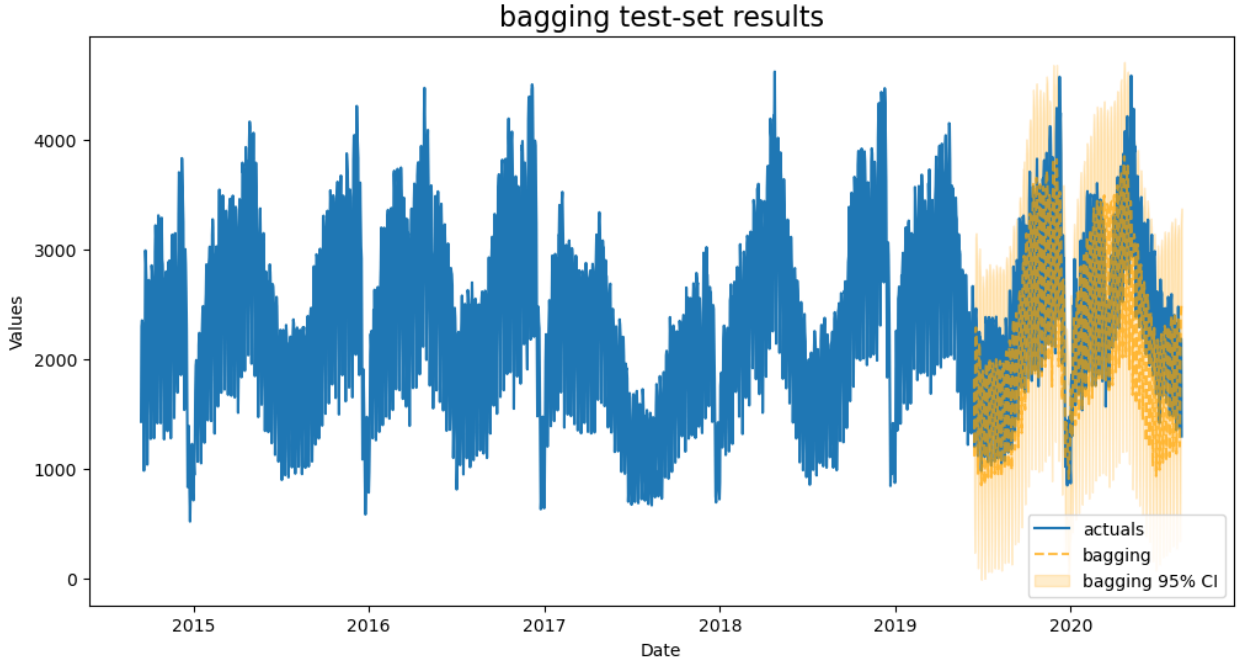
17.11 BaggingRegressor

- Like Random Forest is a bagging ensemble model using decision trees, we can use a bagging estimator that uses a three-layered mulit-level perceptron to see if the added complexity improves where the Random Forest couldn't

[28]:
```
f.set_estimator('bagging')
f.manual_forecast(
    base_estimator = MLPRegressor(
        hidden_layer_sizes=(100,100,100)
        ,solver='lbfgs'
    ),
    max_samples = 0.9,
    max_features = 0.5,
)
```

```
Finished loading model, total used 200 iterations
```

[29]:
```
plot_test_export_summaries(f)
```



[29]:
```
  ModelNickname  \
0       xgboost
1           knn
2       bagging
3         lasso
4         ridge
5           mlr
6           sgd
7      lightgbm
8    elasticnet
9            rf


                                                                                ␣
↪                                        HyperParams  \
0                                          {'n_estimators': 150, 'scale_pos_weight':␣
↪5, 'learning_rate': 0.1, 'gamma': 5, 'subsample': 0.9}
1                                                                               ␣
↪             {'n_neighbors': 20, 'weights': 'uniform'}
2                {'base_estimator': MLPRegressor(hidden_layer_sizes=(100, 100, 100),␣
↪solver='lbfgs'), 'max_samples': 0.9, 'max_features': 0.5}
3                                                                               ␣
↪                       {'alpha': 0.32323232323232326}
4                                                                               ␣
↪                        {'alpha': 0.6060606060606061}
5                                                                               ␣
↪                                                   {}
6                                                                    {'penalty':
↪'elasticnet', 'l1_ratio': 1, 'learning_rate': 'constant'}
7  {'n_estimators': 200, 'boosting_type': 'goss', 'max_depth': 3, 'learning_rate': 0.1,
↪'reg_alpha': 0.25, 'reg_lambda': 1.0, 'num_leaves': 35}
8                                                                               ␣
```

(continues on next page)

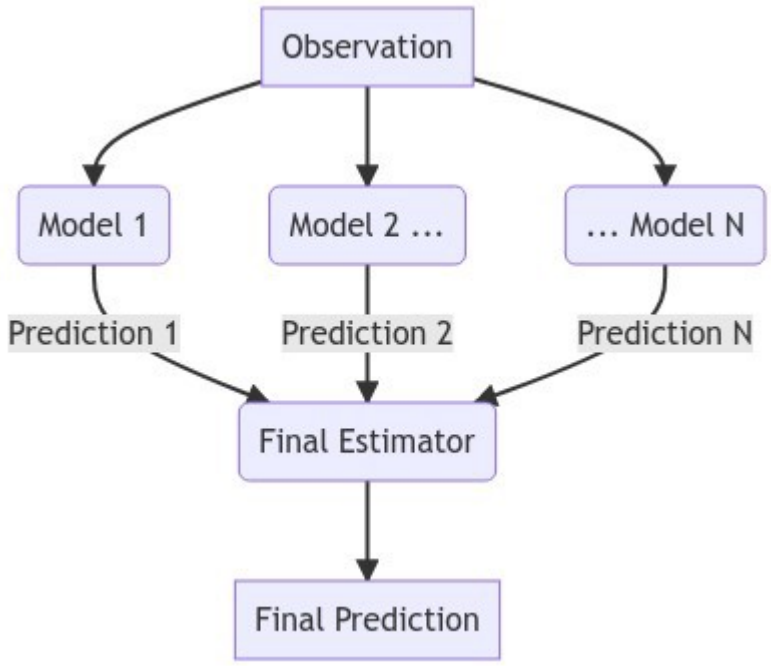
(continued from previous page)

```
↪     {'alpha': 2.0, 'l1_ratio': 0, 'normalizer': None}
9                                                          {'max_depth': 5, 'n_
↪estimators': 200, 'max_features': 'auto', 'max_samples': 0.75}

   TestSetMAPE  TestSetR2  InSampleMAPE  InSampleR2
0     0.115551   0.769377      0.018676    0.995667
1     0.130121   0.713268      0.117198    0.873220
2     0.130137   0.714881      0.050964    0.967984
3     0.139526   0.744279      0.065458    0.952205
4     0.140560   0.743738      0.064592    0.952812
5     0.144850   0.733553      0.064183    0.953482
6     0.166129   0.609554      0.065153    0.951795
7     0.166747   0.592318      0.050726    0.970159
8     0.259322   0.152033      0.092402    0.904719
9     0.318096  -0.916329      0.083374    0.918095
```

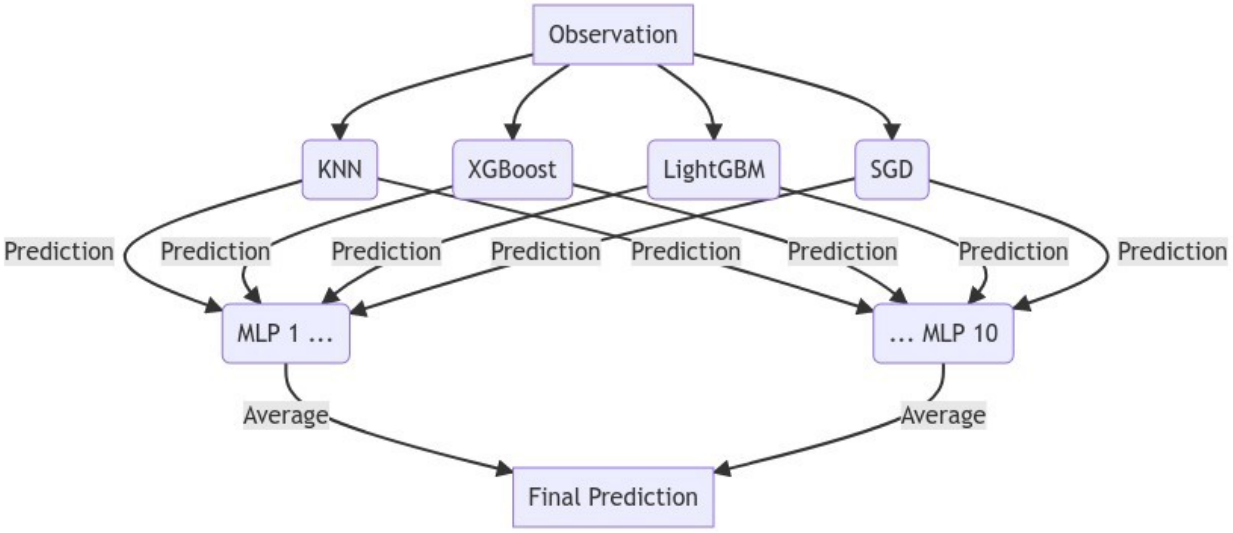
This is the best-performing model yet, with less overfitting than XGBoost and KNN.

17.12 StackingRegressor

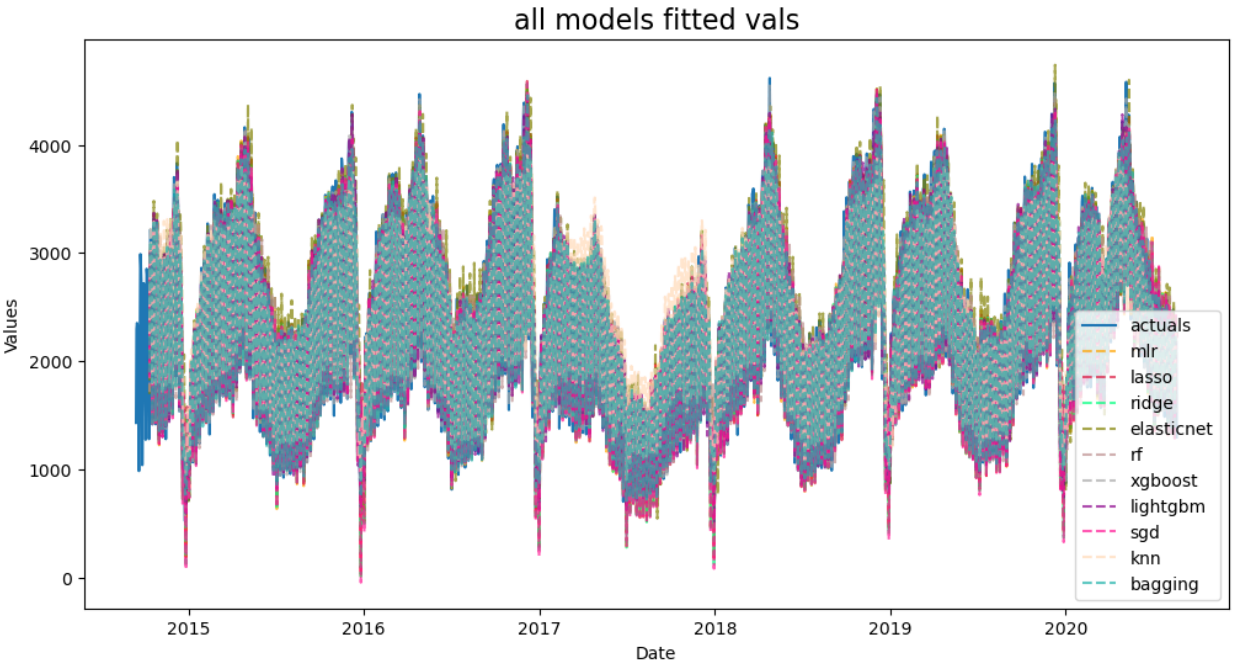
This model will use the predictions of other models as inputs to a "final estimator". It's added complexity can mean better performance, although not always.



The model that we employ uses the bagged MLP regressor from the previous section as its final estimator, and the tuned knn, xgboost, lightgbm, and sgd models as its inputs.



[30]:
```
f.plot_fitted()
plt.title('all models fitted vals',size=16)
plt.show()
```



Sometimes seeing the fitted values can give a good idea of which models to stack, as you want models that can predict both the highs and lows of the series. However, this graph didn't tell us much.

We use four of our best performing models to create the model inputs. We use a bagging estimator as the final model.

[31]:
```
f.set_estimator('stacking')
results = f.export('model_summaries')
estimators = [
    'knn',
    'xgboost',
```

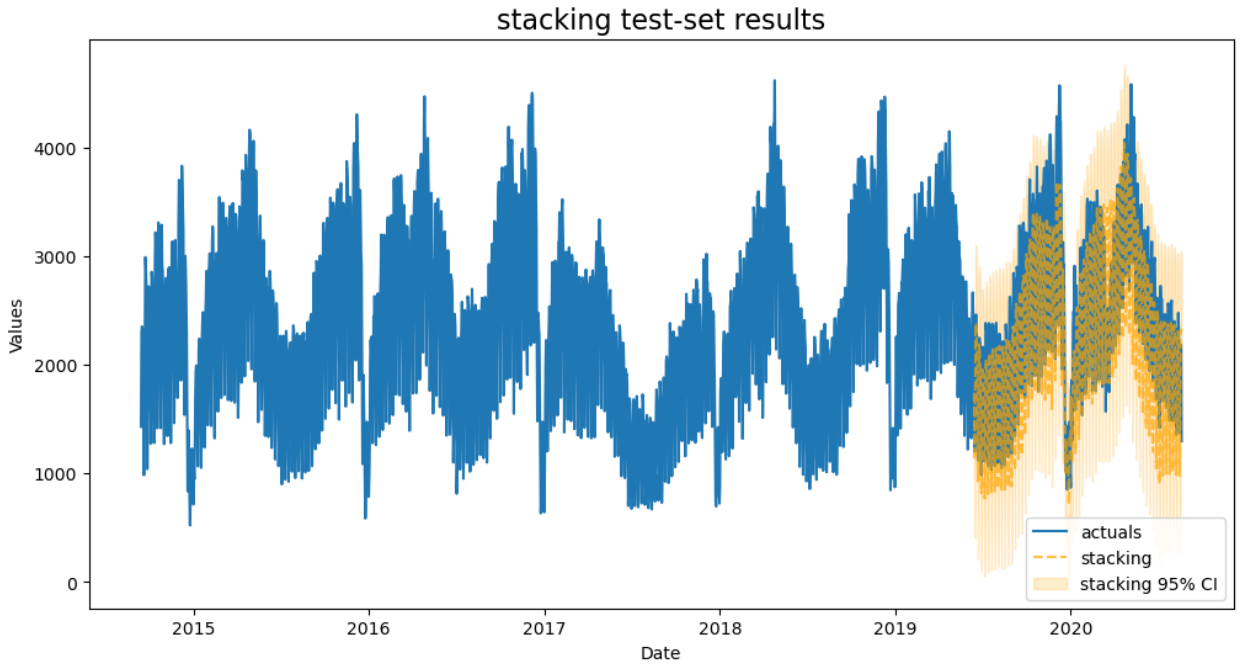
(continues on next page)

(continued from previous page)

```
    'lightgbm',
    'sgd',
]

mlp_stack(f,estimators,call_me='stacking')
```

```
Finished loading model, total used 200 iterations
```

[32]:
```
plot_test_export_summaries(f)
```



[32]:
```
   ModelNickname  \
0        xgboost
1       stacking
2            knn
3        bagging
4          lasso
5          ridge
6            mlr
7            sgd
8       lightgbm
9     elasticnet
10            rf


↪
↪
↪
↪
↪                                                  HyperParams  \
0
↪
```

(continues on next page)

(continued from previous page)

```
↪                                                                                ␣
↪                                                                                ␣
↪                                                    {'n_estimators': 150, 'scale_
↪pos_weight': 5, 'learning_rate': 0.1, 'gamma': 5, 'subsample': 0.9}
1   {'estimators': [('knn', KNeighborsRegressor(n_neighbors=20)), ('xgboost',␣
↪XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=5, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
          ...
2                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                        {'n_neighbors': 20, 'weights': 'uniform'}
3                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                           {'base_estimator': MLPRegressor(hidden_layer_sizes=(100,␣
↪100, 100), solver='lbfgs'), 'max_samples': 0.9, 'max_features': 0.5}
4                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                    {'alpha': 0.32323232323232326}
5                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                     {'alpha': 0.6060606060606061}
6                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                {}
7                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                               {
↪'penalty': 'elasticnet', 'l1_ratio': 1, 'learning_rate': 'constant'}
8                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
↪                                                                                ␣
```

(continues on next page)

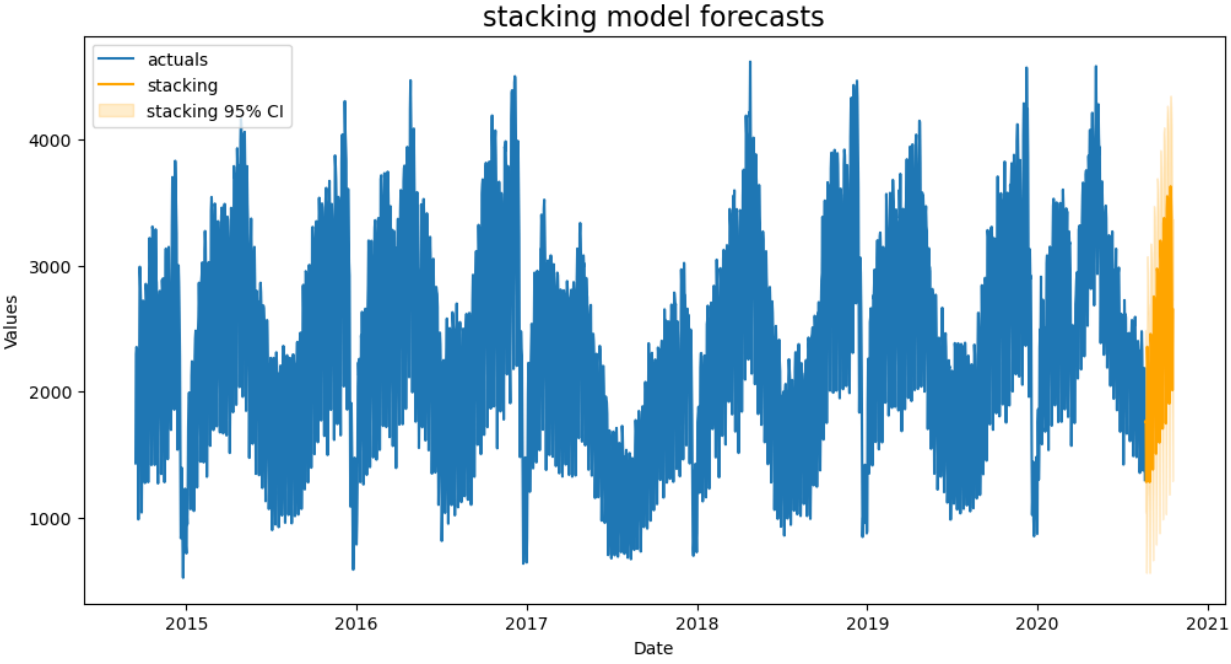
(continued from previous page)

```
↪             {'n_estimators': 200, 'boosting_type': 'goss', 'max_depth': 3, 'learning_
↪rate': 0.1, 'reg_alpha': 0.25, 'reg_lambda': 1.0, 'num_leaves': 35}
9                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                 {'alpha': 2.0, 'l1_ratio': 0, 'normalizer': None}
10                                                                                   ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                  {'max_depth': 5,
↪ 'n_estimators': 200, 'max_features': 'auto', 'max_samples': 0.75}

    TestSetMAPE  TestSetR2  InSampleMAPE  InSampleR2
0      0.115551   0.769377      0.018676    0.995667
1      0.122353   0.784423      0.040845    0.979992
2      0.130121   0.713268      0.117198    0.873220
3      0.130137   0.714881      0.050964    0.967984
4      0.139526   0.744279      0.065458    0.952205
5      0.140560   0.743738      0.064592    0.952812
6      0.144850   0.733553      0.064183    0.953482
7      0.166129   0.609554      0.065153    0.951795
8      0.166747   0.592318      0.050726    0.970159
9      0.259322   0.152033      0.092402    0.904719
10     0.318096  -0.916329      0.083374    0.918095
```

This is our best model with lower amounts of overfitting than some of our other models. We will choose it to make our final forecasts with.

17.13 Plot Forecast

[33]:
```
f.plot('stacking',ci=True)
plt.title('stacking model forecasts',size=16)
plt.show()
```


stacking model forecasts
actuals
stacking
stacking 95% CI
4000
3000
2000
1000
Values
2015
2016
2017
2018
2019
2020
2021
Date

[ ]:

CHAPTER

EIGHTEEN

STACKING MODELS

- Scalecast allows you to stack models of different classes together.
- You can add the predictions of any given model to the `Forecaster` object as a covariate, which scalecast refers to as "signals". This is through the use of the `Forecaster.add_signals()` method. See the documentation.
- These signals can come from any model class available in scalecast and are treated the same as any other covariate. They can be combined with other covariates (such as series lags, seasonal representations, and trends). They can also be added to an `MVForecaster` object for multivariate forecasting. The signals from Meta Prophet or LinkedIn Silverkite, which add holiday effects to models, can be added to the objects to capture the uniqueness of these models' specifications.
- We will use symmetric mean percentage error (SMAPE) to measure the performance of each model in this notebook.
- Requirements:
- `scalecast>=0.17.9`
- `tensorflow`
- `shap`
- Data source: M4

[1]:
```
import pandas as pd
import numpy as np
from scalecast.Forecaster import Forecaster
from scalecast.util import metrics
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
def read_data(idx = 'H1', cis = True, metrics = ['smape']):
    info = pd.read_csv(
        'M4-info.csv',
        index_col=0,
        parse_dates=['StartingDate'],
        dayfirst=True,
    )
    train = pd.read_csv(
        f'Hourly-train.csv',
        index_col=0,
    ).loc[idx]
    test = pd.read_csv(
        f'Hourly-test.csv',
```

(continues on next page)

(continued from previous page)

```
        index_col=0,
    ).loc[idx]
    y = train.values
    sd = info.loc[idx,'StartingDate']
    fcst_horizon = info.loc[idx,'Horizon']
    cd = pd.date_range(
        start = sd,
        freq = 'H',
        periods = len(y),
    )
    f = Forecaster(
        y = y,
        current_dates = cd,
        future_dates = fcst_horizon,
        test_length = fcst_horizon,
        cis = cis,
        metrics = metrics,
    )

    return f, test.values
```

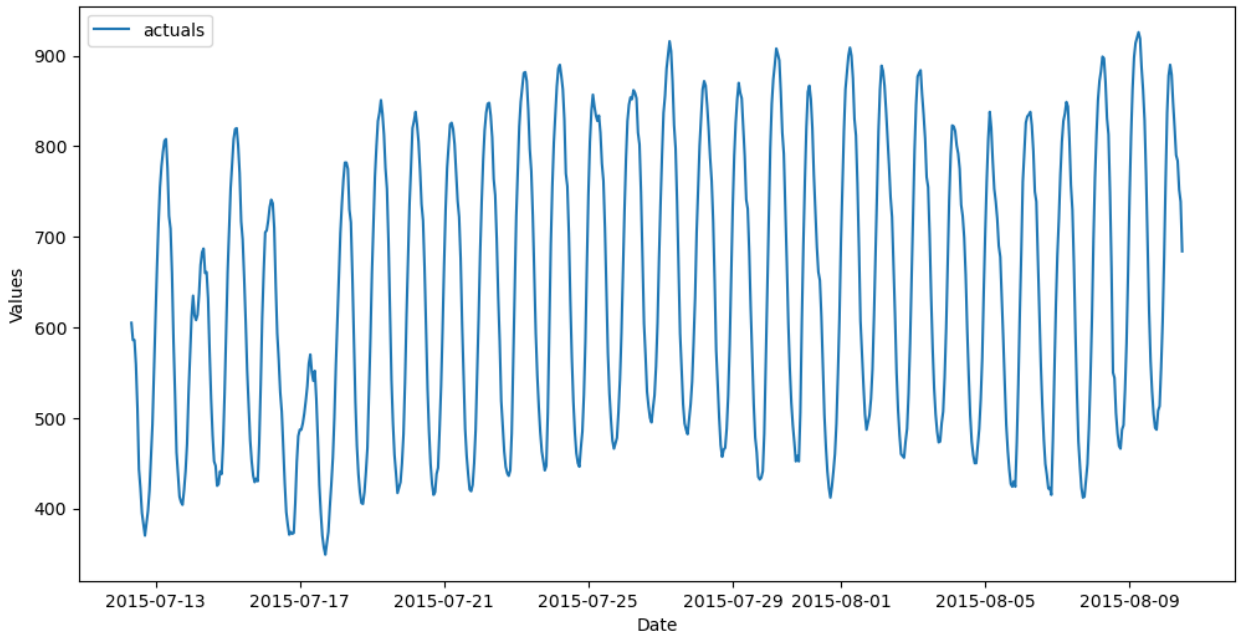
[3]:
```
f, test_set = read_data()
f
```

[3]:
```
Forecaster(
    DateStartActuals=2015-07-12T08:00:00.000000000
    DateEndActuals=2015-08-10T11:00:00.000000000
    Freq=H
    N_actuals=700
    ForecastLength=48
    Xvars=[]
    TestLength=48
    ValidationMetric=smape
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

We are using the H1 series from the M4 competition, but you can change the value passed to the `idx` argument in the function above to test this same analysis with any hourly series in the dataset.

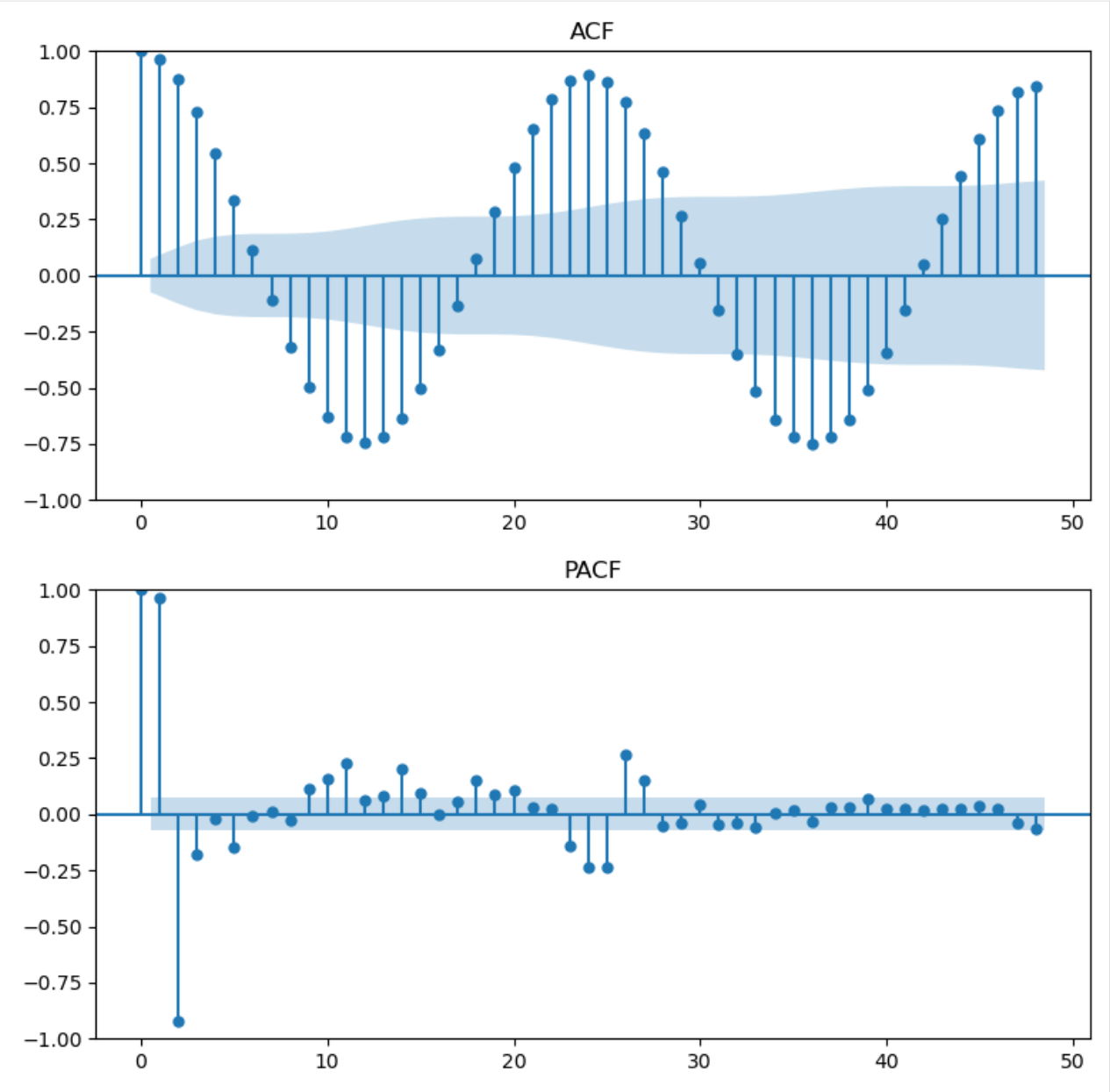
18.1 EDA

[4]:
```
f.plot()
plt.show()
```



18.1.1 ACF/PACF at Series Level

[5]:
```
figs, axs = plt.subplots(2, 1,figsize=(9,9))
f.plot_acf(ax=axs[0],title='ACF',lags=48)
f.plot_pacf(ax=axs[1],title='PACF',lags=48,method='ywm')
plt.show()
```



18.1.2 Augmented Dickey-Fuller Test

[6]:
```
critical_pval = 0.05
print('Augmented Dickey-Fuller results:')
stat, pval, _, _, _, _ = f.adf_test(full_res=True)
print('the test-stat value is: {:.2f}'.format(stat))
print('the p-value is {:.4f}'.format(pval))
print('the series is {}'.format('stationary' if pval < critical_pval else 'not stationary
↪'))
print('-'*100)
```

```
Augmented Dickey-Fuller results:
```

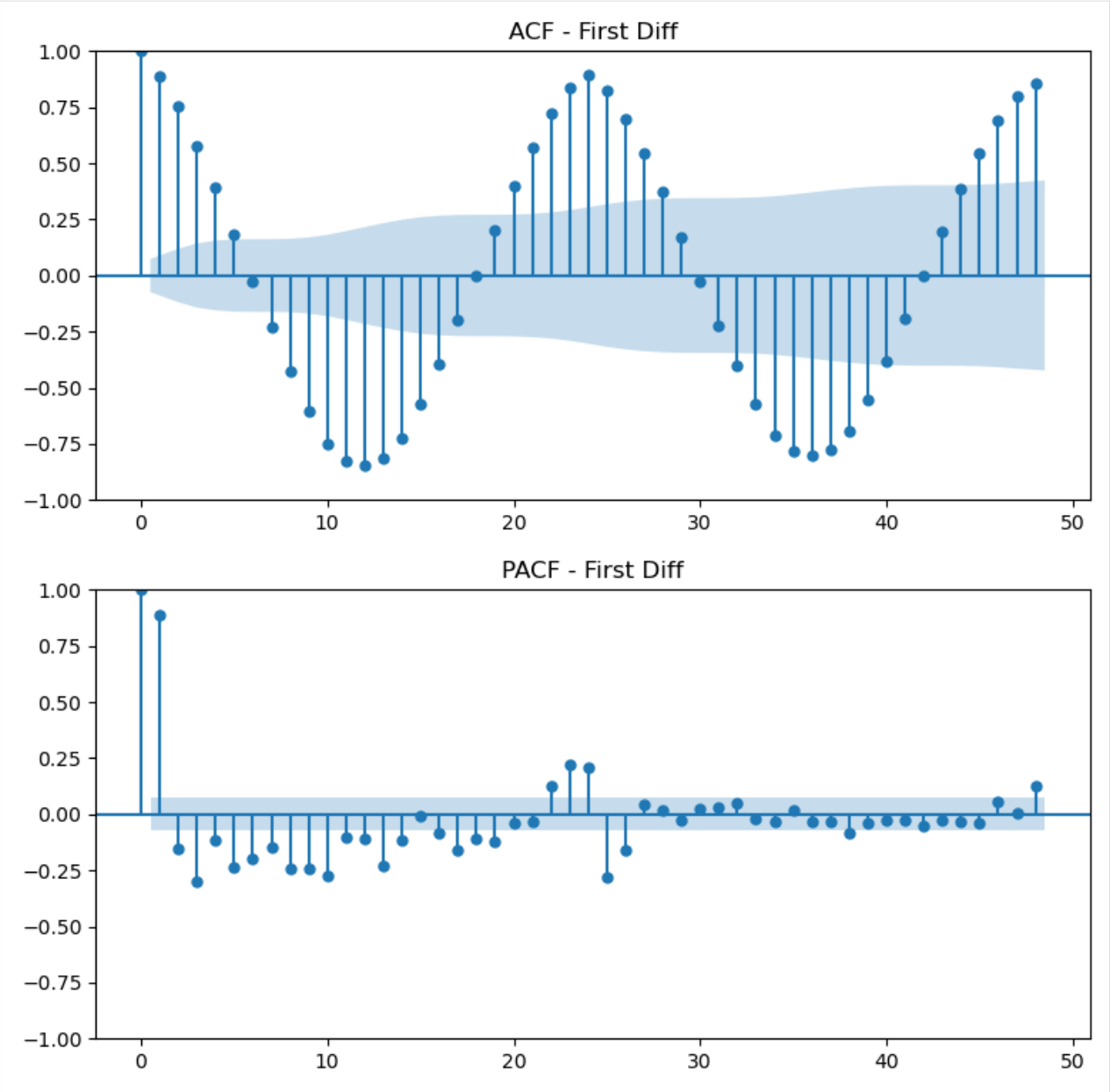
(continues on next page)

(continued from previous page)

```
the test-stat value is: -2.06
the p-value is 0.2623
the series is not stationary
-----------------------------------------------------------------------------------------
↪-----------
```

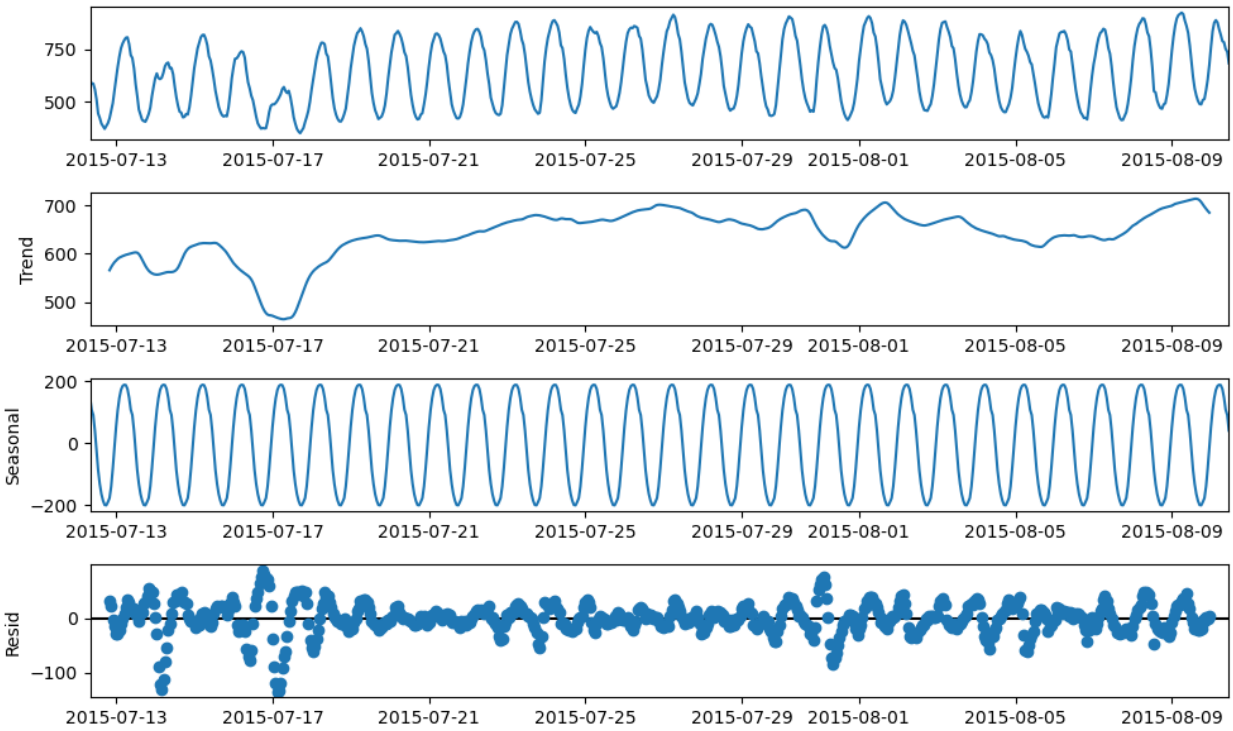
18.1.3 ACF/PACF at Series First Difference

[7]:
```
figs, axs = plt.subplots(2, 1,figsize=(9,9))
f.plot_acf(ax=axs[0],title='ACF - First Diff',lags=48,diffy=1)
f.plot_pacf(ax=axs[1],title='PACF - First Diff',lags=48,diffy=1,method='ywm')
plt.show()
```



18.1.4 Seasonal Decomp

[8]:
```
plt.rc("figure",figsize=(10,6))
f.seasonal_decompose().plot()
plt.show()
```



18.2 Naive

- This will serve as a benchmark model
- It will propagate the last 24 observations in a "seasonal naive" model

[9]:
```
f.set_estimator('naive')
f.manual_forecast(seasonal=True)
```

18.3 ARIMA

18.3.1 Manual ARIMA: (5,1,4) x (1,1,1,24)

[10]:
```
f.set_estimator('arima')
f.manual_forecast(
    order = (5,1,4),
    seasonal_order = (1,1,1,24),
    call_me = 'manual_arima',
)
```

18.4 RNN

18.4.1 Tanh Activation

[11]:
```
f.set_estimator('rnn')
f.manual_forecast(
    lags = 48,
    layers_struct=[
        ('LSTM',{'units':100,'activation':'tanh'}),
        ('LSTM',{'units':100,'activation':'tanh'}),
        ('LSTM',{'units':100,'activation':'tanh'}),
    ],
    optimizer = 'Adam',
    epochs = 15,
    plot_loss = True,
    validation_split=0.2,
    call_me = 'rnn_tanh_activation',
)
```

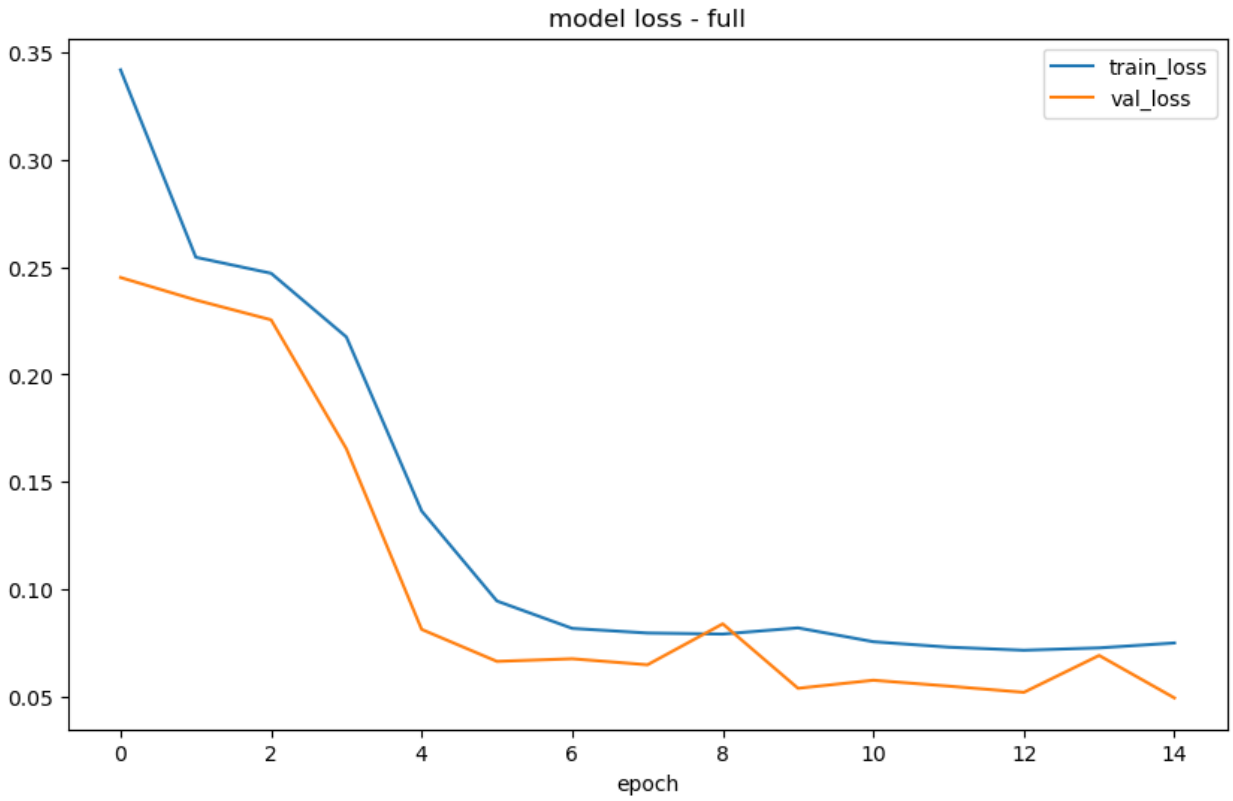
```
2023-04-11 11:12:28.428651: I tensorflow/core/platform/cpu_feature_guard.cc:193] This␣
↪TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use␣
↪the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler␣
↪flags.
2023-04-11 11:12:30.032138: I tensorflow/core/platform/cpu_feature_guard.cc:193] This␣
↪TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use␣
↪the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler␣
↪flags.
```

```
Epoch 1/15
14/14 [==============================] - 4s 110ms/step - loss: 0.3419 - val_loss: 0.2452
Epoch 2/15
14/14 [==============================] - 1s 60ms/step - loss: 0.2546 - val_loss: 0.2347
Epoch 3/15
14/14 [==============================] - 1s 60ms/step - loss: 0.2472 - val_loss: 0.2255
Epoch 4/15
14/14 [==============================] - 1s 60ms/step - loss: 0.2175 - val_loss: 0.1654
Epoch 5/15
14/14 [==============================] - 1s 60ms/step - loss: 0.1365 - val_loss: 0.0813
Epoch 6/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0946 - val_loss: 0.0664
Epoch 7/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0818 - val_loss: 0.0677
Epoch 8/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0796 - val_loss: 0.0649
Epoch 9/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0792 - val_loss: 0.0839
Epoch 10/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0820 - val_loss: 0.0539
Epoch 11/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0756 - val_loss: 0.0577
Epoch 12/15
```

(continues on next page)

(continued from previous page)

```
14/14 [==============================] - 1s 60ms/step - loss: 0.0731 - val_loss: 0.0549
Epoch 13/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0716 - val_loss: 0.0520
Epoch 14/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0727 - val_loss: 0.0692
Epoch 15/15
14/14 [==============================] - 1s 60ms/step - loss: 0.0750 - val_loss: 0.0494
1/1 [==============================] - 1s 597ms/step
```

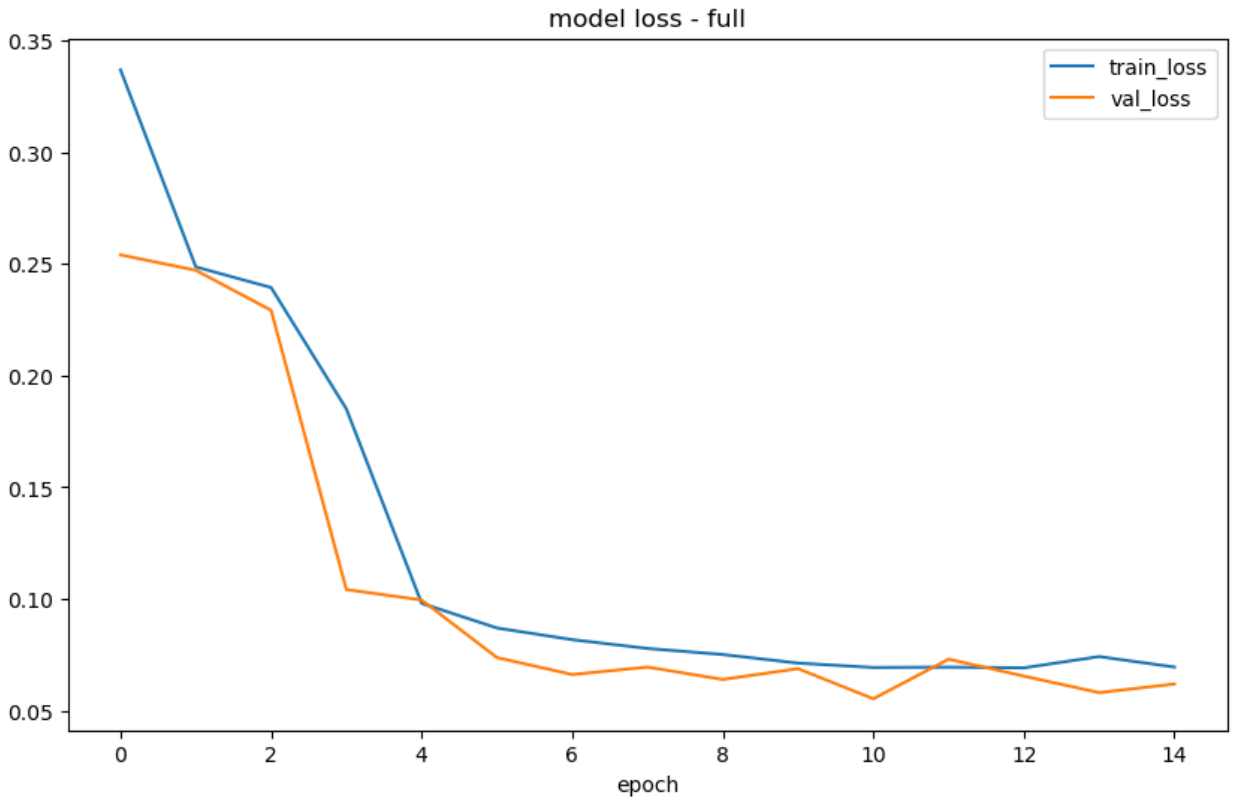



```
Epoch 1/15
16/16 [==============================] - 4s 100ms/step - loss: 0.3369 - val_loss: 0.2541
Epoch 2/15
16/16 [==============================] - 1s 58ms/step - loss: 0.2487 - val_loss: 0.2471
Epoch 3/15
16/16 [==============================] - 1s 58ms/step - loss: 0.2394 - val_loss: 0.2292
Epoch 4/15
16/16 [==============================] - 1s 58ms/step - loss: 0.1850 - val_loss: 0.1042
Epoch 5/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0980 - val_loss: 0.0995
Epoch 6/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0870 - val_loss: 0.0737
Epoch 7/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0817 - val_loss: 0.0661
Epoch 8/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0778 - val_loss: 0.0695
Epoch 9/15
```

(continues on next page)

(continued from previous page)

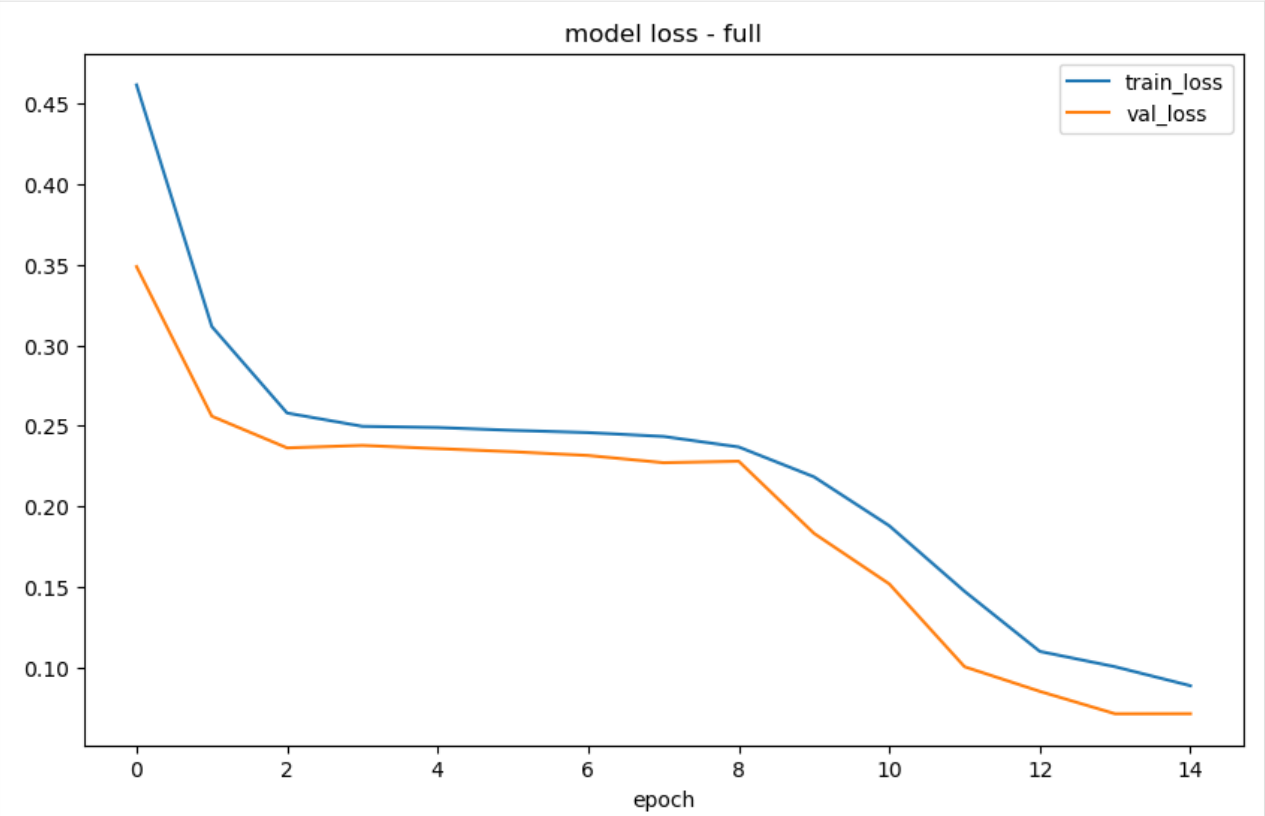
```
16/16 [==============================] - 1s 58ms/step - loss: 0.0751 - val_loss: 0.0639
Epoch 10/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0712 - val_loss: 0.0687
Epoch 11/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0693 - val_loss: 0.0553
Epoch 12/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0694 - val_loss: 0.0730
Epoch 13/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0691 - val_loss: 0.0654
Epoch 14/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0742 - val_loss: 0.0580
Epoch 15/15
16/16 [==============================] - 1s 58ms/step - loss: 0.0695 - val_loss: 0.0619
1/1 [==============================] - 1s 750ms/step
19/19 [==============================] - 0s 18ms/step
```



18.4.2 Relu Activation

[12]:
```
f.manual_forecast(
    lags = 48,
    layers_struct=[
        ('LSTM',{'units':100,'activation':'relu'}),
        ('LSTM',{'units':100,'activation':'relu'}),
        ('LSTM',{'units':100,'activation':'relu'}),
    ],
    optimizer = 'Adam',
    epochs = 15,
    plot_loss = True,
    validation_split=0.2,
    call_me = 'rnn_relu_activation',
)
```

```
Epoch 1/15
14/14 [==============================] - 3s 79ms/step - loss: 0.4613 - val_loss: 0.3487
Epoch 2/15
14/14 [==============================] - 1s 58ms/step - loss: 0.3115 - val_loss: 0.2559
Epoch 3/15
14/14 [==============================] - 1s 58ms/step - loss: 0.2579 - val_loss: 0.2363
Epoch 4/15
14/14 [==============================] - 1s 58ms/step - loss: 0.2497 - val_loss: 0.2378
Epoch 5/15
14/14 [==============================] - 1s 58ms/step - loss: 0.2489 - val_loss: 0.2359
Epoch 6/15
14/14 [==============================] - 1s 59ms/step - loss: 0.2472 - val_loss: 0.2339
Epoch 7/15
14/14 [==============================] - 1s 58ms/step - loss: 0.2458 - val_loss: 0.2316
Epoch 8/15
14/14 [==============================] - 1s 58ms/step - loss: 0.2434 - val_loss: 0.2271
Epoch 9/15
14/14 [==============================] - 1s 58ms/step - loss: 0.2369 - val_loss: 0.2280
Epoch 10/15
14/14 [==============================] - 1s 58ms/step - loss: 0.2184 - val_loss: 0.1833
Epoch 11/15
14/14 [==============================] - 1s 58ms/step - loss: 0.1880 - val_loss: 0.1520
Epoch 12/15
14/14 [==============================] - 1s 58ms/step - loss: 0.1474 - val_loss: 0.1007
Epoch 13/15
14/14 [==============================] - 1s 58ms/step - loss: 0.1102 - val_loss: 0.0854
Epoch 14/15
14/14 [==============================] - 1s 58ms/step - loss: 0.1007 - val_loss: 0.0715
Epoch 15/15
14/14 [==============================] - 1s 58ms/step - loss: 0.0889 - val_loss: 0.0715
1/1 [==============================] - 0s 232ms/step
```

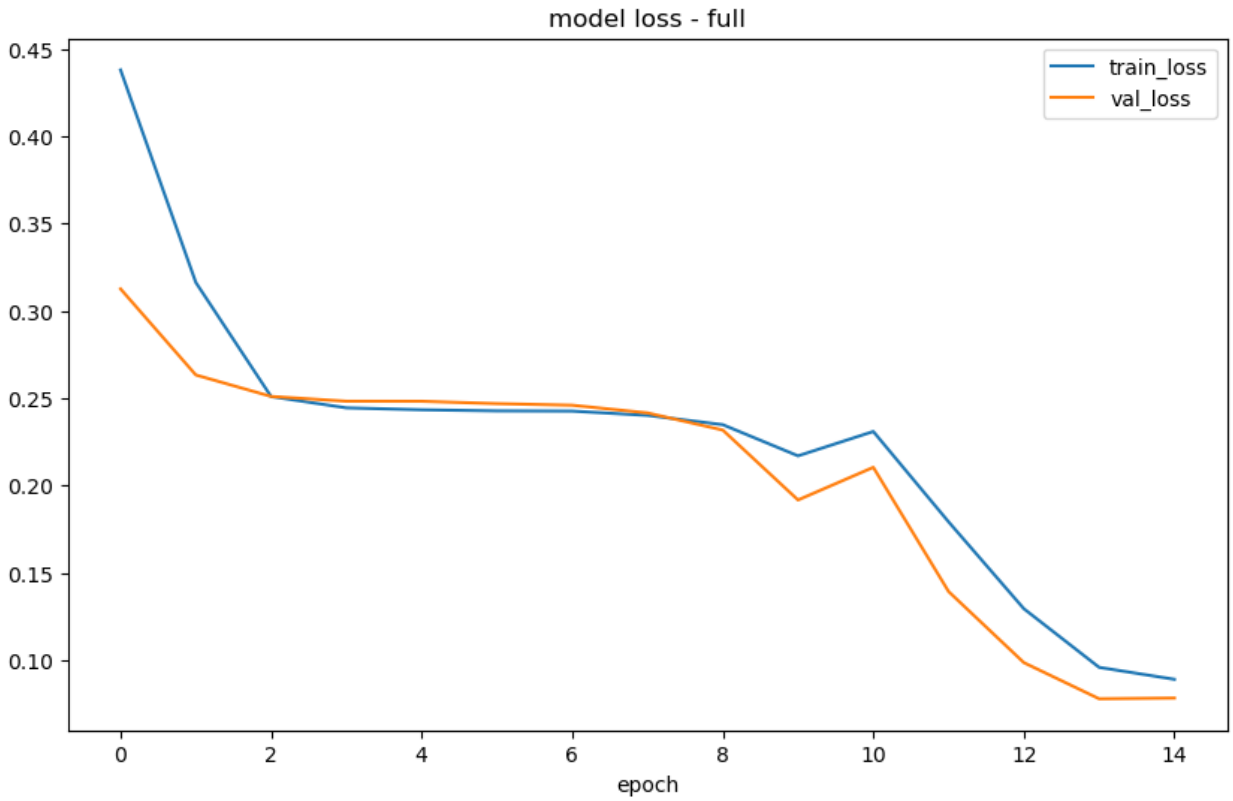



```
Epoch 1/15
16/16 [==============================] - 3s 75ms/step - loss: 0.4379 - val_loss: 0.3125
Epoch 2/15
16/16 [==============================] - 1s 56ms/step - loss: 0.3162 - val_loss: 0.2632
Epoch 3/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2509 - val_loss: 0.2509
Epoch 4/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2444 - val_loss: 0.2482
Epoch 5/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2433 - val_loss: 0.2482
Epoch 6/15
16/16 [==============================] - 1s 57ms/step - loss: 0.2427 - val_loss: 0.2468
Epoch 7/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2426 - val_loss: 0.2459
Epoch 8/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2401 - val_loss: 0.2415
Epoch 9/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2348 - val_loss: 0.2317
Epoch 10/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2169 - val_loss: 0.1917
Epoch 11/15
16/16 [==============================] - 1s 56ms/step - loss: 0.2309 - val_loss: 0.2104
Epoch 12/15
16/16 [==============================] - 1s 56ms/step - loss: 0.1792 - val_loss: 0.1393
Epoch 13/15
16/16 [==============================] - 1s 56ms/step - loss: 0.1294 - val_loss: 0.0986
```

(continues on next page)

(continued from previous page)

```
Epoch 14/15
16/16 [==============================] - 1s 56ms/step - loss: 0.0959 - val_loss: 0.0779
Epoch 15/15
16/16 [==============================] - 1s 56ms/step - loss: 0.0890 - val_loss: 0.0783
1/1 [==============================] - 0s 232ms/step
19/19 [==============================] - 0s 18ms/step
```



18.5 Prophet

[13]:
```
f.set_estimator('prophet')
f.manual_forecast()
```

```
11:13:36 - cmdstanpy - INFO - Chain [1] start processing
11:13:36 - cmdstanpy - INFO - Chain [1] done processing
11:13:36 - cmdstanpy - INFO - Chain [1] start processing
11:13:36 - cmdstanpy - INFO - Chain [1] done processing
```

18.6 Compare Results

[14]:
```
results = f.export(determine_best_by='TestSetSMAPE')
ms = results['model_summaries']
ms[
    [
        'ModelNickname',
        'TestSetLength',
        'TestSetSMAPE',
        'InSampleSMAPE',
    ]
]
```

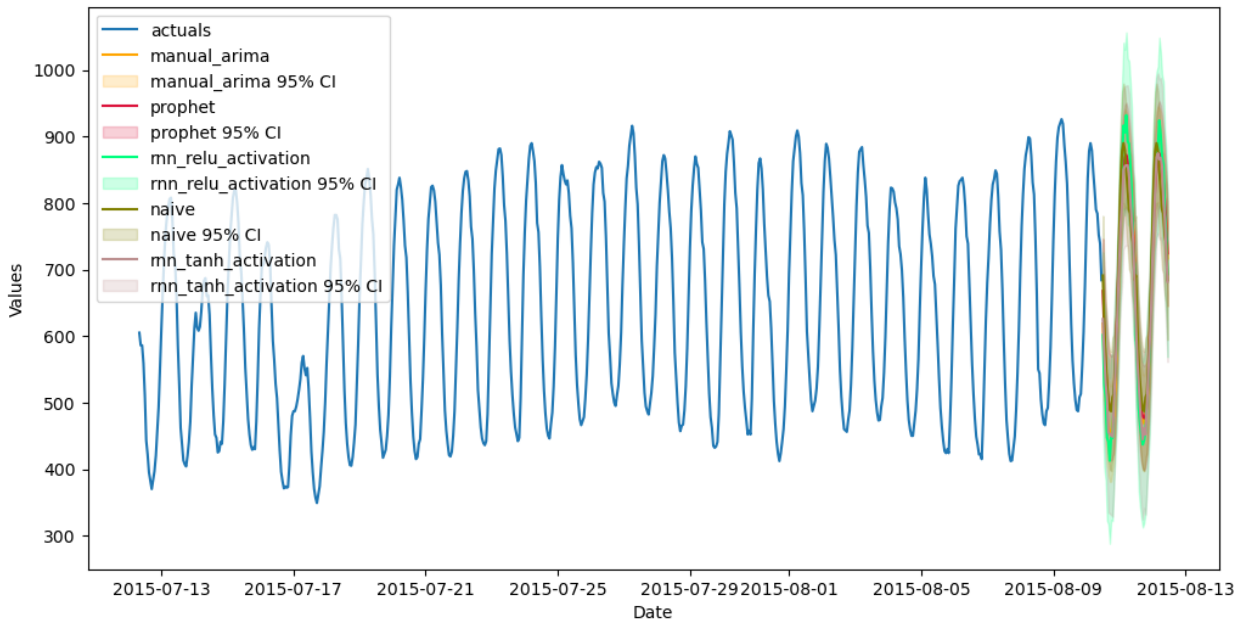
[14]:

	ModelNickname	TestSetLength	TestSetSMAPE	InSampleSMAPE
0	manual_arima	48	0.046551	0.017995
1	prophet	48	0.047237	0.043859
2	rnn_relu_activation	48	0.082511	0.081514
3	naive	48	0.093079	0.067697
4	rnn_tanh_activation	48	0.096998	0.059914

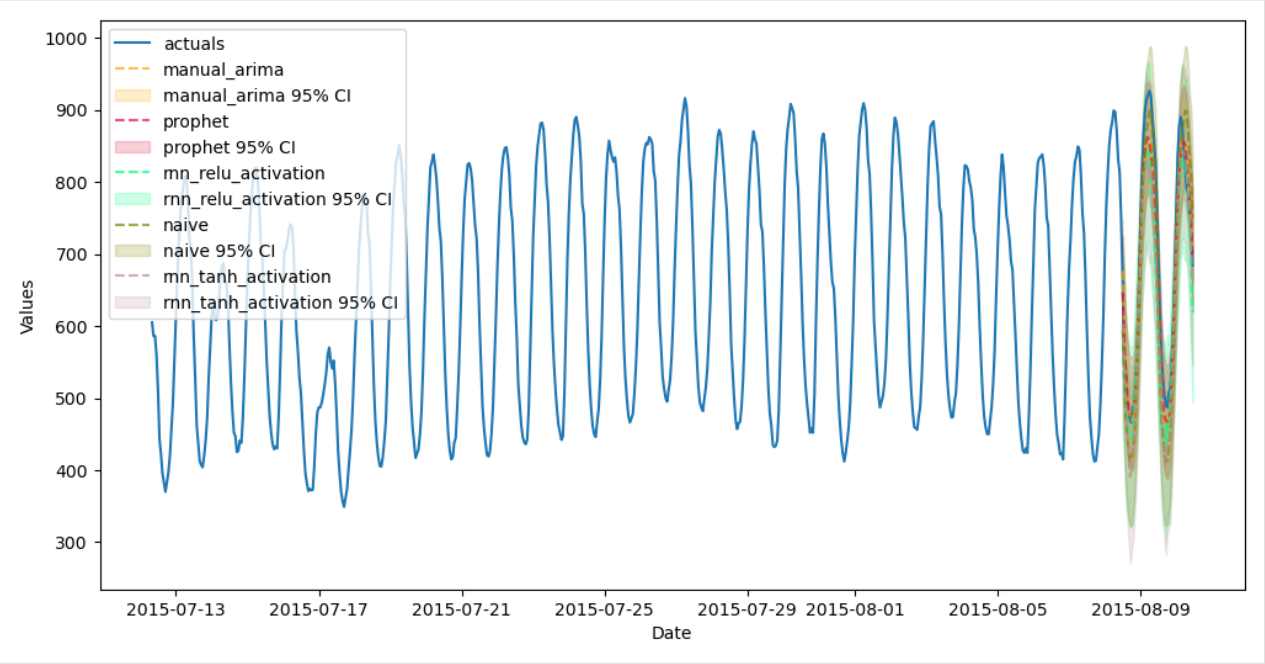
Using the last 48 observations in the `Forecaster` object to test each model, the arima model performed the best.

18.7 Plot Results

[15]:
```
f.plot(order_by="TestSetSMAPE",ci=True)
plt.show()
```



[16]:
```
f.plot_test_set(order_by="TestSetSMAPE",ci=True)
plt.show()
```



18.8 Stack Models

- To stack models in scalecast, you can either use the StackingRegressor from scikit-learn, or you can add the predictions from already-evaluated models into the `Forecaster` object using `Forecaster.add_signals()`. The latter approach is more advantageous as it can do everything that can be done with the StackingRegressor, but it can use non scikit-learn model classes, can more flexibly use other regressors, and is easier to tune.
- In the below example, we are using the signals generated from two LSTM models, an ARIMA model, a naive model, and Meta Prophet. We will also add the last 48 series lags to the object.
- One key point in the function below: we are specifying the `train_only` argument as False. This means that our test set will be compromised as we will introduce leakage from the other models. I chose to leave it False because this approach will ultimately be tested with a separate out-of-sample test set. The rule of thumb around this is to make this argument True if you want to report accurate test-set metrics but leave False when you want to deliver a Forecast into a future horizon. You can run this function twice with each option specified if you want to do both–run first time with `train_only=True`, evaluate models, and check the test-set metrics. Then, rerun with `train_only=False` and re-evaluate models to deliver future point predictions.

[17]:
```
f.add_ar_terms(48)
f.add_signals(
    f.history.keys(),
    #train_only = True, # uncomment to avoid leakage into the test set
)
f.set_estimator('catboost')
f
```

[17]:
```
Forecaster(
    DateStartActuals=2015-07-12T08:00:00.000000000
    DateEndActuals=2015-08-10T11:00:00.000000000
    Freq=H
```

(continues on next page)

(continued from previous page)

```
    N_actuals=700
    ForecastLength=48
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11',
↪ 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21', 'AR22
↪', 'AR23', 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30', 'AR31', 'AR32',
↪'AR33', 'AR34', 'AR35', 'AR36', 'AR37', 'AR38', 'AR39', 'AR40', 'AR41', 'AR42', 'AR43',
↪ 'AR44', 'AR45', 'AR46', 'AR47', 'AR48', 'signal_naive', 'signal_manual_arima',
↪'signal_rnn_tanh_activation', 'signal_rnn_relu_activation', 'signal_prophet']
    TestLength=48
    ValidationMetric=smape
    ForecastsEvaluated=['naive', 'manual_arima', 'rnn_tanh_activation', 'rnn_relu_
↪activation', 'prophet']
    CILevel=0.95
    CurrentEstimator=catboost
    GridsFile=Grids
)
```

Now we can train three catboost models: - One with all added regressors (the model signals and lags)
- One with just the model signals
- One with just the series lags

[18]:
```
f.manual_forecast(
    Xvars='all',
    call_me='catboost_all_reg',
    verbose = False,
)
f.save_feature_importance(method = 'shap') # it would be interesting to see the shapley␣
↪scores (later)
f.manual_forecast(
    Xvars=[x for x in f.get_regressor_names() if x.startswith('AR')],
    call_me = 'catboost_lags_only',
    verbose = False,
)
f.manual_forecast(
    Xvars=[x for x in f.get_regressor_names() if not x.startswith('AR')],
    call_me = 'catboost_signals_only',
    verbose = False,
)
```

[19]:
```
results = f.export(determine_best_by='TestSetSMAPE')
ms = results['model_summaries']
ms[
    [
        'ModelNickname',
        'TestSetLength',
        'TestSetSMAPE',
        'InSampleSMAPE',
    ]
]
```

[19]:

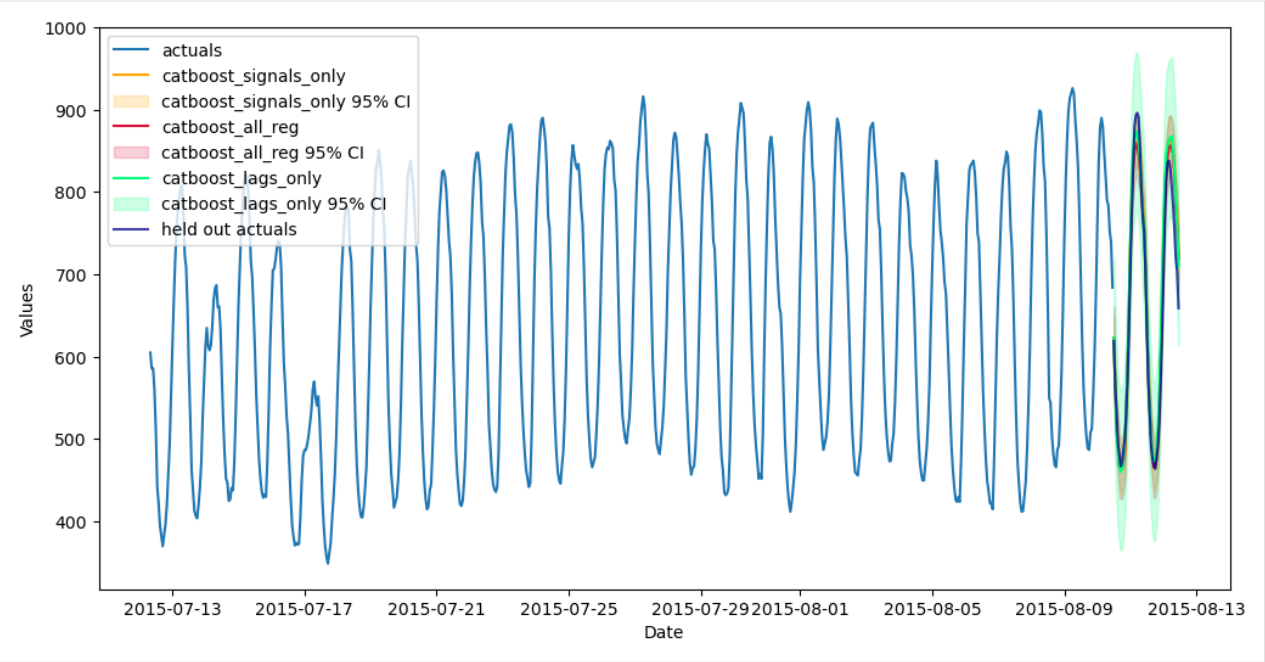
	ModelNickname	TestSetLength	TestSetSMAPE	InSampleSMAPE
0	catboost_signals_only	48	0.014934	0.007005
1	catboost_all_reg	48	0.022299	0.003340
2	manual_arima	48	0.046551	0.017995
3	prophet	48	0.047237	0.043859
4	catboost_lags_only	48	0.060460	0.003447
5	rnn_relu_activation	48	0.082511	0.081514
6	naive	48	0.093079	0.067697
7	rnn_tanh_activation	48	0.096998	0.059914

Unsuprisingly, now our catboost model with just the signals is showing the best test-set scores. But the test set has been compromised for all models that used signals as inputs. A way around that would have been to call `add_signals(train_only=True)`. Another way to really know how these models performed out-of-sample, we need to compare it with a separate out-of-sample test set.

18.9 Check Performance of Forecast on Held-Out Sample

[20]:

```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(
    models = [m for m in f.history if m.startswith('catboost')],
    order_by="TestSetSMAPE",
    ci=True,
    ax = ax
)
sns.lineplot(
    x = f.future_dates,
    y = test_set,
    ax = ax,
    label = 'held out actuals',
    color = 'darkblue',
    alpha = .75,
)
plt.show()
```



[21]:
```
test_results = pd.DataFrame(index = f.history.keys(),columns = ['smape','mase'])
for k, v in f.history.items():
    test_results.loc[k,['smape','mase']] = [
        metrics.smape(test_set,v['Forecast']),
        metrics.mase(test_set,v['Forecast'],m=24,obs=f.y),
    ]

test_results.sort_values('smape')
```

[21]:

	smape	mase
catboost_all_reg	0.028472	0.47471
catboost_signals_only	0.029847	0.508708
rnn_tanh_activation	0.030332	0.482463
manual_arima	0.032933	0.542456
catboost_lags_only	0.035468	0.58522
prophet	0.039312	0.632253
naive	0.052629	0.827014
rnn_relu_activation	0.053967	0.844506

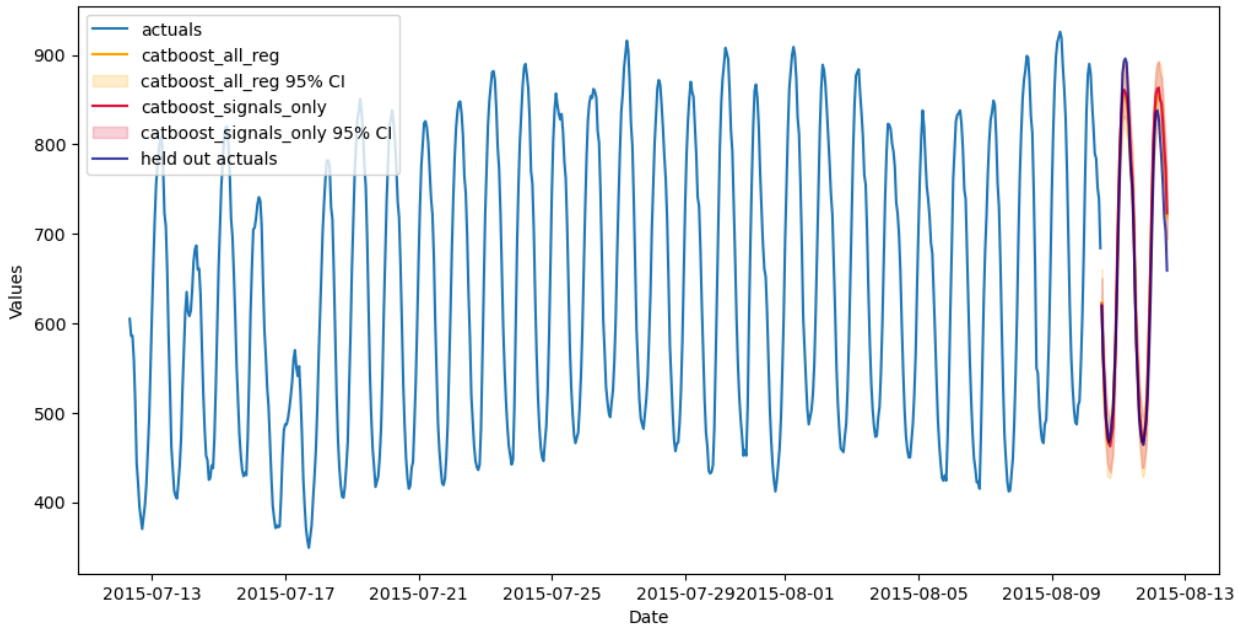
Now, we finally get to the crux of the analysis, where we can see that the catboost that used both the other model signals and the series lags performed best, followed by the catboost model that used only signals. This demonstrates the power of stacking and how it can make good models great.

[22]:
```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(
    models = ['catboost_all_reg','catboost_signals_only'],
    ci=True,
    ax = ax
)
sns.lineplot(
    x = f.future_dates,
    y = test_set,
```

(continues on next page)

(continued from previous page)

```
    ax = ax,
    label = 'held out actuals',
    color = 'darkblue',
    alpha = .75,
)
plt.show()
```



18.10 View each covariate's shapley score

- Looking at the scores below, it is not surprising that the ARIMA signal was deemed the most important covariate in the final catboost model.

[23]:
```
f.export_feature_importance('catboost_all_reg')
```

[23]:

feature	weight	std
signal_manual_arima	43.409875	19.972339
AR1	23.524131	12.268015
signal_prophet	15.545415	7.210582
AR3	5.498856	3.197898
AR48	5.287357	2.185925
signal_rnn_relu_activation	4.786888	1.545580
AR46	4.122075	1.869779
AR24	3.416473	1.909012
AR9	3.280298	1.027637
AR11	3.267605	1.200242
signal_rnn_tanh_activation	3.182172	1.265836
AR2	3.127812	2.230522
AR4	3.036603	1.894097
AR47	2.989912	2.134603

(continues on next page)

(continued from previous page)

```
AR22                    2.936120   1.500898
AR23                    2.886598   1.646012
AR26                    2.857151   1.606732
AR37                    2.742366   1.151931
AR45                    2.487890   1.208743
signal_naive            2.312755   1.719179
AR32                    2.220659   1.607538
AR10                    1.983458   1.114602
AR35                    1.774869   0.631349
AR44                    1.684264   0.795448
AR38                    1.400866   0.727546
AR25                    1.397472   1.024810
AR20                    1.394218   0.849565
AR19                    1.340780   1.050531
AR15                    1.153406   0.850390
AR5                     1.152500   1.552131
AR27                    1.087360   0.692808
AR13                    1.021683   0.483840
AR12                    0.956608   0.690090
AR33                    0.952559   0.603423
AR30                    0.870104   0.833138
AR6                     0.785105   0.742930
AR14                    0.782419   0.431874
AR29                    0.741190   0.518606
AR17                    0.726336   0.489545
AR8                     0.720186   0.613533
AR18                    0.694779   0.615912
AR42                    0.640053   0.504591
AR28                    0.637528   0.431281
AR39                    0.620131   0.728132
AR36                    0.612843   0.565509
AR31                    0.589739   0.449994
AR34                    0.574390   0.679812
AR16                    0.568558   0.539353
AR7                     0.554511   0.520831
AR40                    0.515242   0.359853
AR21                    0.506977   0.641584
AR41                    0.273889   0.300270
AR43                    0.222464   0.264377
```

[ ]:

CHAPTER

NINETEEN

TRANSFER LEARNING

- This notebook shows how to use one fitted model stored in a `Forecaster` object to make predictions on a separate time series.
- Requires >=`0.19.0`
- As of `0.19.1`, univariate sklearn and tensorflow (RNN/LSTM) models are supported for this type of process. Multivariate sklearn, then the rest of the model types will be worked on next.
- See the documentation.

[1]:
```
from scalecast.Forecaster import Forecaster
from scalecast.util import infer_apply_Xvar_selection, find_optimal_transformation
from scalecast.Pipeline import Pipeline, Transformer, Reverter
from scalecast import GridGenerator
import pandas_datareader as pdr
import matplotlib.pyplot as plt
import pandas as pd
```

[2]:
```
GridGenerator.get_example_grids()
```

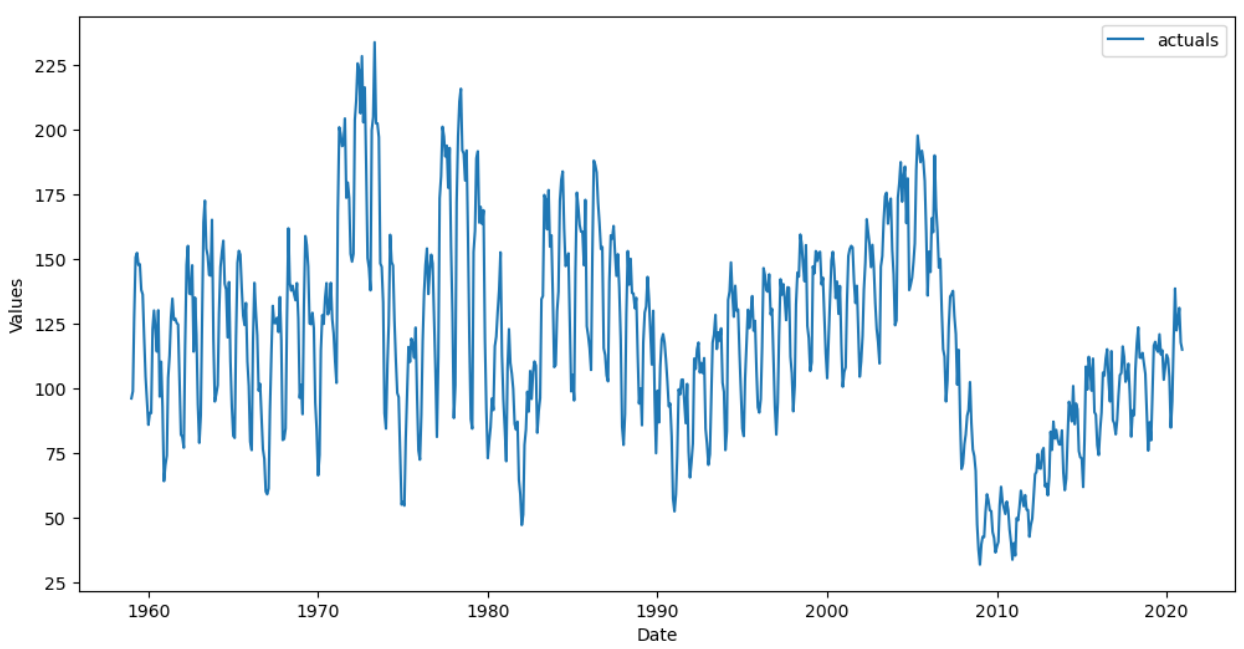
19.1 Initiate the First Forecaster Object

This series ends December, 2020.

[3]:
```
df = pdr.get_data_fred(
    'HOUSTNSA',
    start = '1959-01-01',
    end = '2020-12-31',
)

df.tail()
```

[3]:
```
            HOUSTNSA
DATE
2020-08-01     122.5
2020-09-01     126.3
2020-10-01     131.2
2020-11-01     117.8
2020-12-01     115.1
```

[4]:
```
f = Forecaster(
    y = df.iloc[:,0],
    current_dates = df.index,
    future_dates = 24,
)

f.plot()
plt.show()
```



19.1.1 Automatically add Xvars to the object

[5]:
```
f.auto_Xvar_select()
f
```

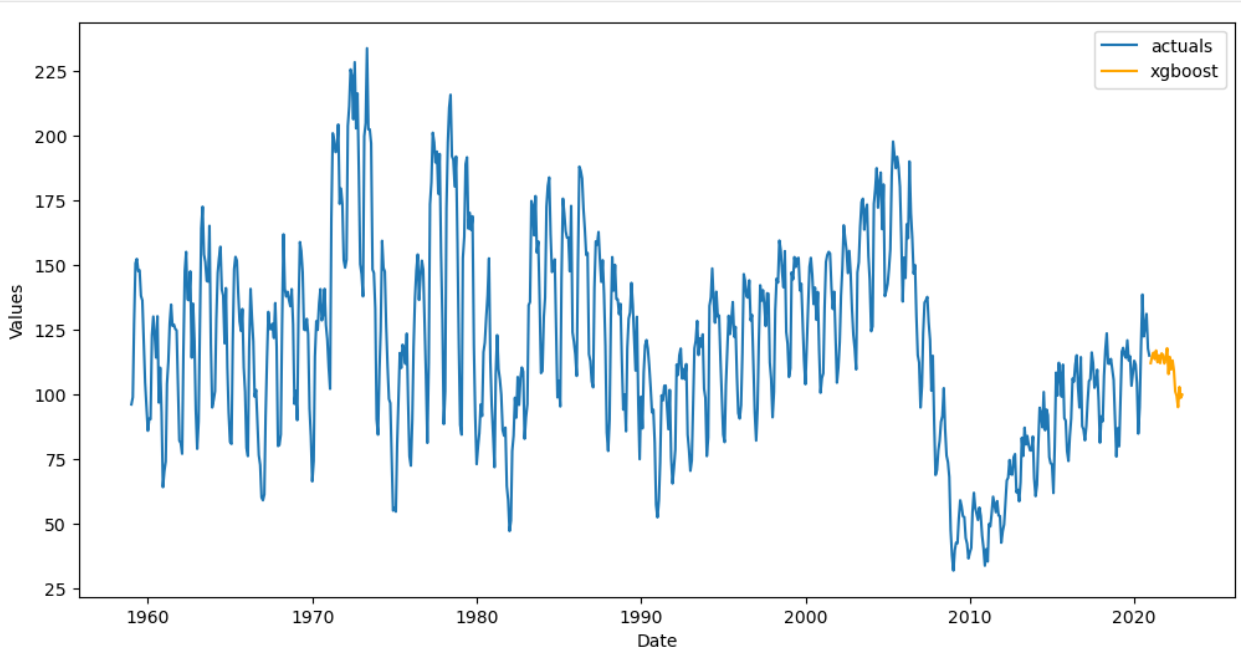
[5]:
```
Forecaster(
    DateStartActuals=1959-01-01T00:00:00.000000000
    DateEndActuals=2020-12-01T00:00:00.000000000
    Freq=MS
    N_actuals=744
    ForecastLength=24
    Xvars=['AR1', 'AR2', 'AR3', 'AR4']
    TestLength=0
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

19.1.2 Fit an XGBoost Model and Make Predictions

[6]:
```
f.set_estimator('xgboost')
f.ingest_grid('xgboost')
f.limit_grid_size(10)
f.cross_validate(k=3,test_length=48)
f.auto_forecast()
```

19.1.3 View the Forecast

[7]:
```
f.plot()
plt.show()
```



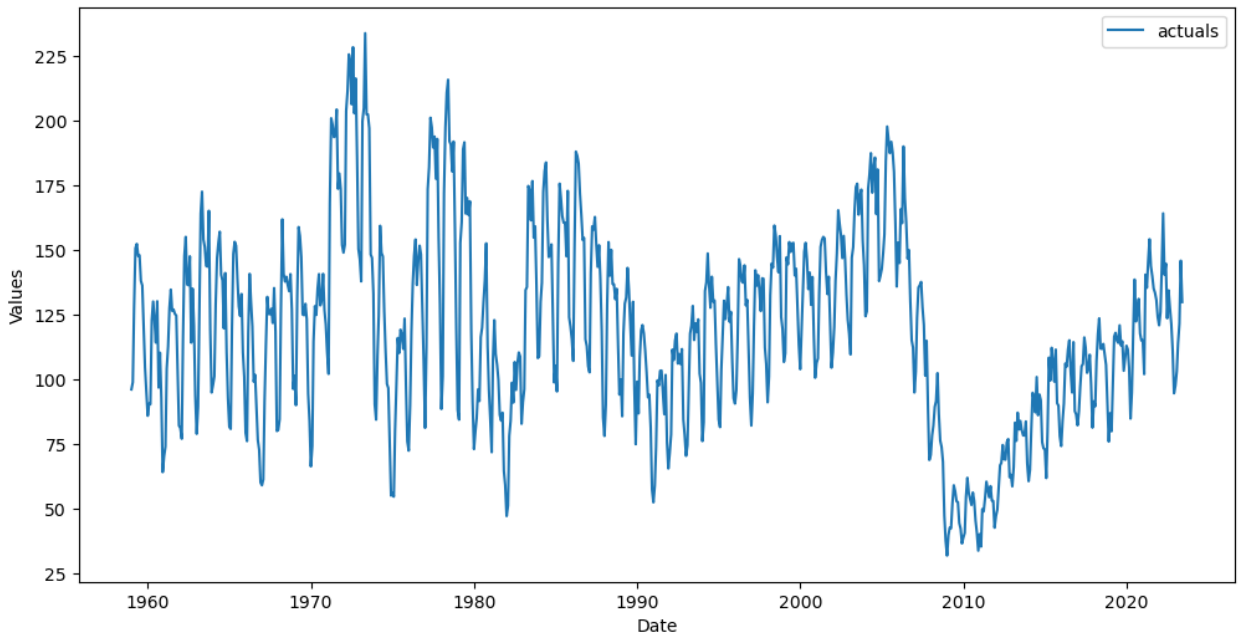
19.2 Initiate the Second Forecaster Object

- Later, if we have more data streaming in, instead of refitting a model, we can use the already-fitted model to make the predictions. This updated series is through June, 2023
- You can use an updated version of the original series, you can use the same series with an extended Forecast horizon, or you can use an entirely different series (as long as it's the same frequency) to perform this process

[8]:
```
df_new = pdr.get_data_fred(
    'HOUSTNSA',
    start = '1959-01-01',
    end = '2023-06-30',
)

df_new.tail()
```

[8]:
```
            HOUSTNSA
DATE
2023-02-01     103.2
2023-03-01     114.0
2023-04-01     121.7
2023-05-01     146.0
2023-06-01     130.0
```

[9]:
```
f_new = Forecaster(
    y = df_new.iloc[:,0],
    current_dates = df_new.index,
    future_dates = 48,
)

f_new.plot()
plt.show()
```



19.2.1 Add the same Xvars to the new Forecaster object

- The helper function below can assist when you automatically added Xvars
- If you manually added Xvars, you can wrap the selection process in a function and run this new `Forecaster` object through the same function.

[10]:
```
f_new = infer_apply_Xvar_selection(infer_from=f,apply_to=f_new)
f_new
```

[10]:
```
Forecaster(
    DateStartActuals=1959-01-01T00:00:00.000000000
    DateEndActuals=2023-06-01T00:00:00.000000000
    Freq=MS
```

(continues on next page)

(continued from previous page)

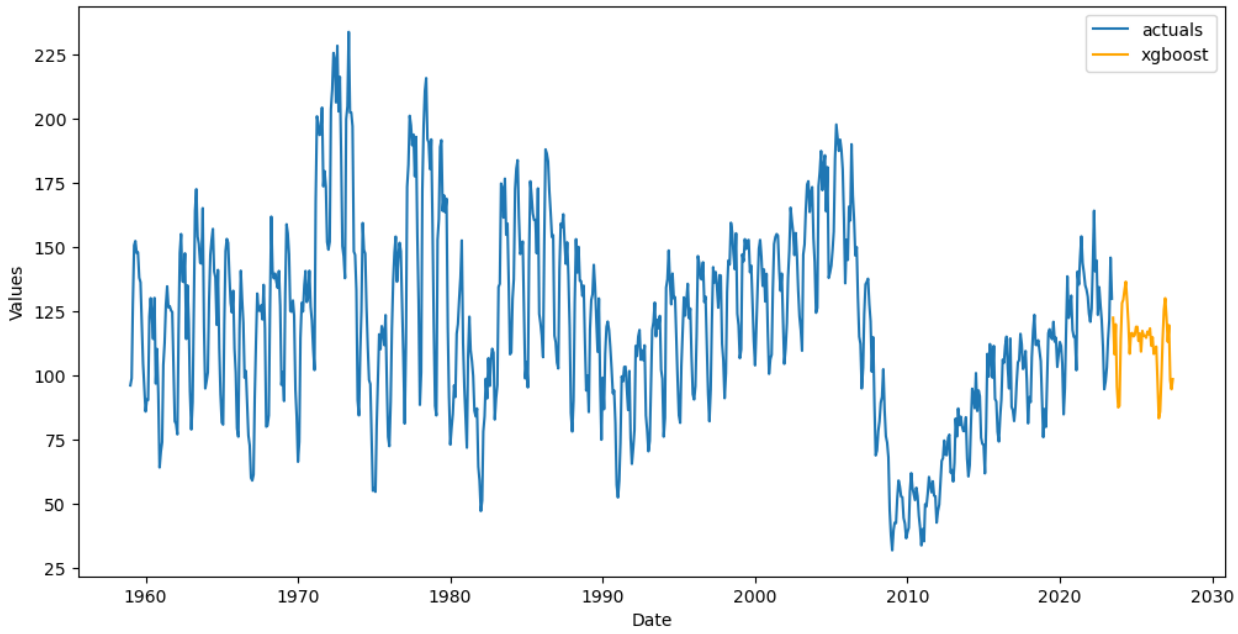
```
    N_actuals=774
    ForecastLength=48
    Xvars=['AR1', 'AR2', 'AR3', 'AR4']
    TestLength=0
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

19.2.2 Apply fitted model from first object onto this new object

[11]:
```
f_new.transfer_predict(transfer_from=f,model='xgboost')
```

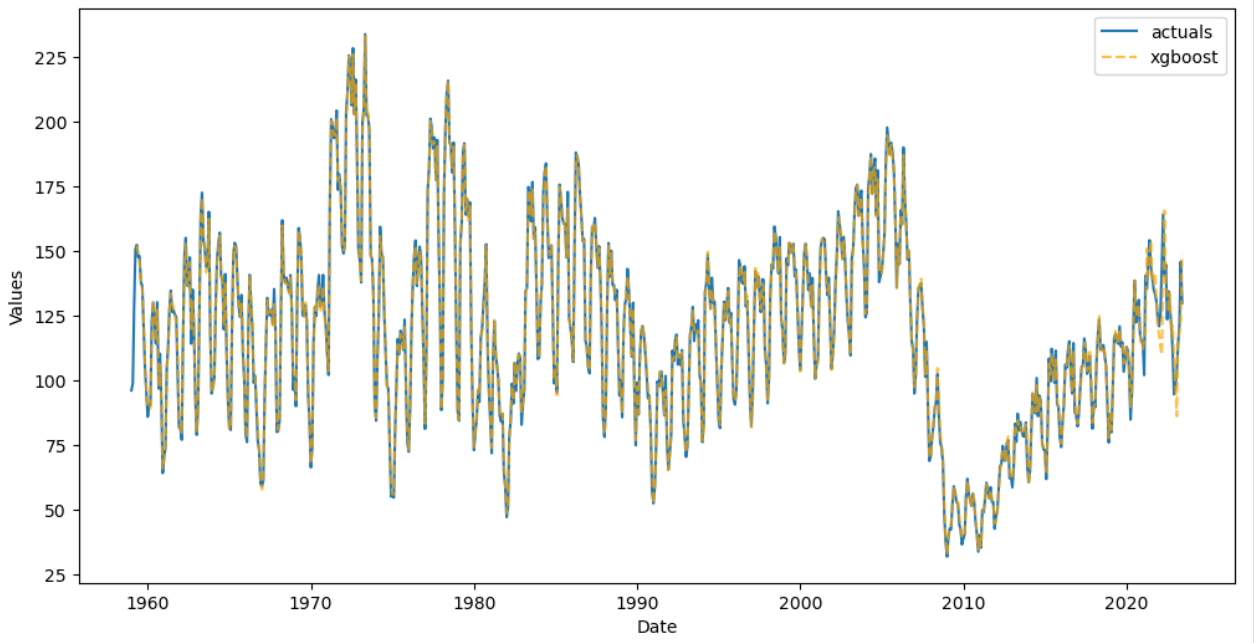
19.2.3 View the new forecast

[13]:
```
f_new.plot()
plt.show()
```



19.2.4 View the in-sample predictions

[14]:
```
f_new.plot_fitted()
plt.show()
```



In the below plot, the model has seen all observations until December, 2020. From January, 2021 through June, 2023, it is seeing those observations for the first time, but because it has the actual y observations, its predictions are expected to be more accurate than the forecast into the unknown horizon.

19.3 Predict over a specific date range

- Instead of storing the model info into the new `Forecaster` object, you can instead get predicted output over a specific date range.

[15]:
```
preds = f_new.transfer_predict(
    transfer_from = f,
    model = 'xgboost',
    dates = pd.date_range(start='2021-01-01',end='2023-12-31',freq='MS'),
    save_to_history=False,
    return_series=True,
)
preds
```

[15]:
```
2021-01-01    112.175186
2021-02-01    115.029518
2021-03-01    118.659706
2021-04-01    151.111618
2021-05-01    140.886765
2021-06-01    147.894882
2021-07-01    154.100571
2021-08-01    137.926544
```

(continues on next page)

(continued from previous page)

```
2021-09-01    137.189224
2021-10-01    141.547058
2021-11-01    130.737900
2021-12-01    120.278030
2022-01-01    117.764030
2022-02-01    112.886353
2022-03-01    110.008720
2022-04-01    139.413696
2022-05-01    165.805817
2022-06-01    135.017914
2022-07-01    134.379486
2022-08-01    126.559814
2022-09-01    133.788025
2022-10-01    118.278656
2022-11-01    121.269691
2022-12-01    110.270599
2023-01-01     98.168579
2023-02-01     86.210205
2023-03-01    115.844292
2023-04-01    118.449600
2023-05-01    123.620918
2023-06-01    148.793396
2023-07-01    122.571846
2023-08-01    108.331818
2023-09-01    120.023163
2023-10-01     95.729309
2023-11-01     87.593613
2023-12-01     88.265671
dtype: float32
```

From January through June, 2021, the predictions are considered in-sample, although the model has never previously seen them (it predicted using the actual y observations over that timespan, and that is a form of leakage for auto-regressive time series models). The rest of the predictions are truly out-of-sample.

19.4 Transfer Predict in a Pipeline

- We can use auto-transformation selection and pipelines to apply predictions from a fitted model into a new Forecaster object. This can be good to apply when new data frequently comes through and you don't want to refit models.

19.4.1 Find optimal set of transformations

[16]:
```
transformer, reverter = find_optimal_transformation(f,verbose=True)
```

```
Using xgboost model to find the best transformation set on 1 test sets, each 24 in␣
↪length.
All transformation tries will be evaluated with 12 lags.
Last transformer tried:
[]
Score (rmse): 19.44337760778588
```

(continues on next page)

(continued from previous page)

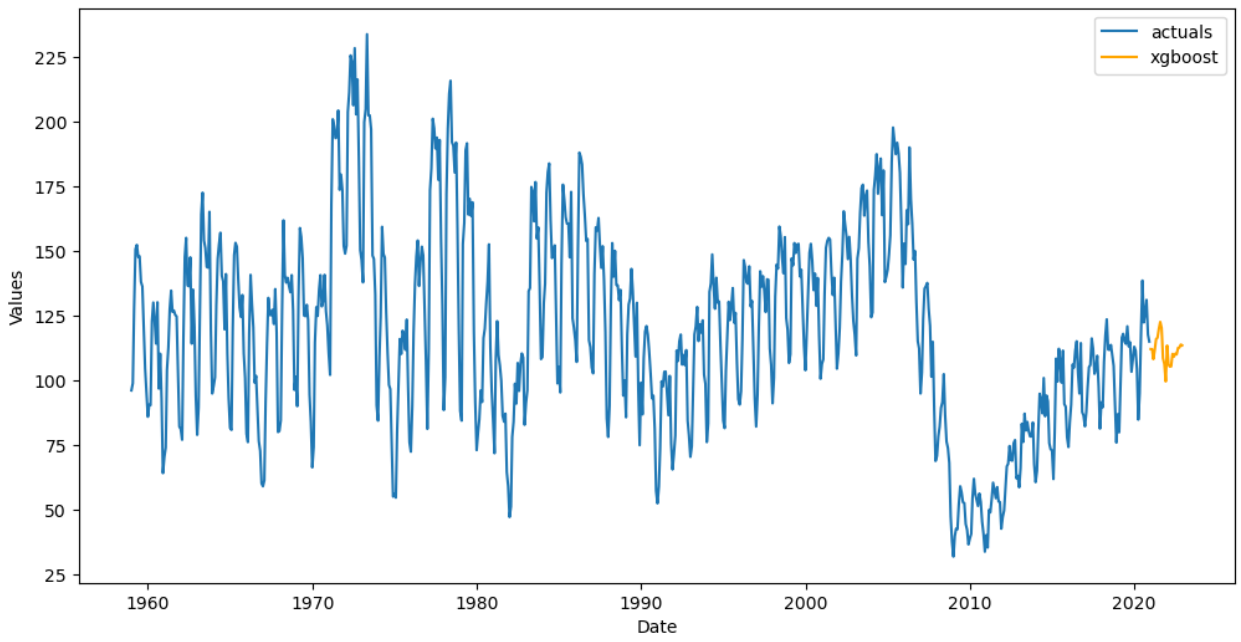
```
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (rmse): 16.47704226214548
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (rmse): 27.038328952689234
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (rmse): 14.152673765108048
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('DeseasonTransform', {'m': 12, 'model': 'add'}
↪)]
Score (rmse): 20.14955556372946
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('DiffTransform', 1)]
Score (rmse): 18.150826589832704
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('DiffTransform', 12)]
Score (rmse): 31.59079798104221
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('ScaleTransform',)]
Score (rmse): 17.27265247612732
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('MinMaxTransform',)]
Score (rmse): 13.685311300863027
-------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('RobustScaleTransform',)]
Score (rmse): 14.963448554311212
-------------------------------------------------
Final Selection:
[('DetrendTransform', {'poly_order': 2}), ('MinMaxTransform',)]
```

19.4.2 Fit the first pipeline

[17]:
```
def forecaster(f):
    f.auto_Xvar_select()
    f.ingest_grid('xgboost')
    f.limit_grid_size(10)
    f.cross_validate(k=3,test_length=48)
    f.auto_forecast()
```

[18]:
```
pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter),
    ]
)

f = pipeline.fit_predict(f)
```

[19]:
```
f.plot()
plt.show()
```

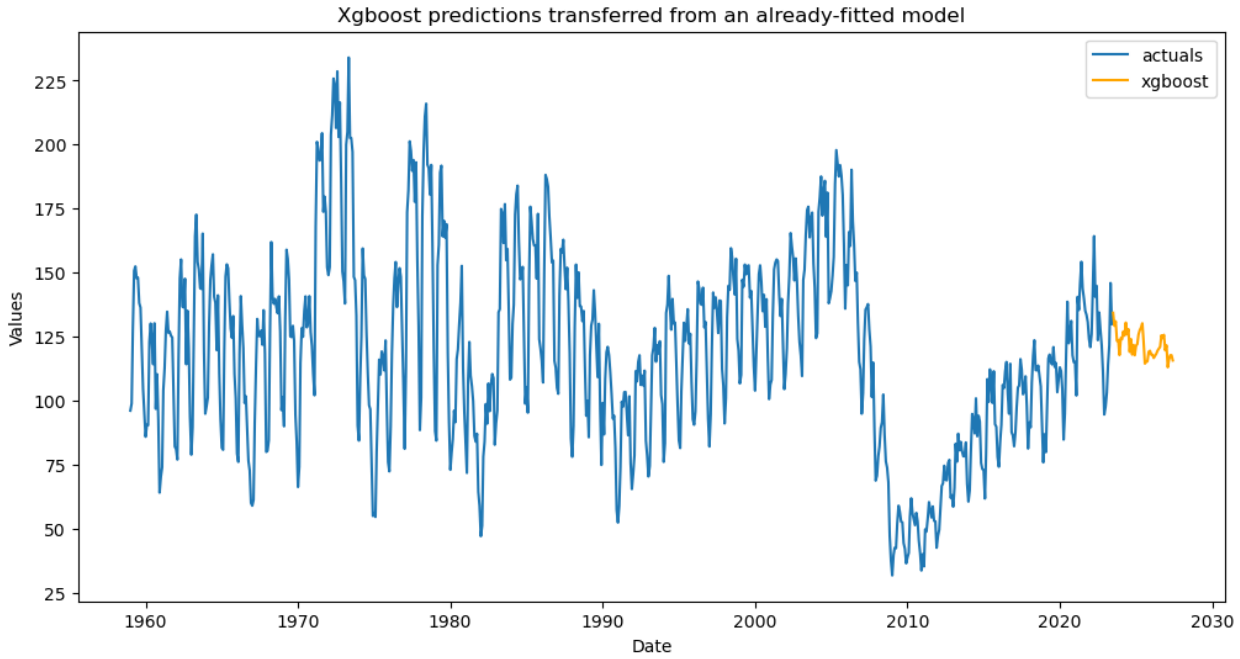


19.4.3 Predict new data

[20]:
```
def transfer_forecast(f,transfer_from):
    infer_apply_Xvar_selection(infer_from=transfer_from,apply_to=f)
    f.transfer_predict(transfer_from=transfer_from,model='xgboost')
```

[21]:
```
pipeline_new = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Transfer Forecast',transfer_forecast),
        ('Revert',reverter),
    ]
)

f_new = pipeline_new.fit_predict(f_new,transfer_from=f) # even though it says fit, no␣
↪model actually gets fit in the pipeline
```

[23]:
```
f_new.plot()
plt.title('Xgboost predictions transferred from an already-fitted model')
plt.show()
```



[ ]:

CHAPTER

TWENTY

TRANSFER LEARNING - TENSORFLOW

- Transfer learning tensorflow (RNN/LSTM) models works slightly different than other model types, due to the difficulty of carrying these models in `Forecaster` object history. `Forecater` objects are meant to be be copied and pickled, both of which can fail with tensorflow models. For that reason, it is good to save out tensorflow models manually before using them to transfer learn.
- This notebook also demonstrates transferring confidence intervals.
- Requires >=`0.19.1`.

[1]:
```
from scalecast.Forecaster import Forecaster
from scalecast.util import infer_apply_Xvar_selection, find_optimal_transformation
from scalecast.Pipeline import Pipeline, Transformer, Reverter
from scalecast import GridGenerator
import pandas_datareader as pdr
import matplotlib.pyplot as plt
import pandas as pd
```

20.1 Initiate the First Forecaster Object

This series ends December, 2020.

[2]:
```
df = pdr.get_data_fred(
    'HOUSTNSA',
    start = '1959-01-01',
    end = '2020-12-31',
)

df.head()
```

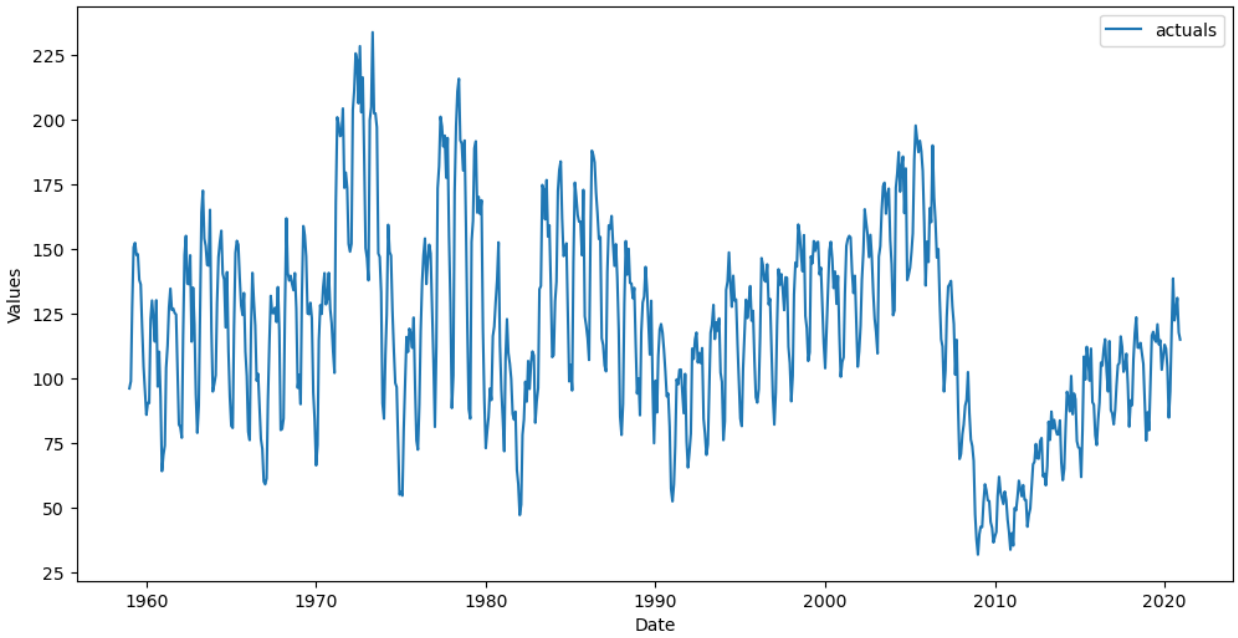
[2]:
```
            HOUSTNSA
DATE
1959-01-01      96.2
1959-02-01      99.0
1959-03-01     127.7
1959-04-01     150.8
1959-05-01     152.5
```

[3]:
```
f = Forecaster(
    y = df.iloc[:,0],
    current_dates = df.index,
```

(continues on next page)

(continued from previous page)

```
    future_dates = 24,
    cis=True,
    test_length = 24,
)

f.plot()
plt.show()
```



20.1.1 Fit the RNN Model

[4]:
```
f.set_estimator('rnn')
f.manual_forecast(epochs=15,lags=24)
```

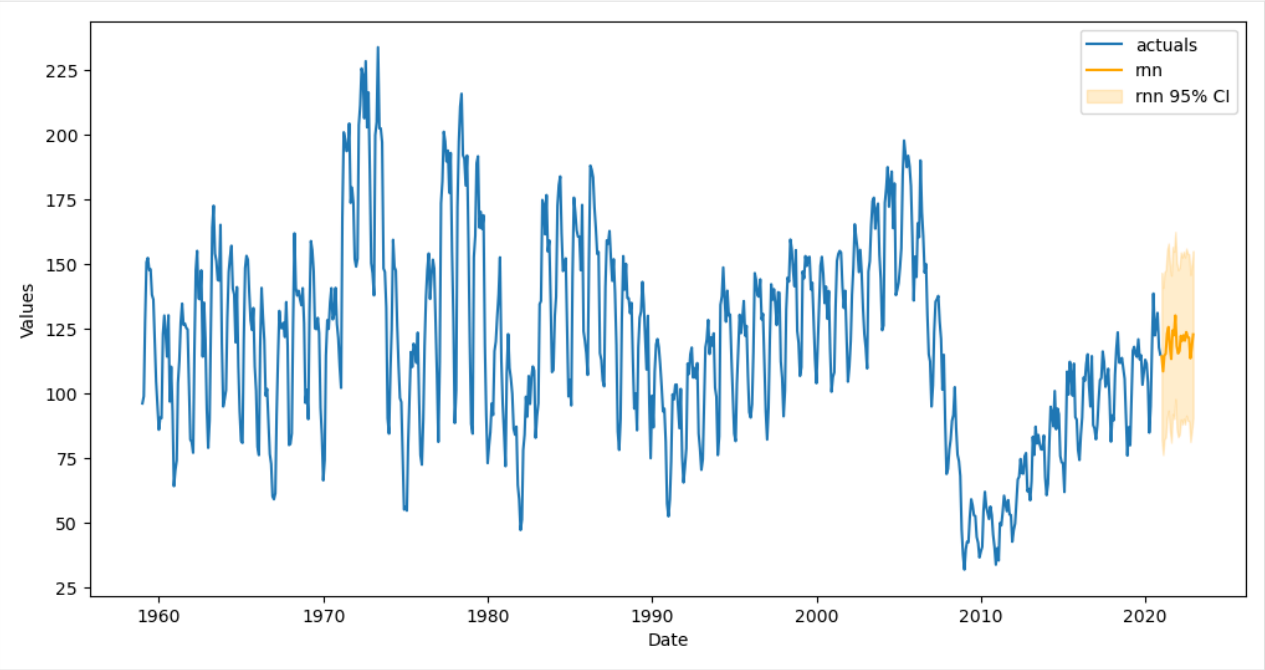
```
Epoch 1/15
21/21 [==============================] - 2s 8ms/step - loss: 0.4615
Epoch 2/15
21/21 [==============================] - 0s 9ms/step - loss: 0.3677
Epoch 3/15
21/21 [==============================] - 0s 8ms/step - loss: 0.2878
Epoch 4/15
21/21 [==============================] - 0s 8ms/step - loss: 0.2330
Epoch 5/15
21/21 [==============================] - 0s 8ms/step - loss: 0.1968
Epoch 6/15
21/21 [==============================] - 0s 8ms/step - loss: 0.1724
Epoch 7/15
21/21 [==============================] - 0s 8ms/step - loss: 0.1555
Epoch 8/15
21/21 [==============================] - 0s 7ms/step - loss: 0.1454
Epoch 9/15
```

(continues on next page)

(continued from previous page)

```
21/21 [==============================] - 0s 7ms/step - loss: 0.1393
Epoch 10/15
21/21 [==============================] - 0s 8ms/step - loss: 0.1347
Epoch 11/15
21/21 [==============================] - 0s 7ms/step - loss: 0.1323
Epoch 12/15
21/21 [==============================] - 0s 7ms/step - loss: 0.1303
Epoch 13/15
21/21 [==============================] - 0s 6ms/step - loss: 0.1279
Epoch 14/15
21/21 [==============================] - 0s 7ms/step - loss: 0.1275
Epoch 15/15
21/21 [==============================] - 0s 7ms/step - loss: 0.1261
1/1 [==============================] - 0s 345ms/step
Epoch 1/15
22/22 [==============================] - 2s 8ms/step - loss: 0.3696
Epoch 2/15
22/22 [==============================] - 0s 7ms/step - loss: 0.3072
Epoch 3/15
22/22 [==============================] - 0s 6ms/step - loss: 0.2600
Epoch 4/15
22/22 [==============================] - 0s 6ms/step - loss: 0.2255
Epoch 5/15
22/22 [==============================] - 0s 6ms/step - loss: 0.2014
Epoch 6/15
22/22 [==============================] - 0s 6ms/step - loss: 0.1840
Epoch 7/15
22/22 [==============================] - 0s 5ms/step - loss: 0.1713
Epoch 8/15
22/22 [==============================] - 0s 5ms/step - loss: 0.1616
Epoch 9/15
22/22 [==============================] - 0s 5ms/step - loss: 0.1557
Epoch 10/15
22/22 [==============================] - 0s 5ms/step - loss: 0.1503
Epoch 11/15
22/22 [==============================] - 0s 5ms/step - loss: 0.1437
Epoch 12/15
22/22 [==============================] - 0s 6ms/step - loss: 0.1404
Epoch 13/15
22/22 [==============================] - 0s 5ms/step - loss: 0.1381
Epoch 14/15
22/22 [==============================] - 0s 6ms/step - loss: 0.1354
Epoch 15/15
22/22 [==============================] - 0s 6ms/step - loss: 0.1339
1/1 [==============================] - 0s 327ms/step
22/22 [==============================] - 0s 3ms/step
```

[5]:
```
f.plot(ci=True)
plt.show()
```



20.1.2 Save the model out

After you fit a tensforflow model, the fit model is attached to the `Forecaster` in the `tf_model` attribute. You can view the model summary:

[6]:
```
f.tf_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 8)                 80

 dense_1 (Dense)             (None, 24)                216

=================================================================
Total params: 296
Trainable params: 296
Non-trainable params: 0
_________________________________________________________________
```

However, immediately after changing estimators or when the object is pickled out, this attribute is lost. Therefore, it is a good idea to save the model out:

[7]:
```
f.save_tf_model(name='model.h5') # default argument
```

Later, when you want to transfer learn with the model, you can re-attach it to the `Forecaster` object:

[8]:
```
f.load_tf_model(name='model.h5') # default argument
```

This re-attaches the `tf_model` attribute.

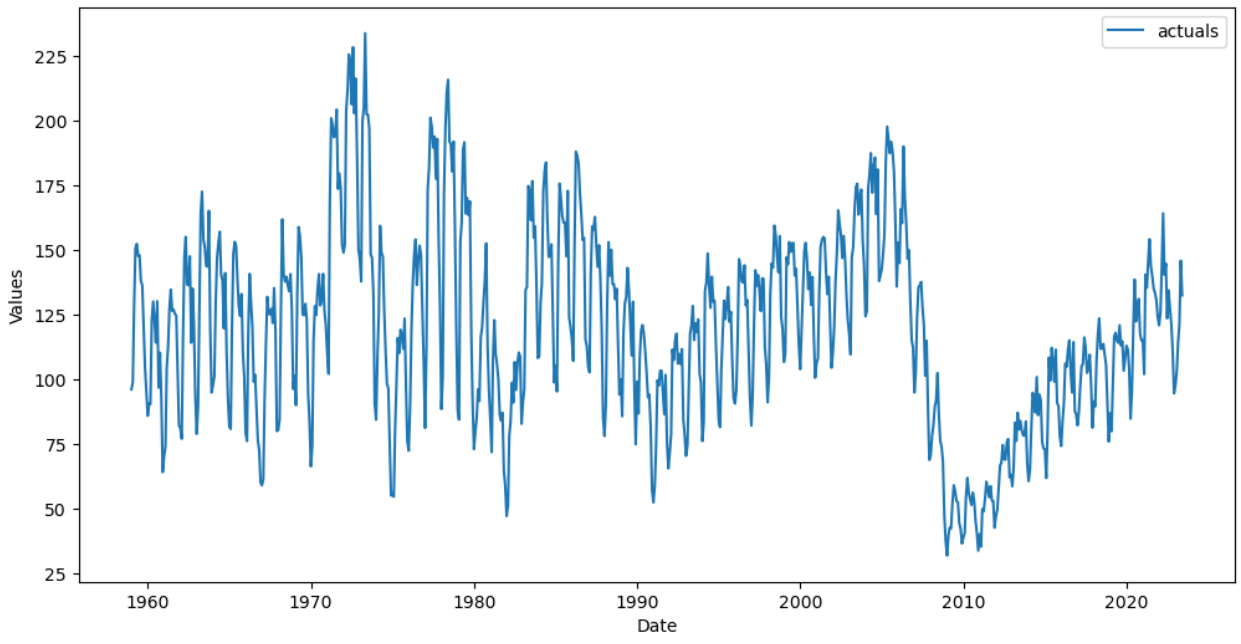
20.2 Initiate the Second Forecaster Object

- Later, if we have more data streaming in, instead of refitting a model, we can use the already-fitted model to make the predictions. This updated series is through June, 2023
- You can use an updated version of the original series, you can use the same series with an extended Forecast horizon, or you can use an entirely different series (as long as it's the same frequency) to perform this process

[9]:
```
df_new = pdr.get_data_fred(
    'HOUSTNSA',
    start = '1959-01-01',
    end = '2023-06-30',
)

df_new.tail()
```

[9]:
```
            HOUSTNSA
DATE
2023-02-01     103.2
2023-03-01     114.0
2023-04-01     121.7
2023-05-01     146.0
2023-06-01     132.6
```

[10]:
```
f_new = Forecaster(
    y = df_new.iloc[:,0],
    current_dates = df_new.index,
    future_dates = 24,
)

f_new.plot()
plt.show()
```



20.2.1 Add the same Xvars to the new Forecaster object

- The helper function below can assist when you automatically added Xvars
- If you manually added Xvars, you can wrap the selection process in a function and run this new `Forecaster` object through the same function.

[11]:
```
infer_apply_Xvar_selection(infer_from=f,apply_to=f_new)
```

[11]:
```
Forecaster(
    DateStartActuals=1959-01-01T00:00:00.000000000
    DateEndActuals=2023-06-01T00:00:00.000000000
    Freq=MS
    N_actuals=774
    ForecastLength=24
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8', 'AR9', 'AR10', 'AR11',
↪ 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19', 'AR20', 'AR21', 'AR22
↪', 'AR23', 'AR24']
    TestLength=0
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

20.2.2 Apply fitted model from first object onto this new object

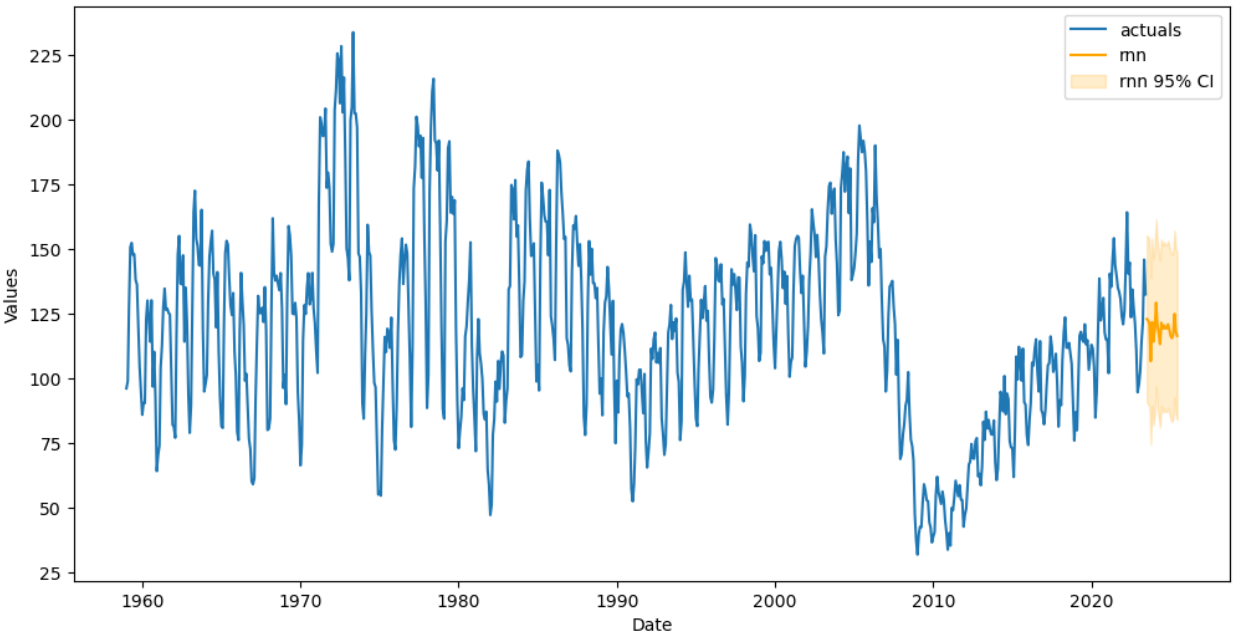
[12]:
```
f_new.transfer_predict(transfer_from=f,model='rnn',model_type='tf')
```

```
1/1 [==============================] - 0s 149ms/step
23/23 [==============================] - 0s 2ms/step
```

20.2.3 Transfer the model's confidence intervals

[13]:
```
f_new.transfer_cis(transfer_from=f,model='rnn') # transfer the model's confidence␣
↪intervals
```

[14]:
```
f_new.plot(ci=True)
plt.show()
```


actuals
mn
rnn 95% CI
225
200
175
150
125
100
75
50
25
Values
1960
1970
1980
1990
2000
2010
2020
Date

[ ]:

CHAPTER

TWENTYONE

TRANSFORMATIONS EXAMPLE

- This is the accompanying code to the medium publication.
- See the documentation for SeriesTransformer, Transformer, and Reverter.

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scalecast.Forecaster import Forecaster
from scalecast.SeriesTransformer import SeriesTransformer
```

[2]:
```
data = pd.read_csv('../lstm/AirPassengers.csv')
```

[3]:
```
data.head()
```

[3]:
```
     Month  #Passengers
0  1949-01          112
1  1949-02          118
2  1949-03          132
3  1949-04          129
4  1949-05          121
```

[4]:
```
f = Forecaster(
    current_dates = data['Month'],
    y = data['#Passengers'],
    future_dates = 24,
)
```

21.1 Create Thumbnail Image

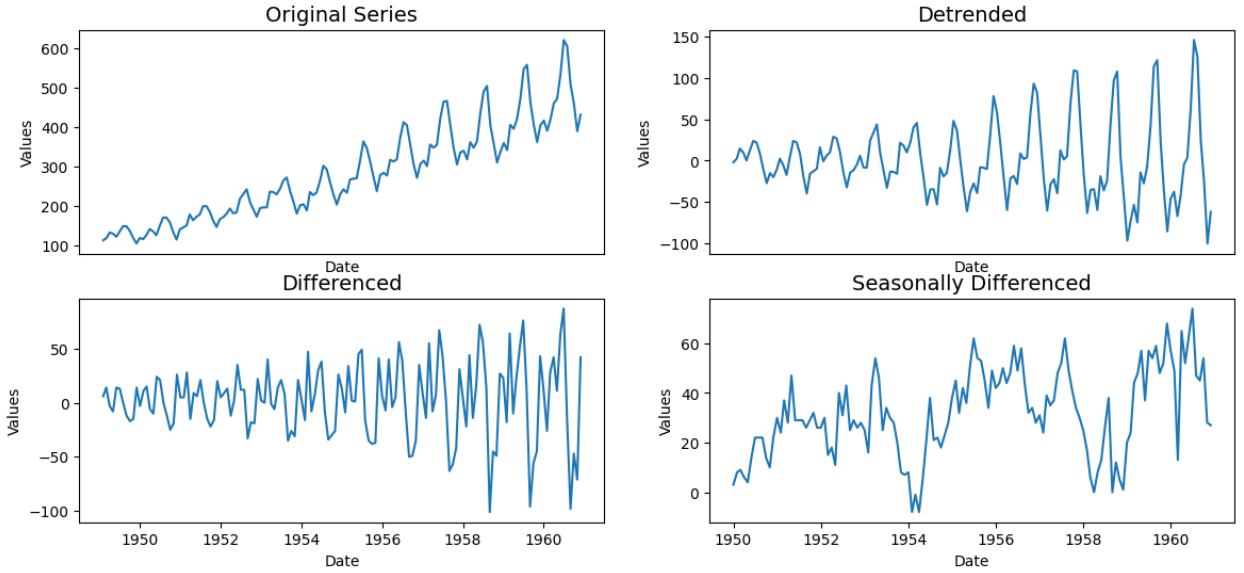
[5]:
```
f_detrended = SeriesTransformer(f).DetrendTransform(poly_order=2)
f_diff = SeriesTransformer(f).DiffTransform()
f_diff_seas = SeriesTransformer(f).DiffTransform(12)
```

[6]:
```
fig, axs = plt.subplots(2,2,figsize=(14,6))
f.plot(ax=axs[0,0])
axs[0,0].set_title('Original Series',size=14)
axs[0,0].tick_params(axis='x',which='both',bottom=False,top=False,labelbottom=False)
axs[0,0].get_legend().remove()
f_detrended.plot(ax=axs[0,1])
```

(continues on next page)

(continued from previous page)

```
axs[0,1].set_title('Detrended',size=14)
axs[0,1].tick_params(axis='x',which='both',bottom=False,top=False,labelbottom=False)
axs[0,1].get_legend().remove()
f_diff.plot(ax=axs[1,0])
axs[1,0].set_title('Differenced',size=14)
axs[1,0].get_legend().remove()
f_diff_seas.plot(ax=axs[1,1])
axs[1,1].set_title('Seasonally Differenced',size=14)
axs[1,1].get_legend().remove()
plt.show()
```

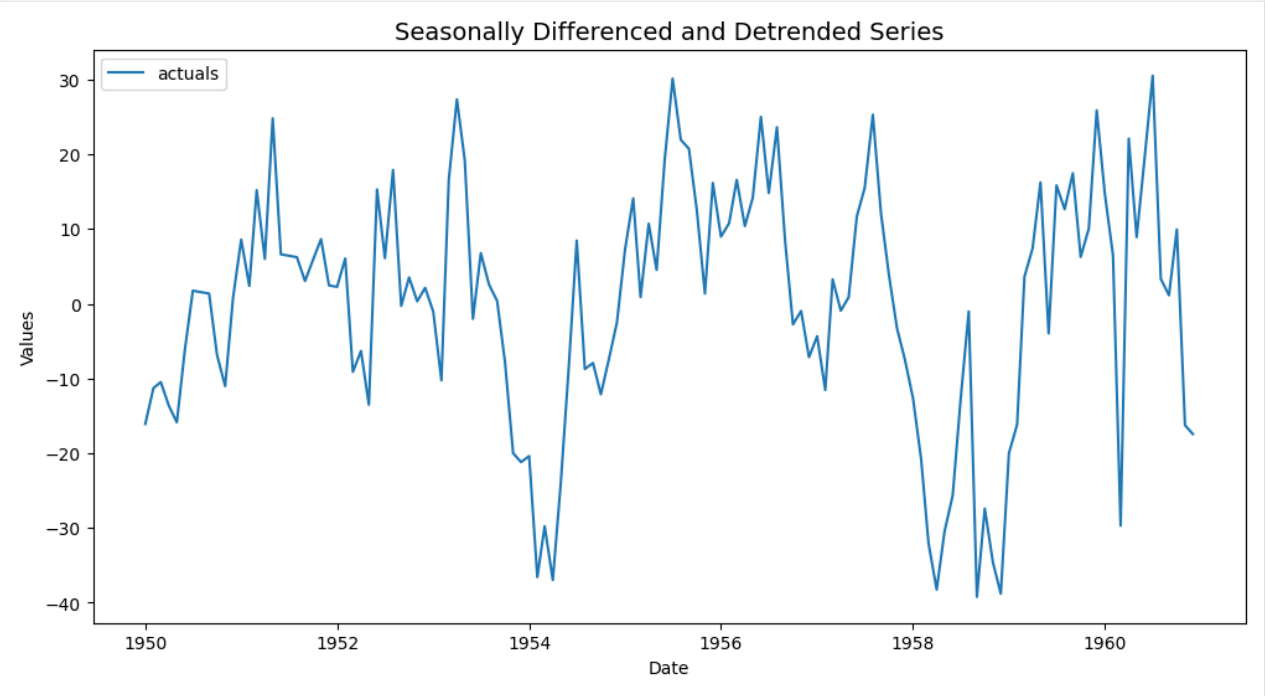


21.2 Create Transformer

[7]:
```
transformer = SeriesTransformer(f)
```

21.3 Apply Transformations

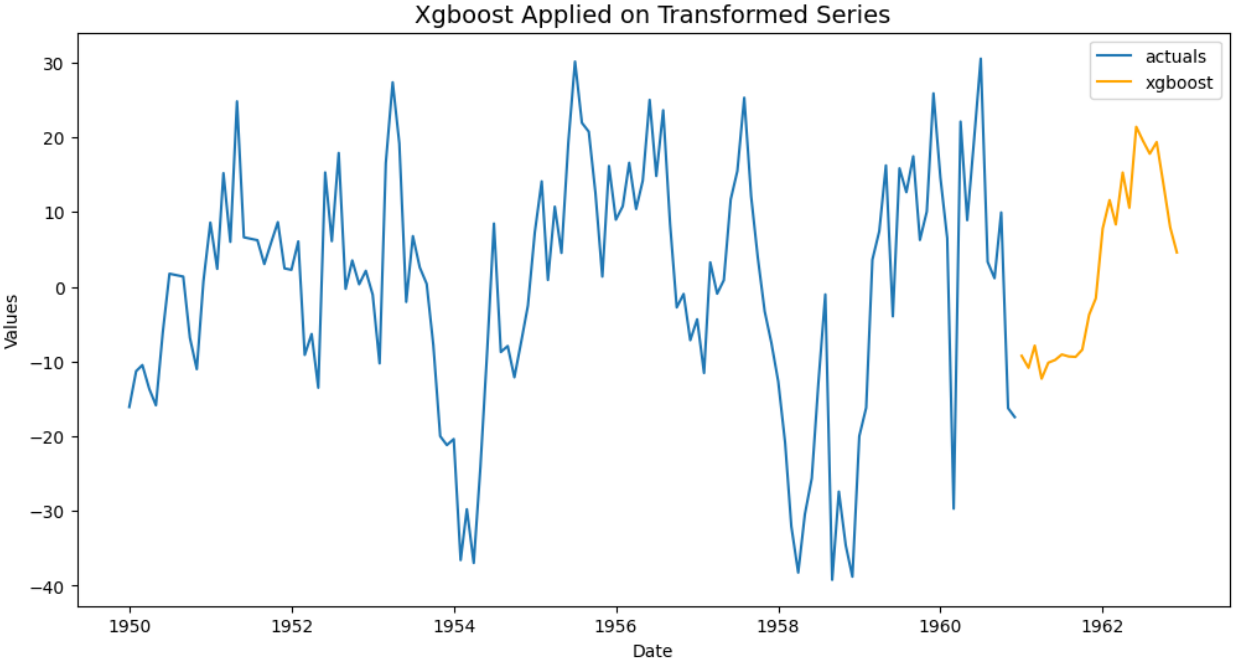
[8]:
```
f = transformer.DiffTransform(12) # 12 periods is one seasonal difference for monthly␣
↪data
f = transformer.DetrendTransform()
f.plot()
plt.title('Seasonally Differenced and Detrended Series',size=14);
```



21.4 Forecast on Transformed Data

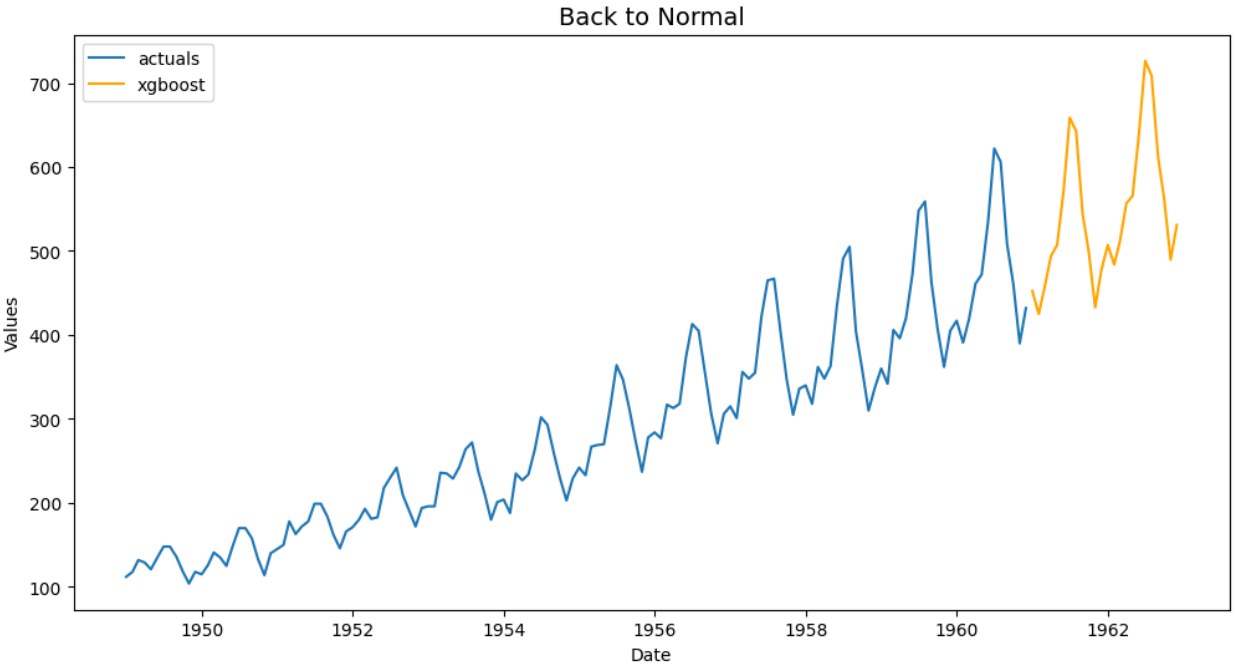
[9]:
```
f.set_estimator('xgboost')
f.add_ar_terms(12)
f.manual_forecast(n_estimators=100,gamma=2)
```

[10]:
```
f.plot()
plt.title('Xgboost Applied on Transformed Series',size=14);
```



21.5 Revert Transformation

[11]:
```
f = transformer.DetrendRevert()
f = transformer.DiffRevert(12)
f.plot()
plt.title('Back to Normal',size=14);
```



Note: After reverting a difference transforamtion, it is a good idea to drop all Xvars from the object and re-add them, especially model lags, since their values are now at a different level and they will have lost some observations from the front.

[12]:
```
f.drop_all_Xvars()
```

21.6 Function to Automatically Find Optimal Transformation

- Documentation

[13]:
```
from scalecast.util import find_optimal_transformation
# default args below
transformer, reverter = find_optimal_transformation(
    f, # Forecaster object to try the transformations on
    estimator=None, # model used to evaluate each transformation, default last estimator␣
↪set in object
    monitor='rmse', # out-of-sample metric to monitor
    test_length = None, # default is the fcst horizon in the Forecaster object
    train_length = None, # default is the max available
    num_test_sets = 1, # number of test sets to iterate through, final transformation␣
↪based on best avg. metric
    space_between_sets = 1, # space between consectutive train sets
    lags='auto', # uses the length of the inferred seasonality
    try_order = ['detrend','seasonal_adj','boxcox','first_diff','first_seasonal_diff',
↪'scale'], # order of transformations to try
    boxcox_lambdas = [-0.5,0,0.5], # box-cox lambas
    detrend_kwargs = [{'loess': True},{'poly_order':1},{'poly_order':2}], # detrender␣
↪transform kwargs (tries as many detrenders as the length of this list)
    scale_type = ['Scale','MinMax'], # scale transformers to try
    m = 'auto', # the seasonal length to try for the seasonal adjusters, accepts multiple
    model = 'add', # the model to use when seasonally adjusting
    verbose = True, # default is False
    # specific model kwargs also accepted
)
```

```
All transformation tries will use 12 lags.
Last transformer tried:
[]
Score (rmse): 73.54387726568602
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'loess': True})]
Score (rmse): 64.9621416143047
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 1})]
Score (rmse): 32.35918665790083
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2})]
Score (rmse): 22.916929563263274
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('DeseasonTransform', {'m': 12, 'model': 'add'}
↪)]
Score (rmse): 36.738799031744186
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('DiffTransform', 1)]
Score (rmse): 55.37104438051655
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('DiffTransform', 12)]
Score (rmse): 46.742630596791805
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('ScaleTransform',)]
Score (rmse): 22.798651665783563
--------------------------------------------------
Last transformer tried:
[('DetrendTransform', {'poly_order': 2}), ('MinMaxTransform',)]
Score (rmse): 21.809089561053717
--------------------------------------------------
Final Selection:
[('DetrendTransform', {'poly_order': 2}), ('MinMaxTransform',)]
```

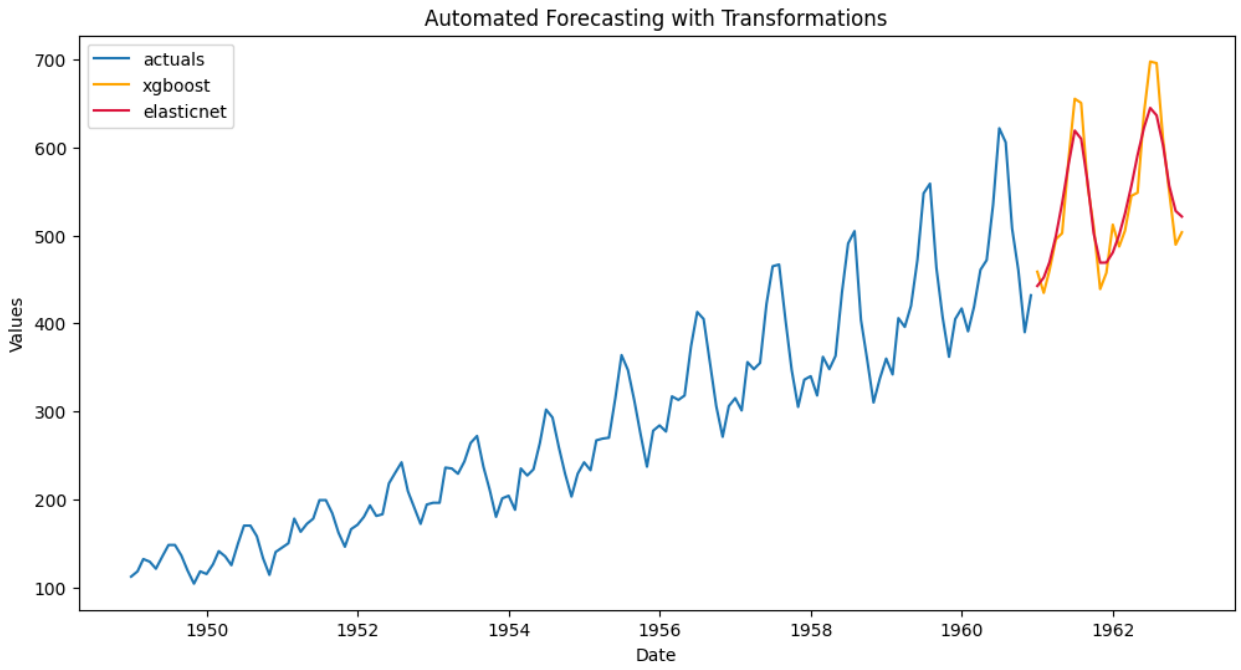
21.7 Automated Forecasting with Pipeline

[14]:
```
from scalecast.Pipeline import Pipeline
from scalecast import GridGenerator
from scalecast.util import find_optimal_transformation

GridGenerator.get_example_grids()

def forecaster(f):
    f.set_validation_length(20)
    f.auto_Xvar_select(max_ar=20)
    f.tune_test_forecast(
        ['elasticnet','xgboost'],
        cross_validate=True,
        limit_grid_size = .2,
    )

pipeline = Pipeline(
    steps = [
        ('Transform',transformer),
        ('Forecast',forecaster),
        ('Revert',reverter),
    ],
)

f = pipeline.fit_predict(f)
```

(continues on next page)

(continued from previous page)

```
f.plot()
plt.title('Automated Forecasting with Transformations');
```




```
[ ]:
```

CHAPTER

TWENTYTWO

THETA

- Read about the theta model
- See the darts implementation
- See the statsmodels implementation
- Download data from GitHub
- Install darts: `pip install darts`
- See the blog post

Scalecast ports the model from darts, which is supposed to be more accurate and is also easier to maintain.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from darts.utils.utils import SeasonalityMode, TrendMode, ModelMode
from scalecast.Forecaster import Forecaster
from scalecast.util import metrics
from scalecast import GridGenerator
```

[2]:
```
train = pd.read_csv('Hourly-train.csv',index_col=0)
test = pd.read_csv('Hourly-test.csv',index_col=0)
y = train.loc['H7'].to_list()
current_dates = pd.date_range(start='2015-01-07 12:00',freq='H',periods=len(y)).to_list()

y_test = test.loc['H7'].to_list()

f = Forecaster(
    y=y,
    current_dates=current_dates,
    metrics = ['smape','r2'],
    test_length = .25,
    future_dates = len(y_test),
    cis = True,
)

f
```

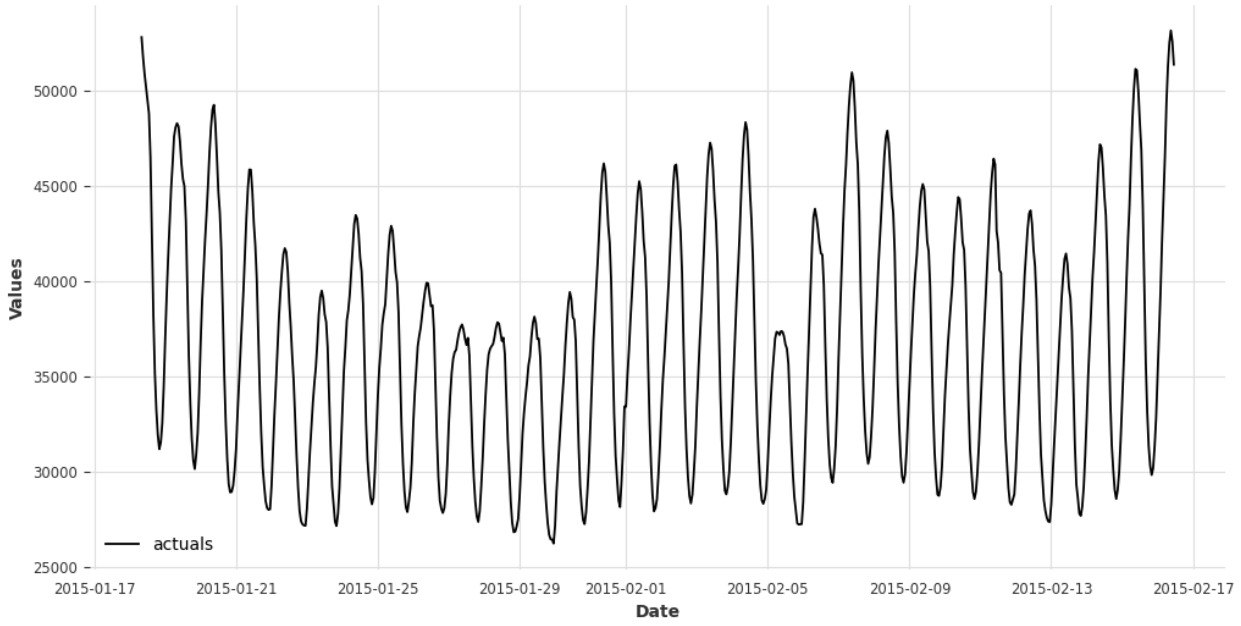
[2]:
```
Forecaster(
    DateStartActuals=2015-01-18T08:00:00.000000000
```

(continues on next page)

(continued from previous page)

```
    DateEndActuals=2015-02-16T11:00:00.000000000
    Freq=H
    N_actuals=700
    ForecastLength=48
    Xvars=[]
    TestLength=240
    ValidationMetric=smape
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[3]:
```
f.plot()
plt.show()
```



22.1 Prepare forecast

- Download theta's validation grid

[6]:
```
GridGenerator.get_grids('theta',out_name='Grids.py')
f.ingest_grid('theta')
```

22.2 Call the forecast

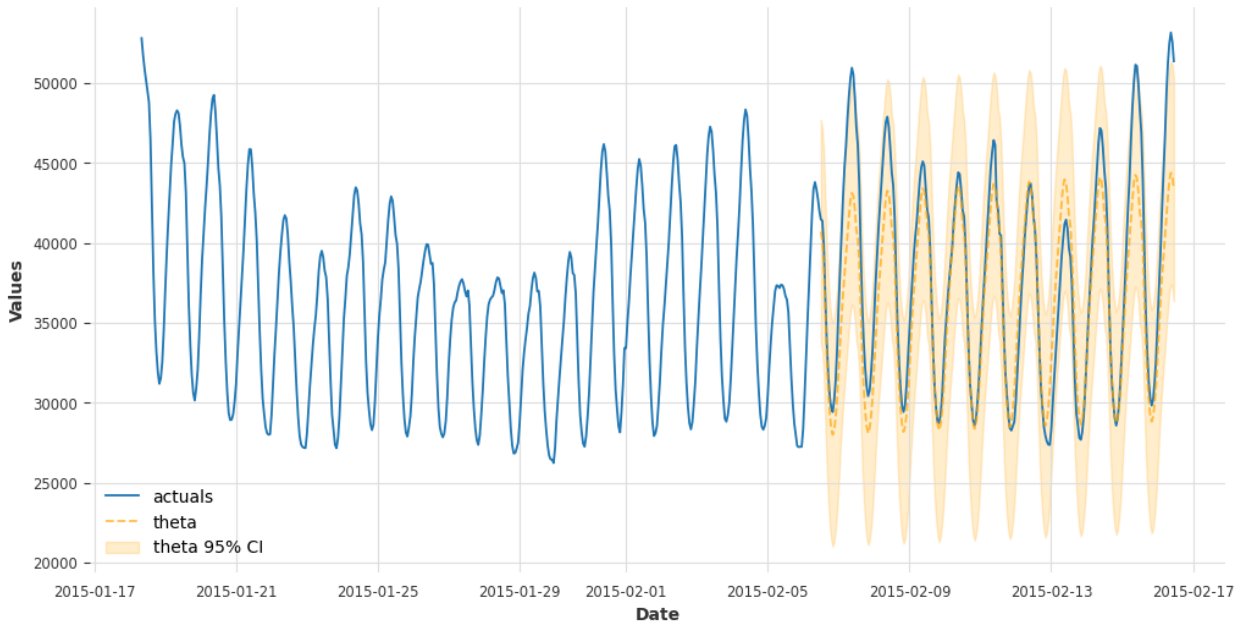
- tune hyperparemters with 3-fold time series cross validation

[8]:
```
f.set_estimator('theta')
f.cross_validate(k=3,verbose=True)
f.auto_forecast()
```

```
Num hyperparams to try for the theta model: 48.
Fold 0: Train size: 345 (2015-01-18 08:00:00 - 2015-02-01 16:00:00). Test Size: 115␣
↪(2015-02-01 17:00:00 - 2015-02-06 11:00:00).
Fold 1: Train size: 230 (2015-01-18 08:00:00 - 2015-01-27 21:00:00). Test Size: 115␣
↪(2015-01-27 22:00:00 - 2015-02-01 16:00:00).
Fold 2: Train size: 115 (2015-01-18 08:00:00 - 2015-01-23 02:00:00). Test Size: 115␣
↪(2015-01-23 03:00:00 - 2015-01-27 21:00:00).
Chosen paramaters: {'theta': 0.5, 'model_mode': <ModelMode.ADDITIVE: 'additive'>,
↪'season_mode': <SeasonalityMode.MULTIPLICATIVE: 'multiplicative'>, 'trend_mode':
↪<TrendMode.EXPONENTIAL: 'exponential'>}.
```

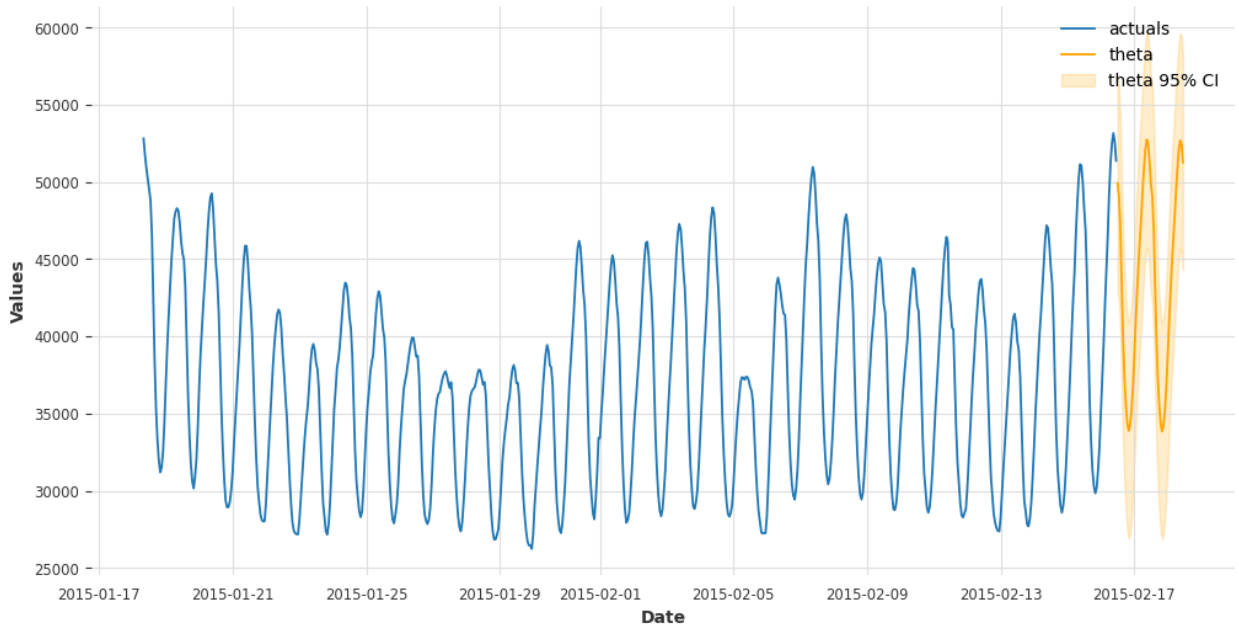
22.3 Visualize test results

[9]:
```
f.plot_test_set(ci=True)
plt.show()
```



22.4 Visualize forecast results

[10]:
```
f.plot(ci=True)
plt.show()
```



22.5 See in-sample and out-of-sample accuracy/error metrics

[12]:
```
results = f.export('model_summaries')
```

[13]:
```
results[
    [
        'TestSetSMAPE',
        'InSampleSMAPE',
        'TestSetR2',
        'InSampleR2',
        'ValidationMetric',
        'ValidationMetricValue',
        'TestSetLength'
    ]
]
```

[13]:

	TestSetSMAPE	InSampleSMAPE	TestSetR2	InSampleR2	ValidationMetric	ValidationMetricValue	TestSetLength
0	0.058274	0.014082	0.786943	0.984652	smape	0.064113	240

The validation metric displayed above is the average SMAPE across the three cross-validation folds.

[14]:
```
validation_grid = f.export_validation_grid('theta')
validation_grid.head()
```

[14]:
```
  theta                model_mode                     season_mode  \
0   0.5        ModelMode.ADDITIVE  SeasonalityMode.MULTIPLICATIVE
1   0.5        ModelMode.ADDITIVE  SeasonalityMode.MULTIPLICATIVE
2   0.5        ModelMode.ADDITIVE        SeasonalityMode.ADDITIVE
3   0.5        ModelMode.ADDITIVE        SeasonalityMode.ADDITIVE
4   0.5  ModelMode.MULTIPLICATIVE  SeasonalityMode.MULTIPLICATIVE

              trend_mode  Fold0Metric  Fold1Metric  Fold2Metric  \
0  TrendMode.EXPONENTIAL     0.056320     0.067335     0.068684
1       TrendMode.LINEAR     0.056510     0.072710     0.084188
2  TrendMode.EXPONENTIAL     0.055679     0.075179     0.073782
3       TrendMode.LINEAR     0.056044     0.081705     0.089734
4  TrendMode.EXPONENTIAL     0.056569     0.071406     0.080493

   AverageMetric MetricEvaluated
0       0.064113           smape
1       0.071136           smape
2       0.068213           smape
3       0.075827           smape
4       0.069489           smape
```

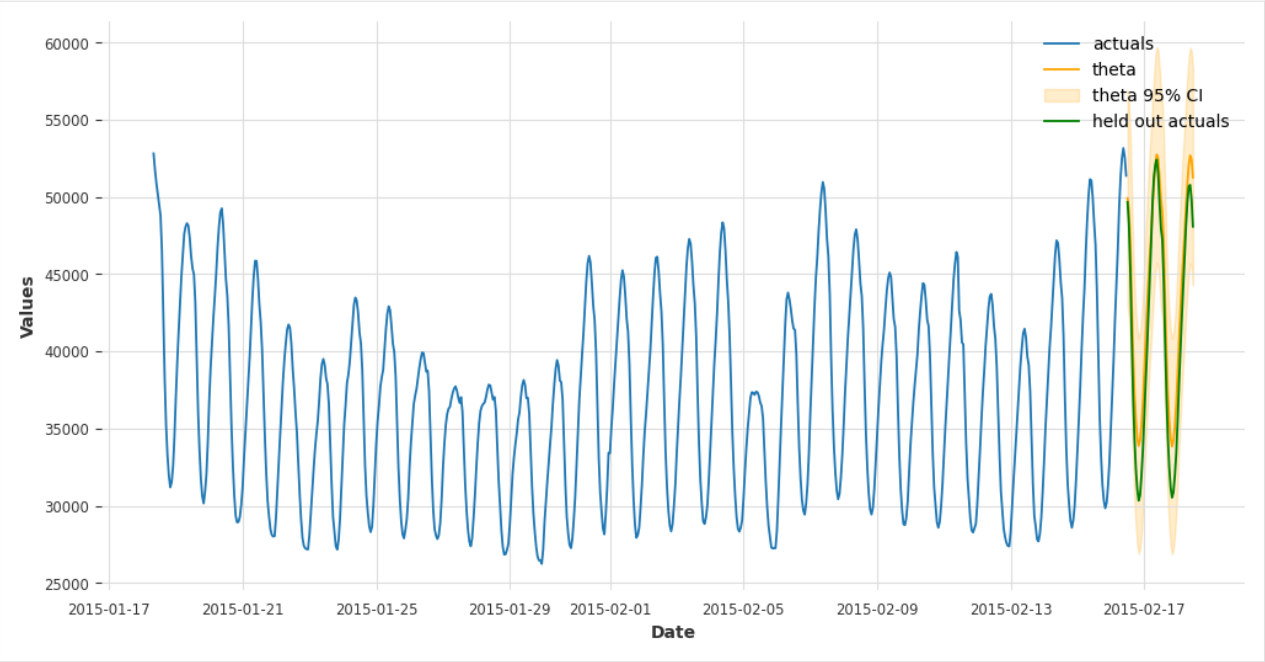
22.6 Test the forecast against out-of-sample data

- this is data the `Forecaster` object has never seen

[15]:
```
fcst = f.export('lvl_fcsts')
fcst.head()
```

[15]:
```
                 DATE         theta
0 2015-02-16 12:00:00  49921.533332
1 2015-02-16 13:00:00  49135.400143
2 2015-02-16 14:00:00  47126.062514
3 2015-02-16 15:00:00  43417.987575
4 2015-02-16 16:00:00  39867.287257
```

[16]:
```
fig, ax = plt.subplots(figsize=(12,6))
f.plot(ax=ax,ci=True)
sns.lineplot(
    x = f.future_dates,
    y = y_test,
    ax = ax,
    label = 'held out actuals',
    color = 'green',
)
plt.show()
```



[17]:
```
smape = metrics.smape(y_test,fcst['theta'])
smape
```

[17]:
```
0.05764507814606441
```

[ ]:
```
```

CHAPTER

TWENTYTHREE

VALIDATION

There are many ways to validate a model with scalecast and this notebook introduces them and overviews the differences between dynamic and non-dynamic tuning/testing, cross-validation, backtesting, and the eye test.

- Download data: https://www.kaggle.com/robervalt/sunspots
- See here for EDA on this dataset: https://scalecast-examples.readthedocs.io/en/latest/rnn/rnn.html
- See here for documentation on cross validation: https://scalecast.readthedocs.io/en/latest/Forecaster/Forecaster.html#src.scalecast.F

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scalecast.Forecaster import Forecaster
import diagram_creator
```

[2]:
```
def prepare_fcst(f, test_length=0.1, fcst_length=120):
    """ adds all variables and sets the test length/forecast length in the object

    Args:
        f (Forecaster): the Forecaster object.
        test_length (int or float): the test length as a size or proportion.
        fcst_length (int): the forecast horizon.

    Returns:
        (Forecaster) the processed object.
    """
    f.generate_future_dates(fcst_length)
    f.set_test_length(test_length)
    f.set_validation_length(f.test_length)
    f.eval_cis()
    f.add_seasonal_regressors("month")
    for i in np.arange(60, 289, 12):  # 12-month cycles from 12 to 288 months
        f.add_cycle(i)
    f.add_ar_terms(120)  # AR 1-120
    f.add_AR_terms((20, 12))  # seasonal AR up to 20 years, spaced one year apart
    f.add_seasonal_regressors("year")
    #f.auto_Xvar_select(irr_cycles=[120],estimator='gbt')
    return f


def export_results(f):
```

(continues on next page)

(continued from previous page)

```
    """ returns a dataframe with all model results given a Forecaster object.

    Args:
        f (Forecaster): the Forecaster object.

    Returns:
        (DataFrame) the dataframe with the pertinent results.
    """
    results = f.export("model_summaries", determine_best_by="TestSetMAE")
    return results[
        [
            "ModelNickname",
            "TestSetRMSE",
            "InSampleRMSE",
            "ValidationMetric",
            "ValidationMetricValue",
            "HyperParams",
            "TestSetLength",
            "DynamicallyTested",
        ]
    ]
```

23.1 Load Forecaster Object

- we choose 120 periods (10 years) for all validation and forecasting
- 10 years of observervations to tune model hyperparameters, 10 years to test, and a forecast horizon of 10 years

[3]:
```
df = pd.read_csv("Sunspots.csv", index_col=0, names=["Date", "Target"], header=0)
f = Forecaster(y=df["Target"], current_dates=df["Date"])
prepare_fcst(f)
```

[3]:
```
Forecaster(
    DateStartActuals=1749-01-31T00:00:00.000000000
    DateEndActuals=2021-01-31T00:00:00.000000000
    Freq=M
    N_actuals=3265
    ForecastLength=120
    Xvars=['month', 'cycle60sin', 'cycle60cos', 'cycle72sin', 'cycle72cos', 'cycle84sin',
↪ 'cycle84cos', 'cycle96sin', 'cycle96cos', 'cycle108sin', 'cycle108cos', 'cycle120sin',
↪ 'cycle120cos', 'cycle132sin', 'cycle132cos', 'cycle144sin', 'cycle144cos',
↪'cycle156sin', 'cycle156cos', 'cycle168sin', 'cycle168cos', 'cycle180sin', 'cycle180cos
↪', 'cycle192sin', 'cycle192cos', 'cycle204sin', 'cycle204cos', 'cycle216sin',
↪'cycle216cos', 'cycle228sin', 'cycle228cos', 'cycle240sin', 'cycle240cos', 'cycle252sin
↪', 'cycle252cos', 'cycle264sin', 'cycle264cos', 'cycle276sin', 'cycle276cos',
↪'cycle288sin', 'cycle288cos', 'AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR8',
↪'AR9', 'AR10', 'AR11', 'AR12', 'AR13', 'AR14', 'AR15', 'AR16', 'AR17', 'AR18', 'AR19',
↪'AR20', 'AR21', 'AR22', 'AR23', 'AR24', 'AR25', 'AR26', 'AR27', 'AR28', 'AR29', 'AR30',
↪ 'AR31', 'AR32', 'AR33', 'AR34', 'AR35', 'AR36', 'AR37', 'AR38', 'AR39', 'AR40', 'AR41
↪', 'AR42', 'AR43', 'AR44', 'AR45', 'AR46', 'AR47', 'AR48', 'AR49', 'AR50', 'AR51',
↪'AR52', 'AR53', 'AR54', 'AR55', 'AR56', 'AR57', 'AR58', 'AR59', 'AR60', 'AR61', 'AR62',
```

(continues on next page)

(continued from previous page)

```
↪ 'AR63', 'AR64', 'AR65', 'AR66', 'AR67', 'AR68', 'AR69', 'AR70', 'AR71', 'AR72', 'AR73
↪', 'AR74', 'AR75', 'AR76', 'AR77', 'AR78', 'AR79', 'AR80', 'AR81', 'AR82', 'AR83',
↪'AR84', 'AR85', 'AR86', 'AR87', 'AR88', 'AR89', 'AR90', 'AR91', 'AR92', 'AR93', 'AR94',
↪ 'AR95', 'AR96', 'AR97', 'AR98', 'AR99', 'AR100', 'AR101', 'AR102', 'AR103', 'AR104',
↪'AR105', 'AR106', 'AR107', 'AR108', 'AR109', 'AR110', 'AR111', 'AR112', 'AR113', 'AR114
↪', 'AR115', 'AR116', 'AR117', 'AR118', 'AR119', 'AR120', 'AR132', 'AR144', 'AR156',
↪'AR168', 'AR180', 'AR192', 'AR204', 'AR216', 'AR228', 'AR240', 'year']
    TestLength=326
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

[4]:
```
f.set_estimator('gbt')
```

In the feature_selection notebook, gbt was chosen as the best model class out of several tried. We will show all examples with this estimator.

23.2 Default Model Parameters

- one with dynamic testing
- one with non-dynamic testing
- the difference can be expressed by taking the case of a simple autoregressive model, such that:

23.2.1 Non-Dynamic Autoregressive Predictions

$$x_t = \alpha * x_{t-1} + e_t$$

Over an indefinite forecast horizon, the above equation would only work if you knew the value for x_{t-1}. Going more than one period into the future, you would stop knowing what that value is. In a test-set of data, of course, you do know all values into the forecast horizon, but to be more realistic, you could write an equation for a two-step forecast like this:

23.2.2 Dynamic Autoregressive Predictions

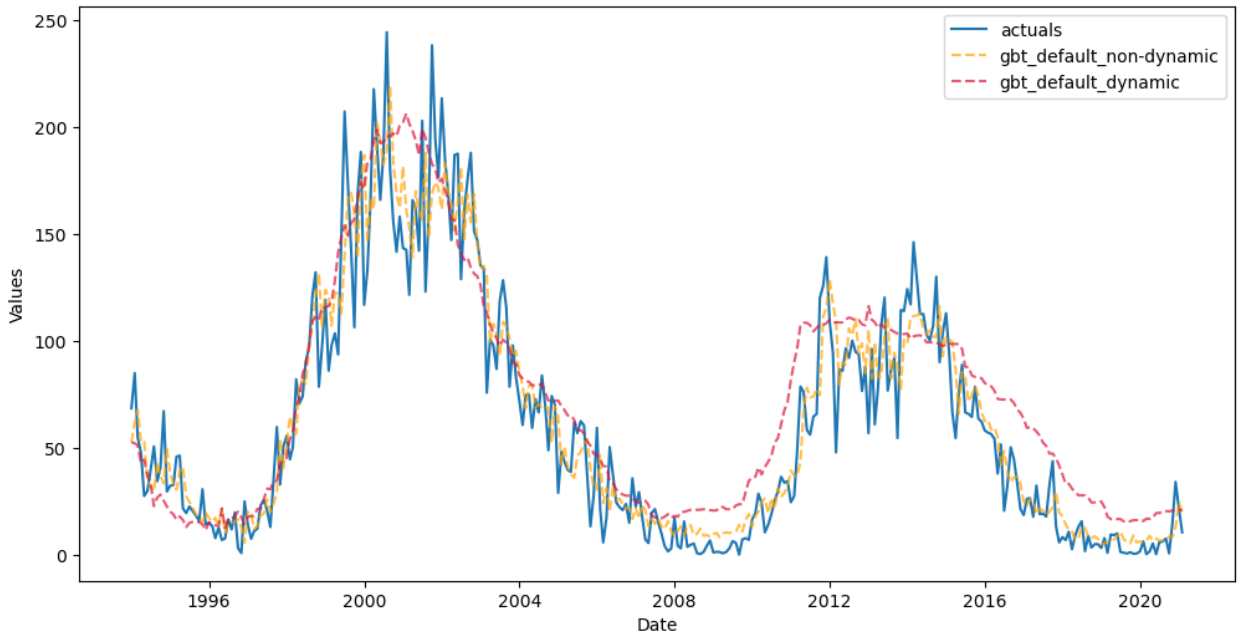
$$\hat{x_t} = \hat{\alpha} * x_{t-1}$$

$$x_{t+1} = \hat{\alpha} * \hat{x}_t + e_{t+1}$$

Using these two equations, which scalecast refers to as dynamic forecasting, you could evaluate any forecast horizon by plugging in predicted values for x_{t-1} or x_t over periods in which you did not know it. This is default behavior for all models tested through scalecast, but it is not default for tuning models. We will explore dynamic tuning soon. First, let's see in practical terms the difference between non-dynamic and dynamic testing.

[5]:
```
f.manual_forecast(call_me="gbt_default_non-dynamic", dynamic_testing=False)
f.manual_forecast(call_me="gbt_default_dynamic")  # default is dynamic testing
```

[6]:
```
f.plot_test_set(
    models=[
        "gbt_default_non-dynamic",
        "gbt_default_dynamic"
    ],
    include_train=False,
)
plt.show()
```



It appears that the non-dynamically tested model performed significantly better than the other, but looks can be deceiving. In essence, the non-dynamic model was only tested for its ability to perform 326 one-step forecasts and its final metric is an average of these one-step forecasts. It could be a good idea to set `dynamic_testing=False` if you want to speed up the testing process or if you only care about how your model would perform one step into the future. But to report the test-set metric from this model as if it could be expected to do that well for the full 326 periods into the future is misleading. The other model that was dynamically tested can be more realistically trusted in that regard.

[7]:
```
export_results(f)
```

[7]:

	ModelNickname	TestSetRMSE	InSampleRMSE	ValidationMetric	ValidationMetricValue	HyperParams	TestSetLength	DynamicallyTested
0	gbt_default_non-dynamic	18.723179	18.841544	NaN	NaN	{}	326	False
1	gbt_default_dynamic	24.456923	18.841544	NaN	NaN	{}	326	True

23.3 Tune the model to find optimal hyperparameters

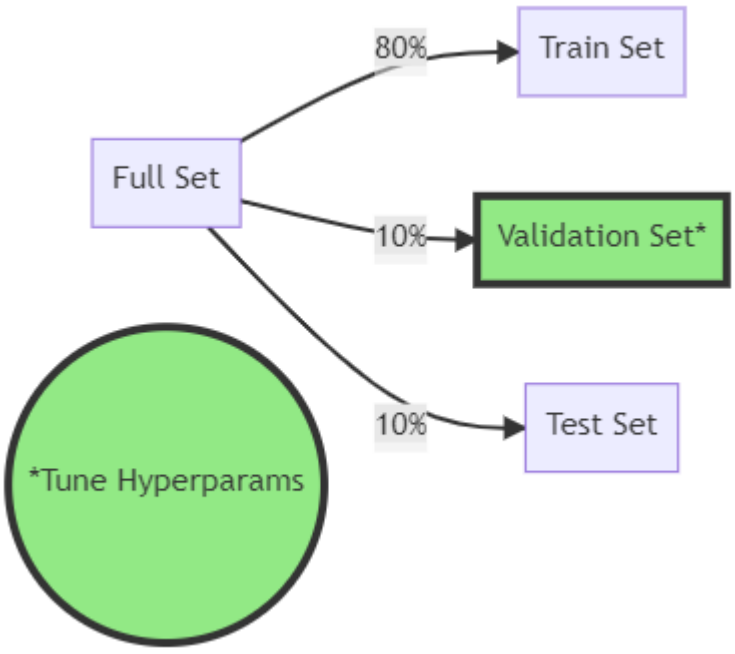
- Create a validation grid
- Try three strategies to tune the parameters:
- Train/validation/test split
  - Hyperparameters are tried on the validation set
- Train/test split with 5-fold time-series cross-validation on training set
  - Training data split 5 times into train/validations set
  - Models trained on training set only
  - Validated out-of-sample
  - All data available before each validation split sequentially used to train the model
- Train/test split with 5-fold time-series rolling cross-validation on training set
  - Rolling is different in that each train/validation split is the same size

[8]:
```
grid = {
    "max_depth": [2, 3, 5],
    "max_features": ["sqrt", "auto"],
    "subsample": [0.8, 0.9, 1],
}
```

[9]:
```
f.ingest_grid(grid)
```

23.3.1 Train/Validation/Test Split

- The data's sequence is always maintained in time-series splits with scalecast



[10]:
```
f.tune(dynamic_tuning=True)
f.auto_forecast(
    call_me="gbt_tuned"
)  # automatically uses optimal paramaeters suggested from the tuning process
```

[11]:
```
f.export_validation_grid("gbt_tuned").sort_values('AverageMetric').head(10)
```

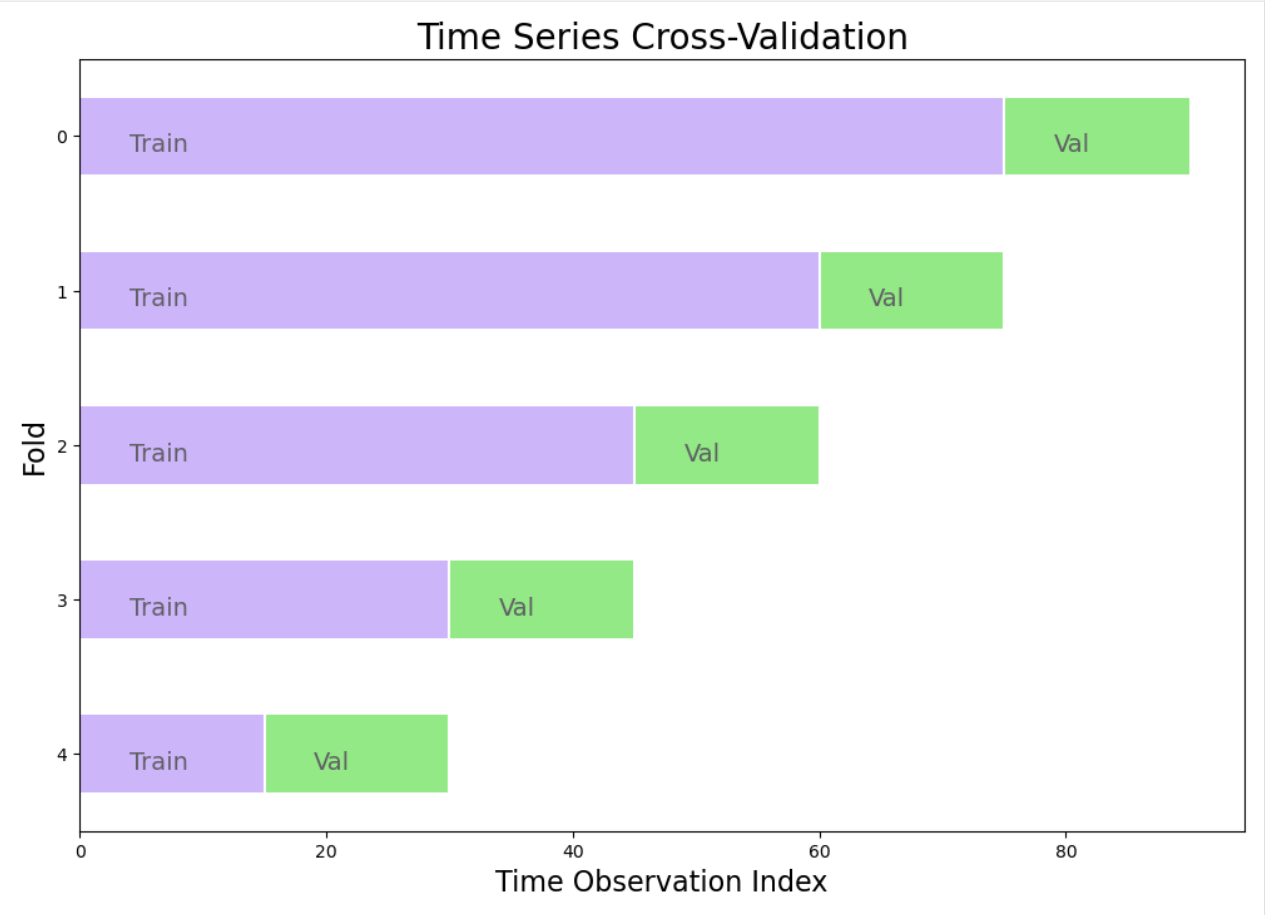
[11]:

	max_depth	max_features	subsample	Fold0Metric	AverageMetric	MetricEvaluated
7	3	sqrt	0.9	43.623057	43.623057	rmse
12	5	sqrt	0.8	43.667072	43.667072	rmse
6	3	sqrt	0.8	44.015361	44.015361	rmse
8	3	sqrt	1	44.807485	44.807485	rmse
0	2	sqrt	0.8	46.093143	46.093143	rmse
2	2	sqrt	1	53.206496	53.206496	rmse
14	5	sqrt	1	60.587243	60.587243	rmse
10	3	auto	0.9	65.424393	65.424393	rmse
15	5	auto	0.8	66.530315	66.530315	rmse
13	5	sqrt	0.9	67.459775	67.459775	rmse

23.3.2 5-Fold Time Series Cross Validation

- Split training set into k (5) folds
- Each validation set is the same size and determined such that: `val_size = n_obs // (folds + 1)`
- The last training set will be the same size or almost the same size as the validation sets
- Model trained and re-trained with all data that came before each validation slice
- Each fold tested out of sample on its validation set
- Final error is an average of the out-of-sample error obtained from each fold
- The chosen hyperparameters are determined by which final error was minimized
- Below is an example with a dataset sized 100 observations and in which 10 observations are held out for testing and the remaining 90 observations are used as the training set:

[12]:
```
diagram_creator.create_cv()
```



The final error, E, can be expressed as an average of the error from each fold i:

$$E = \frac{1}{n}\sum_{i=0}^{n-1}(e_i)$$

[13]:
```
f.cross_validate(
    k=5,
    dynamic_tuning=True,
    test_length = None, # default so that last test and train sets are same size (or␣
↪close to the same)
    train_length = None, # default uses all observations before each test set
    space_between_sets = None, # default adds a length equal to the test set between␣
↪consecutive sets
    verbose = True, # print out info about each fold
)
f.auto_forecast(call_me="gbt_cv")
```

```
Num hyperparams to try for the gbt model: 18.
Fold 0: Train size: 2450 (1749-01-31 00:00:00 - 1953-02-28 00:00:00). Test Size: 489␣
↪(1953-03-31 00:00:00 - 1993-11-30 00:00:00).
Fold 1: Train size: 1961 (1749-01-31 00:00:00 - 1912-05-31 00:00:00). Test Size: 489␣
↪(1912-06-30 00:00:00 - 1953-02-28 00:00:00).
Fold 2: Train size: 1472 (1749-01-31 00:00:00 - 1871-08-31 00:00:00). Test Size: 489␣
```

(continues on next page)

(continued from previous page)

```
↪(1871-09-30 00:00:00 - 1912-05-31 00:00:00).
Fold 3: Train size: 983 (1749-01-31 00:00:00 - 1830-11-30 00:00:00). Test Size: 489␣
↪(1830-12-31 00:00:00 - 1871-08-31 00:00:00).
Fold 4: Train size: 494 (1749-01-31 00:00:00 - 1790-02-28 00:00:00). Test Size: 489␣
↪(1790-03-31 00:00:00 - 1830-11-30 00:00:00).
Chosen paramaters: {'max_depth': 5, 'max_features': 'sqrt', 'subsample': 1}.
```

[14]:
```
f.export_validation_grid("gbt_cv").sort_values("AverageMetric").head(10)
```

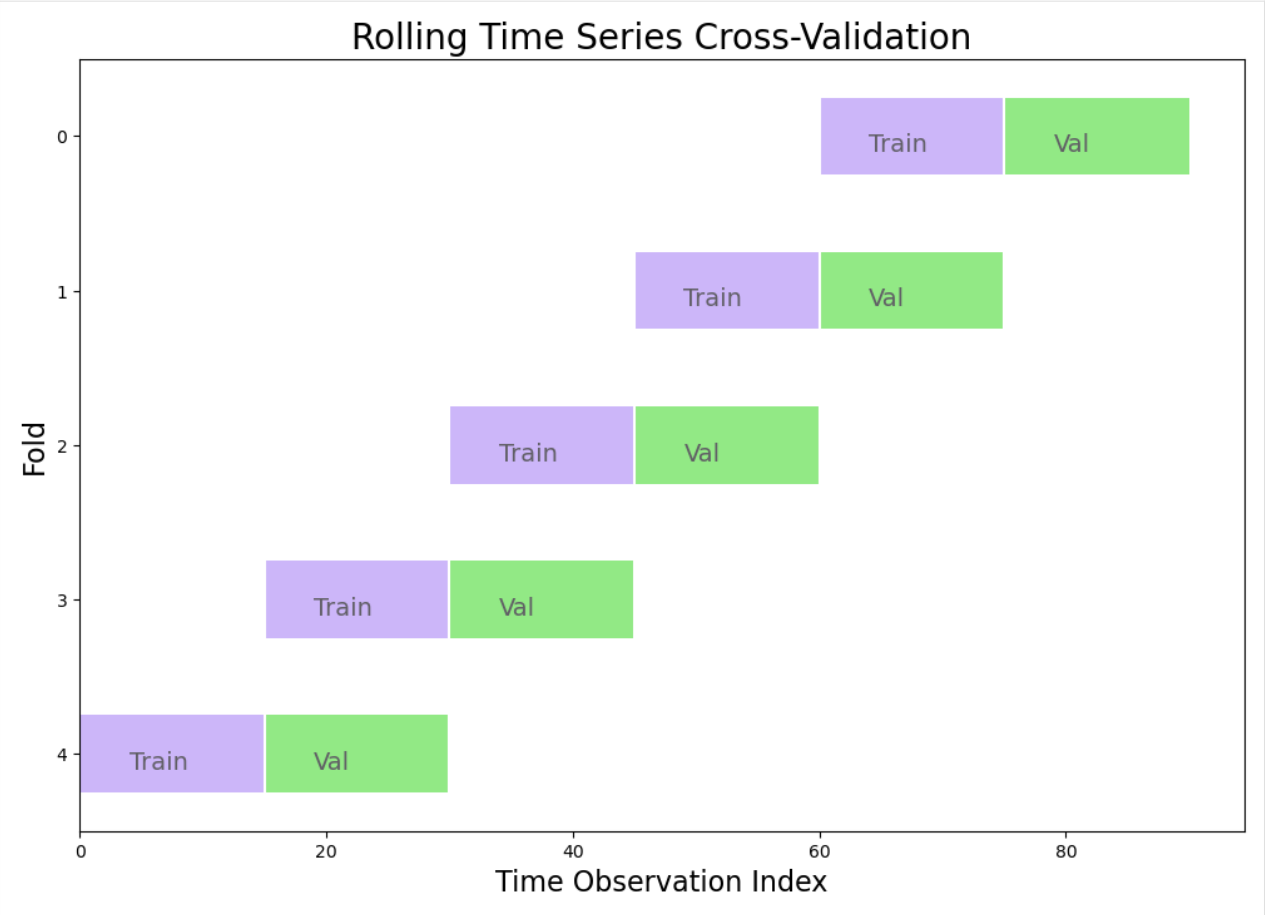
[14]:

	max_depth	max_features	subsample	Fold0Metric	Fold1Metric	Fold2Metric	Fold3Metric	Fold4Metric	AverageMetric	MetricEvaluated
14	5	sqrt	1	75.881634	85.585377	79.162305	83.264294	96.912215	84.161165	rmse
12	5	sqrt	0.8	90.328729	96.703967	48.606034	80.703070	111.586242	85.585609	rmse
13	5	sqrt	0.9	108.327532	95.594713	61.082475	61.798101	108.437289	87.048022	rmse
16	5	auto	0.9	89.047882	92.811981	74.332318	86.321627	112.510827	91.004927	rmse
0	2	sqrt	0.8	105.541098	78.227464	103.386493	77.997451	115.573720	96.145245	rmse
8	3	sqrt	1	64.668019	119.509341	93.877028	93.740770	114.533147	97.265661	rmse
7	3	sqrt	0.9	75.061438	85.898970	122.575339	91.157722	119.005642	98.739822	rmse
2	2	sqrt	1	59.907395	88.529044	113.535870	118.721240	117.856393	99.709989	rmse
17	5	auto	1	111.676983	96.506098	70.688360	102.882864	118.365703	100.024002	rmse
6	3	sqrt	0.8	93.640946	85.464725	100.451257	109.813079	111.415691	100.157140	rmse

23.3.3 5-Fold Rolling Time Series Cross Validation

- Split training set into k (5) folds
- Each validation set is the same size
- Each training set is also the same size as each validation set
- Each fold tested out of sample
- Final error is an average of the out-of-sample error obtained from each folds
- The chosen hyperparameters are determined by which final error was minimized
- Below is an example with a dataset sized 100 observation and in which 10 observations are held out for testing and the remaining 90 observations are used as the training set:

[15]:
```
diagram_creator.create_rolling_cv()
```



[16]:
```
f.cross_validate(
    k=5,
    rolling=True,
    dynamic_tuning=True,
    test_length = None, # with rolling = True, makes all train and test sets the same␣
↪size
    train_length = None, # with rolling = True, makes all train and test sets the same␣
↪size
    space_between_sets = None, # default adds a length equal to the test set between␣
↪consecutive sets
    verbose = True, # print out info about each fold
)
f.auto_forecast(call_me="gbt_rolling_cv")
```

```
Num hyperparams to try for the gbt model: 18.
Fold 0: Train size: 489 (1912-06-30 00:00:00 - 1953-02-28 00:00:00). Test Size: 489␣
↪(1953-03-31 00:00:00 - 1993-11-30 00:00:00).
Fold 1: Train size: 489 (1871-09-30 00:00:00 - 1912-05-31 00:00:00). Test Size: 489␣
↪(1912-06-30 00:00:00 - 1953-02-28 00:00:00).
Fold 2: Train size: 489 (1830-12-31 00:00:00 - 1871-08-31 00:00:00). Test Size: 489␣
↪(1871-09-30 00:00:00 - 1912-05-31 00:00:00).
Fold 3: Train size: 489 (1790-03-31 00:00:00 - 1830-11-30 00:00:00). Test Size: 489␣
↪(1830-12-31 00:00:00 - 1871-08-31 00:00:00).
Fold 4: Train size: 489 (1749-06-30 00:00:00 - 1790-02-28 00:00:00). Test Size: 489␣
```

(continues on next page)

(continued from previous page)

```
↪(1790-03-31 00:00:00 - 1830-11-30 00:00:00).
Chosen paramaters: {'max_depth': 5, 'max_features': 'sqrt', 'subsample': 0.9}.
```

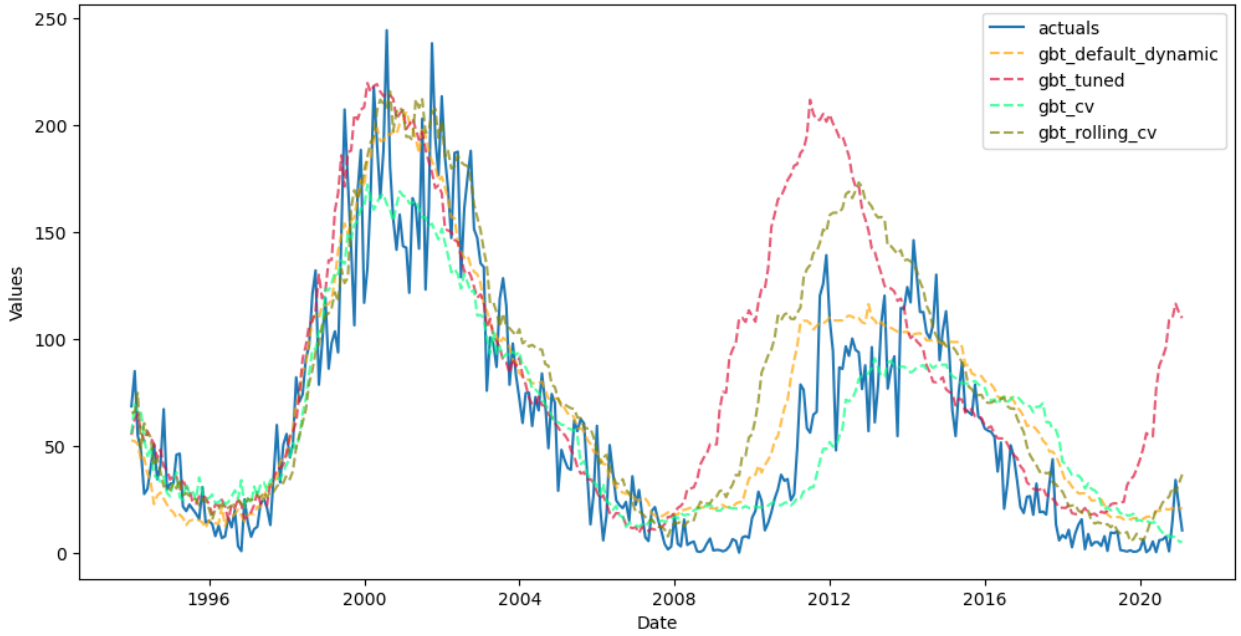
[17]:
```
f.export_validation_grid("gbt_rolling_cv").sort_values("AverageMetric").head(10)
```

[17]:

	max_depth	max_features	subsample	Fold0Metric	Fold1Metric	Fold2Metric	Fold3Metric	Fold4Metric	AverageMetric	MetricEvaluated
13	5	sqrt	0.9	52.067612	75.101589	47.665446	73.458303	110.673685	71.793327	rmse
11	3	auto	1	61.780860	70.554830	48.914548	74.467054	110.351684	73.213795	rmse
9	3	auto	0.8	59.526670	76.807231	48.096324	74.276742	110.628222	73.867038	rmse
17	5	auto	1	68.884924	63.475008	52.178487	73.895434	111.100209	73.906812	rmse
7	3	sqrt	0.9	44.798329	69.131312	54.226665	93.287371	115.134538	75.315643	rmse
8	3	sqrt	1	56.409051	84.081666	49.339527	75.453036	111.791536	75.414963	rmse
3	2	auto	0.8	61.498220	73.383101	48.898466	78.773421	115.272207	75.565083	rmse
12	5	sqrt	0.8	63.732031	78.140692	47.053796	73.473322	115.558499	75.591668	rmse
5	2	auto	1	69.075181	67.852121	53.383812	75.150129	115.413148	76.174878	rmse
14	5	sqrt	1	48.860162	84.807936	49.921226	89.144401	110.995682	76.745881	rmse

23.3.4 View results

[18]:
```
f.plot_test_set(
    models=[
        "gbt_default_dynamic",
        "gbt_tuned",
        "gbt_cv",
        "gbt_rolling_cv",
    ],
    include_train=False,
)
plt.show()
```



[19]:
```
pd.set_option('max_colwidth', 60)
export_results(f)
```

[19]:
```
             ModelNickname  TestSetRMSE  InSampleRMSE ValidationMetric  \
0  gbt_default_non-dynamic    18.723179     18.841544              NaN
1      gbt_default_dynamic    24.456923     18.841544              NaN
2                   gbt_cv    25.604943     13.063379             rmse
3            gbt_rolling_cv    34.248137     12.789779             rmse
4                 gbt_tuned    50.777958     20.342340             rmse

   ValidationMetricValue  \
0                    NaN
1                    NaN
2              84.161165
3              71.793327
4              43.623057

                                                HyperParams  TestSetLength  \
0                                                        {}            326
1                                                        {}            326
2    {'max_depth': 5, 'max_features': 'sqrt', 'subsample': 1}            326
3  {'max_depth': 5, 'max_features': 'sqrt', 'subsample': 0.9}            326
4  {'max_depth': 3, 'max_features': 'sqrt', 'subsample': 0.9}            326

   DynamicallyTested
0              False
1               True
2               True
3               True
4               True
```

23.4 Backtest models

- Backtesting is a process in which the final chosen model is re-validated by seeing its average performance on the last x-number of forecast horizons available in the data
- With scalecast, pipeline objects can be built and backtest all applied models to see the best one over several forecast horizons
- See the documentation for more information

[20]:
```
from scalecast.Pipeline import Pipeline
from scalecast.util import backtest_metrics
```

[21]:
```
def forecaster(f):
    f.set_estimator('gbt')
    f.set_validation_length(int(len(f.y)*.1)) # 10% val length each time
    f.add_seasonal_regressors("month")
    for i in np.arange(60, 289, 12):  # 12-month cycles from 12 to 288 months
        f.add_cycle(i)
    f.add_ar_terms(120)  # AR 1-120
    f.add_AR_terms((20, 12))  # seasonal AR up to 20 years, spaced one year apart
    f.add_seasonal_regressors("year")

    f.manual_forecast(call_me='gbt_default_dynamic')

    f.ingest_grid(grid)

    f.tune(dynamic_tuning = True)
    f.auto_forecast(call_me='gbt_tuned')

    f.cross_validate(dynamic_tuning = True)
    f.auto_forecast(call_me='gbt_cv')

    f.cross_validate(dynamic_tuning = True, rolling = True)
    f.auto_forecast(call_me='gbt_rolling_cv')
```

[22]:
```
pipeline = Pipeline(steps = [('Forecast',forecaster)])
```

[23]:
```
backtest_results = pipeline.backtest(
    f,
    test_length = 0,
    fcst_length = 120,
    jump_back = 120, # place 120 obs between each training set
    cis = False,
    verbose = True,
)
```

[24]:
```
bm = backtest_metrics(backtest_results,mets=['rmse','mae','r2'])
bm
```

[24]:

Model	Metric	Iter0	Iter1	Iter2	Iter3
gbt_default_dynamic	rmse	29.787489	32.631984	40.218024	57.655588

(continues on next page)

(continued from previous page)

```
                    mae     24.725663  25.120353   32.70369  45.658795
                    r2       0.499971   0.718315   0.652106   0.466619
gbt_tuned           rmse    21.793363  38.201181  34.145031  55.637363
                    mae     17.232072  29.314773  26.055781   42.62898
                    r2       0.732345   0.613962   0.749239   0.503307
gbt_cv              rmse    27.290614  30.871908  38.859572  55.141773
                    mae     21.956079  24.026144  30.078019  42.157045
                    r2       0.580286   0.747883   0.675211   0.512116
gbt_rolling_cv      rmse    26.080319    31.8572  34.485633  43.732603
                    mae     21.003643  24.701014  27.536833  37.057153
                    r2       0.616687   0.731533   0.744211   0.693122

                                Iter4    Average
Model               Metric
gbt_default_dynamic rmse    88.550393  49.768696
                    mae     56.387906  36.919281
                    r2      -0.415585   0.384285
gbt_tuned           rmse    99.862839  49.927956
                    mae     63.122565  35.670834
                    r2      -0.800374   0.359696
gbt_cv              rmse    35.648112  37.562396
                    mae      27.85357  29.214172
                    r2       0.770582   0.657215
gbt_rolling_cv      rmse    38.312404  34.893632
                    mae     28.484026  27.756534
                    r2       0.735007   0.704112
```

[25]:
```
bmr = bm.reset_index()
```

[26]:
```
bmrrmse = bmr.loc[bmr['Metric'] == 'rmse'].sort_values('Average')
best_model = bmrrmse.iloc[0,0]
best_model
```

[26]:
```
'gbt_rolling_cv'
```

[27]:
```
rmse = bmr.loc[
    (bmr['Model'] == best_model) & (bmr['Metric'] == 'rmse'),
    'Average',
].values[0]
mae = bmr.loc[
    (bmr['Model'] == best_model) & (bmr['Metric'] == 'mae'),
    'Average',
].values[0]
r2 = bmr.loc[
    (bmr['Model'] == best_model) & (bmr['Metric'] == 'r2'),
    'Average',
].values[0]
```

[28]:
```
print(
    f"The best model, according to the RMSE over 5 backtest iterations was {best_model}."
    " On average, we can expect this model to have an RMSE of"
```

(continues on next page)

(continued from previous page)

```
    f" {rmse:.2f}, MAE of {mae:.2f}, and R2 of {r2:.2%} over a full 120-period"
    " forecast window."
)
```

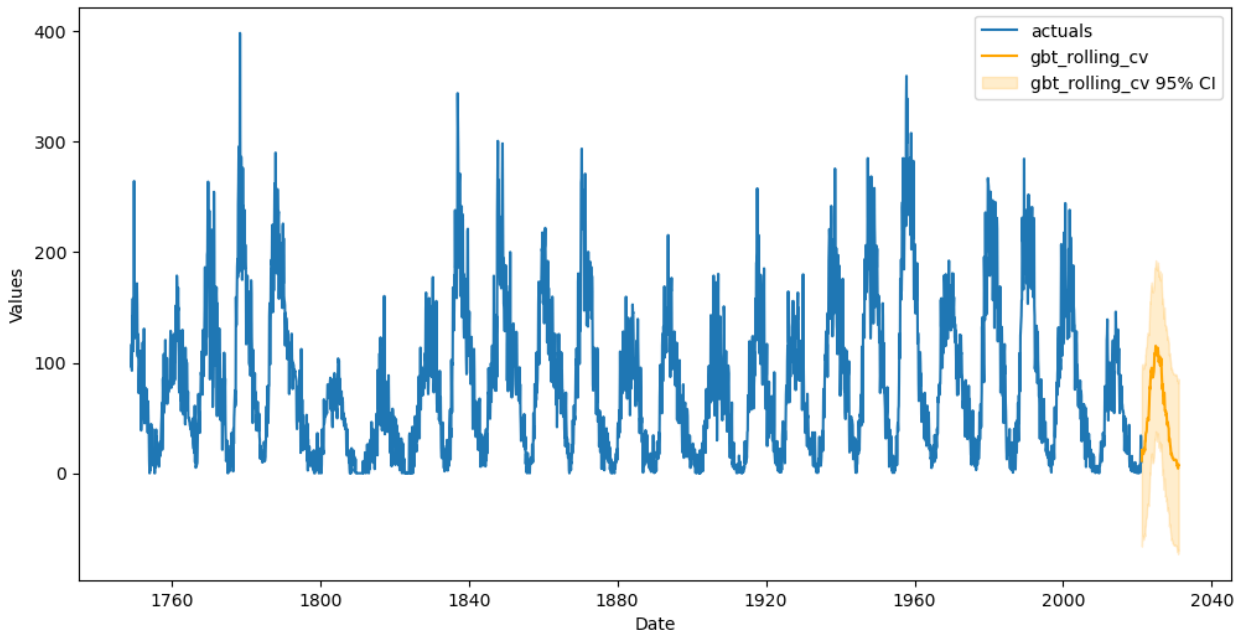
```
The best model, according to the RMSE over 5 backtest iterations was gbt_rolling_cv. On␣
↪average, we can expect this model to have an RMSE of 34.89, MAE of 27.76, and R2 of 70.
↪41% over a full 120-period forecast window.
```

An extension of this analysis could be to choose regressors more carefully (see here) and to use more complex models (see here).

23.5 The Eye Test

In addition to all the objective validation performed in this notebook, one of the most important questions to ask is if the forecast looks reasonable. Does it pass the common sense test? Below, we plot the 120-forecast period horizon from the best model.

[29]:
```
f.plot(models=best_model, ci = True)
plt.show()
```



[ ]:
```

```

CHAPTER

TWENTYFOUR

VECM

- Using a VECM to predict FANG stocks
- See the VECM documentation

[1]:
```
import pandas as pd
import numpy as np
from pandas_datareader import data as pdr
import yfinance as yf
from scalecast.Forecaster import Forecaster
from scalecast.MVForecaster import MVForecaster
from scalecast.Pipeline import Transformer, Reverter, MVPipeline
from scalecast.util import (
    find_optimal_lag_order,
    find_optimal_coint_rank,
    Forecaster_with_missing_vals,
)
from scalecast.auxmodels import vecm
from scalecast.multiseries import export_model_summaries
from scalecast import GridGenerator
import matplotlib.pyplot as plt
```

[2]:
```
yf.pdr_override()
```

24.1 Download data using a public API

[3]:
```
FANG = [
    'META',
    'AMZN',
    'NFLX',
    'GOOG',
]

fs = []
for sym in FANG:
    df = pdr.get_data_yahoo(sym)
    # since the api doesn't send the data exactly in Business-day frequency
    # we can correct it using this function
    f = Forecaster_with_missing_vals(
```

(continues on next page)

(continued from previous page)

```
        y=df['Close'],
        current_dates = df.index,
        future_dates = 65,
        end = '2022-09-30',
        desired_frequency = 'B',
        fill_strategy = 'linear_interp',
        add_noise = True,
        noise_lookback = 5,
    )
    fs.append(f)

mvf = MVForecaster(*fs,names=FANG,test_length=65)
mvf.set_validation_metric('rmse')
mvf.add_sklearn_estimator(vecm,'vecm')

mvf
```

```
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
```

[3]:
```
MVForecaster(
    DateStartActuals=2012-05-18T00:00:00.000000000
    DateEndActuals=2023-08-03T00:00:00.000000000
    Freq=B
    N_actuals=2925
    N_series=4
    SeriesNames=['META', 'AMZN', 'NFLX', 'GOOG']
    ForecastLength=65
    Xvars=[]
    TestLength=65
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    OptimizeOn=mean
    GridsFile=MVGrids
)
```

[4]:
```
mvf.plot()
plt.show()
```

24.2 Augmented Dickey Fuller Tests to Confirm Unit-1 Roots

[5]:
```
for stock, f in zip(FANG,fs):
    adf_result = f.adf_test(full_res=True)
    print('the stock {} is {}stationary at level'.format(
        stock,
        'not ' if adf_result[1] > 0.05 else ''
    )
    )
```

```
the stock META is not stationary at level
the stock AMZN is not stationary at level
the stock NFLX is not stationary at level
the stock GOOG is not stationary at level
```

[6]:
```
for stock, f in zip(FANG,fs):
    adf_result = f.adf_test(diffy=True,full_res=True)
    print('the stock {} is {}stationary at its first difference'.format(
        stock,
        'not ' if adf_result[1] > 0.05 else ''
    )
    )
```

```
the stock META is stationary at its first difference
the stock AMZN is stationary at its first difference
the stock NFLX is stationary at its first difference
the stock GOOG is stationary at its first difference
```

24.3 Measure IC to Find Optimal Lag Order

- this is used to run the cointegration test

[7]:
```
lag_test = find_optimal_lag_order(mvf,train_only=True)
pd.DataFrame(
    {
        'aic':lag_test.aic,
        'bic':lag_test.bic,
        'hqic':lag_test.hqic,
        'fpe':lag_test.fpe,
    },
    index = ['optimal lag order'],
).T
```

[7]:
```
      optimal lag order
aic                  27
bic                   1
hqic                  3
fpe                  27
```

24.4 Johansen cointegration test

[8]:
```
coint_res = find_optimal_coint_rank(
    mvf,
    det_order=1,
    k_ar_diff=10,
    train_only=True,
)
print(coint_res)
coint_res.rank
```

```
Johansen cointegration test using trace test statistic with 5% significance level
=====================================
r_0 r_1 test statistic critical value
-------------------------------------
  0   4          56.60          55.25
  1   4          33.50          35.01
-------------------------------------
```

[8]:
```
1
```

We found a cointegration rank of 1.

24.5 Run VECM

- Now, we can specify a grid that will try more lags, deterministic terms, seasonal fluctuations, and cointegration ranks of 0 and 1

[9]:
```
vecm_grid = dict(
    lags = [0], # required to set this to 0 for the vecm model in scalecast
    freq = ['B'], # only necessary to suppress a warning
    k_ar_diff = range(1,66), # lag orders to try
    coint_rank = [0,1],
    deterministic = ["n","co","lo","li","cili","colo"],
    seasons = [0,5,30,65,260],
)

mvf.set_estimator('vecm')
mvf.ingest_grid(vecm_grid)
mvf.limit_grid_size(100,random_seed=20)
mvf.cross_validate(k=3,verbose=True)
mvf.auto_forecast()

results = mvf.export('model_summaries')
results[[
    'ModelNickname',
    'Series',
    'TestSetRMSE',
    'TestSetMAE',
]]
```

```
Num hyperparams to try for the vecm model: 100.
Fold 0: Train size: 2145 (2012-05-18 00:00:00 - 2020-08-06 00:00:00). Test Size: 715␣
↪(2020-08-07 00:00:00 - 2023-05-04 00:00:00).
Fold 1: Train size: 1430 (2012-05-18 00:00:00 - 2017-11-09 00:00:00). Test Size: 715␣
↪(2017-11-10 00:00:00 - 2020-08-06 00:00:00).
Fold 2: Train size: 715 (2012-05-18 00:00:00 - 2015-02-12 00:00:00). Test Size: 715␣
↪(2015-02-13 00:00:00 - 2017-11-09 00:00:00).
Chosen paramaters: {'lags': 0, 'freq': 'B', 'k_ar_diff': 28, 'coint_rank': 1,
↪'deterministic': 'li', 'seasons': 5}.
```

[9]:

	ModelNickname	Series	TestSetRMSE	TestSetMAE
0	vecm	META	50.913814	44.133838
1	vecm	AMZN	23.752750	22.348669
2	vecm	NFLX	113.705564	105.310047
3	vecm	GOOG	18.846431	18.041912

24.6 View VECM Results

[10]:
```
results['TestSetRMSE'].mean()
```

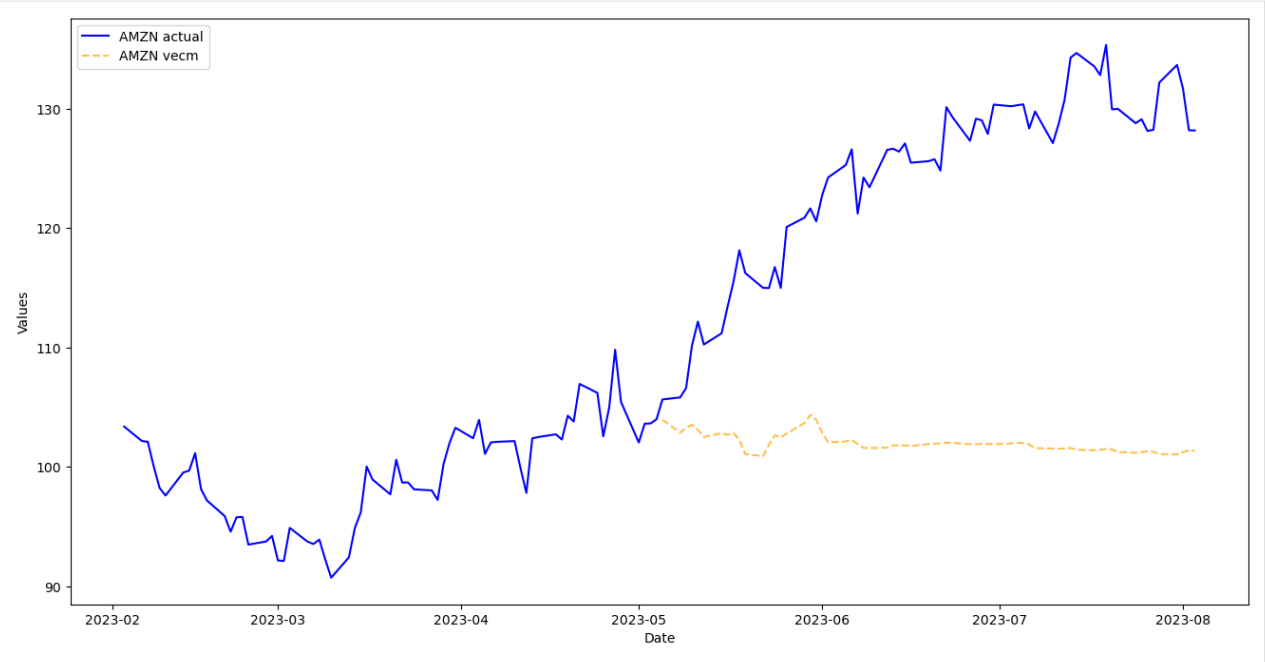
[10]:
```
51.80463965264752
```

[11]:
```
mvf.export_validation_grid('vecm').sample(15)
```

[11]:

	lags	freq	k_ar_diff	coint_rank	deterministic	seasons	Fold0Metric	Fold1Metric	Fold2Metric	AverageMetric	MetricEvaluated	Optimized On
71	0	B	58	0	colo	0	162.599874	44.268484	26.594518	77.820958	rmse	mean
40	0	B	7	0	li	260	NaN	NaN	NaN	NaN	rmse	mean
3	0	B	54	1	lo	30	110.553065	51.443275	19.154490	60.383610	rmse	mean
30	0	B	29	0	co	30	112.035140	44.427934	18.598329	58.353801	rmse	mean
76	0	B	1	1	cili	65	93.945660	51.090754	21.836356	55.624257	rmse	mean
87	0	B	50	0	lo	260	163.152289	46.616014	13.327631	74.365311	rmse	mean
10	0	B	44	0	cili	65	NaN	NaN	NaN	NaN	rmse	mean
9	0	B	47	0	n	0	86.212252	55.639330	36.182421	59.344668	rmse	mean
94	0	B	38	1	colo	5	139.580751	41.852200	24.055439	68.496130	rmse	mean
43	0	B	55	1	colo	260	112.237679	43.782141	63.595936	73.205252	rmse	mean
73	0	B	40	0	lo	0	159.422501	46.642590	18.090453	74.718515	rmse	mean
62	0	B	18	1	n	5	166.241648	54.618358	30.753820	83.871275	rmse	mean
63	0	B	12	0	co	260	108.180519	44.685347	23.286315	58.717394	rmse	mean
60	0	B	21	1	li	260	82.029398	50.186756	31.553748	54.589967	rmse	mean
64	0	B	43	1	co	0	100.291180	46.496187	16.839982	54.542450	rmse	mean

[12]:
```
mvf.plot_test_set(
    series='AMZN',
    models='vecm',
    include_train=130,
    figsize=(16,8)
)
plt.show()
```



24.7 Re-weight Evaluation Metrics and Rerun VECM

[13]:
```
weights = results['TestSetRMSE'] / results['TestSetRMSE'].sum()
weights
```

[13]:
```
0    0.245701
1    0.114627
2    0.548723
3    0.090950
Name: TestSetRMSE, dtype: float64
```

[14]:
```
mvf.set_optimize_on(
    lambda x: (
        x[0]*weights[0] +
        x[1]*weights[1] +
        x[2]*weights[2] +
        x[3]*weights[3]
    )
)
mvf.ingest_grid(vecm_grid)
mvf.limit_grid_size(100,random_seed=20)
mvf.cross_validate(k=3,verbose=True)
mvf.auto_forecast(call_me='vecm_weighted')

results = mvf.export('model_summaries')
results[[
    'ModelNickname',
    'Series',
    'TestSetRMSE',
```

(continues on next page)

(continued from previous page)

```
    'TestSetMAE',
]]
```

```
Num hyperparams to try for the vecm model: 100.
Fold 0: Train size: 2145 (2012-05-18 00:00:00 - 2020-08-06 00:00:00). Test Size: 715␣
↪(2020-08-07 00:00:00 - 2023-05-04 00:00:00).
Fold 1: Train size: 1430 (2012-05-18 00:00:00 - 2017-11-09 00:00:00). Test Size: 715␣
↪(2017-11-10 00:00:00 - 2020-08-06 00:00:00).
Fold 2: Train size: 715 (2012-05-18 00:00:00 - 2015-02-12 00:00:00). Test Size: 715␣
↪(2015-02-13 00:00:00 - 2017-11-09 00:00:00).
Chosen paramaters: {'lags': 0, 'freq': 'B', 'k_ar_diff': 62, 'coint_rank': 1,
↪'deterministic': 'li', 'seasons': 0}.
```

[14]:

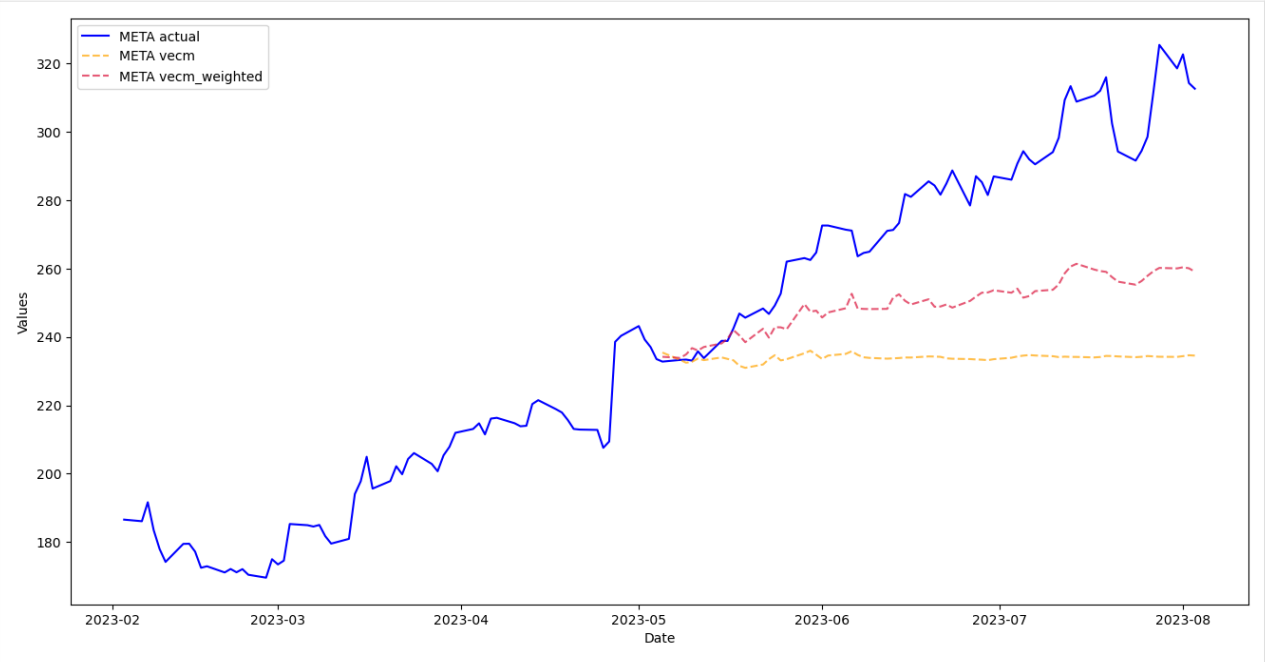
	ModelNickname	Series	TestSetRMSE	TestSetMAE
0	vecm	META	50.913814	44.133838
1	vecm_weighted	META	33.772797	28.520031
2	vecm	AMZN	23.752750	22.348669
3	vecm_weighted	AMZN	15.835426	14.617043
4	vecm	NFLX	113.705564	105.310047
5	vecm_weighted	NFLX	59.887563	53.210215
6	vecm	GOOG	18.846431	18.041912
7	vecm_weighted	GOOG	16.757831	16.121594

[15]:
```
results.loc[results['ModelNickname'] == 'vecm_weighted','TestSetRMSE'].mean()
```

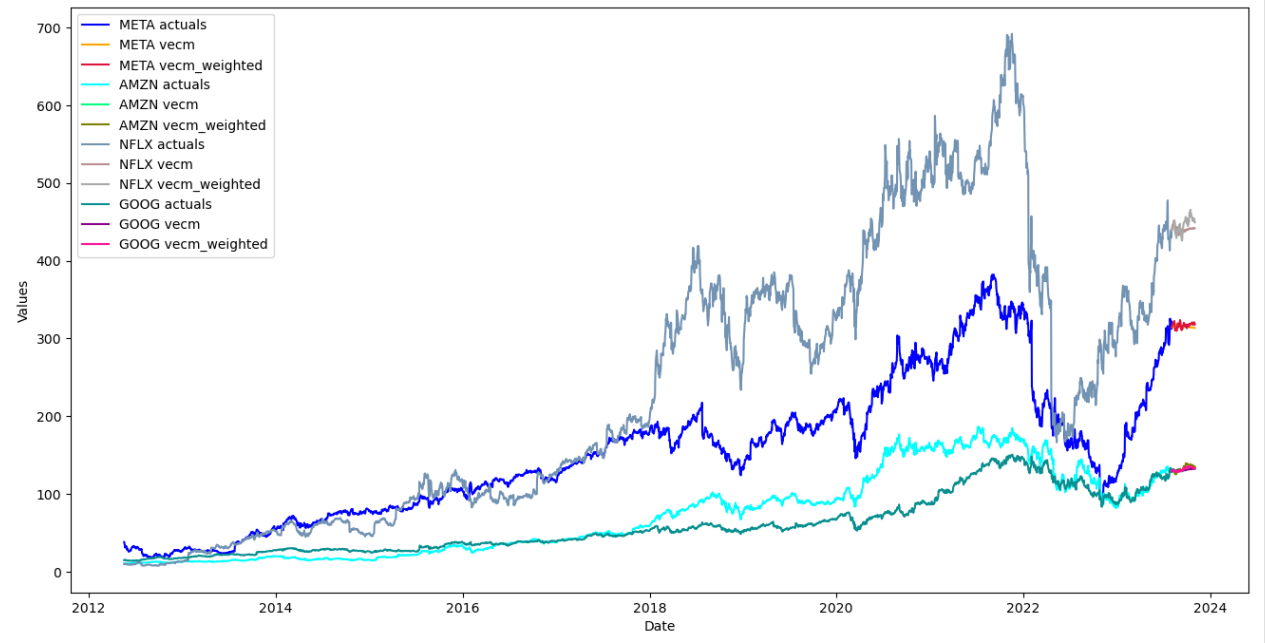
[15]:
```
31.56340410699967
```

An improvement by weighting the optimizer!

[16]:
```
mvf.plot_test_set(
    series='META',
    models='all',
    include_train=130,
    figsize=(16,8)
)
plt.show()
```



[17]:
```
mvf.plot(
    series='all',
    models='all',
    figsize=(16,8)
)
plt.show()
```



24.8 Try Other MV Models

[31]:
```
GridGenerator.get_mv_grids()
# open MVGrids.py and manually change all lags arguments to range(1,66)
```

[32]:
```
transformers = []
reverters = []
for stock, f in zip(FANG,fs):
    transformer = Transformer(
        transformers = [('DiffTransform',)]
    )
    reverter = Reverter(
        reverters = [('DiffRevert',)],
        base_transformer = transformer,
    )
    transformers.append(transformer)
    reverters.append(reverter)
```

[33]:
```
def Xvar_select(f):
    f.set_validation_length(65)
    f.auto_Xvar_select(
        estimator='gbt',
        max_depth=2,
        max_ar=0, # in mv modeling, lags are a hyperparameter, not a regressor in the␣
↪MVForecaster object
    )

def mvforecaster(mvf):
    models = (
        'mlr',
        'elasticnet',
        'gbt',
        'xgboost',
        'lightgbm',
        'knn',
    )
    mvf.set_test_length(65)
    mvf.tune_test_forecast(
        models,
        limit_grid_size=10,
        cross_validate=True,
        k=3,
    )
```

[34]:
```
pipeline = MVPipeline(
    steps = [
        ('Transform',transformers),
        ('Xvar Select',[Xvar_select]*4),
        ('Forecast',mvforecaster),
        ('Revert',reverters),
    ],
    names = FANG,
```

(continues on next page)

(continued from previous page)

```
)
```

[35]:
```
fs = pipeline.fit_predict(*fs)
```

```
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
Finished loading model, total used 150 iterations
```

[36]:
```
results = export_model_summaries(dict(zip(FANG,fs)))
```

24.9 View Results

[37]:
```
model_rmses = results.groupby('ModelNickname')['TestSetRMSE'].mean().sort_values().reset_
↪index()
model_rmses
```

[37]:
```
  ModelNickname  TestSetRMSE
0       xgboost    27.918698
1           knn    41.832711
2           mlr    42.519262
3           gbt    43.895070
4    elasticnet    44.320649
5      lightgbm    44.554279
```

The above table is the mean mape performance from each model over all series.

[38]:
```
series_rmses = results.groupby('Series')['TestSetRMSE'].min().reset_index()
series_rmses['Model'] = [
    results.loc[
        results['TestSetRMSE'] == i,
        'ModelNickname'
    ].values[0] for i in series_rmses['TestSetRMSE']
]
series_rmses
```

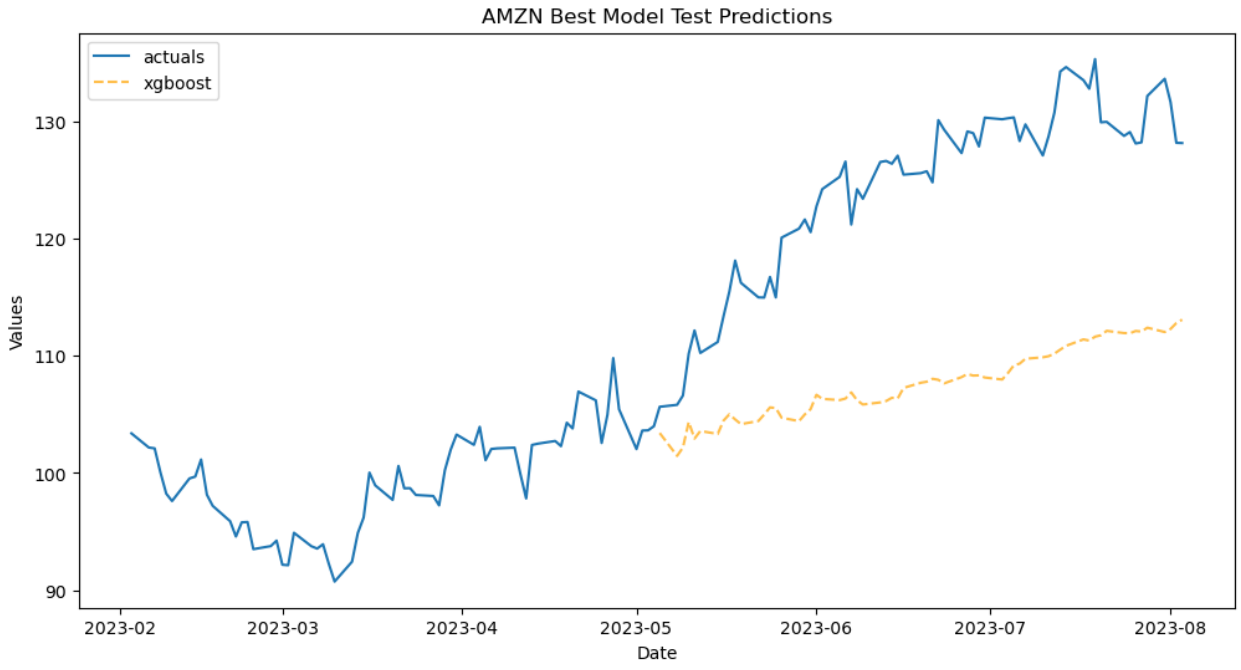
[38]:
```
  Series  TestSetRMSE    Model
0   AMZN    17.433315  xgboost
1   GOOG    12.892690  xgboost
2   META    10.509525  xgboost
3   NFLX    70.839262  xgboost
```

[39]:
```
series_rmses['TestSetRMSE'].mean()
```

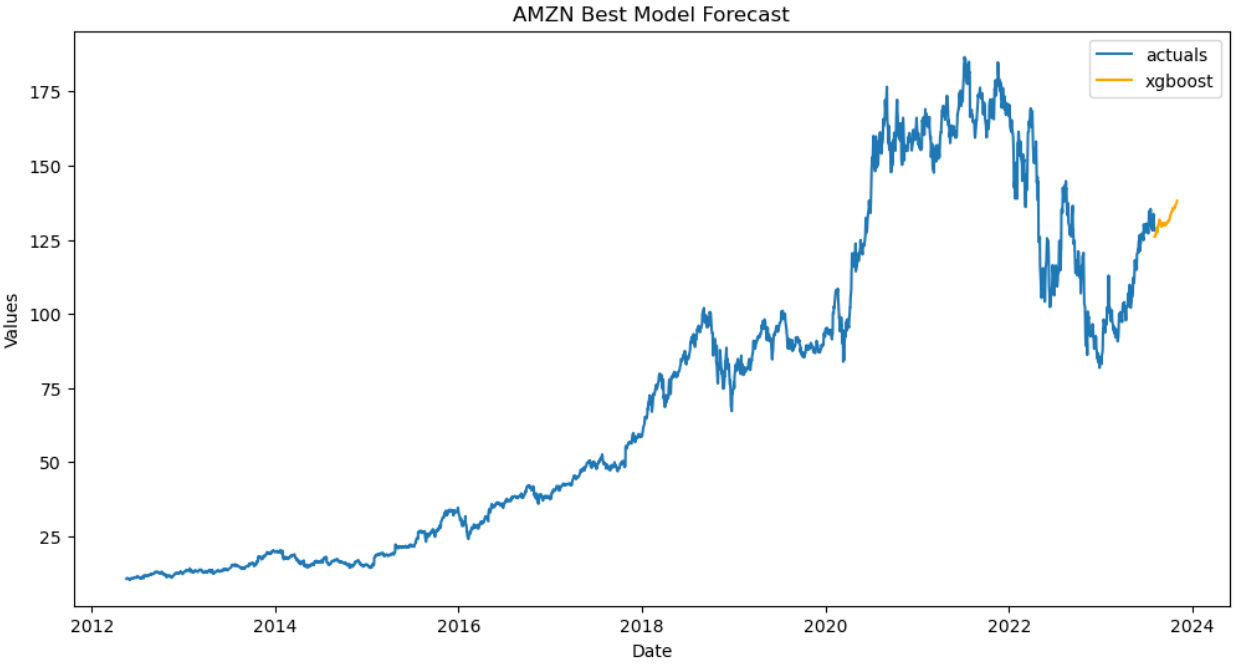
[39]:
```
27.918697955026953
```

The above table shows the best model for each series and its derived RMSE. The average RMSE of all these models applied to the individual series is 27.9, but being so dependent on the test set to choose the model probably leads to overfitting.

[41]:
```
fs[1].plot_test_set(
    models='xgboost',
    include_train=130,
)
plt.title('AMZN Best Model Test Predictions')
plt.show()
```



[42]:
```
fs[1].plot(
    models='xgboost',
)
plt.title('AMZN Best Model Forecast')
plt.show()
```


AMZN Best Model Forecast
actuals
xgboost
Values
175
150
125
100
75
50
25
2012
2014
2016
2018
2020
2022
2024
Date

[ ]: