```
!pip install prophet
!pip install pmdarima
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.1
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/d
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/d
Requirement already satisfied: holidays>=0.25 in /usr/local/lib/python3.10/
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: importlib-resources in /usr/local/lib/python
Requirement already satisfied: stanio~=0.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pmdarima in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/di
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/loc
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/d
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/di
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/li
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pyt
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packag
```

```
import lightgbm as lgb
import numpy as np
import pandas as pd

from prophet import Prophet
from matplotlib import pyplot as plt
!pip install statsmodels
!pip install pmdarima
from pmdarima import auto_arima
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dis
Requirement already satisfied: numpy>=1.18 in /usr/local/lib/python3.10/dis
Requirement already satisfied: scipy!=1.9.2,>=1.4 in /usr/local/lib/python3
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pyt
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pmdarima in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/di
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/loc
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/d
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/di
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/li
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pyt
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packag
```

Data Preparations

```
df = pd.read_csv("/content/HDFC.csv")
df.set_index("Date", drop=False, inplace=True)
df.head()
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Vol
Date											
2000-01-03	2000-01-03	HDFC	EQ	271.75	293.5	293.50	293.5	293.5	293.50	293.50	2
2000-01-04	2000-01-04	HDFC	EQ	293.50	317.0	317.00	297.0	304.0	304.05	303.62	25
2000-01-05	2000-01-05	HDFC	EQ	304.05	290.0	303.90	285.0	295.0	292.80	294.53	26

```
df.shape
```

```
(5306, 15)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5306 entries, 2000-01-03 to 2021-04-30
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Date                5306 non-null   object 
 1   Symbol              5306 non-null   object 
 2   Series              5306 non-null   object 
 3   Prev Close          5306 non-null   float64
 4   Open                5306 non-null   float64
 5   High                5306 non-null   float64
 6   Low                 5306 non-null   float64
 7   Last                5306 non-null   float64
 8   Close               5306 non-null   float64
 9   VWAP                5306 non-null   float64
 10  Volume              5306 non-null   int64  
 11  Turnover            5306 non-null   float64
 12  Trades              2456 non-null   float64
 13  Deliverable Volume  4797 non-null   float64
 14  %Deliverble         4797 non-null   float64
dtypes: float64(11), int64(1), object(3)
memory usage: 663.2+ KB
```

```
df.describe()
```

	Prev Close	Open	High	Low	Last	Close
count	5306.000000	5306.000000	5306.000000	5306.000000	5306.000000	5306.000000
mean	1283.666114	1284.393074	1304.269732	1263.297842	1283.885017	1284.071005
std	709.395090	709.703665	721.308080	697.450309	709.250204	709.430515
min	271.750000	284.000000	290.500000	273.250000	282.850000	283.850000
25%	668.650000	669.712500	677.512500	660.000000	669.000000	668.662500
50%	1136.275000	1135.400000	1156.725000	1119.000000	1135.000000	1136.675000
75%	1811.475000	1813.812500	1835.000000	1783.075000	1812.000000	1811.787500
max	3180.150000	3148.000000	3262.000000	3100.550000	3178.000000	3180.150000

```
print(df.isnull().sum())
```

```
Date                    0
Symbol                  0
Series                  0
Prev Close              0
Open                    0
High                    0
Low                     0
Last                    0
Close                   0
VWAP                    0
Volume                  0
Turnover                0
Trades               2850
Deliverable Volume    509
%Deliverble           509
dtype: int64
```

```
df = df.dropna()
```

Feature Engineering

```
df.reset_index(drop=True, inplace=True)
lag_features = ["High", "Low", "Volume", "Turnover", "Trades"]
window1 = 3
```

```
window2 = 7
window3 = 30

df_rolled_3d = df[lag_features].rolling(window=window1, min_periods=0)
df_rolled_7d = df[lag_features].rolling(window=window2, min_periods=0)
df_rolled_30d = df[lag_features].rolling(window=window3, min_periods=0)

df_mean_3d = df_rolled_3d.mean().shift(1).reset_index().astype(np.float32)
df_mean_7d = df_rolled_7d.mean().shift(1).reset_index().astype(np.float32)
df_mean_30d = df_rolled_30d.mean().shift(1).reset_index().astype(np.float32)

df_std_3d = df_rolled_3d.std().shift(1).reset_index().astype(np.float32)
df_std_7d = df_rolled_7d.std().shift(1).reset_index().astype(np.float32)
df_std_30d = df_rolled_30d.std().shift(1).reset_index().astype(np.float32)

for feature in lag_features:
    df[f"{feature}_mean_lag{window1}"] = df_mean_3d[feature]
    df[f"{feature}_mean_lag{window2}"] = df_mean_7d[feature]
    df[f"{feature}_mean_lag{window3}"] = df_mean_30d[feature]

    df[f"{feature}_std_lag{window1}"] = df_std_3d[feature]
    df[f"{feature}_std_lag{window2}"] = df_std_7d[feature]
    df[f"{feature}_std_lag{window3}"] = df_std_30d[feature]

df.fillna(df.mean(), inplace=True)

df.set_index("Date", drop=False, inplace=True)
df.head()
```

```
<ipython-input-37-6c7b34204c4b>:28: FutureWarning: The default value of num
  df.fillna(df.mean(), inplace=True)
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP
Date										
2011-06-01	2011-06-01	HDFC	EQ	684.05	676.55	692.95	676.55	689.00	689.10	688.38
2011-06-02	2011-06-02	HDFC	EQ	689.10	681.05	684.70	676.60	680.25	680.00	680.53
2011-06-03	2011-06-03	HDFC	EQ	680.00	678.50	683.05	658.25	659.15	660.05	668.24
2011-06-06	2011-06-06	HDFC	EQ	660.05	659.95	674.10	659.15	671.00	670.65	668.56

2011-06-07	2011-06-07	HDFC	EQ	670.65	668.00	674.65	662.30	667.35	669.20	669.01

5 rows × 45 columns

```
Warning: Total number of columns (45) exceeds max_columns (20) limiting to
Warning: Total number of columns (45) exceeds max_columns (20) limiting to
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/google/colab/data_table.py"
    dataframe = self._preprocess_dataframe()
  File "/usr/local/lib/python3.10/dist-packages/google/colab/data_table.py"
    dataframe = dataframe.reset_index()
  File "/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py"
    return func(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py", line
    new_obj.insert(
  File "/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py", line
    raise ValueError(f"cannot insert {column}, already exists")
ValueError: cannot insert Date, already exists
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/google/colab/data_table.py"
    return self._gen_js(self._preprocess_dataframe())
  File "/usr/local/lib/python3.10/dist-packages/google/colab/data_table.py"
    dataframe = dataframe.reset_index()
  File "/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py"
    return func(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py", line
    new_obj.insert(
  File "/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py", line
    raise ValueError(f"cannot insert {column}, already exists")
ValueError: cannot insert Date, already exists
```

```
import numpy as np
import pandas as pd  # Assuming pandas is imported

# Assuming df is your DataFrame

# Your initial code
df.reset_index(drop=True, inplace=True)
lag_features = ["High", "Low", "Volume", "Turnover", "Trades"]
window1 = 3
window2 = 7
window3 = 30

df_rolled_3d = df[lag_features].rolling(window=window1, min_periods=0)
df_rolled_7d = df[lag_features].rolling(window=window2, min_periods=0)
df_rolled_30d = df[lag_features].rolling(window=window3, min_periods=0)

df_mean_3d = df_rolled_3d.mean().shift(1).reset_index().astype(np.float32)
```

```
df_mean_7d = df_rolled_7d.mean().shift(1).reset_index().astype(np.float32)
df_mean_30d = df_rolled_30d.mean().shift(1).reset_index().astype(np.float32)

df_std_3d = df_rolled_3d.std().shift(1).reset_index().astype(np.float32)
df_std_7d = df_rolled_7d.std().shift(1).reset_index().astype(np.float32)
df_std_30d = df_rolled_30d.std().shift(1).reset_index().astype(np.float32)

for feature in lag_features:
    df[f"{feature}_mean_lag{window1}"] = df_mean_3d[feature]
    df[f"{feature}_mean_lag{window2}"] = df_mean_7d[feature]
    df[f"{feature}_mean_lag{window3}"] = df_mean_30d[feature]

    df[f"{feature}_std_lag{window1}"] = df_std_3d[feature]
    df[f"{feature}_std_lag{window2}"] = df_std_7d[feature]
    df[f"{feature}_std_lag{window3}"] = df_std_30d[feature]

# Additional window periods
window4 = 60
window5 = 90
window6 = 120

# Calculate rolling statistics for the new window periods
df_rolled_60d = df[lag_features].rolling(window=window4, min_periods=0)
df_rolled_90d = df[lag_features].rolling(window=window5, min_periods=0)
df_rolled_120d = df[lag_features].rolling(window=window6, min_periods=0)

# Calculate mean and standard deviation for the new window periods
df_mean_60d = df_rolled_60d.mean().shift(1).reset_index().astype(np.float32)
df_mean_90d = df_rolled_90d.mean().shift(1).reset_index().astype(np.float32)
df_mean_120d = df_rolled_120d.mean().shift(1).reset_index().astype(np.float32)

df_std_60d = df_rolled_60d.std().shift(1).reset_index().astype(np.float32)
df_std_90d = df_rolled_90d.std().shift(1).reset_index().astype(np.float32)
df_std_120d = df_rolled_120d.std().shift(1).reset_index().astype(np.float32)

# Add the new mean and standard deviation values to the DataFrame
for feature in lag_features:
    df[f"{feature}_mean_lag{window4}"] = df_mean_60d[feature]
    df[f"{feature}_mean_lag{window5}"] = df_mean_90d[feature]
    df[f"{feature}_mean_lag{window6}"] = df_mean_120d[feature]

    df[f"{feature}_std_lag{window4}"] = df_std_60d[feature]
    df[f"{feature}_std_lag{window5}"] = df_std_90d[feature]
    df[f"{feature}_std_lag{window6}"] = df_std_120d[feature]

# Fill missing values with the mean of each column
df.fillna(df.mean(), inplace=True)
```

```
# Set the index to "Date" column
df.set_index("Date", drop=False, inplace=True)

# Display the updated DataFrame
print(df.head())  # Displaying the updated DataFrame
```

```
                 Date Symbol Series  Prev Close    Open    High     Low  \
Date
2011-06-01 2011-06-01   HDFC     EQ      684.05  676.55  692.95  676.55
2011-06-02 2011-06-02   HDFC     EQ      689.10  681.05  684.70  676.60
2011-06-03 2011-06-03   HDFC     EQ      680.00  678.50  683.05  658.25
2011-06-06 2011-06-06   HDFC     EQ      660.05  659.95  674.10  659.15
2011-06-07 2011-06-07   HDFC     EQ      670.65  668.00  674.65  662.30

              Last   Close    VWAP  ...  Trades_mean_lag60  Trades_mean_lag
Date                                ...
2011-06-01  689.00  689.10  688.38  ...      100803.664062      100075.9921
2011-06-02  680.25  680.00  680.53  ...       38210.000000       38210.0000
2011-06-03  659.15  660.05  668.24  ...       29255.000000       29255.0000
2011-06-06  671.00  670.65  668.56  ...       30750.666016       30750.6660
2011-06-07  667.35  669.20  669.01  ...       32750.250000       32750.2500

            Trades_mean_lag120  Trades_std_lag60  Trades_std_lag90  \
Date
2011-06-01        99373.390625      35707.285156      36564.824219
2011-06-02        38210.000000      35707.285156      36564.824219
2011-06-03        29255.000000      12664.282227      12664.282227
2011-06-06        30750.666016       9322.182617       9322.182617
2011-06-07        32750.250000       8598.181641       8598.181641

            Trades_std_lag120  month  week  day  day_of_week
Date
2011-06-01       37117.847656      6    22    1            2
2011-06-02       37117.847656      6    22    2            3
2011-06-03       12664.282227      6    22    3            4
2011-06-06        9322.182617      6    23    6            0
2011-06-07        8598.181641      6    23    7            1

[5 rows x 79 columns]
<ipython-input-54-cd2dba1f3d1d>:64: FutureWarning: DataFrame.mean and DataF
  df.fillna(df.mean(), inplace=True)
<ipython-input-54-cd2dba1f3d1d>:64: FutureWarning: The default value of num
  df.fillna(df.mean(), inplace=True)
```

```
df.Date = pd.to_datetime(df.Date, format="%Y-%m-%d")
df["month"] = df.Date.dt.month
df["week"] = df.Date.dt.week
df["day"] = df.Date.dt.day
df["day_of_week"] = df.Date.dt.dayofweek
df.head()
```

```
<ipython-input-55-aaf895c467cb>:3: FutureWarning: Series.dt.weekofyear and
  df["week"] = df.Date.dt.week
```

Date	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP
2011-06-01	2011-06-01	HDFC	EQ	684.05	676.55	692.95	676.55	689.00	689.10	688.38
2011-06-02	2011-06-02	HDFC	EQ	689.10	681.05	684.70	676.60	680.25	680.00	680.53
2011-06-03	2011-06-03	HDFC	EQ	680.00	678.50	683.05	658.25	659.15	660.05	668.24
2011-06-06	2011-06-06	HDFC	EQ	660.05	659.95	674.10	659.15	671.00	670.65	668.56
2011-06-07	2011-06-07	HDFC	EQ	670.65	668.00	674.65	662.30	667.35	669.20	669.01

5 rows × 79 columns

```
df_train = df[df.Date < "2021"]
df_valid = df[df.Date >= "2021"]

exogenous_features = ["High_mean_lag3", "High_std_lag3", "Low_mean_lag3", "Low_s
                      "Volume_mean_lag3", "Volume_std_lag3", "Turnover_mean_lag3
                      "Turnover_std_lag3", "Trades_mean_lag3", "Trades_std_lag3"
                      "High_mean_lag7", "High_std_lag7", "Low_mean_lag7", "Low_s
                      "Volume_mean_lag7", "Volume_std_lag7", "Turnover_mean_lag7
                      "Turnover_std_lag7", "Trades_mean_lag7", "Trades_std_lag7"
                      "High_mean_lag30", "High_std_lag30", "Low_mean_lag30", "Lo
                      "Volume_mean_lag30", "Volume_std_lag30", "Turnover_mean_la
                      "Turnover_std_lag30", "Trades_mean_lag30", "Trades_std_lag
                      "month", "week", "day", "day_of_week"]
```

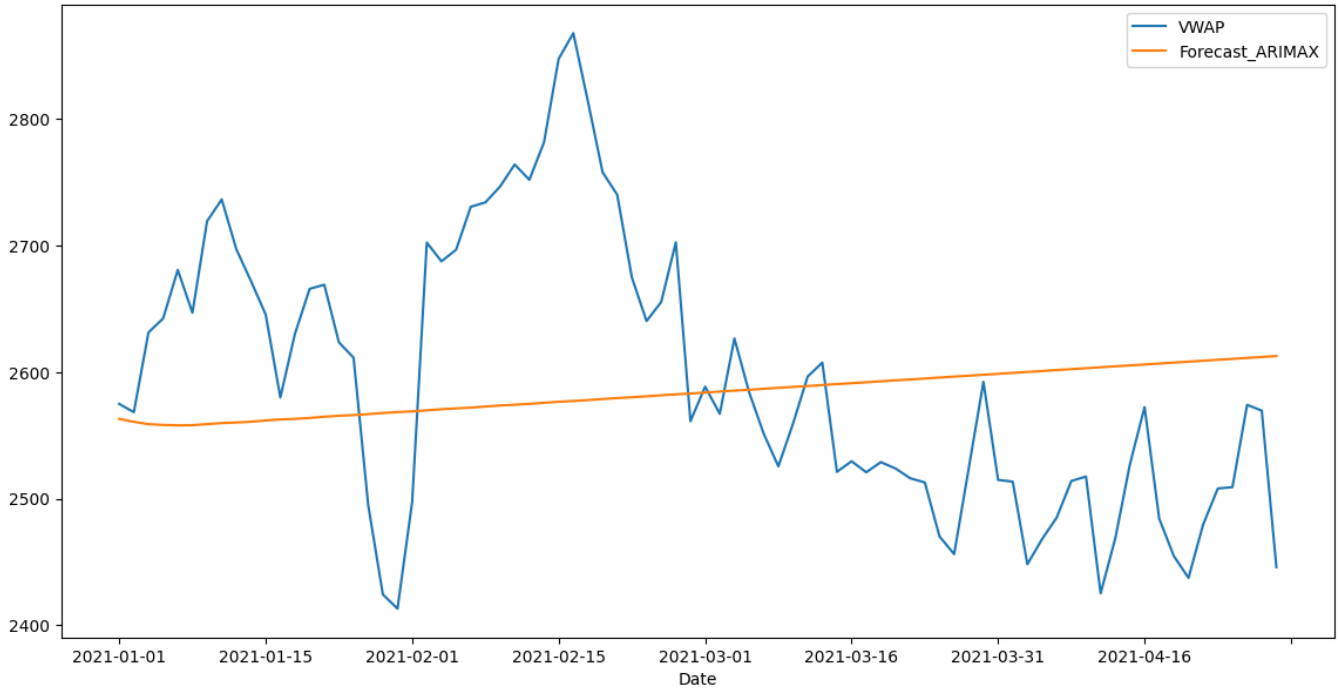
Auto ARIMAX

```
model = auto_arima(df_train.VWAP, exogenous=df_train[exogenous_features], trace=
model.fit(df_train.VWAP, exogenous=df_train[exogenous_features])

# Generate forecasts for the validation set
forecast = model.predict(n_periods=len(df_valid), exogenous=df_valid[exogenous_f
df_valid["Forecast_ARIMAX"] = forecast.values
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=21407.414, Time=3.57 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=21487.514, Time=0.10 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=21424.854, Time=0.11 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=21417.358, Time=0.86 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=21488.472, Time=0.07 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=21412.900, Time=2.14 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=21409.329, Time=2.35 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=21406.385, Time=5.20 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=21409.961, Time=1.10 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=21403.323, Time=8.54 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=21411.963, Time=1.46 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=8.59 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=21401.942, Time=7.79 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=21405.673, Time=8.17 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=21403.390, Time=9.78 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=21405.895, Time=9.63 sec
 ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=10.03 sec
 ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=21405.766, Time=10.13 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=21402.753, Time=3.20 sec

Best model:  ARIMA(4,1,3)(0,0,0)[0] intercept
Total fit time: 92.843 seconds
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:8
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:8
  return get_prediction_index(
<ipython-input-57-4670a86bb652>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  df_valid["Forecast_ARIMAX"] = forecast.values
```

```
# Plot the original VWAP and the ARIMAX forecast
df_valid[["VWAP", "Forecast_ARIMAX"]].plot(figsize=(14, 7))
```

```
<Axes: xlabel='Date'>
```




```
print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.VWAP, df_valid.Forec
```

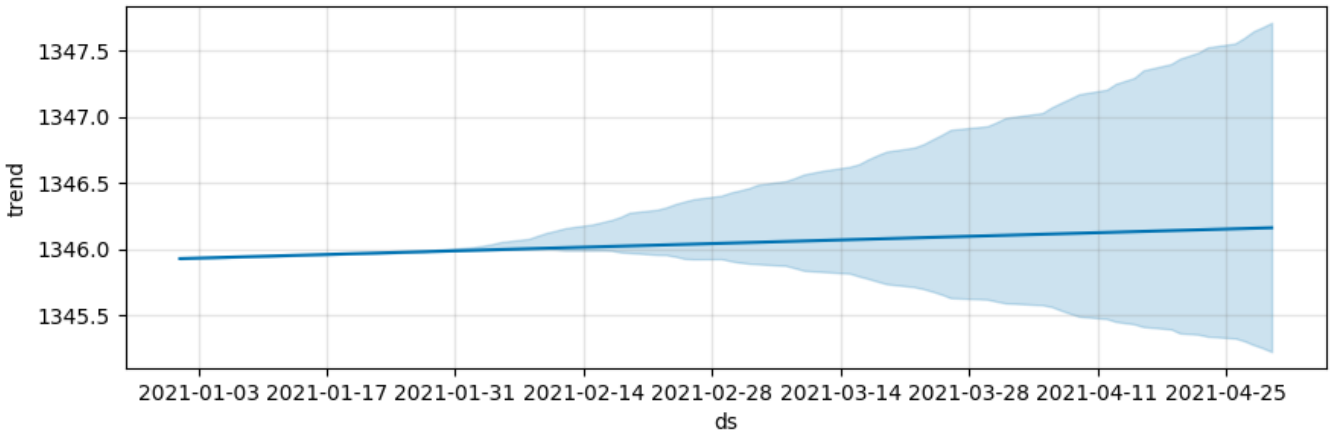
```
RMSE of Auto ARIMAX: 118.40437981485415

MAE of Auto ARIMAX: 101.19843889375443
```

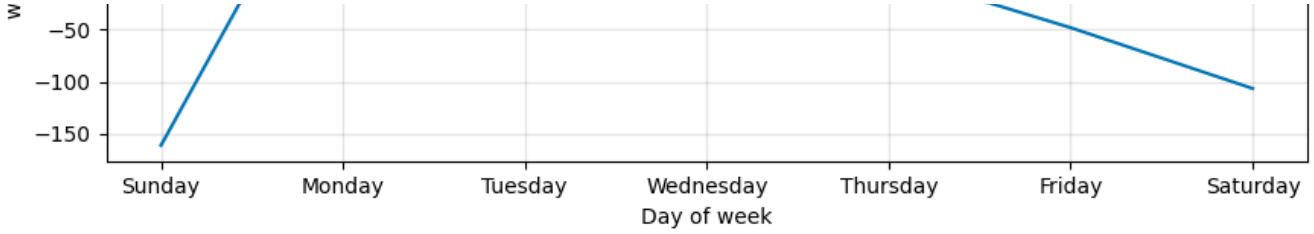
Prophet

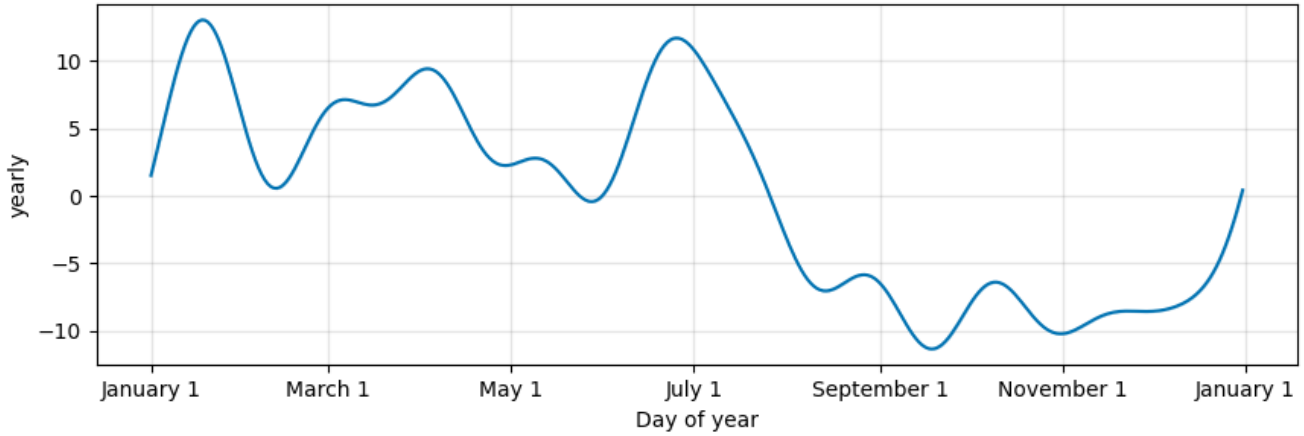
```
model_fbp = Prophet()
for feature in exogenous_features:
    model_fbp.add_regressor(feature)

model_fbp.fit(df_train[["Date", "VWAP"] + exogenous_features].rename(columns={"D

forecast = model_fbp.predict(df_valid[["Date", "VWAP"] + exogenous_features].ren
df_valid["Forecast_Prophet"] = forecast.yhat.values
```

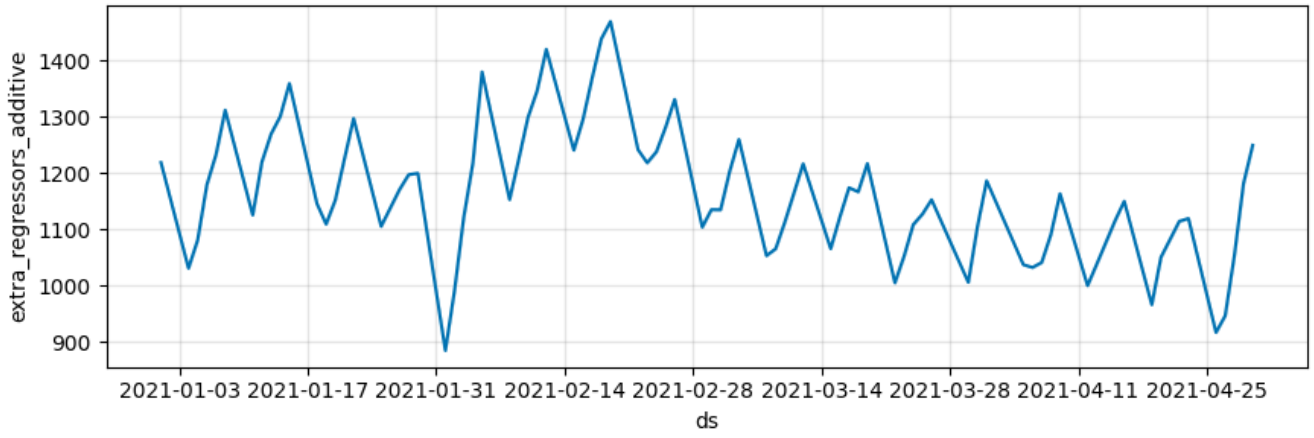
```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonalit
DEBUG:cmdstanpy:input tempfile: /tmp/tmprznchvlk/ntgqx30x.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmprznchvlk/5i1rw6ne.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/pro
15:23:33 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
15:23:35 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<ipython-input-45-113dc461dcde>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  df_valid["Forecast_Prophet"] = forecast.yhat.values
```

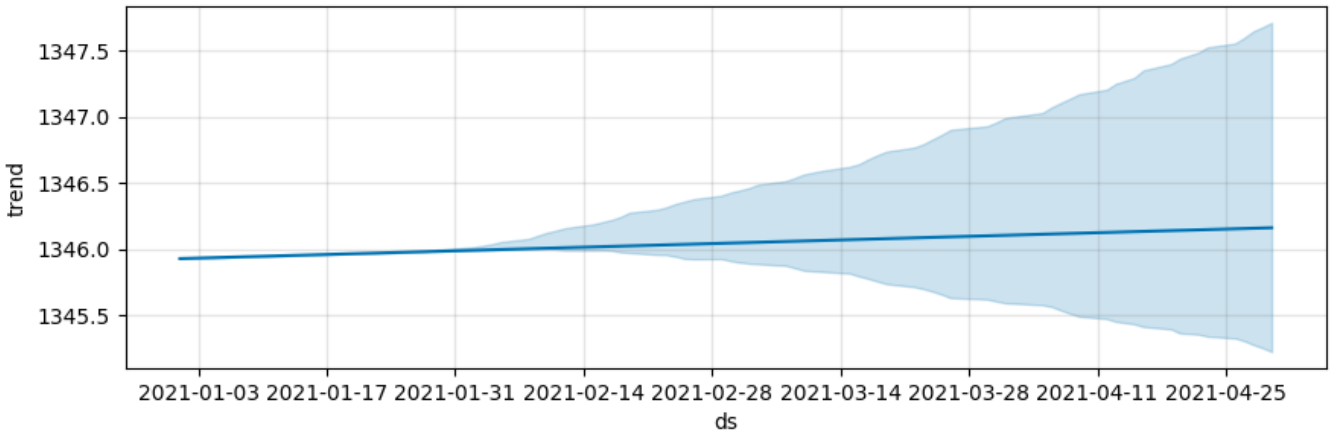
```
model_fbp.plot_components(forecast)
```

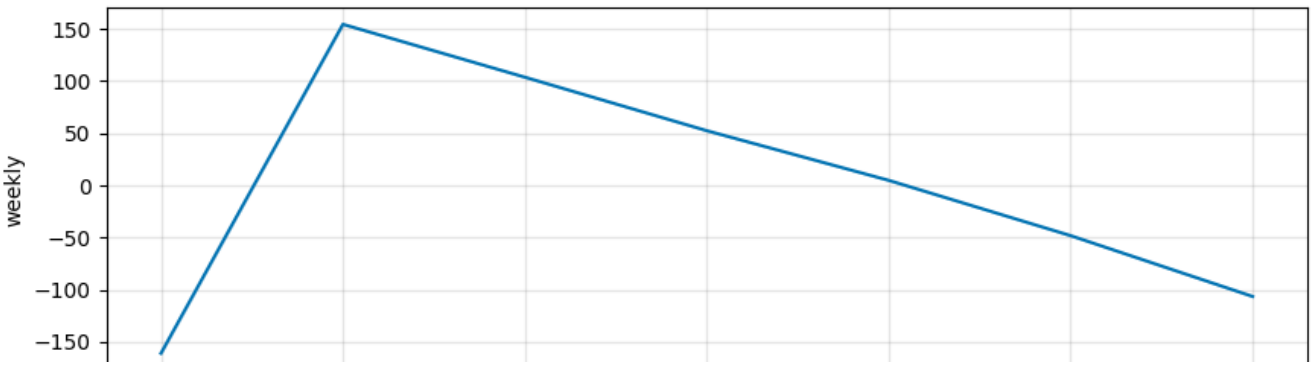


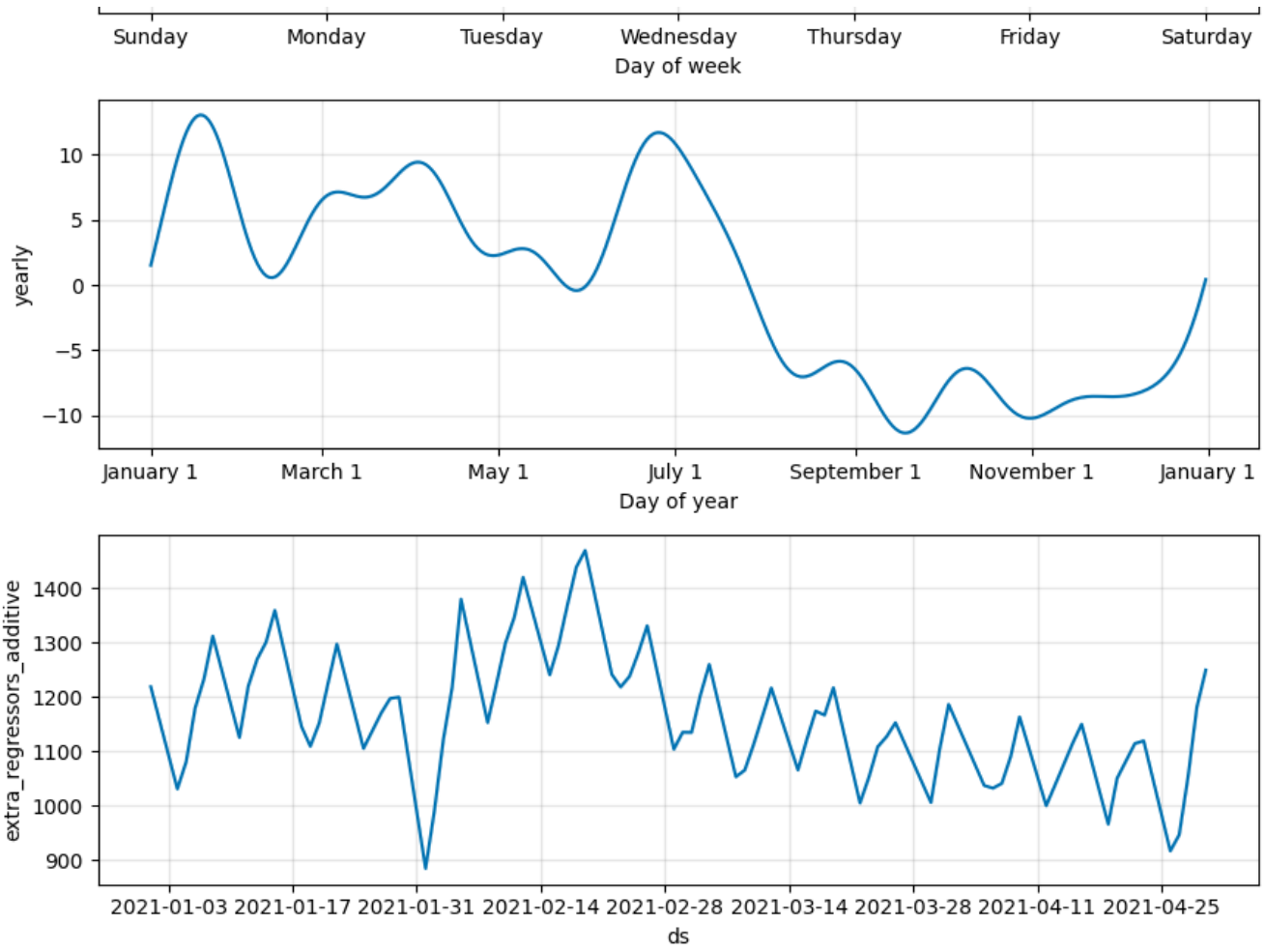

−50
−100
−150
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week


yearly
10
5
0
−5
−10
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year


extra_regressors_additive
1400
1300
1200
1100
1000
900
2021-01-03 2021-01-17 2021-01-31 2021-02-14 2021-02-28 2021-03-14 2021-03-28 2021-04-11 2021-04-25
ds

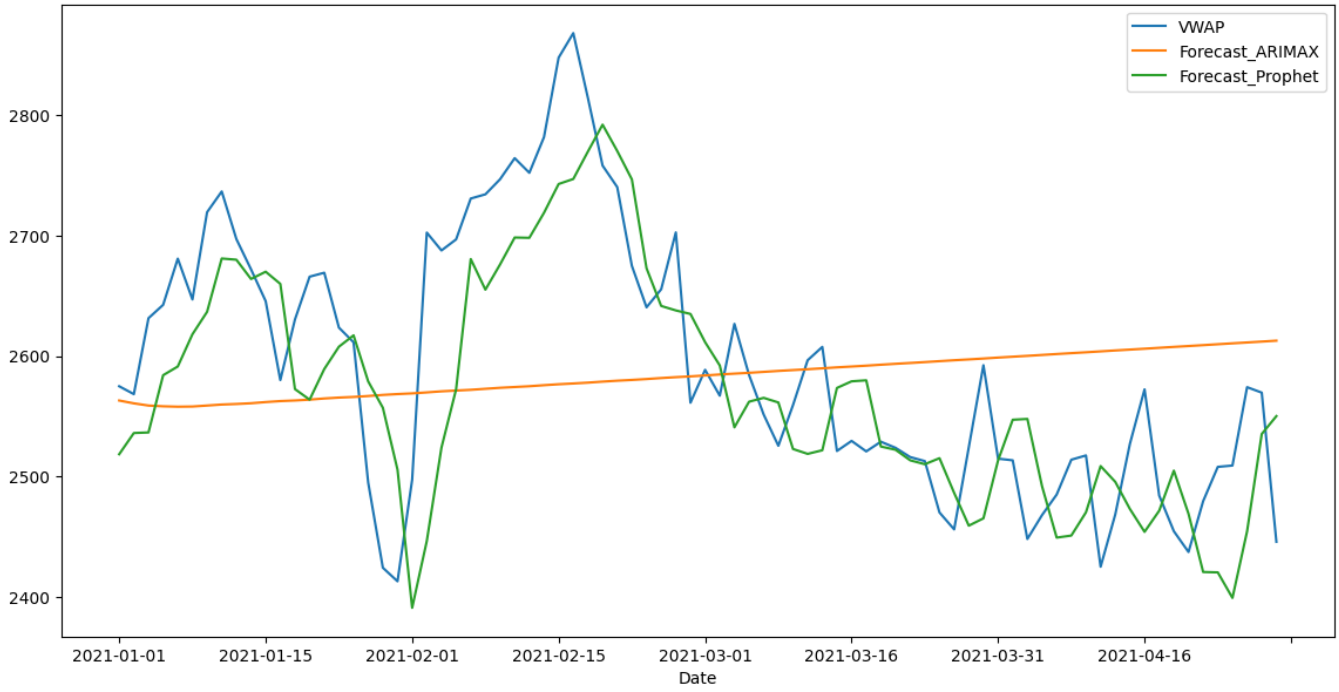

trend
1347.5
1347.0
1346.5
1346.0
1345.5
2021-01-03 2021-01-17 2021-01-31 2021-02-14 2021-02-28 2021-03-14 2021-03-28 2021-04-11 2021-04-25
ds


weekly
150
100
50
0
−50
−100
−150


Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week
yearly
10
5
0
−5
−10
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
extra_regressors_additive
1400
1300
1200
1100
1000
900
2021-01-03
2021-01-17
2021-01-31
2021-02-14
2021-02-28
2021-03-14
2021-03-28
2021-04-11
2021-04-25
ds

```
df_valid[["VWAP", "Forecast_ARIMAX", "Forecast_Prophet"]].plot(figsize=(14, 7))
```

```
<Axes: xlabel='Date'>
```




```
print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid
print("RMSE of Prophet:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.For
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.VWAP, df_valid.Forec
print("MAE of Prophet:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_Pr
```

```
RMSE of Auto ARIMAX: 118.40437981485415
RMSE of Prophet: 74.16819010675829

MAE of Auto ARIMAX: 101.19843889375443
MAE of Prophet: 60.77932568054791
```

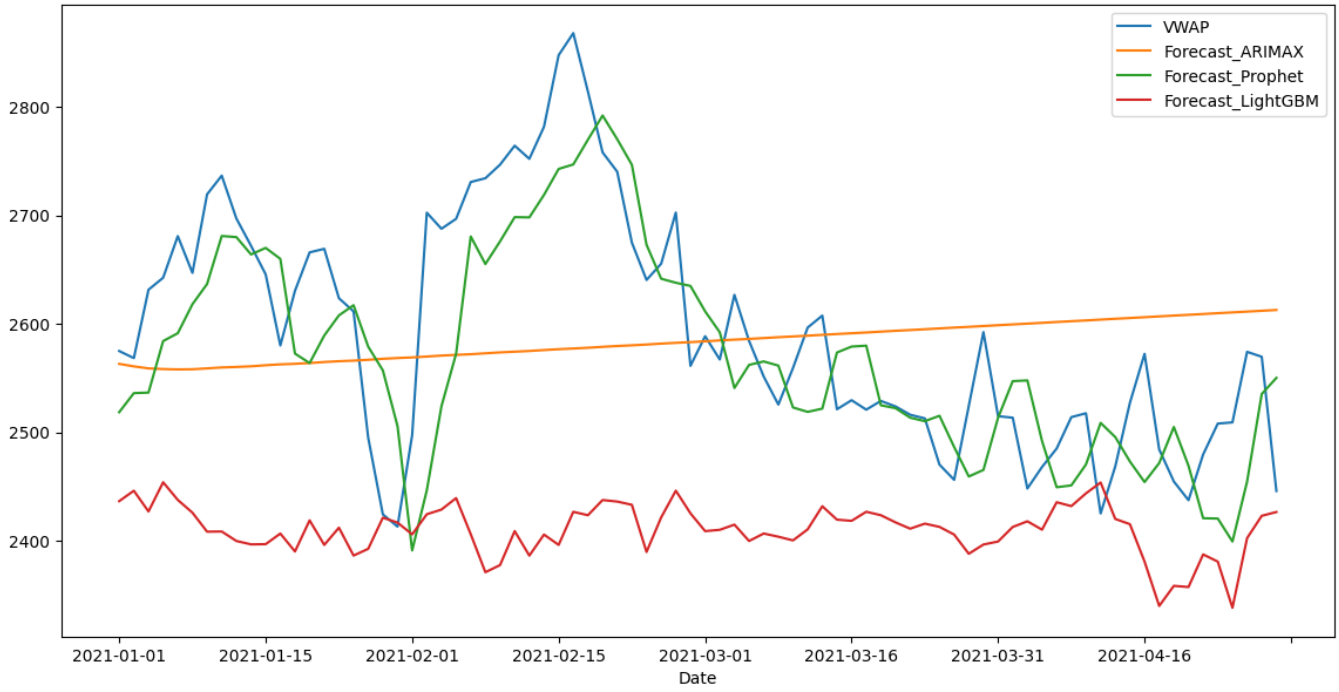
LightGBM

```
params = {"objective": "regression"}

dtrain = lgb.Dataset(df_train[exogenous_features], label=df_train.VWAP.values)
dvalid = lgb.Dataset(df_valid[exogenous_features])

model_lgb = lgb.train(params, train_set=dtrain)

forecast = model_lgb.predict(df_valid[exogenous_features])
df_valid["Forecast_LightGBM"] = forecast
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of t
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7756
[LightGBM] [Info] Number of data points in the train set: 2376, number of u
[LightGBM] [Info] Start training from score 1345.451970
<ipython-input-49-307b1df82ff0>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  df_valid["Forecast_LightGBM"] = forecast
```

```
df_valid[["VWAP", "Forecast_ARIMAX", "Forecast_Prophet", "Forecast_LightGBM"]].p
```

```
<Axes: xlabel='Date'>
```



```
print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid
print("RMSE of Prophet:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.For
print("RMSE of LightGBM:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Fo
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.VWAP, df_valid.Forec
print("MAE of Prophet:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_Pr
print("MAE of LightGBM:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_L
```

```
RMSE of Auto ARIMAX: 118.40437981485415
RMSE of Prophet: 74.16819010675829
RMSE of LightGBM: 212.53225653628058

MAE of Auto ARIMAX: 101.19843889375443
MAE of Prophet: 60.77932568054791
MAE of LightGBM: 184.2514020914423
```

```
# Calculate RMSE and MAE
rmse_auto_arimax = np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forecast_A
rmse_prophet = np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forecast_Proph
rmse_lightgbm = np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forecast_Ligh

mae_auto_arimax = mean_absolute_error(df_valid.VWAP, df_valid.Forecast_ARIMAX)
mae_prophet = mean_absolute_error(df_valid.VWAP, df_valid.Forecast_Prophet)
mae_lightgbm = mean_absolute_error(df_valid.VWAP, df_valid.Forecast_LightGBM)

# Create a dictionary
metrics_dict = {
    "RMSE of Auto ARIMAX": rmse_auto_arimax,
    "RMSE of Prophet": rmse_prophet,
    "RMSE of LightGBM": rmse_lightgbm,
    "MAE of Auto ARIMAX": mae_auto_arimax,
    "MAE of Prophet": mae_prophet,
    "MAE of LightGBM": mae_lightgbm
}

# Print the dictionary
print(metrics_dict)
```

```
{'RMSE of Auto ARIMAX': 118.40437981485415, 'RMSE of Prophet': 74.168190106
```