2j8x112qd

March 3, 2024

[1]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in c:\users\vince\anaconda3\lib\site-
packages (1.1.5)
Requirement already satisfied: cmdstanpy>=1.0.4 in
c:\users\vince\anaconda3\lib\site-packages (from prophet) (1.2.1)
Requirement already satisfied: numpy>=1.15.4 in
c:\users\vince\anaconda3\lib\site-packages (from prophet) (1.24.3)
Requirement already satisfied: matplotlib>=2.0.0 in
c:\users\vince\anaconda3\lib\site-packages (from prophet) (3.7.1)
Requirement already satisfied: pandas>=1.0.4 in
c:\users\vince\anaconda3\lib\site-packages (from prophet) (1.5.3)
Requirement already satisfied: holidays>=0.25 in
c:\users\vince\anaconda3\lib\site-packages (from prophet) (0.43)
Requirement already satisfied: tqdm>=4.36.1 in
c:\users\vince\anaconda3\lib\site-packages (from prophet) (4.65.0)
Requirement already satisfied: importlib-resources in
c:\users\vince\anaconda3\lib\site-packages (from prophet) (6.1.2)
Requirement already satisfied: stanio~=0.3.0 in
c:\users\vince\anaconda3\lib\site-packages (from cmdstanpy>=1.0.4->prophet)
(0.3.0)
Requirement already satisfied: python-dateutil in
c:\users\vince\anaconda3\lib\site-packages (from holidays>=0.25->prophet)
(2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\vince\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(1.0.5)
Requirement already satisfied: cycler>=0.10 in
c:\users\vince\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\vince\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\vince\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(1.4.4)
Requirement already satisfied: packaging>=20.0 in
c:\users\vince\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
```

```
(23.0)
Requirement already satisfied: pillow>=6.2.0 in
c:\users\vince\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\vince\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(3.0.9)
Requirement already satisfied: pytz>=2020.1 in
c:\users\vince\anaconda3\lib\site-packages (from pandas>=1.0.4->prophet)
(2022.7)
Requirement already satisfied: colorama in c:\users\vince\anaconda3\lib\site-
packages (from tqdm>=4.36.1->prophet) (0.4.6)
Requirement already satisfied: six>=1.5 in c:\users\vince\anaconda3\lib\site-
packages (from python-dateutil->holidays>=0.25->prophet) (1.16.0)
```

[2]:
```
# Import Libraries and packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
import datetime
import math
import warnings
import prophet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.models import Sequential

warnings.filterwarnings("ignore")
```

```
WARNING:tensorflow:From C:\Users\vince\anaconda3\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

[3]:
```
# import and display csv file
df = pd.read_csv('AAPL.csv')
df.head()
```

[3]:
```
         Date      Open      High       Low     Close  Adj Close     Volume
0  1980-12-12  0.128348  0.128906  0.128348  0.128348   0.100323  469033600
1  1980-12-15  0.122210  0.122210  0.121652  0.121652   0.095089  175884800
2  1980-12-16  0.113281  0.113281  0.112723  0.112723   0.088110  105728000
3  1980-12-17  0.115513  0.116071  0.115513  0.115513   0.090291   86441600
4  1980-12-18  0.118862  0.119420  0.118862  0.118862   0.092908   73449600
```

[4]:
```
# display bottom rows of csv file
df.tail()
```

[4]:
```
             Date        Open        High         Low       Close   Adj Close  \
10404  2022-03-18  160.509995  164.479996  159.759995  163.979996  163.979996
10405  2022-03-21  163.509995  166.350006  163.009995  165.380005  165.380005
10406  2022-03-22  165.509995  169.419998  164.910004  168.820007  168.820007
10407  2022-03-23  167.990005  172.639999  167.649994  170.210007  170.210007
10408  2022-03-24  171.059998  174.139999  170.210007  174.070007  174.070007

          Volume
10404  123351200
10405   95811400
10406   81532000
10407   98062700
10408   90018700
```

[5]:
```
# checking shape of dataframe
df.shape
```

[5]: (10409, 7)

[6]:
```
#checking data types of data frame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10409 entries, 0 to 10408
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       10409 non-null  object
 1   Open       10409 non-null  float64
 2   High       10409 non-null  float64
 3   Low        10409 non-null  float64
 4   Close      10409 non-null  float64
 5   Adj Close  10409 non-null  float64
 6   Volume     10409 non-null  int64
dtypes: float64(5), int64(1), object(1)
memory usage: 569.4+ KB
```

[7]:
```python
#renaming columns to python casing.
col_head = {
    'Date': 'date',
    'Open': 'open',
    'High': 'high',
    'Close': 'close',
    'Low': 'low',
    'Adj Close': 'adj_close',
    'Volume': 'volume'}
df.rename(columns=col_head, inplace=True)
```

[8]:
```python
# checking for NULL values
print("Null Values:\n", df.isna().sum())

# dropping null values
df = df.dropna()

# verifying null values were dropped
print("Null Values after dropping:\n", df.isna().sum())
```

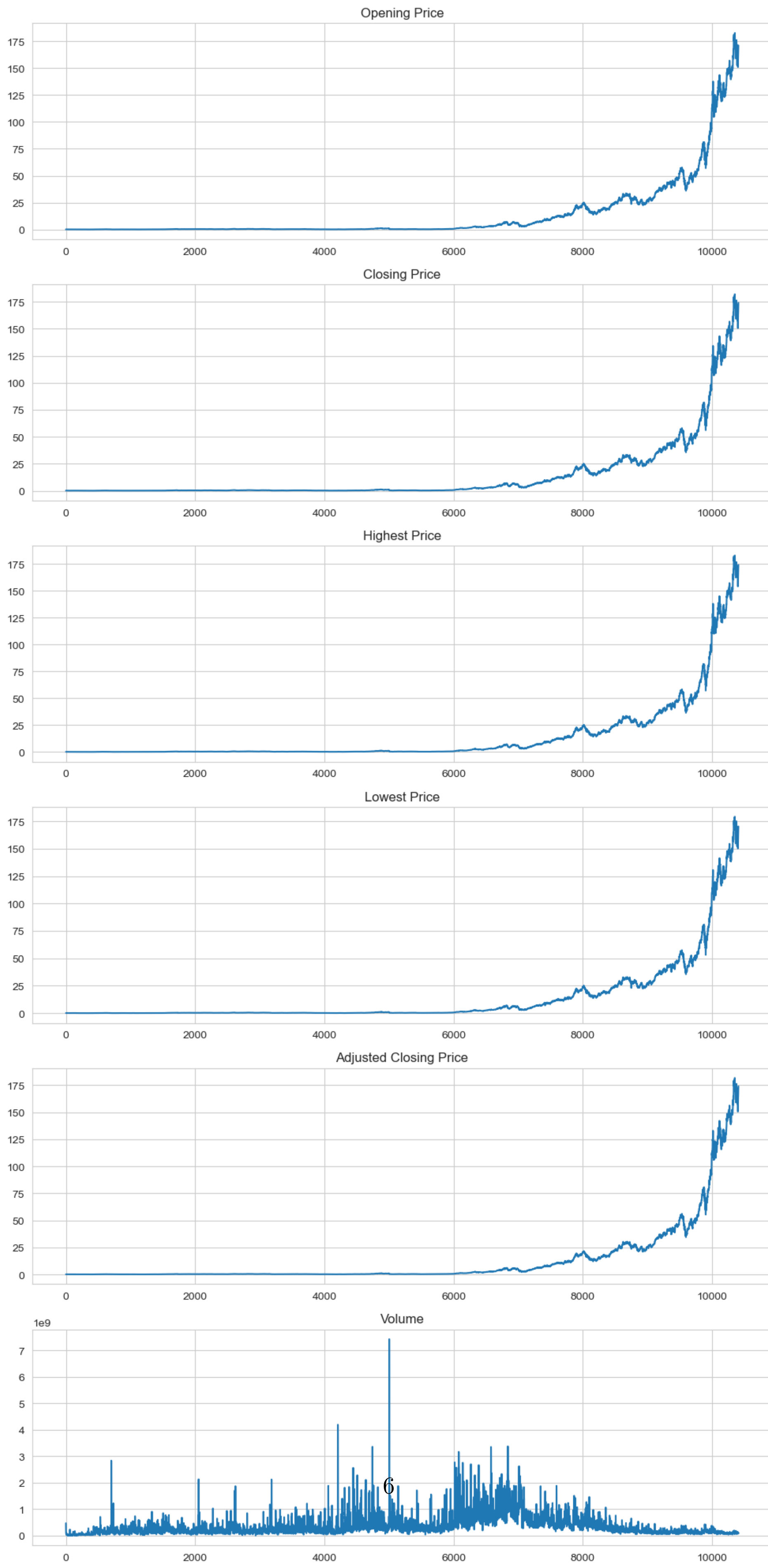
```
Null Values:
 date         0
open         0
high         0
low          0
close        0
adj_close    0
volume       0
dtype: int64
Null Values after dropping:
 date         0
open         0
high         0
low          0
close        0
adj_close    0
volume       0
dtype: int64
```

[9]:
```python
# checking for duplicate values
print("Duplicate Values:\n", df.duplicated().sum())

# dropping duplicate values
df = df.drop_duplicates()

# verifying duplicates were dropped
print("Duplicate Values after dropping:\n", df.duplicated().sum())
```

```
Duplicate Values:
 0
Duplicate Values after dropping:
 0
```

[10]:

```python
# Create a figure and a grid of subplots
fig, axs = plt.subplots(nrows=6, ncols=1, figsize=(10, 20))

# plot each trace on the corresponding subplot
axs[0].plot(df.index, df["open"])
axs[0].set_title("Opening Price")
axs[1].plot(df.index, df["close"])
axs[1].set_title("Closing Price")
axs[2].plot(df.index, df["high"])
axs[2].set_title("Highest Price")
axs[3].plot(df.index, df["low"])
axs[3].set_title("Lowest Price")
axs[4].plot(df.index, df["adj_close"])
axs[4].set_title("Adjusted Closing Price")
axs[5].plot(df.index, df["volume"])
axs[5].set_title("Volume")

# Adjust layout
plt.tight_layout()

# Show the plot
```


Opening Price
Closing Price
Highest Price
Lowest Price
Adjusted Closing Price
Volume
1e9
0
2000
4000
6000
8000
10000
175
150
125
100
75
50
25

[11]:
```python
# printing summary statistics
df.describe()
```

[11]:
```
                open          high           low         close     adj_close  \
count  10409.000000  10409.000000  10409.000000  10409.000000  10409.000000
mean      13.959910     14.111936     13.809163     13.966757     13.350337
std       30.169244     30.514878     29.835055     30.191696     29.911132
min        0.049665      0.049665      0.049107      0.049107      0.038384
25%        0.281964      0.287946      0.274554      0.281250      0.234799
50%        0.468750      0.477679      0.459821      0.468750      0.386853
75%       14.217857     14.364286     14.043571     14.206071     12.188149
max      182.630005    182.940002    179.119995    182.009995    181.778397

             volume
count  1.040900e+04
mean   3.321778e+08
std    3.393344e+08
min    0.000000e+00
25%    1.247604e+08
50%    2.199680e+08
75%    4.126108e+08
max    7.421641e+09
```

[12]:
```python
# importing plotly for an interactive graph
import plotly.graph_objects as go
import plotly.express as px

# create a figure
fig = px.line(df, x=df.index, y=['open', 'high', 'low', 'close'], title='Apple␣
 ↪Stock Price 1980-2022')

# add labels
fig.update_layout(xaxis_title='Date', yaxis_title='Price (USD)')

# show the interactive plot
fig.show()
```

[13]:
```python
# calculate the moving average
moving_avg = df['close'].rolling(window=100).mean()

# create a figure
fig = go.Figure()

# add the moving average trace
```

```
fig.add_trace(go.Scatter(x=df.index, y=moving_avg, mode='lines', name='Simple␣
↪Moving Average'))

# add the closing price trace
fig.add_trace(go.Scatter(x=df.index, y=df["close"], mode='lines', name='Closing␣
↪Price'))

# update layout
fig.update_layout(title='Moving Average of Closing Price',
                  xaxis_title='Date',
                  yaxis_title='Price (USD)',
                  xaxis=dict(tickangle=-45),
                  showlegend=True)

# show the plot
fig.show()
```

[14]:
```
# create subset data frame that contains just the date and close column
df_close = df[['date','close']]
df_close.head()
```

[14]:

	date	close
0	1980-12-12	0.128348
1	1980-12-15	0.121652
2	1980-12-16	0.112723
3	1980-12-17	0.115513
4	1980-12-18	0.118862

[15]:
```
# changing datatype of date column
df_close["date"] = pd.to_datetime(pd.to_datetime(df["date"]).dt.date)
df_close.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10409 entries, 0 to 10408
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    10409 non-null  datetime64[ns]
 1   close   10409 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 244.0 KB
```

[16]:
```
# updating column names to the Prophet casing convention
df_new = df_close.rename(mapper={"date": "ds", "close": "y"}, axis="columns")
df_new.head()
```

[16]:

```
           ds         y
0 1980-12-12  0.128348
1 1980-12-15  0.121652
2 1980-12-16  0.112723
3 1980-12-17  0.115513
4 1980-12-18  0.118862
```

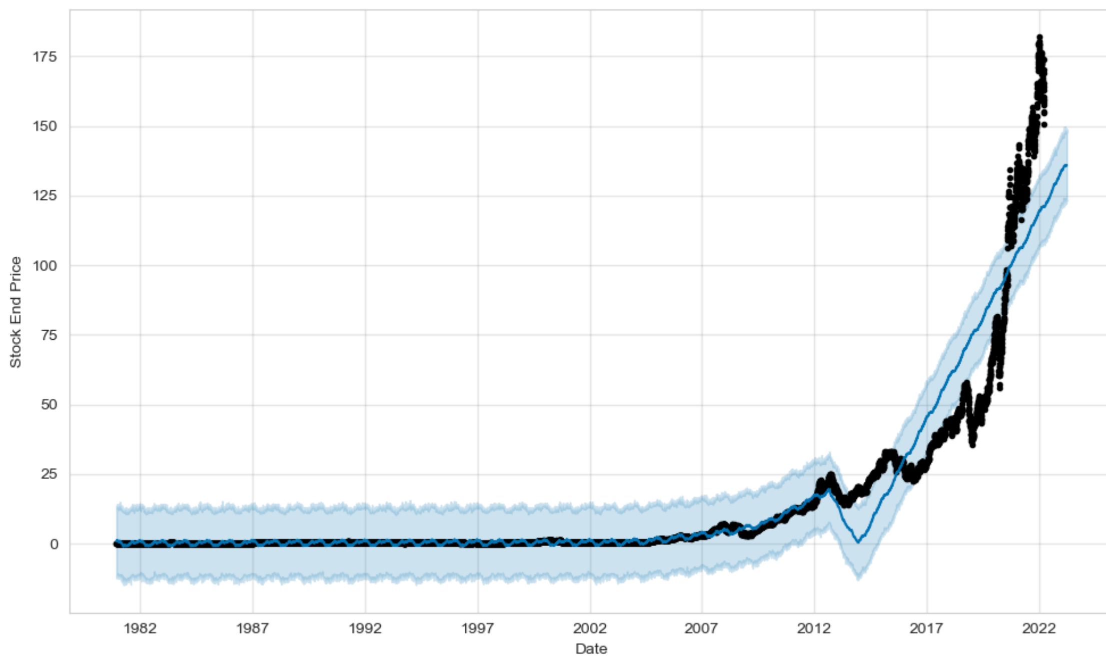
[17]:

```
# training and forecasting the model using Facebook prophet model
op = prophet.Prophet()
op.fit(df_new)
forecast = op.make_future_dataframe(periods=365)
forecast = op.predict(forecast)
```

```
17:00:34 - cmdstanpy - INFO - Chain [1] start processing
17:00:39 - cmdstanpy - INFO - Chain [1] done processing
```

[18]:

```
# visualizing forecast results
op.plot(forecast, xlabel="Date", ylabel="Stock End Price")
plt.show()
```



[19]:

```
# first forecast was poor, removing data prior to 2015
df_new['ds'] = pd.to_datetime(df_new['ds'])
df_new = df_new[df_new['ds'].dt.year >= 2015]
```

[20]:

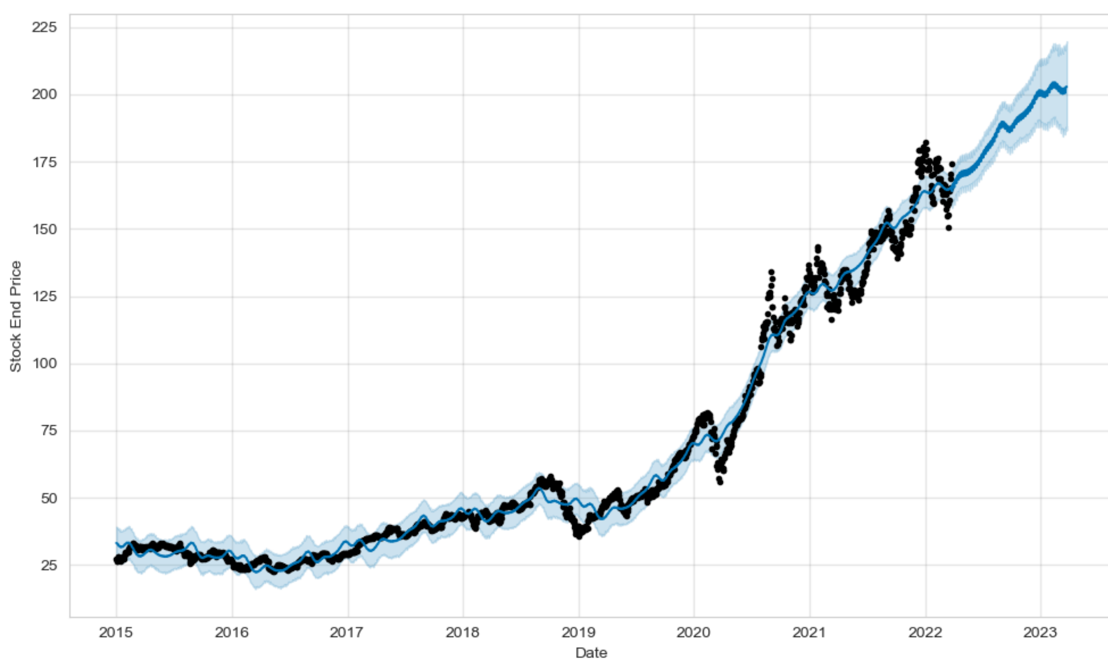
```
df_new
```

[20]:
```
                ds           y
8589   2015-01-02   27.332500
8590   2015-01-05   26.562500
8591   2015-01-06   26.565001
8592   2015-01-07   26.937500
8593   2015-01-08   27.972500
…             …          …
10404 2022-03-18  163.979996
10405 2022-03-21  165.380005
10406 2022-03-22  168.820007
10407 2022-03-23  170.210007
10408 2022-03-24  174.070007

[1820 rows x 2 columns]
```

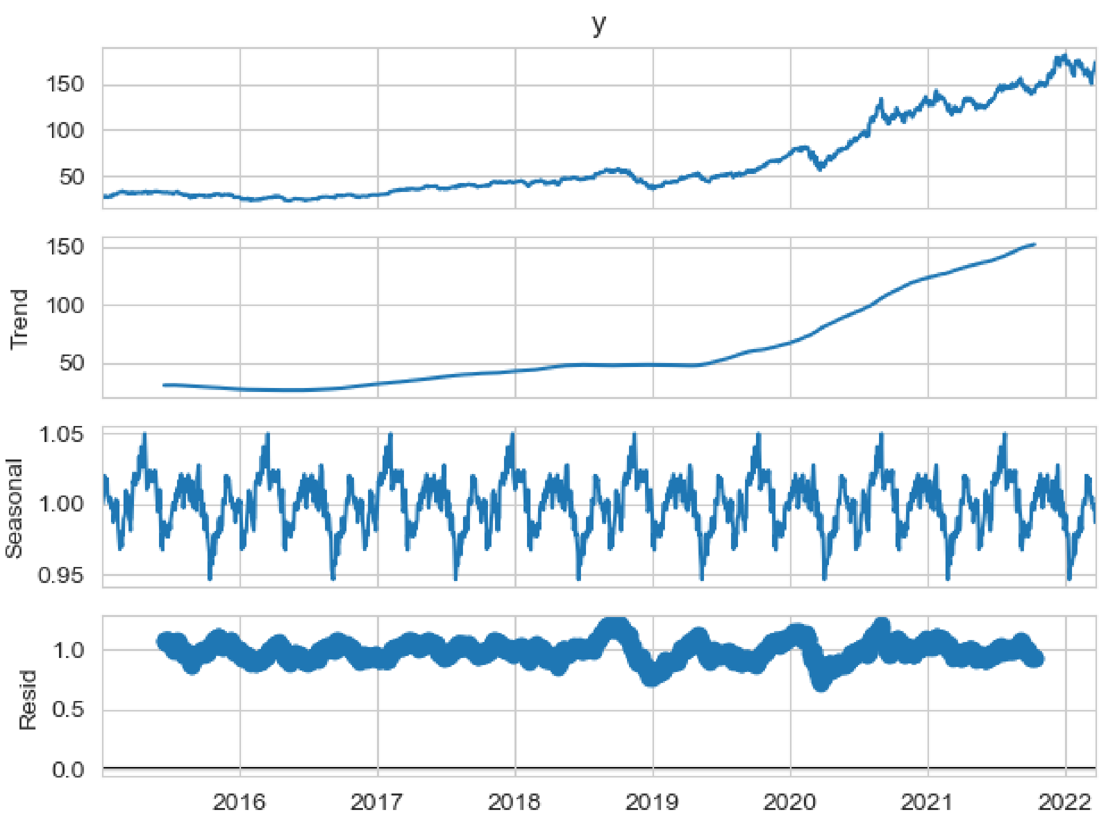
[21]:
```python
# training and forecasting the model using Facebook prophet model
op = prophet.Prophet()
op.fit(df_new)
forecast = op.make_future_dataframe(periods=365)
forecast = op.predict(forecast)
```

```
17:00:47 - cmdstanpy - INFO - Chain [1] start processing
17:00:48 - cmdstanpy - INFO - Chain [1] done processing
```

[22]:
```python
# visualizing second forecast results
op.plot(forecast, xlabel="Date", ylabel="Stock End Price")
plt.show()
```

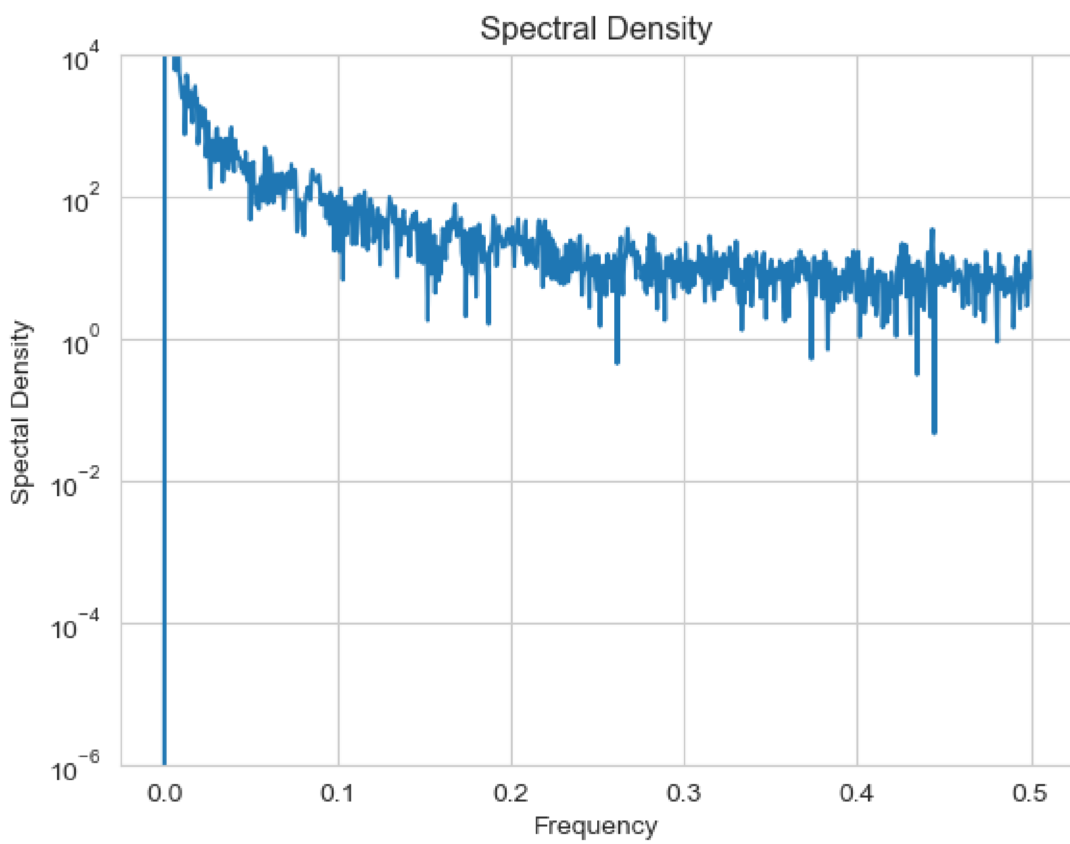


[23]:
```python
# import seasonal decompose from statsmodels library
from statsmodels.tsa.seasonal import seasonal_decompose

# index on date
df_new.set_index("ds", inplace=True)

# Decomposition
result = seasonal_decompose(df_new.y, model='multiplicative', period=225)

# Plot Trend, Seasonality, Residuals
result.plot()
plt.show()
```



[24]:
```python
import matplotlib.gridspec as gridspec

# Power spectral density
from scipy import signal
f, Pxx_den = signal.periodogram(df_new['y'])
```

```
plt.semilogy(f, Pxx_den)
plt.ylim([1e-6, 1e4])
plt.title('Spectral Density')
plt.xlabel('Frequency')
plt.ylabel('Spectal Density')
plt.show()
```



[39]:
```
# first forecast was poor, removing data prior to 2015
df_close = df_close[df_close['date'].dt.year >= 2015]

# printing the shape of the new model
df_close.shape
```

[39]: (1820, 2)

[40]:
```
df_close.head()
```

[40]:

	date	close
8589	2015-01-02	27.332500
8590	2015-01-05	26.562500

```
8591 2015-01-06  26.565001
8592 2015-01-07  26.937500
8593 2015-01-08  27.972500
```

[41]:
```
df_close.tail()
```

[41]:
```
            date       close
10404 2022-03-18  163.979996
10405 2022-03-21  165.380005
10406 2022-03-22  168.820007
10407 2022-03-23  170.210007
10408 2022-03-24  174.070007
```

[42]:
```
# splitting the data into training and testing sets 80/20 split
X_train = df_close[:1456]
X_test = df_close[1456:]

# print the shape of the datasets
print("X_train Shape", X_train.shape)
print("X_test Shape", X_test.shape)
```

```
X_train Shape (1456, 2)
X_test Shape (364, 2)
```

[43]:
```
# indexing dataset on date
X_train = X_train[["date", "close"]]
X_test = X_test[["date", "close"]]

X_train.set_index("date", inplace=True)
X_test.set_index("date", inplace=True)
```

[44]:
```
# creating forecast for next twelve months
fy_forecast = pd.date_range(X_test.index[-1], freq="MS", periods=12)
fy_forecast
```

[44]:
```
DatetimeIndex(['2022-04-01', '2022-05-01', '2022-06-01', '2022-07-01',
               '2022-08-01', '2022-09-01', '2022-10-01', '2022-11-01',
               '2022-12-01', '2023-01-01', '2023-02-01', '2023-03-01'],
              dtype='datetime64[ns]', freq='MS')
```

[45]:
```
# performing ADF test to check for stationarity
def define_d(data):
    adf_result = adfuller(data, autolag="AIC")
    adf_statistic = adf_result[0]
    pvalue = adf_result[1]
    critical_value = adf_result[4]["5%"]
```

```
    print(f"ADF Test Statistic: {adf_statistic}")
    print(f"p-value: {pvalue}")
    print(f"Critical Value (5%): {critical_value}")

    if pvalue < 0.05:
        print("** Data is stationary, proceed to plotting ACF and PACF.**")
    else:
        print("** Data is not stationary, and needs to be differenced! **")

define_d(X_train)
```

```
ADF Test Statistic: 2.3635665531945196
p-value: 0.9989921192624683
Critical Value (5%): -2.863561857668172
** Data is not stationary, and needs to be differenced! **
```

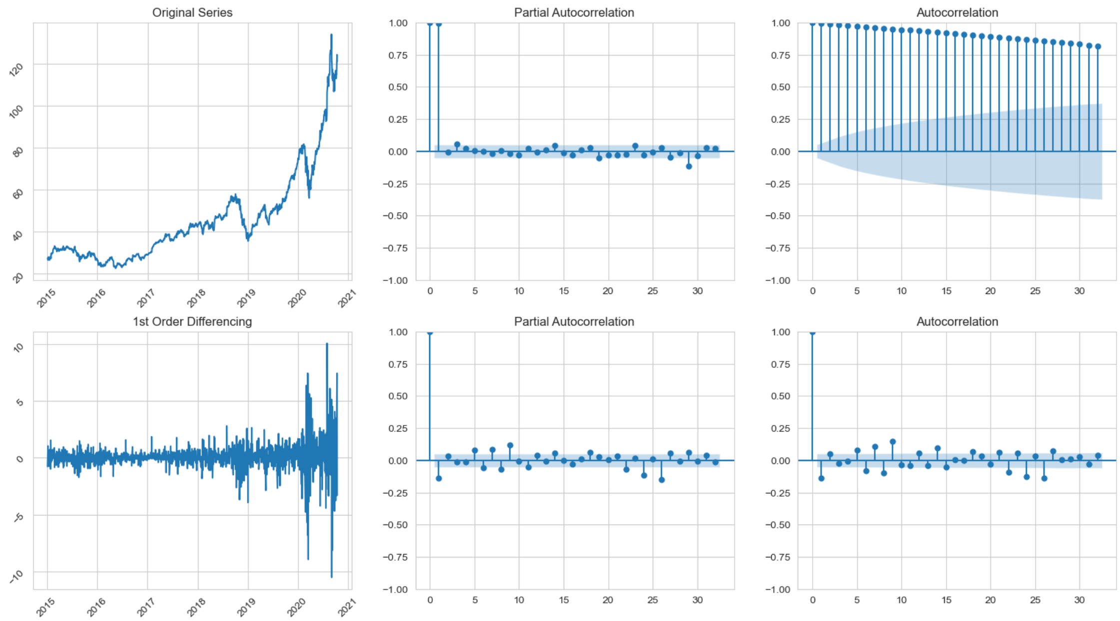
[46]:

```
# performing differencing to make data staionary
data_diff = X_train.diff()
data_diff.dropna(inplace=True)
define_d(data_diff)
```

```
ADF Test Statistic: -7.488965986870958
p-value: 4.5589820085980184e-11
Critical Value (5%): -2.863561857668172
** Data is stationary, proceed to plotting ACF and PACF.**
```

[47]:

```
# Original dataframe visualization
fig, axes = plt.subplots(2, 3)
fig.set_figheight(10)
fig.set_figwidth(19)
axes[0, 0].plot(X_train)
axes[0, 0].tick_params(labelrotation=45)
axes[0, 0].set_title("Original Series")
plot_pacf(X_train, ax=axes[0, 1])
plot_acf(X_train, ax=axes[0, 2])

# visualizing data post first differencing
axes[1, 0].plot(X_train.diff())
axes[1, 0].set_title("1st Order Differencing")
axes[1, 0].tick_params(labelrotation=45)
plot_pacf(X_train.diff().dropna(), ax=axes[1, 1])
plot_acf(X_train.diff().dropna(), ax=axes[1, 2])

plt.show();
```



[49]:
```
price_validate = df_close["close"][1456:]
price_validate
```

[49]:
```
10045    121.190002
10046    120.709999
10047    119.019997
10048    115.980003
10049    117.510002
            …
10404    163.979996
10405    165.380005
10406    168.820007
10407    170.210007
10408    174.070007
Name: close, Length: 364, dtype: float64
```

[51]:
```
# calculating the Root Mean Squared Error (RMSE)
def forecast_accuracy(forecast, actual):
    mape = (np.mean(np.abs(forecast - actual) / np.abs(actual)) * 100).round(2)
    rmse = np.sqrt(((forecast - actual) ** 2).mean())
    return {"Mean Absolute Percentage Error (%)": mape, "Root Mean Squared␣
↪Error": rmse}
```

0.0.1 Arima Model

[52]:
```
model = ARIMA(X_train)
model = model.fit()
print(model.summary())

fc = model.forecast(364)
forecast_accuracy(fc, price_validate.values)
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  close   No. Observations:                 1456
Model:                          ARIMA   Log Likelihood               -6503.753
Date:                Sat, 02 Mar 2024   AIC                          13011.507
Time:                        17:20:09   BIC                          13022.074
Sample:                             0   HQIC                         13015.449
                               - 1456
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         45.3707      0.857     52.959      0.000      43.692      47.050
sigma2       444.0009     16.194     27.417      0.000     412.260     475.741
================================================================================
===
Ljung-Box (L1) (Q):                 1441.09   Jarque-Bera (JB):
1240.27
Prob(Q):                               0.00   Prob(JB):
0.00
Heteroskedasticity (H):                3.13   Skew:
1.70
Prob(H) (two-sided):                   0.00   Kurtosis:
5.97
================================================================================
===

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

[52]:
```
{'Mean Absolute Percentage Error (%)': 67.55,
 'Root Mean Squared Error': 98.41523686774337}
```

[53]:
```
model = ARIMA(X_train, order=(1, 0, 1))
model = model.fit()
print(model.summary())

fc = model.forecast(364)
```

```
forecast_accuracy(fc, price_validate.values)
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  close   No. Observations:                 1456
Model:                 ARIMA(1, 0, 1)   Log Likelihood               -2251.763
Date:                Sat, 02 Mar 2024   AIC                           4511.525
Time:                        17:21:16   BIC                           4532.659
Sample:                             0   HQIC                          4519.410
                               - 1456
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         46.2916    193.929      0.239      0.811    -333.802     426.385
ar.L1          0.9998      0.001   1292.451      0.000       0.998       1.001
ma.L1         -0.1219      0.011    -11.080      0.000      -0.143      -0.100
sigma2         1.2840      0.014     90.019      0.000       1.256       1.312
================================================================================
===
Ljung-Box (L1) (Q):                   0.15   Jarque-Bera (JB):
26074.85
Prob(Q):                              0.70   Prob(JB):
0.00
Heteroskedasticity (H):              16.45   Skew:
-0.27
Prob(H) (two-sided):                  0.00   Kurtosis:
23.72
================================================================================
===

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

[53]:
```
{'Mean Absolute Percentage Error (%)': 15.75,
 'Root Mean Squared Error': 30.536918644529457}
```

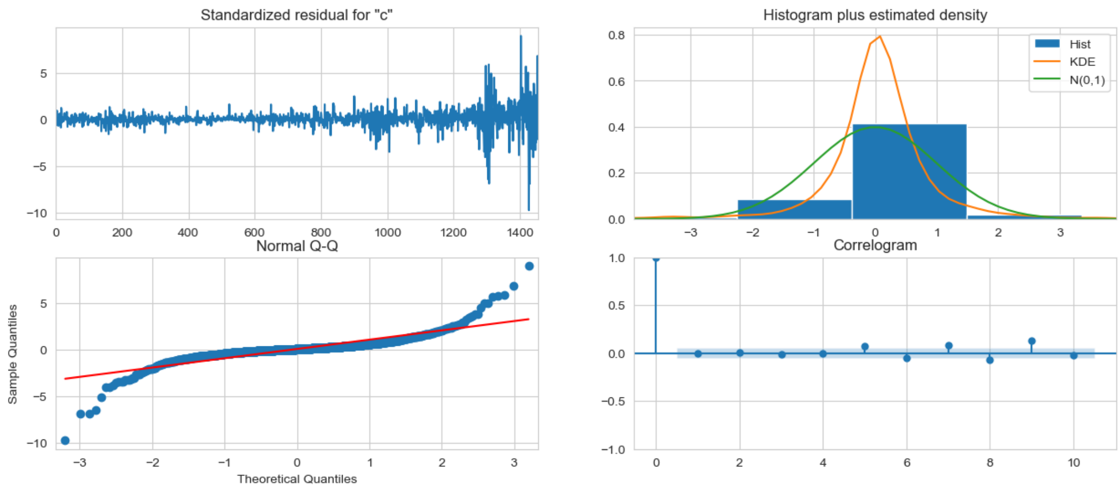
[54]:
```
model = ARIMA(X_train, order=(1, 1, 1))
model = model.fit()
print(model.summary())

fc = model.forecast(364)
forecast_accuracy(fc, price_validate.values)
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  close   No. Observations:                 1456
```

```
Model:                   ARIMA(1, 1, 1)   Log Likelihood               -2244.252
Date:                  Sat, 02 Mar 2024   AIC                           4494.504
Time:                          17:21:20   BIC                           4510.352
Sample:                               0   HQIC                          4500.417
                                 - 1456
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3705      0.067     -5.551      0.000      -0.501      -0.240
ma.L1          0.2398      0.070      3.413      0.001       0.102       0.377
sigma2         1.2802      0.014     90.495      0.000       1.252       1.308
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
26453.30
Prob(Q):                              0.94   Prob(JB):
0.00
Heteroskedasticity (H):              16.24   Skew:
-0.25
Prob(H) (two-sided):                  0.00   Kurtosis:
23.88
================================================================================
===

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

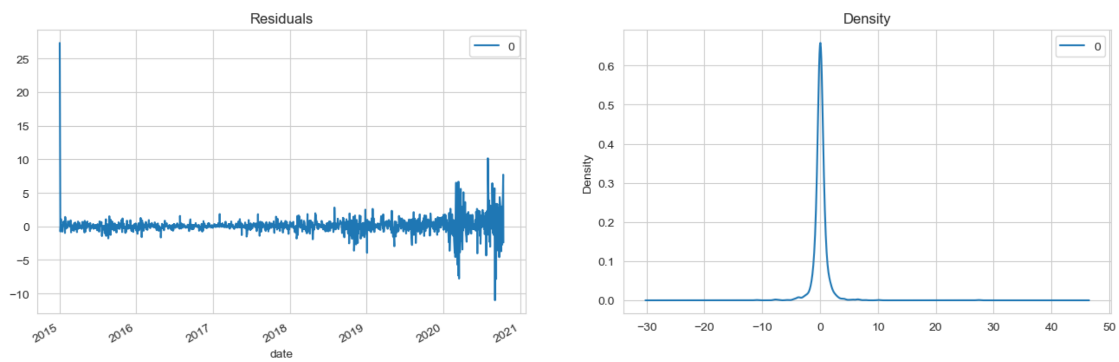
[54]:
```
{'Mean Absolute Percentage Error (%)': 13.94,
 'Root Mean Squared Error': 27.427856712866763}
```

[55]:
```
model.plot_diagnostics(figsize=(15,6))
plt.show()
```



[56]:

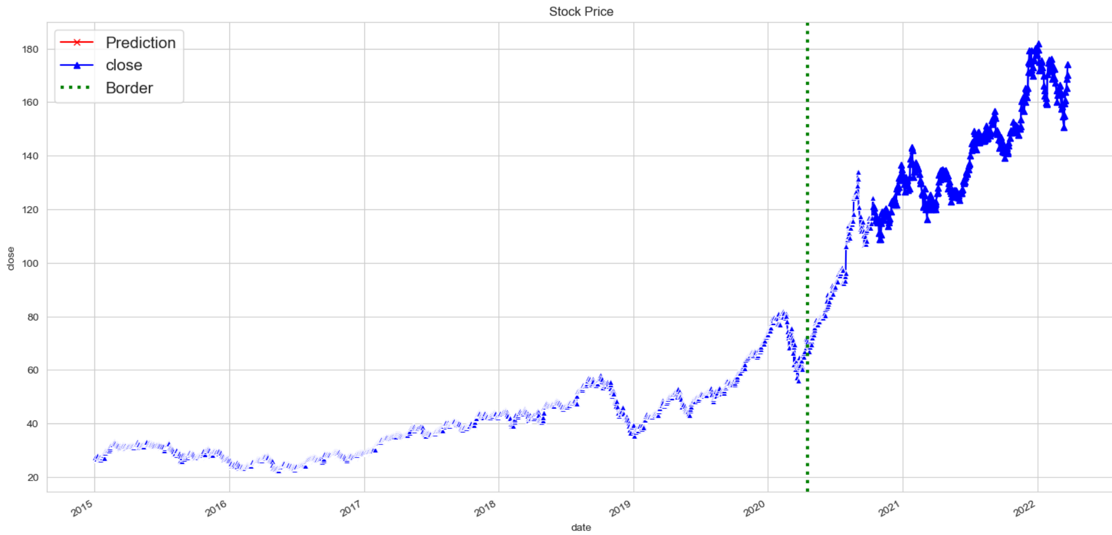
```
# plotting residuals and density
fig, ax = plt.subplots(1, 2)
fig.set_figwidth(16)
residuals = pd.DataFrame(model.resid)
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind="kde", title="Density", ax=ax[1])
plt.show()
```



[57]:

```
price_forecast = model.forecast(12)
price_forecast = pd.Series(price_forecast, index=fy_forecast)
price_forecast = price_forecast.rename("Prediction")

fig, ax = plt.subplots(figsize=(18, 9))
sns.lineplot(x="date", y="close", data=df_close, color="blue", marker="^")
price_forecast.plot(ax=ax, c="red", marker="x", label="Prediction")
X_test.plot(ax=ax, c="blue", marker="^")
plt.title("Stock Price")
```

```
ax.axvline(x=18370, ls=":", linewidth=3, c="green", label="Border")
plt.legend(loc=0, fontsize=15)
plt.show()
```



[58]:
```
# printing forecast predictions for next twelve months
df = pd.DataFrame({"Predictions": model.forecast(12)})
df["date"] = fy_forecast
df.set_index("date")
```

[58]:

date	Predictions
2022-04-01	121.747739
2022-05-01	121.507763
2022-06-01	121.596670
2022-07-01	121.563732
2022-08-01	121.575935
2022-09-01	121.571414
2022-10-01	121.573088
2022-11-01	121.572468
2022-12-01	121.572698
2023-01-01	121.572613
2023-02-01	121.572644
2023-03-01	121.572633

[59]:
```
# splitting data into training and testing for an LTSM model
x_train, y_train = X_train.index, X_train.close
x_test, y_test = X_test.index, X_test.close

print(x_train.shape), print(x_test.shape)
```

```
(1456,)
(364,)
```

[59]:
```
(None, None)
```

[60]:
```python
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(100, 1)))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))

model.compile(loss="mean_squared_error", optimizer="adam")
model.summary()
```

```
WARNING:tensorflow:From C:\Users\vince\anaconda3\Lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\vince\anaconda3\Lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 lstm_1 (LSTM)               (None, 100, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50851 (198.64 KB)
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

model = Sequential(): Creates a new sequential model, which is a linear stack of layers.

model.add(LSTM(50, return_sequences=True, input_shape=(100, 1))): Adds the first LSTM layer with 50 units (neurons), return_sequences=True indicates that the layer should return sequences rather than a single output, and input_shape=(100, 1) specifies the input shape for the first layer (100 time steps with 1 feature).

model.add(LSTM(50, return_sequences=True)): Adds a second LSTM layer with 50 units and return_sequences=True, which means it also returns sequences.

model.add(LSTM(50)): Adds a third LSTM layer with 50 units, but return_sequences is not specified, so it defaults to False, meaning this layer will return a single output for the whole sequence.

model.add(Dense(1)): Adds a dense (fully connected) layer with 1 unit, which is the output layer for the model.

model.compile(loss="mean_squared_error", optimizer="adam"): Compiles the model with a mean squared error loss function and the Adam optimizer.

model.summary(): Prints a summary of the model, showing the architecture and the number of parameters in each layer.

[61]:
```python
# fitting the model on the testing data
model.fit(
    X_train,
    y_train,
    validation_data=(X_test, y_test),
    epochs=100,
    batch_size=64,
    verbose=1,
)
```

```
Epoch 1/100
WARNING:tensorflow:From C:\Users\vince\anaconda3\Lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

23/23 [==============================] - 15s 148ms/step - loss: 2488.9712 -
val_loss: 20388.2852
Epoch 2/100
23/23 [==============================] - 1s 22ms/step - loss: 2381.2502 -
val_loss: 19654.4473
Epoch 3/100
23/23 [==============================] - 0s 14ms/step - loss: 2043.4578 -
val_loss: 18389.5742
Epoch 4/100
23/23 [==============================] - 0s 12ms/step - loss: 1755.6875 -
val_loss: 17672.7148
Epoch 5/100
23/23 [==============================] - 1s 24ms/step - loss: 1612.3430 -
val_loss: 17255.2969
Epoch 6/100
23/23 [==============================] - 1s 23ms/step - loss: 1518.8396 -
val_loss: 16935.4551
Epoch 7/100
23/23 [==============================] - 0s 14ms/step - loss: 1443.6001 -
val_loss: 16650.6055
Epoch 8/100
```

```
23/23 [==============================] - 0s 15ms/step - loss: 1377.0129 -
val_loss: 16385.7891
Epoch 9/100
23/23 [==============================] - 0s 12ms/step - loss: 1316.1627 -
val_loss: 16136.6631
Epoch 10/100
23/23 [==============================] - 0s 12ms/step - loss: 1259.8538 -
val_loss: 15901.1621
Epoch 11/100
23/23 [==============================] - 0s 12ms/step - loss: 1207.6207 -
val_loss: 15675.2227
Epoch 12/100
23/23 [==============================] - 0s 13ms/step - loss: 1158.9547 -
val_loss: 15456.3301
Epoch 13/100
23/23 [==============================] - 0s 12ms/step - loss: 1112.8175 -
val_loss: 15247.6523
Epoch 14/100
23/23 [==============================] - 0s 13ms/step - loss: 1070.1613 -
val_loss: 15041.8857
Epoch 15/100
23/23 [==============================] - 0s 13ms/step - loss: 1029.0826 -
val_loss: 14847.8857
Epoch 16/100
23/23 [==============================] - 0s 13ms/step - loss: 991.2211 -
val_loss: 14656.4824
Epoch 17/100
23/23 [==============================] - 0s 13ms/step - loss: 954.9692 -
val_loss: 14472.6143
Epoch 18/100
23/23 [==============================] - 0s 12ms/step - loss: 921.1567 -
val_loss: 14291.8037
Epoch 19/100
23/23 [==============================] - 0s 12ms/step - loss: 889.0429 -
val_loss: 14117.3115
Epoch 20/100
23/23 [==============================] - 0s 15ms/step - loss: 858.5605 -
val_loss: 13950.3721
Epoch 21/100
23/23 [==============================] - 0s 12ms/step - loss: 830.1619 -
val_loss: 13786.1104
Epoch 22/100
23/23 [==============================] - 0s 12ms/step - loss: 803.4775 -
val_loss: 13625.8311
Epoch 23/100
23/23 [==============================] - 0s 13ms/step - loss: 777.9852 -
val_loss: 13472.5938
Epoch 24/100
```

```
23/23 [==============================] - 0s 13ms/step - loss: 754.1510 -
val_loss: 13323.9248
Epoch 25/100
23/23 [==============================] - 0s 13ms/step - loss: 731.8716 -
val_loss: 13177.8086
Epoch 26/100
23/23 [==============================] - 0s 12ms/step - loss: 710.6928 -
val_loss: 13036.5957
Epoch 27/100
23/23 [==============================] - 0s 17ms/step - loss: 690.9562 -
val_loss: 12899.5293
Epoch 28/100
23/23 [==============================] - 0s 14ms/step - loss: 672.3544 -
val_loss: 12766.9521
Epoch 29/100
23/23 [==============================] - 0s 13ms/step - loss: 654.9308 -
val_loss: 12638.7168
Epoch 30/100
23/23 [==============================] - 1s 23ms/step - loss: 638.6453 -
val_loss: 12513.7861
Epoch 31/100
23/23 [==============================] - 0s 16ms/step - loss: 623.3749 -
val_loss: 12392.7334
Epoch 32/100
23/23 [==============================] - 0s 13ms/step - loss: 609.1682 -
val_loss: 12274.8926
Epoch 33/100
23/23 [==============================] - 0s 13ms/step - loss: 595.8460 -
val_loss: 12161.9893
Epoch 34/100
23/23 [==============================] - 0s 12ms/step - loss: 583.5210 -
val_loss: 12051.2197
Epoch 35/100
23/23 [==============================] - 0s 12ms/step - loss: 572.0040 -
val_loss: 11943.7871
Epoch 36/100
23/23 [==============================] - 0s 13ms/step - loss: 561.0577 -
val_loss: 11844.2129
Epoch 37/100
23/23 [==============================] - 0s 16ms/step - loss: 551.3561 -
val_loss: 11741.2607
Epoch 38/100
23/23 [==============================] - 0s 13ms/step - loss: 541.8694 -
val_loss: 11646.7129
Epoch 39/100
23/23 [==============================] - 0s 15ms/step - loss: 533.4210 -
val_loss: 11552.8994
Epoch 40/100
```

```
23/23 [==============================] - 0s 12ms/step - loss: 525.3023 -
val_loss: 11466.8115
Epoch 41/100
23/23 [==============================] - 0s 12ms/step - loss: 518.1283 -
val_loss: 11377.9570
Epoch 42/100
23/23 [==============================] - 0s 13ms/step - loss: 511.2739 -
val_loss: 11294.5342
Epoch 43/100
23/23 [==============================] - 0s 17ms/step - loss: 505.0655 -
val_loss: 11213.3955
Epoch 44/100
23/23 [==============================] - 0s 17ms/step - loss: 499.2321 -
val_loss: 11138.4824
Epoch 45/100
23/23 [==============================] - 0s 15ms/step - loss: 494.0583 -
val_loss: 11063.6357
Epoch 46/100
23/23 [==============================] - 0s 13ms/step - loss: 489.1286 -
val_loss: 10994.4023
Epoch 47/100
23/23 [==============================] - 0s 15ms/step - loss: 484.7482 -
val_loss: 10925.1025
Epoch 48/100
23/23 [==============================] - 0s 12ms/step - loss: 480.6475 -
val_loss: 10859.4590
Epoch 49/100
23/23 [==============================] - 0s 13ms/step - loss: 476.9771 -
val_loss: 10793.3574
Epoch 50/100
23/23 [==============================] - 0s 16ms/step - loss: 473.5978 -
val_loss: 10731.1602
Epoch 51/100
23/23 [==============================] - 0s 17ms/step - loss: 470.3553 -
val_loss: 10679.4883
Epoch 52/100
23/23 [==============================] - 0s 15ms/step - loss: 467.6563 -
val_loss: 10622.9922
Epoch 53/100
23/23 [==============================] - 0s 12ms/step - loss: 465.1718 -
val_loss: 10566.1650
Epoch 54/100
23/23 [==============================] - 0s 12ms/step - loss: 462.7691 -
val_loss: 10516.8828
Epoch 55/100
23/23 [==============================] - 0s 12ms/step - loss: 460.7469 -
val_loss: 10467.3359
Epoch 56/100
```

```
23/23 [==============================] - 0s 13ms/step - loss: 458.9226 -
val_loss: 10418.1943
Epoch 57/100
23/23 [==============================] - 0s 14ms/step - loss: 457.1200 -
val_loss: 10378.5713
Epoch 58/100
23/23 [==============================] - 0s 13ms/step - loss: 455.6925 -
val_loss: 10334.2988
Epoch 59/100
23/23 [==============================] - 0s 12ms/step - loss: 454.2611 -
val_loss: 10295.1631
Epoch 60/100
23/23 [==============================] - 0s 13ms/step - loss: 453.0477 -
val_loss: 10255.7773
Epoch 61/100
23/23 [==============================] - 0s 12ms/step - loss: 451.9084 -
val_loss: 10221.2090
Epoch 62/100
23/23 [==============================] - 0s 13ms/step - loss: 450.9286 -
val_loss: 10187.3184
Epoch 63/100
23/23 [==============================] - 0s 15ms/step - loss: 449.9744 -
val_loss: 10158.1250
Epoch 64/100
23/23 [==============================] - 0s 16ms/step - loss: 437.2979 -
val_loss: 10145.8506
Epoch 65/100
23/23 [==============================] - 0s 12ms/step - loss: 367.6434 -
val_loss: 10047.4893
Epoch 66/100
23/23 [==============================] - 0s 14ms/step - loss: 354.7472 -
val_loss: 9940.2900
Epoch 67/100
23/23 [==============================] - 0s 15ms/step - loss: 343.8152 -
val_loss: 9829.5586
Epoch 68/100
23/23 [==============================] - 0s 15ms/step - loss: 334.7818 -
val_loss: 9720.6641
Epoch 69/100
23/23 [==============================] - 0s 15ms/step - loss: 325.9932 -
val_loss: 9618.6777
Epoch 70/100
23/23 [==============================] - 0s 15ms/step - loss: 317.7364 -
val_loss: 9517.3125
Epoch 71/100
23/23 [==============================] - 0s 16ms/step - loss: 309.6134 -
val_loss: 9423.3691
Epoch 72/100
```

```
23/23 [==============================] - 0s 16ms/step - loss: 302.2291 -
val_loss: 9324.5146
Epoch 73/100
23/23 [==============================] - 0s 12ms/step - loss: 294.8651 -
val_loss: 9232.8252
Epoch 74/100
23/23 [==============================] - 0s 16ms/step - loss: 288.1949 -
val_loss: 9139.9346
Epoch 75/100
23/23 [==============================] - 0s 18ms/step - loss: 281.7228 -
val_loss: 9048.2471
Epoch 76/100
23/23 [==============================] - 0s 13ms/step - loss: 275.0583 -
val_loss: 8963.2119
Epoch 77/100
23/23 [==============================] - 1s 22ms/step - loss: 269.1318 -
val_loss: 8870.8984
Epoch 78/100
23/23 [==============================] - 0s 14ms/step - loss: 263.1180 -
val_loss: 8784.3125
Epoch 79/100
23/23 [==============================] - 0s 15ms/step - loss: 257.3171 -
val_loss: 8700.8691
Epoch 80/100
23/23 [==============================] - 0s 14ms/step - loss: 251.8498 -
val_loss: 8617.4131
Epoch 81/100
23/23 [==============================] - 1s 22ms/step - loss: 246.5084 -
val_loss: 8535.0449
Epoch 82/100
23/23 [==============================] - 0s 21ms/step - loss: 241.2217 -
val_loss: 8458.7432
Epoch 83/100
23/23 [==============================] - 0s 12ms/step - loss: 236.2263 -
val_loss: 8377.5000
Epoch 84/100
23/23 [==============================] - 0s 12ms/step - loss: 231.2125 -
val_loss: 8298.8252
Epoch 85/100
23/23 [==============================] - 0s 12ms/step - loss: 226.4357 -
val_loss: 8221.0684
Epoch 86/100
23/23 [==============================] - 0s 13ms/step - loss: 221.7800 -
val_loss: 8145.6196
Epoch 87/100
23/23 [==============================] - 0s 12ms/step - loss: 217.3391 -
val_loss: 8067.7920
Epoch 88/100
```

```
23/23 [==============================] - 0s 13ms/step - loss: 212.9360 -
val_loss: 7993.5112
Epoch 89/100
23/23 [==============================] - 0s 11ms/step - loss: 208.5522 -
val_loss: 7923.5181
Epoch 90/100
23/23 [==============================] - 0s 16ms/step - loss: 204.4295 -
val_loss: 7850.1909
Epoch 91/100
23/23 [==============================] - 0s 21ms/step - loss: 200.2717 -
val_loss: 7775.6313
Epoch 92/100
23/23 [==============================] - 0s 14ms/step - loss: 196.1105 -
val_loss: 7704.5234
Epoch 93/100
23/23 [==============================] - 0s 13ms/step - loss: 192.1878 -
val_loss: 7634.9185
Epoch 94/100
23/23 [==============================] - 1s 22ms/step - loss: 188.4186 -
val_loss: 7565.0781
Epoch 95/100
23/23 [==============================] - 0s 21ms/step - loss: 184.5862 -
val_loss: 7495.7261
Epoch 96/100
23/23 [==============================] - 1s 22ms/step - loss: 180.8233 -
val_loss: 7431.9038
Epoch 97/100
23/23 [==============================] - 0s 22ms/step - loss: 177.3417 -
val_loss: 7363.1421
Epoch 98/100
23/23 [==============================] - 1s 23ms/step - loss: 173.8050 -
val_loss: 7295.7637
Epoch 99/100
23/23 [==============================] - 0s 14ms/step - loss: 170.3596 -
val_loss: 7229.7388
Epoch 100/100
23/23 [==============================] - 0s 15ms/step - loss: 166.9794 -
val_loss: 7164.0039
```

[61]:
```
<keras.src.callbacks.History at 0x2396af23110>
```

[62]:
```
# valdating the model
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
math.sqrt(mean_squared_error(y_train, train_predict))
```

```
46/46 [==============================] - 3s 4ms/step
12/12 [==============================] - 0s 9ms/step
```

[62]: 12.850884128834732

[63]:
```
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

train_rmse = math.sqrt(mean_squared_error(y_train, train_predict))
test_rmse = math.sqrt(mean_squared_error(y_test, test_predict))

print(f"Training RMSE: {train_rmse}")
print(f"Test RMSE: {test_rmse}")
```

```
46/46 [==============================] - 0s 4ms/step
12/12 [==============================] - 0s 8ms/step
Training RMSE: 12.850884128834732
Test RMSE: 84.6404395343665
```

[64]:
```
# print the first ten predicted values of training set
train_predict[:10]
```

[64]:
```
array([[27.319466],
       [26.542522],
       [26.545076],
       [26.922934],
       [27.95377 ],
       [27.983282],
       [27.299484],
       [27.541065],
       [27.436642],
       [26.687561]], dtype=float32)
```

[12]:
```
import os
output_file = r'C:\path\to\local\directory\Untitled Folder\capstone_project_vdt.
 ↪pdf'
os.system(f'jupyter nbconvert --to pdf --allow-chromium-download --output␣
 ↪{output_file} capstone_project_vdt.ipynb')
```

[12]: 1