Time Series Analisys on Geoespatial Data with Python

Author: João Otavio Nascimento Firigato

email: joaootavionf007@gmail.com

LinkedIn: https://www.linkedin.com/in/jo%C3%A3o-otavio-firigato-4876b3aa/

First instructions:

✅ Access the link to join our private WhatsApp community for students: https://chat.whatsapp.com/EPn27ZgR07lF3e1vnj8FiI

❗ It is important to access the Whatsapp Group to get the Colab Notebooks, as the PDF files are protected from text copying.

Chapter 10 - Time Series Forecast - Part 3

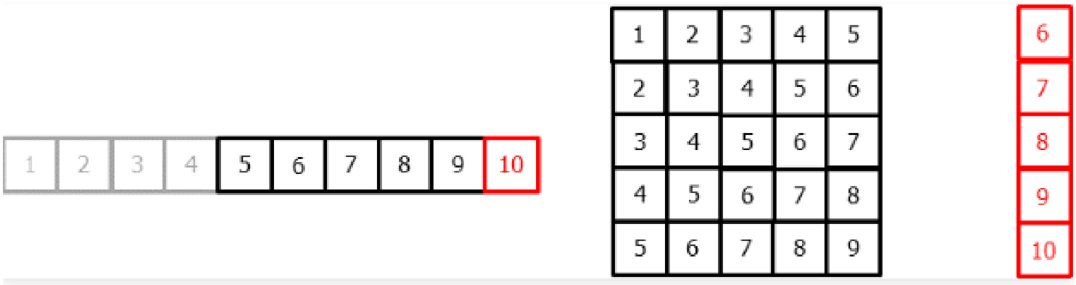
Machine Learning Models

While most of the machine learning algorithms available in scikit-learn (and several other compatible libraries, such as LightGBM) are commonly used for regression and tabular classification, there is no inherent reason why they cannot be effectively applied to time series forecasting.

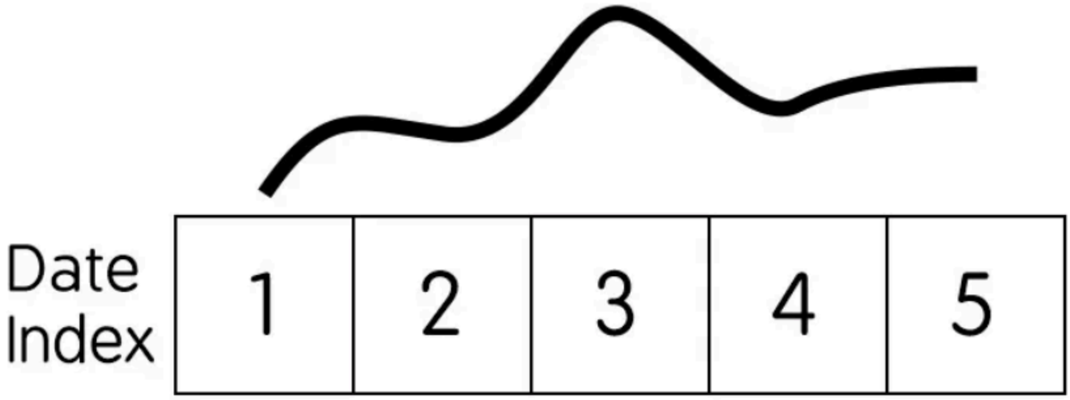
From a theoretical perspective, these approaches may not necessarily satisfy the conditions required for a regression model. Time series data typically exhibit some degree of autocorrelation, meaning that data observed at time t is related to previous data (t-1) and possibly even more. A typical regression model typically assumes independent observations, so they may not be well suited to capture autocorrelation. However, it is still possible to use lagged series as predictors and allow the model to learn from existing autocorrelation or other patterns.

Scikit-learn offers a wide range of regression models, ranging from basic linear regression to highly advanced boosted trees. Along with this, hyperparameter tuning and implementing cross-validation are common practices to ensure better model generality and accuracy.

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com



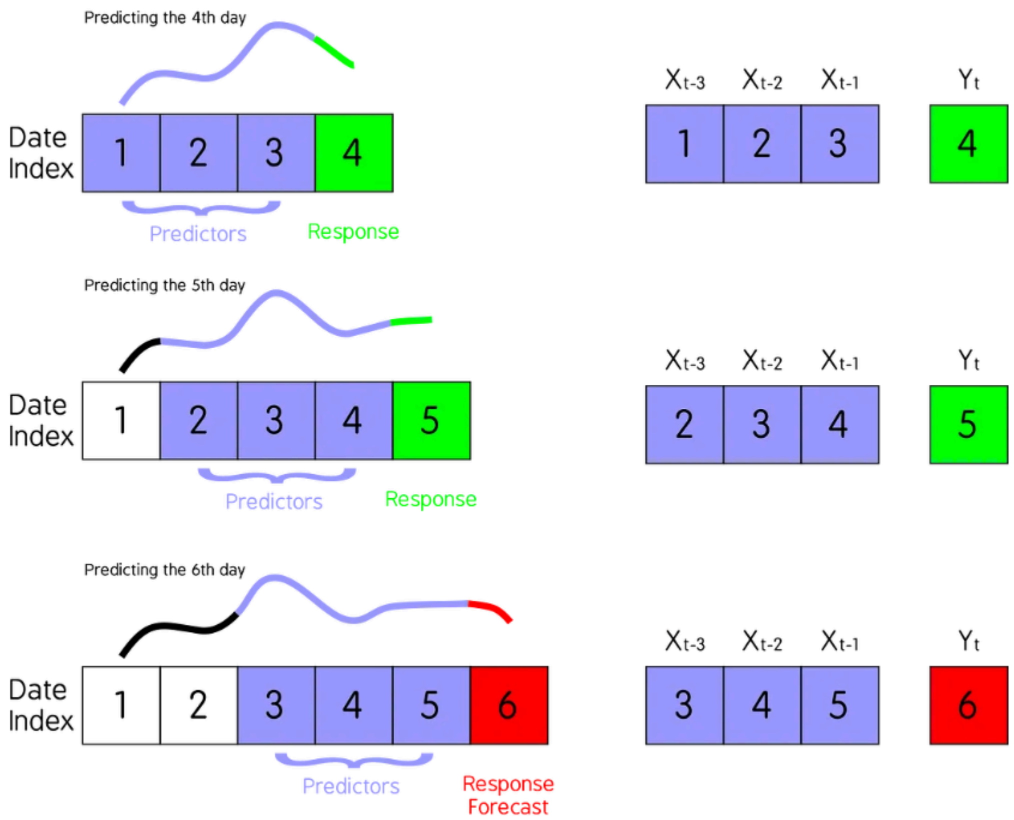
Since we are treating time series forecasting as a regression problem, we would need to have a predictor. Here, we assume that we only have a univariate series with 5 days



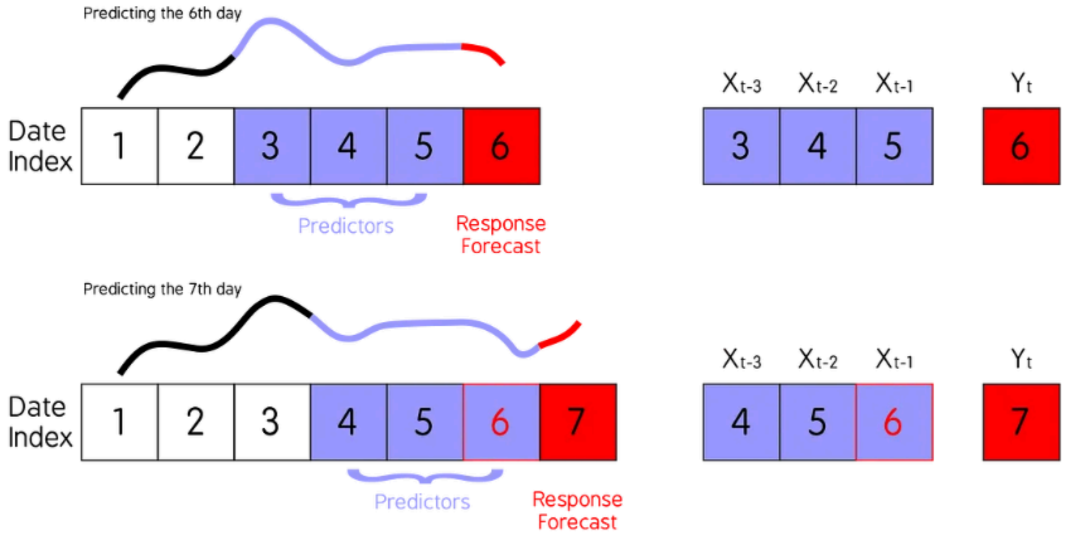
At the beginning, we only have data up to day 5. Let's say we use the previous 3 days as the predictor. This means we are using up to 3 lagged data. To predict day 4, we use data from days 1 to 3. To predict day 5, we use data from days 2 to 4. Then, to predict day 6, we use data from days 3 to 5.

In tabular terms, we can write the predictor and the response as follows. We label yesterday's data as Xt-1, two days ago as Xt-2, and three days ago as Xt-3.

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com



We have established a way to predict one step into the future. How about predicting 2 days ahead? Or 3? This is where we introduce the concept of recursive prediction. In short, to predict day 7, we make a prediction for day 6 and then use that predicted value as our Xt-1. Essentially, we are treating the previous prediction as a new predictor for the future. We do this as often as necessary and it is quite efficient.

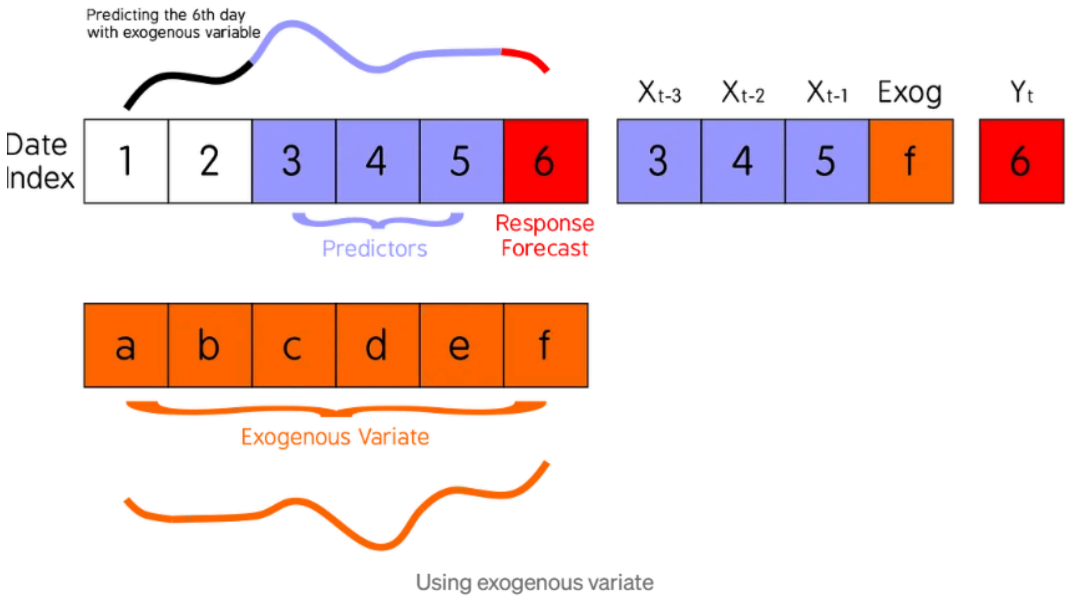


Recursive Forecasting

However, there are several problems with this approach. If we want to predict far into the future, the model may give a relatively flat prediction. It may also introduce new bias, as the 6th predicted data may be far from the actual value.

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

Again, it is possible to include exogenous variables in the model. Keep in mind that to predict the 6th data, we must also provide the exogenous variable for that same index.



Let's get some time series data. Let's use Sentinel 5P data in this example:

In [ ]:
```
import ee
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import geemap
import datetime
```

In [ ]:
```
ee.Authenticate()
ee.Initialize(project='my-project-1527255156007')
```

We select an area of analysis:

In [ ]:
```
italy = ee.Geometry.Polygon([[[8.290016551188252, 45.79137351065613],
          [8.290016551188252, 45.0588767633071],
          [10.168678660563252, 45.0588767633071],
          [10.168678660563252, 45.79137351065613]]])
```

The start and end date:

In [ ]:
```
startDate = '2023-01-01'
endDate = '2023-12-31'
```

Then we will get our collection of NO2 images:

In [ ]:
```
s5p_no = ee.ImageCollection('COPERNICUS/S5P/NRTI/L3_NO2').filterDate(startDate,
```

Let's now extract the time series and convert it into a dataframe:

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

```python
In [ ]: def extract_time_series(image_collection, column):
          def reduce_region_func(image):
            reduced_image = image.reduceRegion(
                reducer=ee.Reducer.sum(),
                geometry=italy,
                scale=1113,
                maxPixels=1e13
            )
            return image.set(reduced_image)

          reduced_collection = image_collection.map(reduce_region_func)

          time_series_time = reduced_collection.aggregate_array('system:time_start').get
          time_series_values = reduced_collection.aggregate_array(column).getInfo()

          # Create a DataFrame
          time_series_df = pd.DataFrame({'time_start': time_series_time, 'value': time_s
          time_series_df['time_start'] = pd.to_datetime(time_series_df['time_start'], un
          time_series_df.set_index('time_start', inplace=True)
          return time_series_df

        no2_time_series = extract_time_series(s5p_no,'NO2_column_number_density' )
```

```python
In [ ]: no2_time_series.index = no2_time_series.index.strftime('%Y-%m-%d')
```

```python
In [ ]: no2_time_series_grouped = no2_time_series.groupby(no2_time_series.index).sum()
```

```python
In [ ]: df = pd.DataFrame(columns=['NO2'])
        df['NO2'] = no2_time_series_grouped['value']
```

```python
In [ ]: df.dropna(axis=0, inplace=True)
```

Thus, we have our time series of the sum of NO2 in our region of interest:

```python
In [ ]: df
```

Out[ ]:

	NO2
time_start	
2023-01-01	0.000000
2023-01-02	0.000000
2023-01-03	0.000000
2023-01-04	0.611969
2023-01-05	2.356767
...	...
2023-12-26	0.148370
2023-12-27	0.389392
2023-12-28	0.013645
2023-12-29	0.000000
2023-12-30	2.165453

357 rows × 1 columns

We will transform our time series into a dataframe adjusted for the classification task by applying shift:

In [ ]:
```
df['NO2_shift1'] = df['NO2'].shift(1)
df['NO2_shift2'] = df['NO2'].shift(2)
df['NO2_shift3'] = df['NO2'].shift(3)
df['NO2_shift4'] = df['NO2'].shift(4)
df['NO2_shift5'] = df['NO2'].shift(5)
df['NO2_shift6'] = df['NO2'].shift(6)
df['NO2_shift7'] = df['NO2'].shift(7)
```

We will remove the days where we do not have all the full shifts:

In [ ]:
```
df.dropna(axis=0, inplace=True)
```

So our DataFrame is ready:

In [ ]:
```
df
```

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

Out[ ]:

time_start	NO2	NO2_shift1	NO2_shift2	NO2_shift3	NO2_shift4	NO2_shift5	NO2
2023-01-08	0.000000	0.133376	0.244905	2.356767	0.611969	0.000000	0.
2023-01-09	1.812042	0.000000	0.133376	0.244905	2.356767	0.611969	0.
2023-01-10	1.831945	1.812042	0.000000	0.133376	0.244905	2.356767	0.
2023-01-11	0.048411	1.831945	1.812042	0.000000	0.133376	0.244905	2.
2023-01-12	3.785621	0.048411	1.831945	1.812042	0.000000	0.133376	0.
...	...	...	...	...	...	...	...
2023-12-26	0.148370	3.243424	2.299797	1.327712	0.956265	1.357426	3.
2023-12-27	0.389392	0.148370	3.243424	2.299797	1.327712	0.956265	1.
2023-12-28	0.013645	0.389392	0.148370	3.243424	2.299797	1.327712	0.
2023-12-29	0.000000	0.013645	0.389392	0.148370	3.243424	2.299797	1.
2023-12-30	2.165453	0.000000	0.013645	0.389392	0.148370	3.243424	2.

350 rows × 8 columns

From the Shift columns we will predict the next value which is our target column NO2:

In [ ]:
```
X = df[['NO2', 'NO2_shift1', 'NO2_shift2', 'NO2_shift3', 'NO2_shift4', 'NO2_shif

y = df['NO2_shift7']
```

We split the data into training and testing:

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random
```

And here we will use the Decision Tree Regressor:

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor
classifier = DecisionTreeRegressor(random_state=42)
classifier.fit(X_train, y_train)

from sklearn.metrics import r2_score
print(r2_score(list(y_test), list(classifier.predict(X_test))))
```

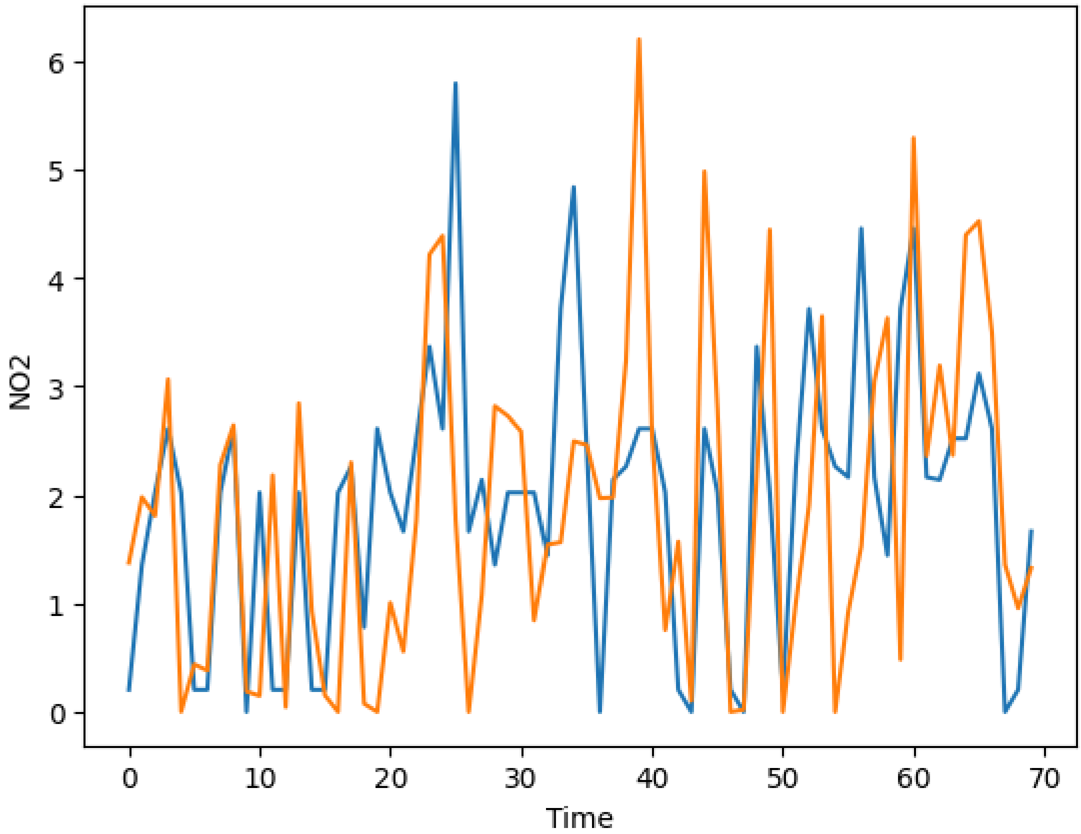
```
-0.9301139049419489
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV

my_dt = GridSearchCV(DecisionTreeRegressor(random_state=44),
                  {'min_samples_split': list(range(20,50, 2)),
                  'max_features': [0.6, 0.7, 0.8, 0.9, 1.]},
                 scoring = 'r2', n_jobs = -1)

my_dt.fit(X_train, y_train)
print(r2_score(list(y_test), list(my_dt.predict(X_test))))
```

```
0.040874925952024244
```

We can check the predicted data compared to the actual data:

In [ ]:
```
fcst = my_dt.predict(X_test)

plt.plot(list(fcst))
plt.plot(list(y_test))
plt.ylabel('NO2')
plt.xlabel('Time')
plt.show()
```



Recursive multi-step forecasting

Let's now use recursive multistep forecasting. To do this, we will select a point and obtain LST time series data:

In [ ]:
```
geometry_test = ee.Geometry.Point([-55.05744298464864, -13.547299531642683])
```

```
In [ ]: startDate = '2016-01-01'
        endDate = '2024-12-31'
```

We generate our image collection according to geographic point and start and end dates:

```
In [ ]: modisLst = ee.ImageCollection('MODIS/061/MOD11A1').filterDate(startDate, endDate
```

Let's get the monthly average between 2016 and 2024:

```
In [ ]: months = ee.List.sequence(1,12)
        years = ee.List.sequence(2016, 2024)
```

```
In [ ]: def monthly(collection):
          img_coll = ee.ImageCollection([])
          for y in years.getInfo():
            for m in months.getInfo():
              filtered = collection.filter(ee.Filter.calendarRange(y, y, 'year')).filter
              filtered = filtered.median()
              img_coll = img_coll.merge(filtered.set('year', y).set('month', m).set('sys
          return img_coll
```

```
In [ ]: modisLst = monthly(modisLst)
```

```
In [ ]: def meanLST(image):
            image = ee.Image(image)
            meanDict = image.reduceRegion(reducer = ee.Reducer.mean().setOutputs(['LST_D
                geometry = geometry_test,
                scale = image.projection().nominalScale().getInfo(),
                                            maxPixels = 1e13,
                                            bestEffort = True);
            return meanDict.get('LST_Day_1km').getInfo()
```

```
In [ ]: listOfImages_modis = modisLst.select('LST_Day_1km').toList(modisLst.size())

        lst_coll = []

        for i in range(listOfImages_modis.length().getInfo()):
            image = ee.Image(listOfImages_modis.get(i-1))
            temp_lst = meanLST(image)
            lst_coll.append(temp_lst)
```

After extracting we convert it into a DataFrame:

```
In [ ]: dates = np.array(modisLst.aggregate_array("system:time_start").getInfo())
        day = [datetime.datetime.fromtimestamp(i/1000).strftime('%Y-%m-%d') for i in (da
```

```
In [ ]: lst_df = pd.DataFrame(lst_coll, index = day, columns = ['LST'])
        lst_df.index = pd.to_datetime(lst_df.index)
        lst_df.sort_index(ascending = True, inplace = True)
```

```
In [ ]: lst_df
```

Out[ ]:

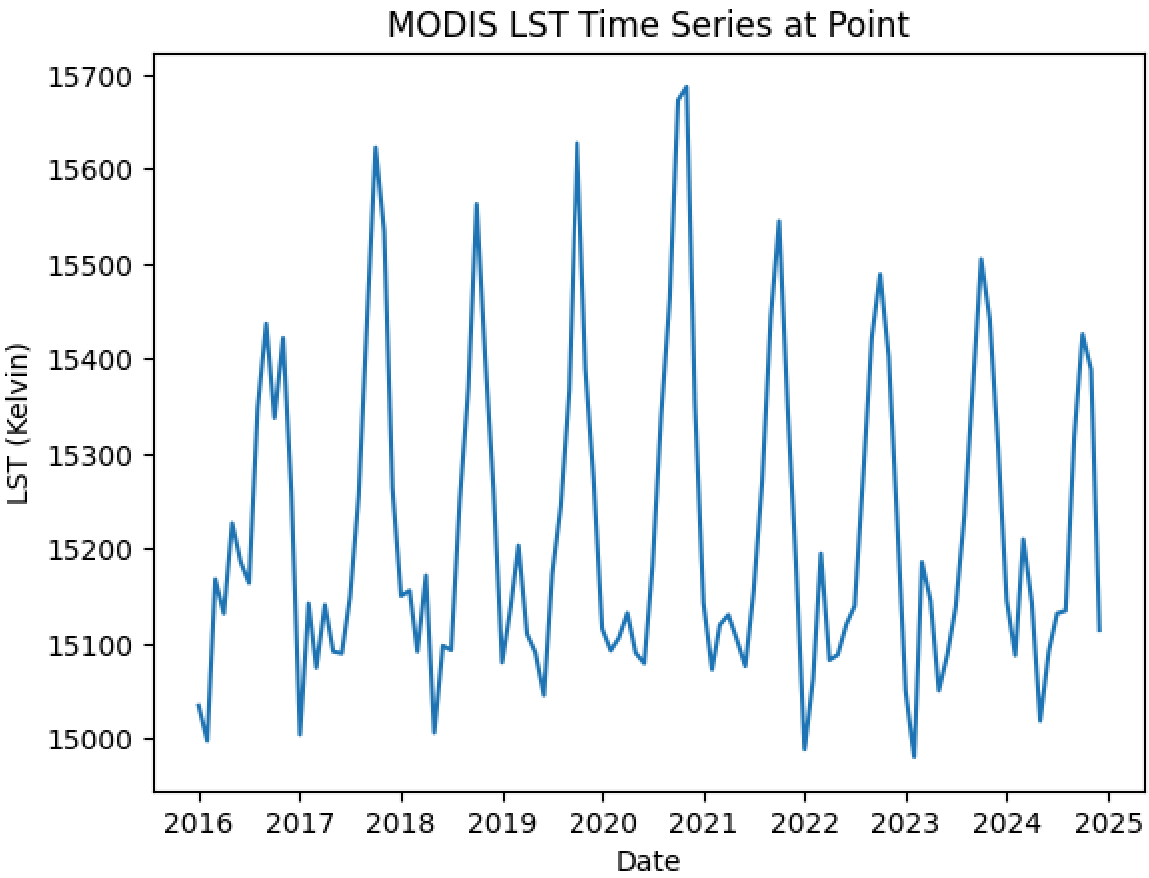
	LST
2016-01-01	15035.0
2016-02-01	14998.0
2016-03-01	15168.0
2016-04-01	15132.0
2016-05-01	15227.0
...	...
2024-08-01	15135.0
2024-09-01	15318.0
2024-10-01	15426.0
2024-11-01	15388.5
2024-12-01	15114.5

108 rows × 1 columns

We present the time series:

In [ ]:

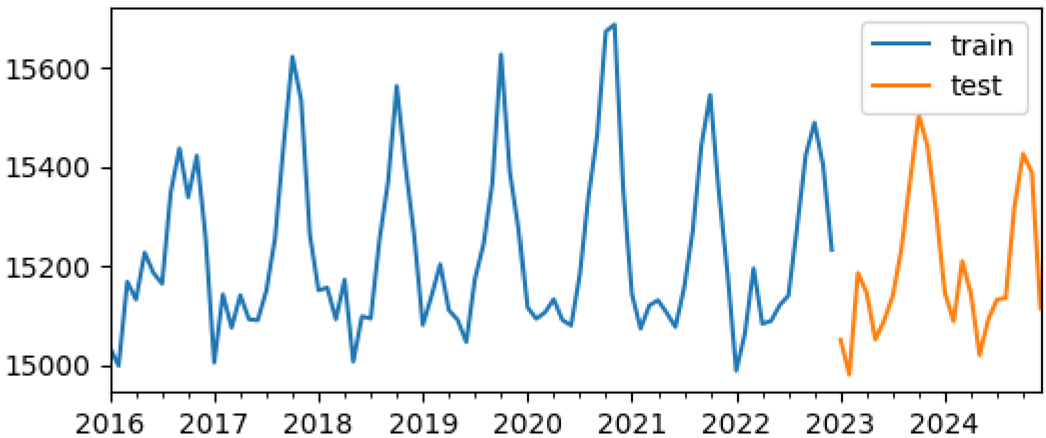
```
plt.plot(lst_df['LST'])
plt.xlabel('Date')
plt.ylabel('LST (Kelvin)')
plt.title('MODIS LST Time Series at Point')
plt.show()
```



We divide it into training and testing:

In [ ]:
```
steps = 24
data_train = lst_df[:-steps]
data_test  = lst_df[-steps:]
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(6, 2.5))
data_train['LST'].plot(ax=ax, label='train')
data_test['LST'].plot(ax=ax, label='test')
ax.legend();
```



Let's use the skforest implementation:

In [ ]:
```
!pip install skforecast
```

```
Collecting skforecast
  Downloading skforecast-0.16.0-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-packages (from skforecast) (2.0.2)
Requirement already satisfied: pandas>=1.5 in /usr/local/lib/python3.11/dist-packages (from skforecast) (2.2.2)
Requirement already satisfied: tqdm>=4.57 in /usr/local/lib/python3.11/dist-packages (from skforecast) (4.67.1)
Requirement already satisfied: scikit-learn>=1.2 in /usr/local/lib/python3.11/dist-packages (from skforecast) (1.6.1)
Collecting optuna>=2.10 (from skforecast)
  Downloading optuna-4.3.0-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: joblib>=1.1 in /usr/local/lib/python3.11/dist-packages (from skforecast) (1.5.0)
Requirement already satisfied: numba>=0.59 in /usr/local/lib/python3.11/dist-packages (from skforecast) (0.60.0)
Requirement already satisfied: rich>=13.9 in /usr/local/lib/python3.11/dist-packages (from skforecast) (13.9.4)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba>=0.59->skforecast) (0.43.0)
Collecting alembic>=1.5.0 (from optuna>=2.10->skforecast)
  Downloading alembic-1.16.1-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna>=2.10->skforecast)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from optuna>=2.10->skforecast) (24.2)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.11/dist-packages (from optuna>=2.10->skforecast) (2.0.40)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages (from optuna>=2.10->skforecast) (6.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.5->skforecast) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.5->skforecast) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.5->skforecast) (2025.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=13.9->skforecast) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=13.9->skforecast) (2.19.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.2->skforecast) (1.15.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.2->skforecast) (3.6.0)
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages (from alembic>=1.5.0->optuna>=2.10->skforecast) (1.1.3)
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.11/dist-packages (from alembic>=1.5.0->optuna>=2.10->skforecast) (4.13.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=13.9->skforecast) (0.1.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.5->skforecast) (1.17.0)
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-packages (from sqlalchemy>=1.4.2->optuna>=2.10->skforecast) (3.2.2)
Downloading skforecast-0.16.0-py3-none-any.whl (814 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 815.0/815.0 kB 13.4 MB/s eta 0:00:00
Downloading optuna-4.3.0-py3-none-any.whl (386 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 386.6/386.6 kB 26.5 MB/s eta 0:00:00
Downloading alembic-1.16.1-py3-none-any.whl (242 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 242.5/242.5 kB 16.7 MB/s eta 0:00:00
```

```
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, alembic, optuna, skforecast
Successfully installed alembic-1.16.1 colorlog-6.9.0 optuna-4.3.0 skforecast-0.1
6.0
```

In [ ]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler
import skforecast
from skforecast.recursive import ForecasterRecursive
from skforecast.direct import ForecasterDirect
from skforecast.plot import plot_prediction_intervals
from sklearn.linear_model import Ridge
```

In [ ]:
```python
forecaster = ForecasterRecursive(
                 regressor = RandomForestRegressor(random_state=123),
                 lags      = 12
             )
forecaster.fit(y=data_train['LST'])
forecaster
```

```
IndexWarning
Series has a pandas DatetimeIndex without a frequency. The index will be
a RangeIndex starting from 0 with a step of 1. To avoid this warning, se
frequency of the DatetimeIndex using `y = y.asfreq('desired_frequency',
fill_value=np.nan)`.

Category : IndexWarning
Location : /usr/local/lib/python3.11/dist-packages/skforecast/utils/util
Suppress : warnings.simplefilter('ignore', category=IndexWarning)
```

Out[ ]:

ForecasterRecursive

▼ **General Information**

- **Regressor:** RandomForestRegressor
- **Lags:** [1 2 3 4 5 6 7 8 9 10 11 12]
- **Window features:** None
- **Window size:** 12
- **Exogenous included:** False
- **Weight function included:** False
- **Differentiation order:** None
- **Creation date:** 2025-03-28 15:14:32
- **Last fit date:** 2025-03-28 15:14:32
- **Skforecast version:** 0.15.1
- **Python version:** 3.11.11
- **Forecaster id:** None

▶ **Exogenous Variables**

▶ **Data Transformations**

▶ **Training Information**

▶ **Regressor Parameters**

▶ **Fit Kwargs**

ⓘ **API Reference** 🗎 **User Guide**

We can apply the prediction 24 steps into the future:

In [ ]:
```
steps = 24
predictions = forecaster.predict(steps=steps)
```

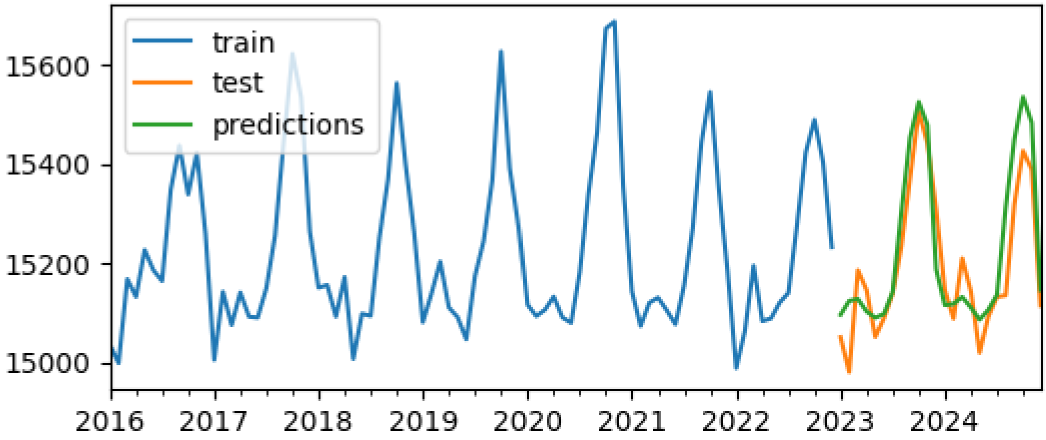
```
IndexWarning
`last_window` has a pandas DatetimeIndex without a frequency. The index
replaced by a RangeIndex starting from 0 with a step of 1. To avoid this
set the frequency of the DatetimeIndex using `last_window =
last_window.asfreq('desired_frequency', fill_value=np.nan)`.

Category : IndexWarning
Location : /usr/local/lib/python3.11/dist-packages/skforecast/utils/util
Suppress : warnings.simplefilter('ignore', category=IndexWarning)
```

In [ ]:
```
predictions.index = data_test.index
```

Then we compare the predicted and test values:

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

In [ ]:
```
fig, ax = plt.subplots(figsize=(6, 2.5))
data_train['LST'].plot(ax=ax, label='train')
data_test['LST'].plot(ax=ax, label='test')
predictions.plot(ax=ax, label='predictions')
ax.legend();
```



In [ ]:
```
error_mse = mean_squared_error(
                y_true = data_test['LST'],
                y_pred = predictions
            )
print(f"Test error (MSE): {error_mse}")
```

```
Test error (MSE): 5892.410045833322
```

In [ ]:
```
importance = forecaster.get_feature_importances()
importance.head(10)
```

Out[ ]:

	feature	importance
11	lag_12	0.802108
3	lag_4	0.029228
0	lag_1	0.027335
8	lag_9	0.026142
4	lag_5	0.024027
10	lag_11	0.019620
5	lag_6	0.018887
7	lag_8	0.017335
9	lag_10	0.010976
6	lag_7	0.009295

Probabilistic forecasting

Probabilistic forecasting, as opposed to point forecasting, is a family of techniques that allows the prediction of the expected distribution of the outcome rather than a single future value. This type of forecasting provides much richer information because it allows

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

the creation of prediction intervals, the range of likely values where the true value might fall. More formally, a prediction interval defines the range within which the true value of the response variable is expected to be found with a given probability.

In [ ]:
```
forecaster = ForecasterRecursive(
                 regressor = Ridge(alpha=0.1, random_state=765),
                 lags      = 12
             )
forecaster.fit(y=data_train['LST'], store_in_sample_residuals=True)
```

```
IndexWarning
Series has a pandas DatetimeIndex without a frequency. The index will be
a RangeIndex starting from 0 with a step of 1. To avoid this warning, se
frequency of the DatetimeIndex using `y = y.asfreq('desired_frequency',
fill_value=np.nan)`.

Category : IndexWarning
Location : /usr/local/lib/python3.11/dist-packages/skforecast/utils/util
Suppress : warnings.simplefilter('ignore', category=IndexWarning)
```

In [ ]:
```
predictions = forecaster.predict_interval(
                  steps    = steps,
                  interval = [5, 95],
                  method   = 'bootstrapping',
                  n_boot   = 500
              )
```

```
IndexWarning
`last_window` has a pandas DatetimeIndex without a frequency. The index
replaced by a RangeIndex starting from 0 with a step of 1. To avoid this
set the frequency of the DatetimeIndex using `last_window =
last_window.asfreq('desired_frequency', fill_value=np.nan)`.

Category : IndexWarning
Location : /usr/local/lib/python3.11/dist-packages/skforecast/utils/util
Suppress : warnings.simplefilter('ignore', category=IndexWarning)
```


```
IndexWarning
`last_window` has a pandas DatetimeIndex without a frequency. The index
replaced by a RangeIndex starting from 0 with a step of 1. To avoid this
set the frequency of the DatetimeIndex using `last_window =
last_window.asfreq('desired_frequency', fill_value=np.nan)`.

Category : IndexWarning
Location : /usr/local/lib/python3.11/dist-packages/skforecast/utils/util
Suppress : warnings.simplefilter('ignore', category=IndexWarning)
```


In [ ]:
```
predictions.index = data_test.index
```

We obtain the predictions and the lower and upper bounds:

In [ ]:
```
predictions
```

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

Out[ ]:

	pred	lower_bound	upper_bound
2023-01-01	15109.370584	14993.803688	15182.299833
2023-02-01	15124.248798	15065.542411	15215.041477
2023-03-01	15133.052543	15014.965621	15195.294893
2023-04-01	15128.439272	15000.971818	15196.639739
2023-05-01	15113.279702	15005.450686	15187.140760
2023-06-01	15098.032501	14997.979767	15178.710010
2023-07-01	15152.621431	15075.536143	15220.744993
2023-08-01	15282.585015	15186.972088	15365.500965
2023-09-01	15422.720289	15338.270998	15510.739895
2023-10-01	15472.790271	15367.630714	15693.779511
2023-11-01	15389.852859	15298.388845	15624.129453
2023-12-01	15246.983814	15106.316030	15349.391264
2024-01-01	15154.417397	15015.188259	15239.597005
2024-02-01	15130.802768	15003.052666	15203.689374
2024-03-01	15140.042226	15016.065847	15207.470452
2024-04-01	15137.960633	15014.165835	15210.952289
2024-05-01	15109.307079	15001.225510	15191.242650
2024-06-01	15096.971809	15003.205270	15192.750144
2024-07-01	15154.010989	15069.247311	15225.941791
2024-08-01	15282.447015	15132.192463	15365.489238
2024-09-01	15412.078872	15283.236457	15579.609366
2024-10-01	15452.423666	15325.937450	15731.706050
2024-11-01	15381.352714	15256.086419	15624.354466
2024-12-01	15262.682807	15108.464266	15388.784724

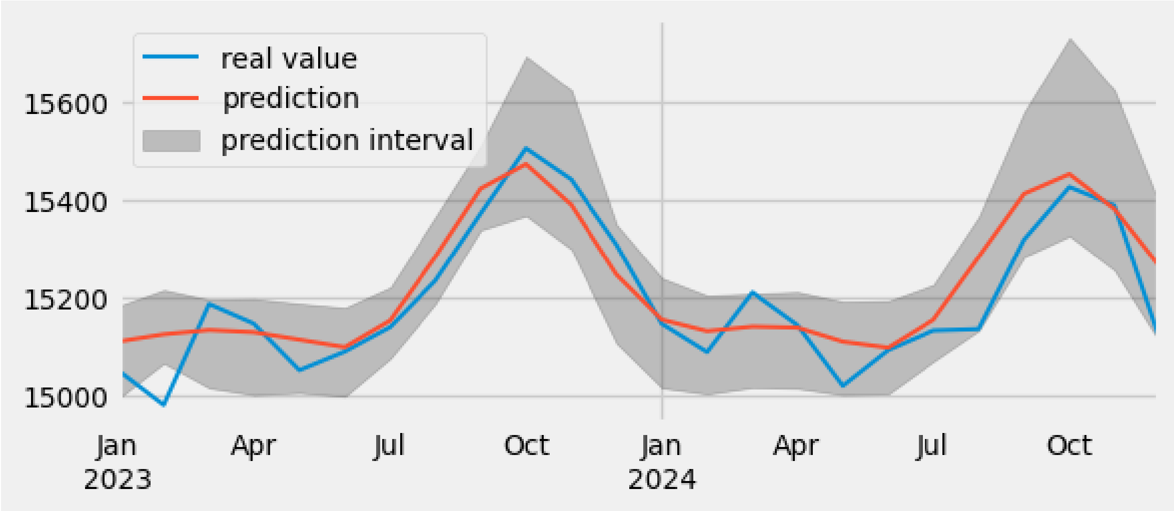
In [ ]:

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.rcParams['lines.linewidth'] = 1.5
plt.rcParams['font.size'] = 10
```

In [ ]:

```
fig, ax = plt.subplots(figsize=(6, 2.5))
plot_prediction_intervals(
    predictions     = predictions,
    y_true          = data_test,
    target_variable = "LST",
    ax              = ax
)
```

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com



Forecasting with exogenous variables

We can add exogenous variables to the model. Let's use the month index as an exogenous variable:

In [ ]:
```
lst_df['month'] = lst_df.index.month
```

In [ ]:
```
lst_df
```

Out[ ]:

	LST	month
2016-01-01	15035.0	1
2016-02-01	14998.0	2
2016-03-01	15168.0	3
2016-04-01	15132.0	4
2016-05-01	15227.0	5
...	...	...
2024-08-01	15135.0	8
2024-09-01	15318.0	9
2024-10-01	15426.0	10
2024-11-01	15388.5	11
2024-12-01	15114.5	12

108 rows × 2 columns

In [ ]:
```
lst_df.index = pd.to_datetime(lst_df.index)
lst_df.index.freq = 'MS'
```

In [ ]:
```
steps = 24
data_train = lst_df[:-steps]
data_test  = lst_df[-steps:]
```

We will use XGBRegressor as the algorithm to be used in recursive prediction:

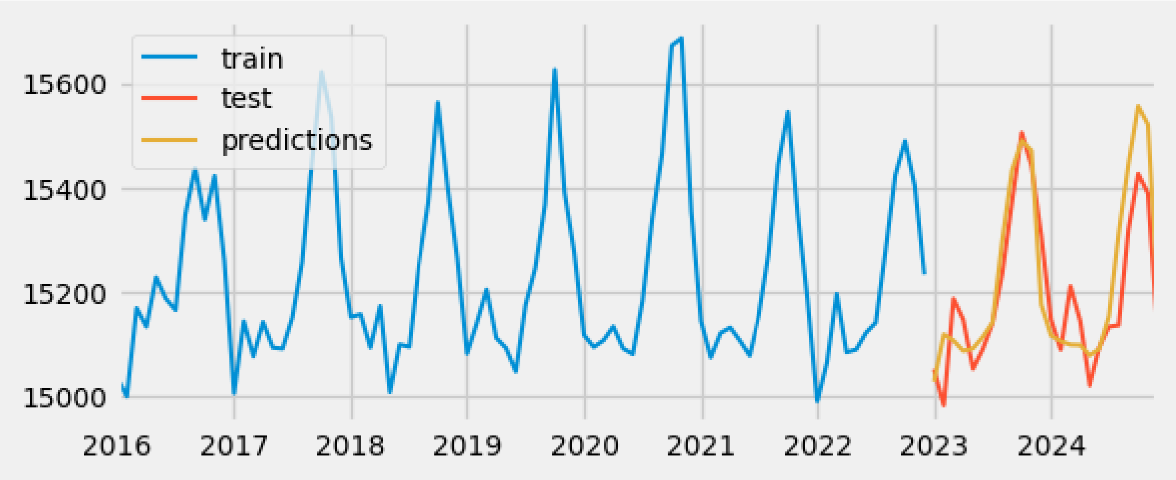
```
In [ ]: from xgboost import XGBRegressor
```

```
In [ ]: forecaster = ForecasterRecursive(
                     regressor = XGBRegressor(random_state=42),
                     lags      = 12
                 )
        forecaster.fit(y=data_train['LST'], exog=data_train['month'])
```

```
In [ ]: predictions = forecaster.predict(steps=steps, exog=data_test['month'])
```

We generate the results and compare them with the reference data:

```
In [ ]: fig, ax = plt.subplots(figsize=(6, 2.5))
        data_train['LST'].plot(ax=ax, label='train')
        data_test['LST'].plot(ax=ax, label='test')
        predictions.plot(ax=ax, label='predictions')
        ax.legend();
```



Prophet

Prophet is an open-source tool released by Facebook's Data Science team that produces time-series forecasting data based on an additive model where a non-linear trend adjusts for seasonality and holiday effects. The design principles allow parameter adjustments without much knowledge of the underlying model, making the method applicable to teams with less statistical expertise.

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

Prophet is designed to forecast univariate time series data based on decomposition components (trend+seasonality+holidays). Prophet is easy to use and automatically finds a good set of hyperparameters in an effort to make skillful predictions for trending data without requiring special domain knowledge.

Prophet also provides easy and customizable ways to tune the hyperparameters, even for someone who has no experience in forecasting models to make skillful predictions for a variety of problems in a business scenario.

Prophet is particularly well-suited for business forecasting applications and has gained popularity due to its ease of use and effectiveness in handling a wide range of time series data. As with all tools, keep in mind that while Prophet is powerful, the choice of forecasting method depends on the specific characteristics of the data and the goals of the analysis. In general, Prophet is not guaranteed to perform better than other models. However, Prophet does come with some useful features, for example, reflecting pre- and post-COVID seasonality changes or treating lockdowns as one-off holidays.

In [ ]:
```
!pip install prophet
```

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

```
Requirement already satisfied: prophet in /usr/local/lib/python3.11/dist-packages
(1.1.6)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.11/dist
-packages (from prophet) (1.2.5)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.11/dist-pa
ckages (from prophet) (2.0.2)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.11/dis
t-packages (from prophet) (3.10.0)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.11/dist-pa
ckages (from prophet) (2.2.2)
Requirement already satisfied: holidays<1,>=0.25 in /usr/local/lib/python3.11/dis
t-packages (from prophet) (0.72)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.11/dist-pac
kages (from prophet) (4.67.1)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.11/d
ist-packages (from prophet) (6.5.2)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.11/
dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-
packages (from holidays<1,>=0.25->prophet) (2.9.0.post0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib>=2.0.0->prophet) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pac
kages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib>=2.0.0->prophet) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib>=2.0.0->prophet) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-
packages (from matplotlib>=2.0.0->prophet) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packag
es (from matplotlib>=2.0.0->prophet) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib>=2.0.0->prophet) (3.2.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pac
kages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-package
s (from python-dateutil->holidays<1,>=0.25->prophet) (1.17.0)
```

We create the columns 'ds' and 'y':

In [ ]:
```
train_prophet = pd.DataFrame()
train_prophet['ds'] = data_train.index
train_prophet['y'] = data_train.LST.values
```

We can apply Prophet to our dataset:

In [ ]:
```
from prophet import Prophet


model = Prophet( yearly_seasonality=True, seasonality_mode = 'additive')
model.fit(train_prophet)
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=Tr
ue to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ebnlkdx/jedv4oec.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ebnlkdx/jgsp0py5.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=47993', 'data', 'file=/tmp/tmp9ebnl
kdx/jedv4oec.json', 'init=/tmp/tmp9ebnlkdx/jgsp0py5.json', 'output', 'file=/tmp/t
mp9ebnlkdx/prophet_modelx_dtpwel/prophet_model-20250523005614.csv', 'method=optim
ize', 'algorithm=newton', 'iter=10000']
00:56:14 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
00:56:15 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

Out[ ]: <prophet.forecaster.Prophet at 0x7c9ebb61b6d0>

In [ ]:
```
future = model.make_future_dataframe(periods = 24, freq = 'M')
```

```
/usr/local/lib/python3.11/dist-packages/prophet/forecaster.py:1854: FutureWarnin
g: 'M' is deprecated and will be removed in a future version, please use 'ME' ins
tead.
  dates = pd.date_range(
```

In [ ]:
```
future
```

Out[ ]:

	ds
0	2016-01-01
1	2016-02-01
2	2016-03-01
3	2016-04-01
4	2016-05-01
...	...
103	2024-07-31
104	2024-08-31
105	2024-09-30
106	2024-10-31
107	2024-11-30

108 rows × 1 columns

We apply prediction to generate future data:

In [ ]:
```
forecast = model.predict(future)
```

In [ ]:
```
forecast.index = lst_df.index
```

In [ ]:
```
forecast.iloc[-24:][['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
```

Out[ ]:

	ds	yhat	yhat_lower	yhat_upper
2023-01-01	2022-12-31	15087.575398	15028.129569	15145.838750
2023-02-01	2023-01-31	14998.490457	14942.658680	15057.143660
2023-03-01	2023-02-28	15251.216540	15195.016855	15309.554991
2023-04-01	2023-03-31	15103.989810	15044.341049	15162.383139
2023-05-01	2023-04-30	14955.641776	14893.764485	15018.697603
2023-06-01	2023-05-31	15054.463797	14993.486580	15118.273605
2023-07-01	2023-06-30	15176.739494	15108.195946	15238.086212
2023-08-01	2023-07-31	15186.660825	15115.538625	15258.944642
2023-09-01	2023-08-31	15353.880081	15278.573998	15429.496362
2023-10-01	2023-09-30	15774.148481	15688.782219	15856.575931
2023-11-01	2023-10-31	15184.954162	15098.792028	15277.608946
2023-12-01	2023-11-30	15240.807458	15141.782868	15340.482775
2024-01-01	2023-12-31	15091.274342	14983.956448	15209.026973
2024-02-01	2024-01-31	14978.854083	14865.165465	15091.734949
2024-03-01	2024-02-29	15170.726868	15037.049632	15302.817624
2024-04-01	2024-03-31	15098.205656	14964.917165	15237.537292
2024-05-01	2024-04-30	15009.440170	14858.314214	15159.637421
2024-06-01	2024-05-31	15053.346201	14891.291081	15213.968421
2024-07-01	2024-06-30	15140.731284	14970.788539	15320.500233
2024-08-01	2024-07-31	15214.886729	15036.138927	15405.688779
2024-09-01	2024-08-31	15363.467145	15166.712685	15579.422830
2024-10-01	2024-09-30	15635.976918	15417.114672	15857.213569
2024-11-01	2024-10-31	15305.473954	15092.972224	15528.042735
2024-12-01	2024-11-30	15227.157523	14991.446347	15474.223684

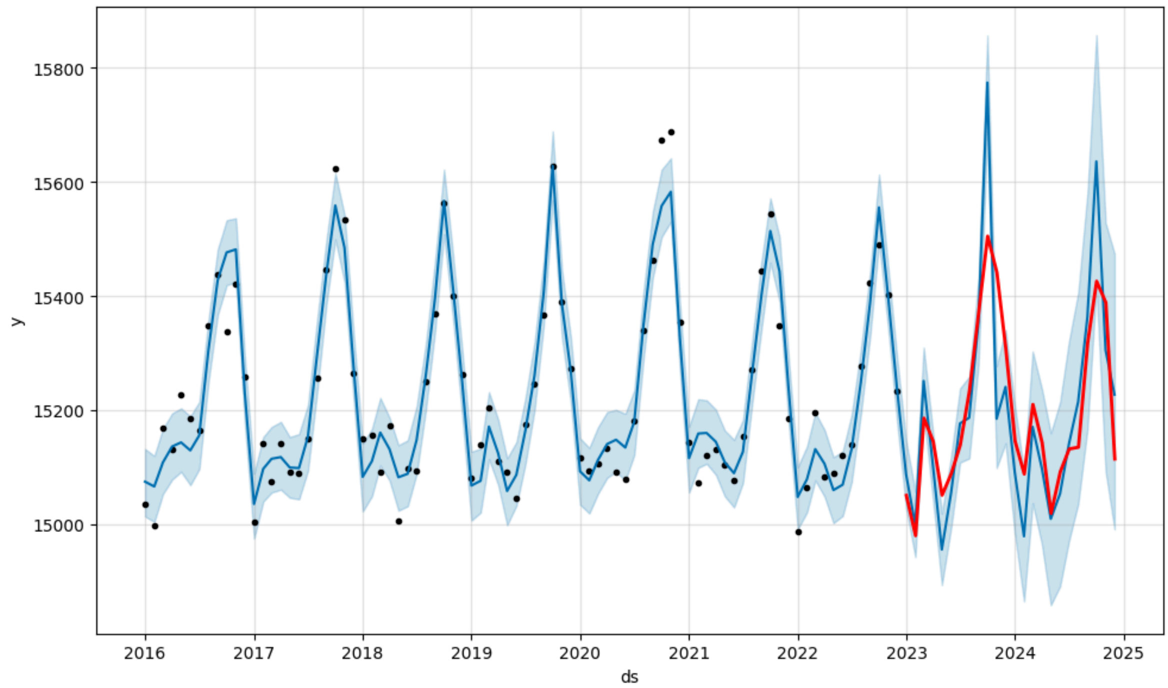
Let's compare with the reference values:

In [ ]:
```
from sklearn.metrics import r2_score
print(r2_score(list(data_test['LST']), list(forecast.iloc[-24:]['yhat'])))
```

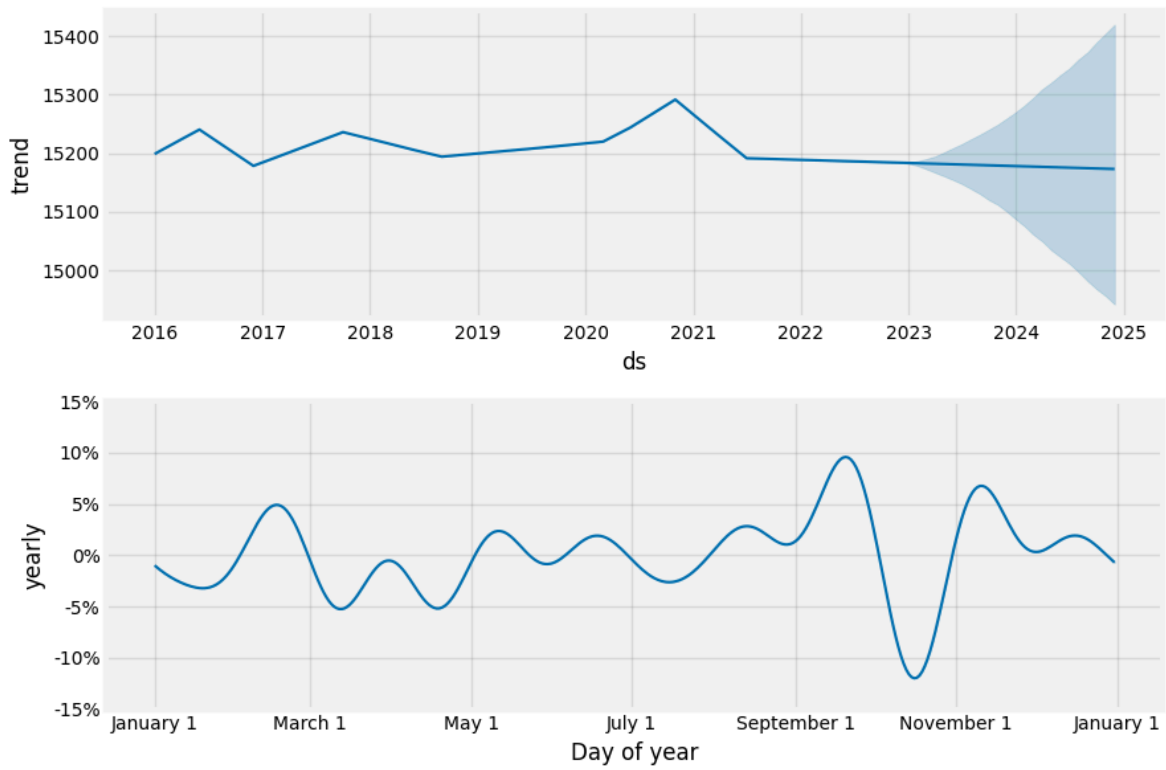
```
0.4705484740015937
```

In [ ]:
```
fig = model.plot(forecast)
#plot the predictions for validation set

plt.plot(data_test['LST'], label='Valid', color = 'red', linewidth = 2)
```

```python
plt.show()
```



In [ ]:
```python
model.plot_components(forecast);
```



Thank you! See you in the next Chapter!

References:

https://medium.com/@mouse3mic3/a-practical-guide-on-scikit-learn-for-time-series-forecasting-bbd15b611a5d

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com

https://github.com/ashishpatel26/Introduction-to-Time-Series-forecasting/blob/master/Time%20Series%20in%20Python.ipynb

https://cienciadedatos.net/documentos/py27-time-series-forecasting-python-scikitlearn

JOAO OTAVIO NASCIMENTO FIRIGATO, joaootavionf007@gmail.com