Stock Market Time Series Data Analysis with Auto ARIMAX,Prophet and LightGBM

In [2]:
```
import numpy as np
import pandas as pd
import lightgbm as lgb

from fbprophet import Prophet
from matplotlib import pyplot as plt
from pmdarima import auto_arima
from sklearn.metrics import mean_absolute_error, mean_squared_error

myfavouritenumber = 13
seed = myfavouritenumber
np.random.seed(seed)
```

Nifty-50 Stock Market Data

The dataset used is stock market data of the Nifty-50 index from NSE (National Stock Exchange) India over the last 20 years (2000 - 2019)

The historic **VWAP (Volume Weighted Average Price)** is the target variable to predict. VWAP is a trading benchmark used by traders that gives the average price the stock has traded at throughout the day, based on both volume and price.

Data Preparation

Reading the market data of BAJAJFINSV stock and preparing a training dataset and validation dataset.

In [3]:
```
df = pd.read_csv("/kaggle/input/nifty50-stock-market-data/BAJAJFINSV.csv")
df.set_index("Date", drop=False, inplace=True)
df.head()
```

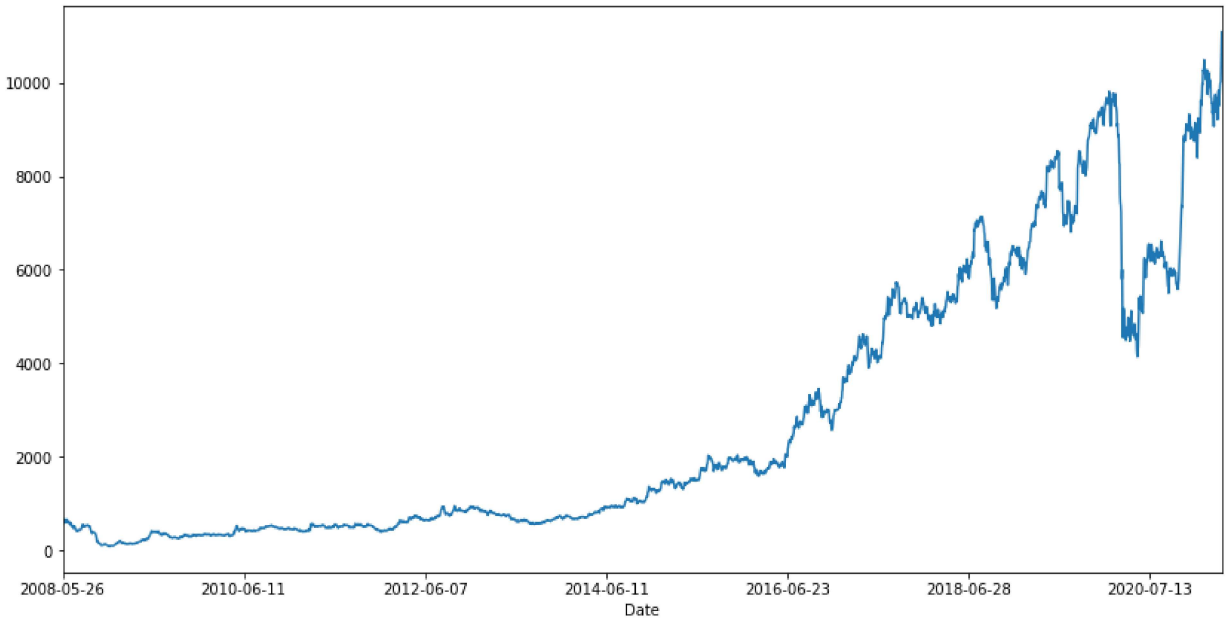
Out[3]:

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	
Date												
2008-05-26	2008-05-26	BAJAJFINSV	EQ	2101.05	600.00	619.00	501.0	505.1	509.10	548.85	3145446	1.726
2008-05-27	2008-05-27	BAJAJFINSV	EQ	509.10	505.00	610.95	491.1	564.0	554.65	572.15	4349144	2.488
2008-05-28	2008-05-28	BAJAJFINSV	EQ	554.65	564.00	665.60	564.0	643.0	640.95	618.37	4588759	2.837
2008-05-29	2008-05-29	BAJAJFINSV	EQ	640.95	656.65	703.00	608.0	634.5	632.40	659.60	4522302	2.982
2008-05-30	2008-05-30	BAJAJFINSV	EQ	632.40	642.40	668.00	588.3	647.0	644.00	636.41	3057669	1.945

Plotting the target variable **VWAP** over time

In [4]:
```
df.VWAP.plot(figsize=(14, 7))
```

Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7a60d782f9e8>
```



Feature Engineering

Almost every time series problem requires the incorporation of external features or internal feature engineering to assist the model.

Let's include some basic features, such as lag values of the available numeric features, which are widely utilized in time series problems. Since our goal is to predict the stock price for a given day, we cannot use the feature values from the same day, as they won't be available during

actual inference. Instead, we need to rely on statistics, such as the mean and standard deviation of their lagged values.

We will employ three sets of lagged values: one from the previous day, another looking back seven days, and a third looking back 30 days, serving as proxies for last week and last month metrics.

In [5]:
```
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()
```

Out[5]:

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	...	Turnover_r
Date												
2008-05-26	2008-05-26	BAJAJFINSV	EQ	2101.05	600.00	619.00	501.0	505.1	509.10	548.85	...	9.3
2008-05-27	2008-05-27	BAJAJFINSV	EQ	509.10	505.00	610.95	491.1	564.0	554.65	572.15	...	1.7
2008-05-28	2008-05-28	BAJAJFINSV	EQ	554.65	564.00	665.60	564.0	643.0	640.95	618.37	...	2.1
2008-05-29	2008-05-29	BAJAJFINSV	EQ	640.95	656.65	703.00	608.0	634.5	632.40	659.60	...	2.3
2008-05-30	2008-05-30	BAJAJFINSV	EQ	632.40	642.40	668.00	588.3	647.0	644.00	636.41	...	2.5

5 rows × 45 columns

When using boosting models, it is highly beneficial to incorporate datetime features such as hour, day, and month, where applicable. These features provide the model with essential information about the temporal aspect of the data. While time series models do not explicitly demand the inclusion of these features, we will include them in this notebook for the sake of ensuring that all models are compared using the exact same set of features.

In [6]:
```
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()
```

Out[6]:

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	...	Trades_me
Date												
2008-05-26	2008-05-26	BAJAJFINSV	EQ	2101.05	600.00	619.00	501.0	505.1	509.10	548.85	...	20805
2008-05-27	2008-05-27	BAJAJFINSV	EQ	509.10	505.00	610.95	491.1	564.0	554.65	572.15	...	20805
2008-05-28	2008-05-28	BAJAJFINSV	EQ	554.65	564.00	665.60	564.0	643.0	640.95	618.37	...	20805
2008-05-29	2008-05-29	BAJAJFINSV	EQ	640.95	656.65	703.00	608.0	634.5	632.40	659.60	...	20805
2008-05-30	2008-05-30	BAJAJFINSV	EQ	632.40	642.40	668.00	588.3	647.0	644.00	636.41	...	20805

5 rows × 49 columns

Splitting the data into train and validation along with features.

- **train:** Data from 26th May, 2008 to 31st December, 2018.
- **valid:** Data from 1st January, 2019 to 31st December, 2019.

In [7]:
```
df_train = df[df.Date < "2019"]
df_valid = df[df.Date >= "2019"]

exogenous_features = ["High_mean_lag3", "High_std_lag3", "Low_mean_lag3", "Low_std_lag
                      "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_std_lag
                      "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", "Low_std_
                      "Volume_mean_lag30", "Volume_std_lag30", "Turnover_mean_lag30",
                      "Turnover_std_lag30", "Trades_mean_lag30", "Trades_std_lag30",
                      "month", "week", "day", "day_of_week"]
```

The additional features supplied to time series problems are called exogenous regressors.

Auto ARIMAX

ARIMA (AutoRegressive Integrated Moving Average) models explain a given time series based on its historical values, including its own lags and lagged forecast errors. This equation allows us to forecast future values.

ARIMA models require specific input parameters: 'p' for the AR(p) component, 'q' for the MA(q) component, and 'd' for the I(d) component. Fortunately, there is an automated process for selecting these parameters known as Auto ARIMA.

When exogenous regressors are employed alongside ARIMA, the model is commonly referred to as ARIMAX.

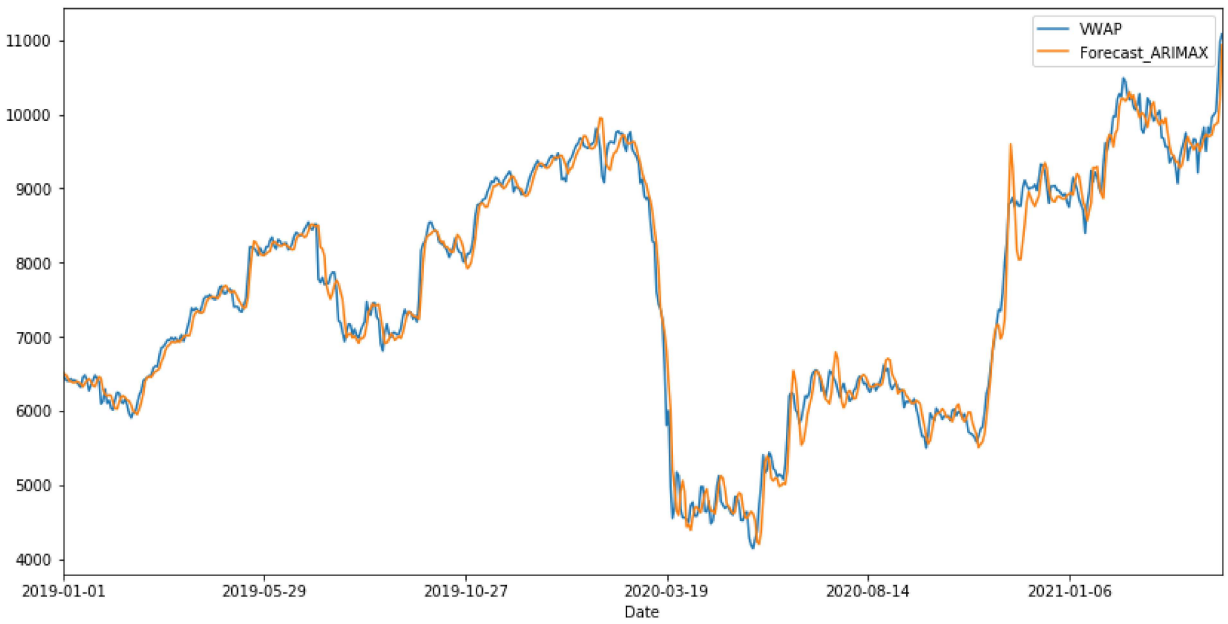
In [8]:
```
model = auto_arima(df_train.VWAP, exogenous=df_train[exogenous_features], trace=True,
model.fit(df_train.VWAP, exogenous=df_train[exogenous_features])

forecast = model.predict(n_periods=len(df_valid), exogenous=df_valid[exogenous_feature
df_valid["Forecast_ARIMAX"] = forecast
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=29826.093, Time=6.48 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=29811.894, Time=4.62 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=29683.281, Time=4.34 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=30130.952, Time=6.40 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=47844.587, Time=4.17 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=29792.143, Time=5.93 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=29883.595, Time=5.41 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=29821.510, Time=5.72 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=29680.564, Time=4.23 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=29790.393, Time=6.38 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=29881.485, Time=5.24 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=30128.815, Time=5.19 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=29819.700, Time=5.44 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0]
Total fit time: 69.607 seconds
```

```
/opt/conda/lib/python3.6/site-packages/statsmodels/tsa/base/tsa_model.py:379: ValueWa
rning:

No supported index is available. Prediction results will be given with an integer ind
ex beginning at `start`.

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: SettingWithCopyWarnin
g:


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: http://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy
```

In [9]:
```
df_valid[["VWAP", "Forecast_ARIMAX"]].plot(figsize=(14, 7))
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7a60cc1399e8>
```



In [10]:
```
print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forec
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_AR
```

```
RMSE of Auto ARIMAX: 224.32773138505723

MAE of Auto ARIMAX: 160.22216424342696
```

The Auto ARIMAX model seems to do a fairly good job in predicting the stock price given data till the previous day.

Facebook Prophet

Prophet is an open-source time series model developed by Facebook and released in early 2017. As described on its homepage:

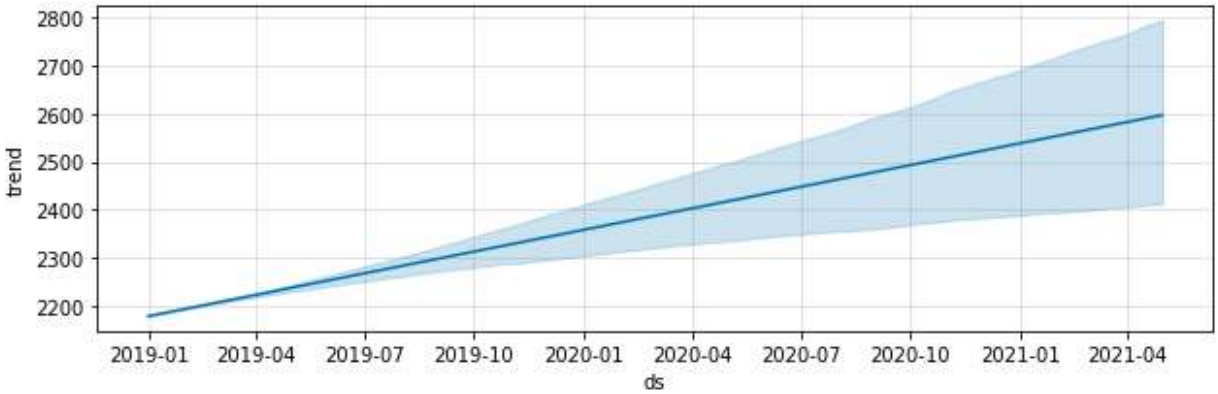
Prophet is a procedure for forecasting time series data based on an additive model. It fits non-linear trends with yearly, weekly, and daily seasonality, along with accounting for holiday effects. Prophet excels with time series data that exhibit strong seasonal patterns and have several seasons of historical data. It demonstrates robustness when handling missing data, shifts in trends, and outliers, making it a valuable tool for time series forecasting.

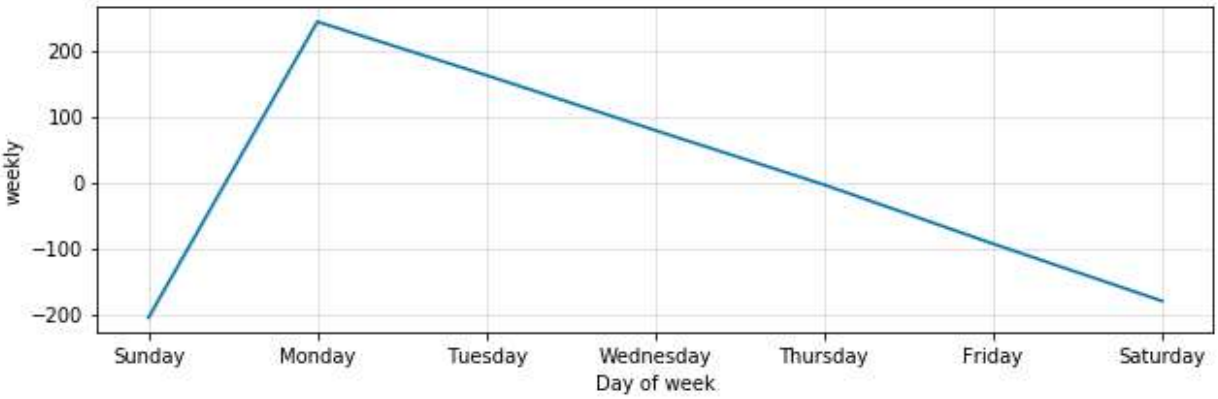
In [11]:
```
model_fbp = Prophet()
for feature in exogenous_features:
    model_fbp.add_regressor(feature)

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

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

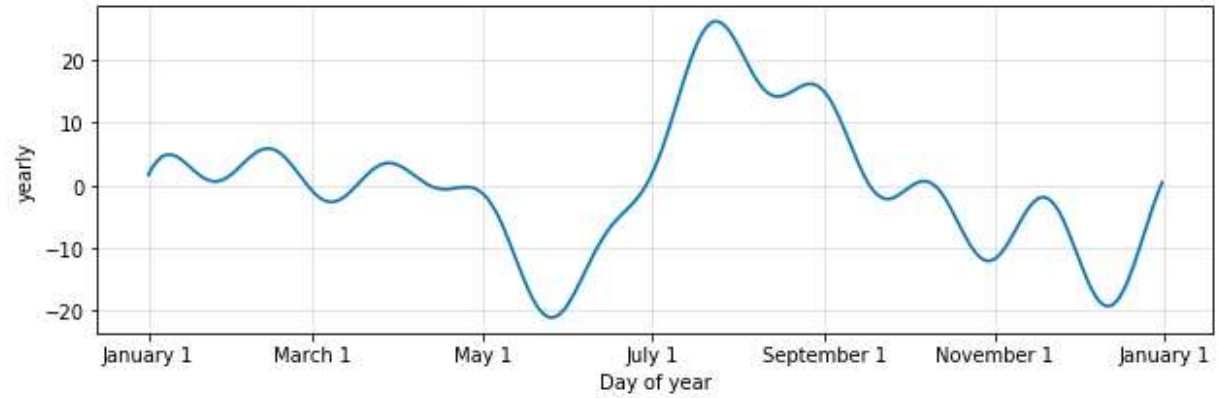
```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:8: SettingWithCopyWarnin
g:


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: http://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy
```

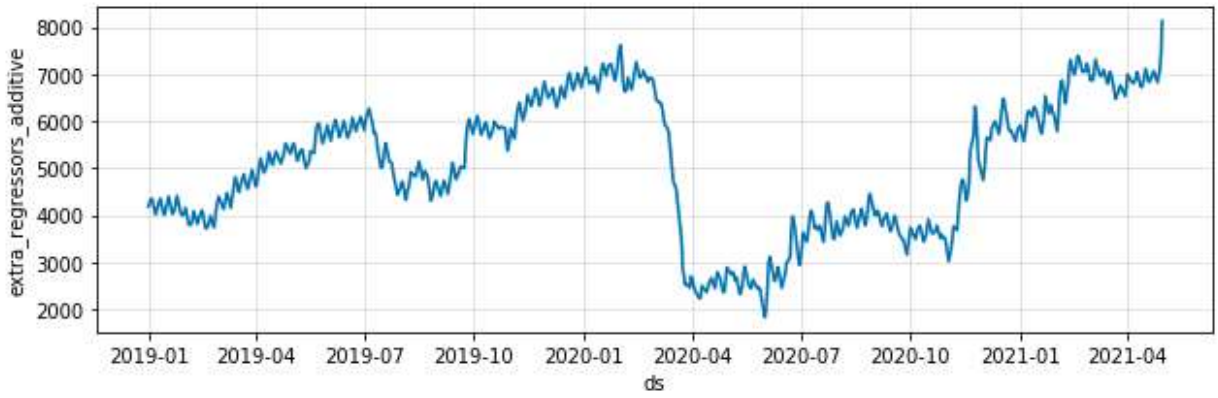
In [12]:
```
model_fbp.plot_components(forecast)
```

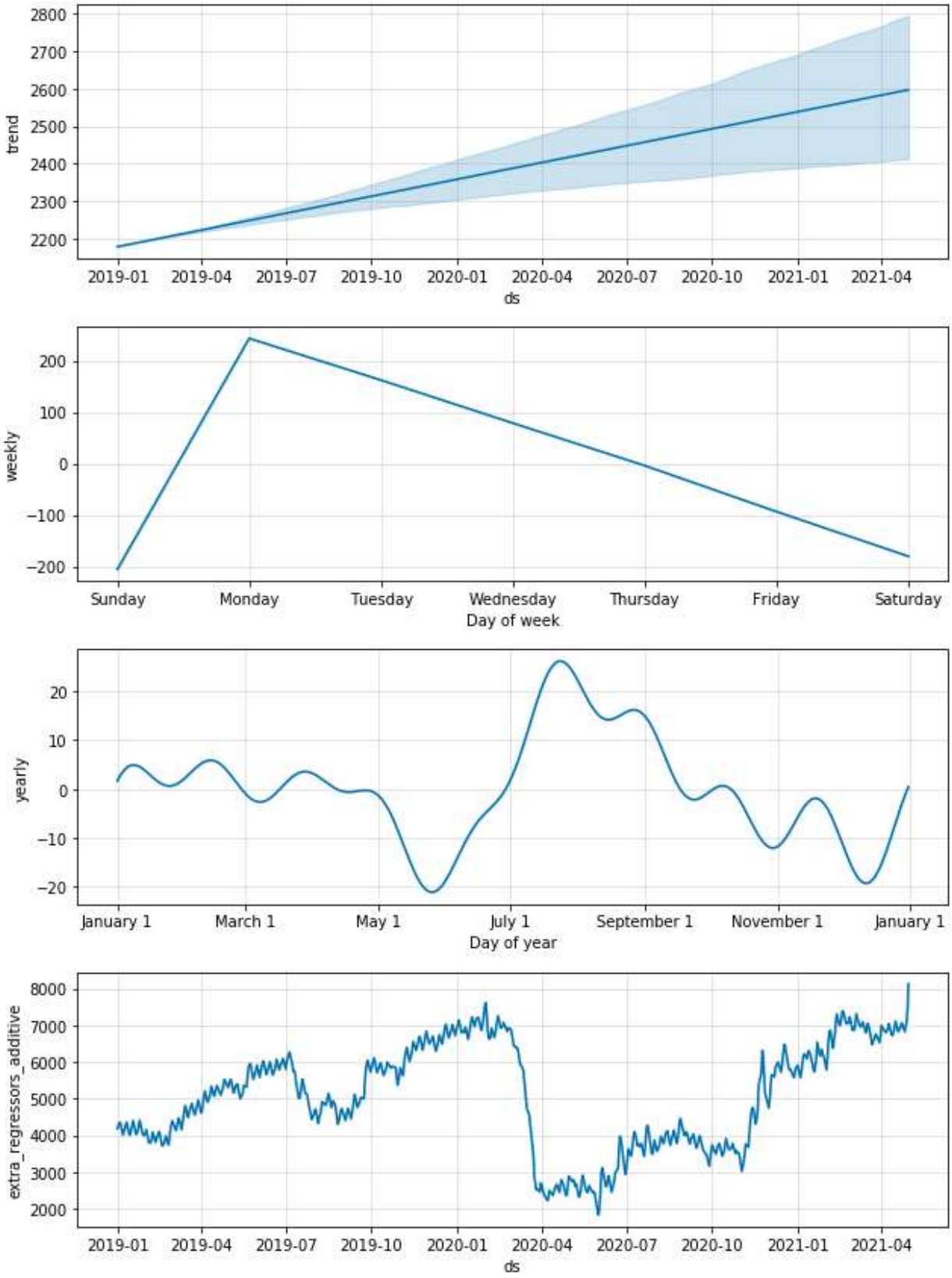
Out[12]:


trend
2800
2700
2600
2500
2400
2300
2200
2019-01
2019-04
2019-07
2019-10
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
ds


weekly
200
100
0
−100
−200
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week

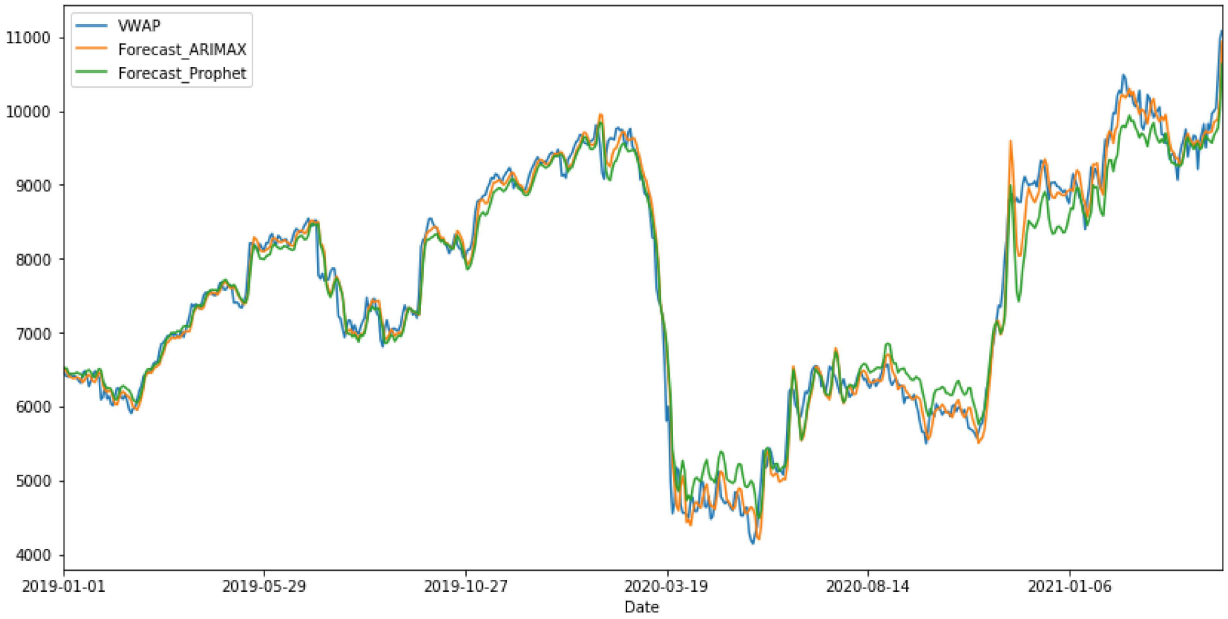

yearly
20
10
0
−10
−20
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year


extra_regressors_additive
8000
7000
6000
5000
4000
3000
2000
2019-01
2019-04
2019-07
2019-10
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
ds



In [13]:
```
df_valid[["VWAP", "Forecast_ARIMAX", "Forecast_Prophet"]].plot(figsize=(14, 7))
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7a60cc007a58>
```



In [14]:
```
print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forec
print("RMSE of Prophet:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forecast_
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_AR
print("MAE of Prophet:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_Prophet)
```

```
RMSE of Auto ARIMAX: 224.32773138505723
RMSE of Prophet: 314.9590544138107

MAE of Auto ARIMAX: 160.22216424342696
MAE of Prophet: 231.8030739337764
```

Auto ARIMAX performs better than Prophet in this case.

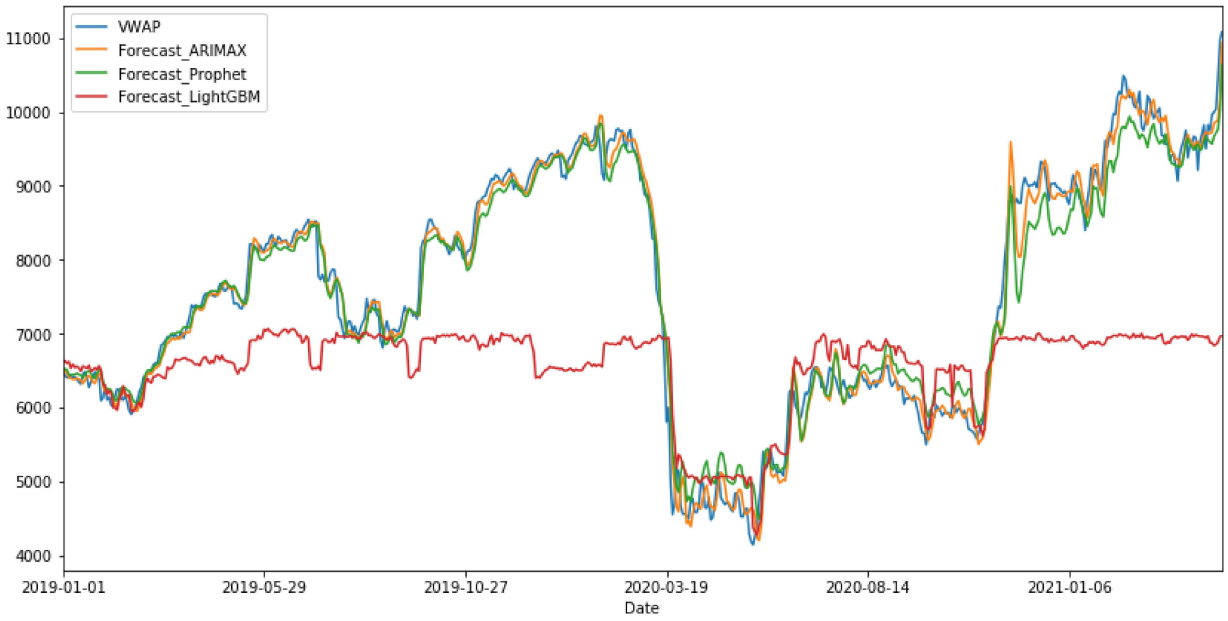
LightGBM

In [15]:
```
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
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:9: SettingWithCopyWarnin
g:


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: http://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy
```

In [16]:
```
df_valid[["VWAP", "Forecast_ARIMAX", "Forecast_Prophet", "Forecast_LightGBM"]].plot(fi
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7a60cc1d1470>



In [17]:
```
print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forec
print("RMSE of Prophet:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forecast_
print("RMSE of LightGBM:", np.sqrt(mean_squared_error(df_valid.VWAP, df_valid.Forecast
print("\nMAE of Auto ARIMAX:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_AR
print("MAE of Prophet:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_Prophet)
print("MAE of LightGBM:", mean_absolute_error(df_valid.VWAP, df_valid.Forecast_LightGB
```

```
RMSE of Auto ARIMAX: 224.32773138505723
RMSE of Prophet: 314.9590544138107
RMSE of LightGBM: 1612.3693263639284

MAE of Auto ARIMAX: 160.22216424342696
MAE of Prophet: 231.8030739337764
MAE of LightGBM: 1238.612339162908
```

LightGBM performs poorly in this context, which underscores a crucial aspect of using boosting models for time series analysis. It's essential to remember that boosting models are constrained to predict within the range of target values present in the training data. In our case, the maximum price value in the training data is around 7100, so LightGBM is incapable of predicting values beyond this threshold.

However, this raises the question: why are boosting methods still so popular? Boosting methods tend to struggle only in cases where the trend component is exceptionally strong. There are numerous use cases where the trend is weak, and the expected forecasts fall within the range of values from the past. For example, stock prices often exhibit strong trend components, especially when observed over the course of several years.

Conclusions

Auto ARIMAX is a great baseline model, but newer algorithms like Facebook's Prophet are continually advancing in power and sophistication. Don't hesitate to explore and experiment with these new techniques.

Establishing an appropriate validation framework is of paramount importance. It empowers you to experiment with various models and make objective comparisons among them.

Lag-based features are invaluable for extracting trend information from time series data. Rolling statistics provide a common method for generating such features.

Exogenous regressors play a crucial role in incorporating external information into time series models and are often essential for accurate forecasting.

Boosting models, such as LightGBM, are limited to predicting within the range of target variable values present in the training data. They do not extrapolate when faced with a strong trend.

A common approach to building solutions involves transforming a time series into a stationary form before modeling, which can significantly enhance the quality of results.