```
### Time Series Experis - Dart3
```

```
####!pip install pandas-datareader
```

```
import pandas_datareader as pdr
import pandas as pd
from datetime import datetime
```

```
#!pip install pycryptodome pycryptodomex
#!pip uninstall pandas-datareader
#!pip install git+https://github.com/raphi6/pandas-datareader.git@ea66d6b981554f9d0262038aef2106dda7138316
```

```
import datetime as dt
import yfinance as yf

company = 'MAN'

# Define a start date and End Date
start = dt.datetime(2015,1,1)
end =  dt.datetime(2023,4,4)
```

```
# Read Stock Price Data
data = yf.download(company, start , end ,ignore_tz=True)

data.tail(10)
```

```
data.shape
```

```
(2077, 6)
```

```
####! pip install tensorflow
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn import preprocessing
import matplotlib.pyplot as plt
tf.random.set_seed(123)
np.random.seed(123)
```

```
import pandas as pd
#import fbprophet
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
data.to_csv("/content/manpower.csv")
```

```
df=pd.read_csv('/content/manpower.csv')
df.head(3)
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-01-02	68.550003	68.809998	66.709999	67.470001	56.032017	346800
1	2015-01-05	66.919998	67.120003	64.949997	65.879997	54.711571	587400
2	2015-01-06	66.190002	66.459999	63.980000	65.320000	54.246517	791700

```
# Installing darts
####!pip install darts
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn import preprocessing
import matplotlib.pyplot as plt
tf.random.set_seed(123)
np.random.seed(123)
```

```
df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

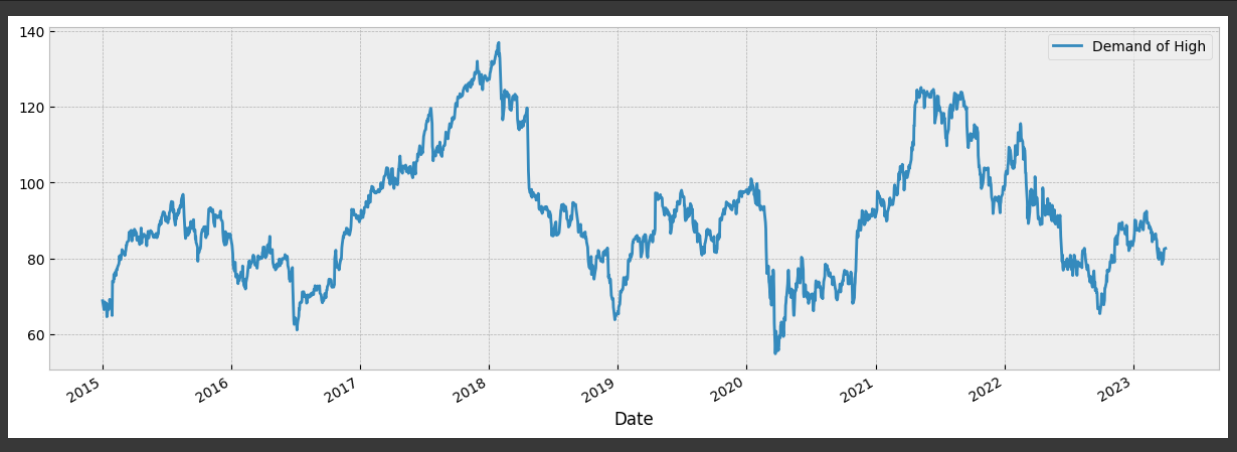
```
# Basic packages
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import random as rd # generating random numbers
import datetime # manipulating date formats
# Viz
import matplotlib.pyplot as plt # basic plotting
import seaborn as sns # for prettier plots
# TIME SERIES
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.stattools import adfuller, acf, pacf,arma_order_select_ic
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
# settings
import warnings
warnings.filterwarnings("ignore")
```

```
df.dtypes
```

```
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

```
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index("Date")
```

```
plt.style.use("bmh")
plt.figure(figsize=(15,5))
df["High"].plot(label='Demand of High')
plt.legend();
```




```
list(df.columns)
```

```
['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
```

```
#Resampling demand at daily level
daily_experis_stock = pd.DataFrame(df[['High']].resample('D', origin = 'start').sum())
# daily_demand.reset_index(inplace=True)
daily_experis_stock.head()
```

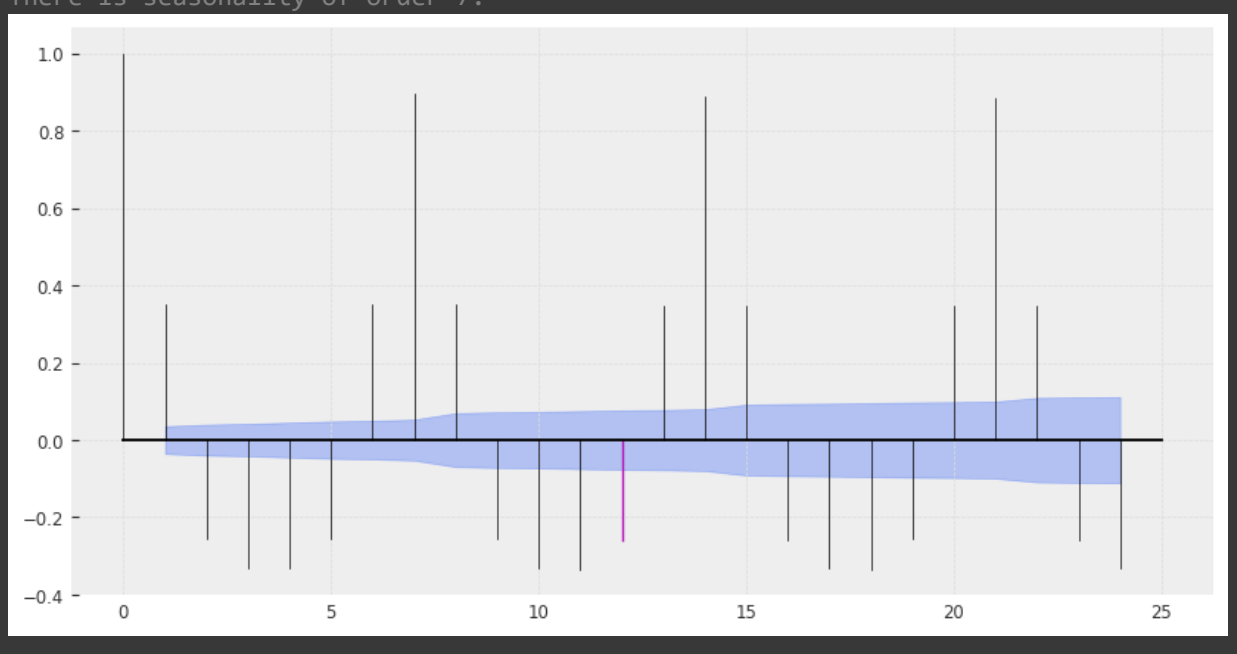
Date	High
2015-01-02	68.809998
2015-01-03	0.000000

```
from darts import TimeSeries
series_High = TimeSeries.from_dataframe(daily_experis_stock, value_cols='High')
series_High
```

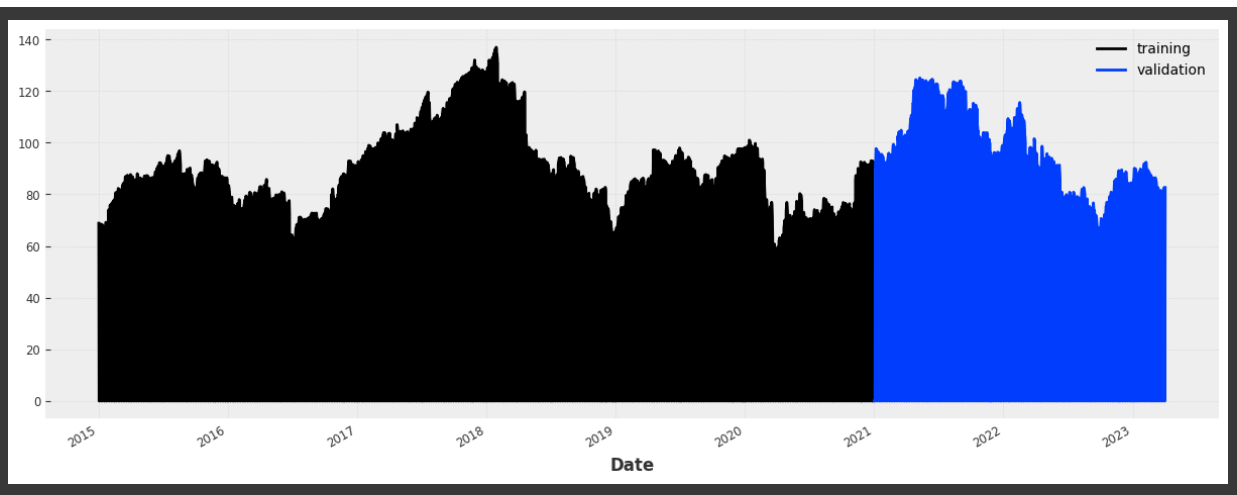
```
TimeSeries (DataArray)  (Date: 3014, component: 1, sample: 1)
array([[[68.80999756]],

       [[ 0.        ]],

       [[ 0.        ]],

       ...,

       [[ 0.        ]],

       [[ 0.        ]],

       [[82.62999725]]])
▼ Coordinates:
  Date       (Date)       datetime64[ns]  2015-01-02 ... 2023-04-03
  component  (component)  object          'High'
► Indexes: (2)
▼ Attributes:
    static_covariates :  None
    hierarchy :          None
```

```
# Check for seasonality
from darts.utils.statistics import plot_acf, check_seasonality

for m in range(2, 25):
    is_seasonal, mseas = check_seasonality(series_High, m=m, alpha=0.05)
    if is_seasonal:
        break

print("seasonal? " + str(is_seasonal))
if is_seasonal:
    print('There is seasonality of order {}.'.format(mseas))

plot_acf(series_High, m=12, alpha=0.05)
```

```
seasonal? True
There is seasonality of order 7.
```




```
train, val = series_High.split_before(pd.Timestamp("20210101"))
plt.figure(figsize=(15,5))
train.plot(label="training")
val.plot(label="validation")
```




```
from datetime import datetime
```

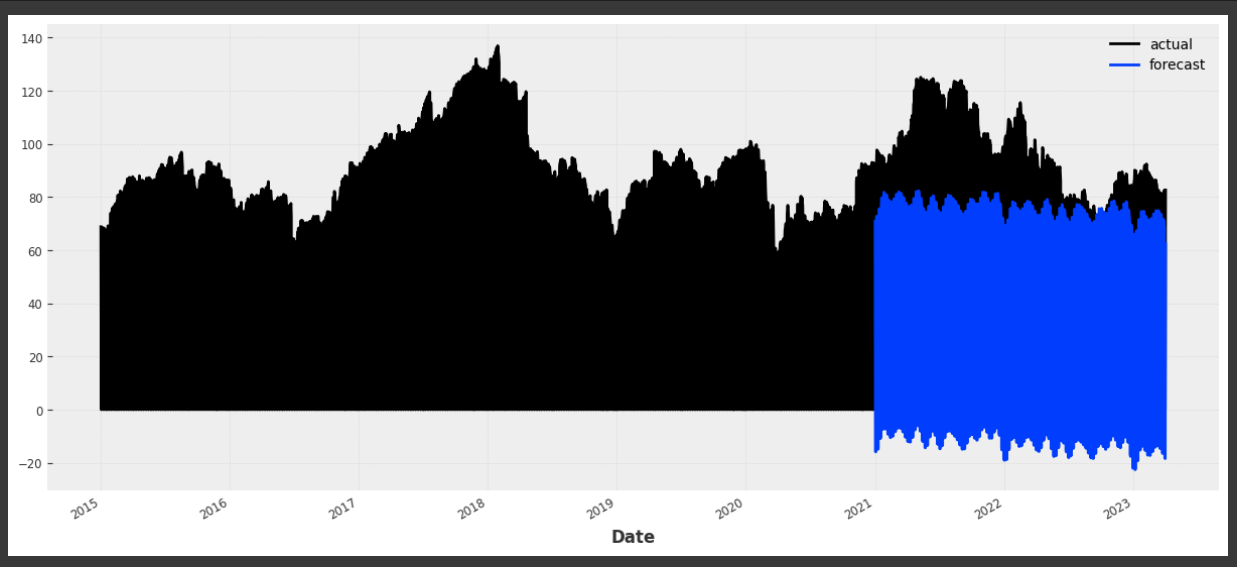
```
from darts.models import ExponentialSmoothing, FFT, Prophet, AutoARIMA, Theta
from darts.metrics import mape, r2_score

def eval_model(model):
    start_time = datetime.now()
    model.fit(train)
    time_elapsed = datetime.now() - start_time
    forecast = model.predict(len(val))
    print("model ",forecast)


eval_model(ExponentialSmoothing())
eval_model(FFT())
eval_model(Prophet())
eval_model(AutoARIMA())
### eval_model(Theta())
```

```
    * component  (component) object 'High'
Dimensions without coordinates: sample
Attributes:
    static_covariates:  None
    hierarchy:          None
```

```
model_prophet = Prophet()
model_prophet.fit(train)
```

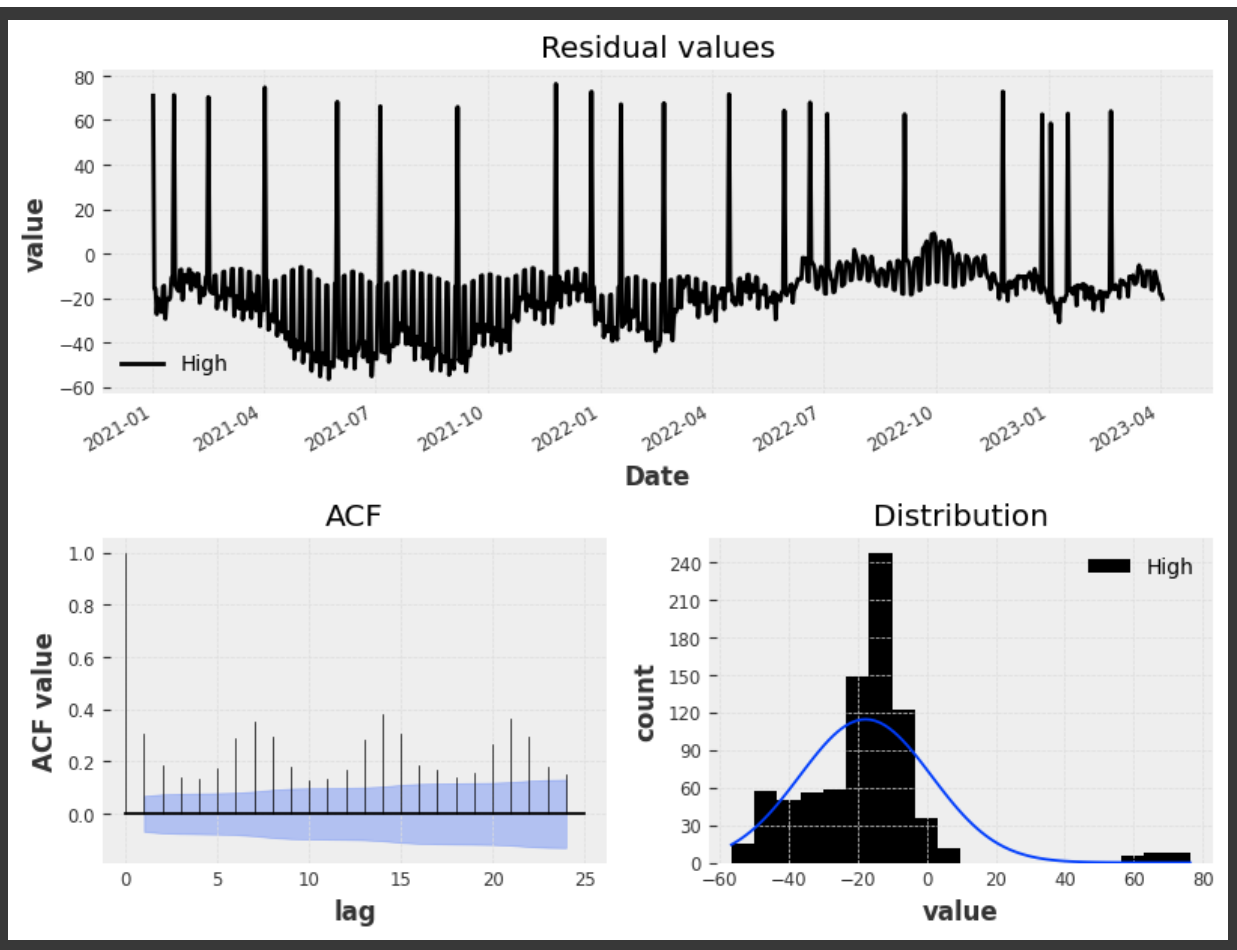
```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpcb8s77en/fxxwvxtj.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpcb8s77en/h09vyoix.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.9/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 's
06:49:33 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
06:49:33 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet(add_seasonalities=None, country_holidays=None, suppress_stdout_stderror=True, add_encoders=None, cap=None,
floor=None)
```

```
# use the model to make a forecast
pred_prophet = model_prophet.predict(len(val))

plt.figure(figsize=(15, 6))
series_High.plot(label="actual")
pred_prophet.plot(label="forecast")
```




```
def display_forecast(pred_series, ts_transformed, forecast_type, start_date=None):
    plt.figure(figsize=(15, 5))
    if start_date:
        ts_transformed = ts_transformed.drop_before(start_date)
    ts_transformed.univariate_component(0).plot(label="actual")
    pred_series.plot(label=("historic " + forecast_type + " forecasts"))
    plt.title(
        "R2: {}".format(r2_score(ts_transformed.univariate_component(0), pred_series))
    )
    plt.legend()

pred_series = model_prophet.historical_forecasts(
    series_High,
    start=pd.Timestamp("20210101"),
    forecast_horizon=30,
    stride=5,
    verbose=True,
)
display_forecast(pred_series, series_High, "30 day", start_date=pd.Timestamp("20190101"))
```

```
# investigate the residuals in the validation dataset
from darts.utils.statistics import plot_residuals_analysis
resid = pred_prophet - val
plot_residuals_analysis(resid);
```




```
train.columns
```

```
Index(['High'], dtype='object', name='component')
```

```
df.columns
```

```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

```
df2 = pd.DataFrame(df[['Volume']].resample('D', origin = 'start').mean())
df2.head()
```

Date	Volume
2015-01-02	346800.0
2015-01-03	NaN
2015-01-04	NaN
2015-01-05	587400.0
2015-01-06	791700.0

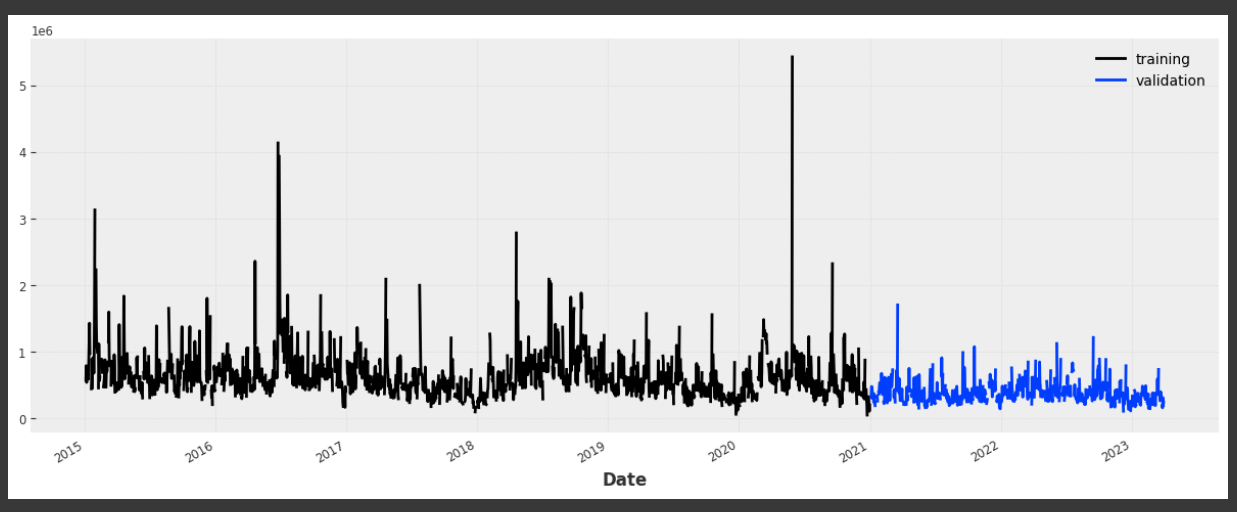
```
data3 = pd.DataFrame(df2[['Volume']].resample('D', origin = 'start').mean())
data3.head()
```

Date	Volume
2015-01-02	346800.0
2015-01-03	NaN
2015-01-04	NaN
2015-01-05	587400.0
2015-01-06	791700.0

```
from darts import TimeSeries
series_volume = TimeSeries.from_dataframe(data3, value_cols='Volume')
series_volume
```

```
TimeSeries (DataArray)  (Date: 3014, component: 1, sample: 1)
array([[[346800.]],

       [[    nan]],

       [[    nan]],

       ...,

       [[    nan]],

       [[    nan]],

       [[298000.]]])
```

```
train, val = series_volume.split_before(pd.Timestamp("20210101"))
plt.figure(figsize=(15,5))
train.plot(label="training")
val.plot(label="validation")
```




```
model_future_covs = Prophet()
model_future_covs.fit(train, future_covariates=series_volume)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpcb8s77en/a1uw28qk.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpcb8s77en/mve2xg3r.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.9/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 's
06:51:41 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
06:51:42 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
Prophet(add_seasonalities=None, country_holidays=None, suppress_stdout_stderror=True, add_encoders=None, cap=None,
floor=None)
```

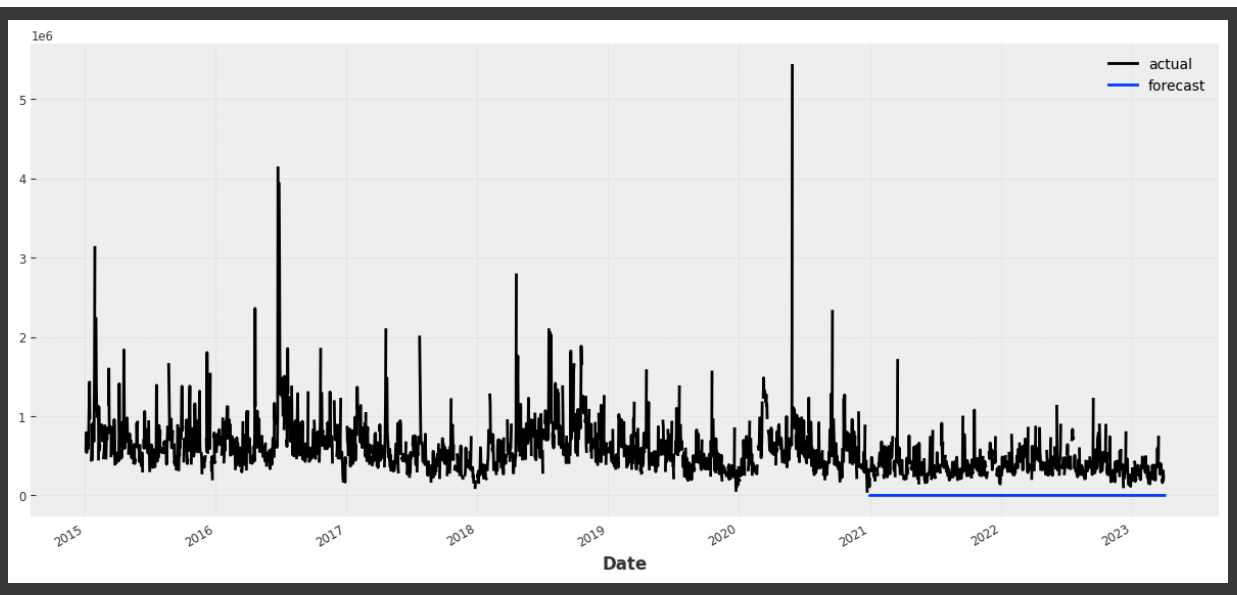
```
pred_series = model_prophet.historical_forecasts(
    series_volume,
    start=pd.Timestamp("20210101"),
    forecast_horizon=30,
    stride=5,
    verbose=True, future_covariates=series_High
)
display_forecast(pred_series, series_volume, "30 day", start_date=pd.Timestamp("20210101"))
```

```
from sklearn.metrics import mean_absolute_error as mae
```

```
print(train.columns, val.columns)
```

```
Index(['Volume'], dtype='object', name='component') Index(['Volume'], dtype='object', name='component')
```

```
# use the model to make a forecast
pred_volume = model_prophet.predict(len(val))

plt.figure(figsize=(15, 6))
series_volume.plot(label="actual")
pred_volume.plot(label="forecast")
#plt.title("MAE: {}".format(mae(val, pred_volume)))
```



Colab paid products - Cancel contracts here

✓ 1s completed at 12:39 PM