Data Project - Stock Market Analysis

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
%matplotlib inline

from datetime import datetime

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import Dense, LSTM
import ta
import warnings
warnings.filterwarnings("ignore")
from datetime import date
```

In [2]:
```
stock_data = pd.read_csv(r"C:\Users\chitt\Downloads\GOOG.csv")
stock_data= pd.DataFrame(stock_data)
stock_data
```

Out[2]:

	symbol	date	close	high	low	open	volume	adjClose
0	GOOG	2016-06-14 00:00:00+00:00	718.27	722.470	713.1200	716.48	1306065	718.27
1	GOOG	2016-06-15 00:00:00+00:00	718.92	722.980	717.3100	719.00	1214517	718.92
2	GOOG	2016-06-16 00:00:00+00:00	710.36	716.650	703.2600	714.91	1982471	710.36
3	GOOG	2016-06-17 00:00:00+00:00	691.72	708.820	688.4515	708.65	3402357	691.72
4	GOOG	2016-06-20 00:00:00+00:00	693.71	702.480	693.4100	698.77	2082538	693.71
...	...	...	...	...	...	...	...	...
1253	GOOG	2021-06-07 00:00:00+00:00	2466.09	2468.000	2441.0725	2451.32	1192453	2466.09
1254	GOOG	2021-06-08 00:00:00+00:00	2482.85	2494.495	2468.2400	2479.90	1253253	2482.85
1255	GOOG	2021-06-09 00:00:00+00:00	2491.40	2505.000	2487.3300	2499.50	1006337	2491.40
1256	GOOG	2021-06-10 00:00:00+00:00	2521.60	2523.260	2494.0000	2494.01	1561733	2521.60
1257	GOOG	2021-06-11 00:00:00+00:00	2513.93	2526.990	2498.2900	2524.92	1262309	2513.93

1258 rows × 14 columns

Dataset Attributes

symbol : Name of the company (in this case Google).

date : year and date.

close: closing of stock value.

high: highest value of stock at that day.

low: lowest value of stock at that day.

open: The opening price of the stock on the given date.

volume:The trading volume (number of shares) of the stock on the given date.

adjClose: The adjusted closing price of the stock on the given date.

adjHigh: The adjusted highest price reached by the stock on the given date.

adjLow: The adjusted lowest price reached by the stock on the given date.

adjOpen: The adjusted opening price of the stock on the given date.

adjVolume: The adjusted trading volume (number of shares) of the stock on the given date.

divCash: Dividends paid out on the given date (if any).

splitFactor: The split factor applied on the given date (if any).

EDA (Exploratory Data Analysis)

In [3]:
```
stock_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   symbol       1258 non-null   object 
 1   date         1258 non-null   object 
 2   close        1258 non-null   float64
 3   high         1258 non-null   float64
 4   low          1258 non-null   float64
 5   open         1258 non-null   float64
 6   volume       1258 non-null   int64  
 7   adjClose     1258 non-null   float64
 8   adjHigh      1258 non-null   float64
 9   adjLow       1258 non-null   float64
 10  adjOpen      1258 non-null   float64
 11  adjVolume    1258 non-null   int64  
 12  divCash      1258 non-null   float64
 13  splitFactor  1258 non-null   float64
dtypes: float64(10), int64(2), object(2)
memory usage: 137.7+ KB
```

In [5]:
```
stock_data.describe()
```

Out[5]:

	close	high	low	open	volume	adjClose
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1258.000000
mean	1216.317067	1227.430934	1204.176430	1215.260779	1.601590e+06	1216.317067
std	383.333358	387.570872	378.777094	382.446995	6.960172e+05	383.333358
min	668.260000	672.300000	663.284000	671.000000	3.467530e+05	668.260000
25%	960.802500	968.757500	952.182500	959.005000	1.173522e+06	960.802500
50%	1132.460000	1143.935000	1117.915000	1131.150000	1.412588e+06	1132.460000
75%	1360.595000	1374.345000	1348.557500	1361.075000	1.812156e+06	1360.595000
max	2521.600000	2526.990000	2498.290000	2524.920000	6.207027e+06	2521.600000

In [6]:

```
stock_data.isnull().sum()
```

Out[6]:

```
symbol         0
date           0
close          0
high           0
low            0
open           0
volume         0
adjClose       0
adjHigh        0
adjLow         0
adjOpen        0
adjVolume      0
divCash        0
splitFactor    0
dtype: int64
```

In [7]:

```
stock_data = stock_data.drop(['symbol'],axis=1)### removing the stock symbol inf
```

In [8]:

```
### spliting the values in the 'date' column by the space character (" ") using
### The parameter n = 1 indicates that the splitting should happen only once
### expand = True ensures that the split parts are expanded into separate column
stock_data['date']= stock_data['date'].str.split(" ", n = 1, expand = True)[0]
###The selected date part is then converted to a datetime format using
stock_data['date']= pd.to_datetime(stock_data['date'])
stock_data
```

Out[8]:

	date	close	high	low	open	volume	adjClose	adjHigh	adjl
0	2016-06-14	718.27	722.470	713.1200	716.48	1306065	718.27	722.470	713.1
1	2016-06-15	718.92	722.980	717.3100	719.00	1214517	718.92	722.980	717.3
2	2016-06-16	710.36	716.650	703.2600	714.91	1982471	710.36	716.650	703.2
3	2016-06-17	691.72	708.820	688.4515	708.65	3402357	691.72	708.820	688.4
4	2016-06-20	693.71	702.480	693.4100	698.77	2082538	693.71	702.480	693.4
...	...	...	...	...	...	...	...	...	
1253	2021-06-07	2466.09	2468.000	2441.0725	2451.32	1192453	2466.09	2468.000	2441.0
1254	2021-06-08	2482.85	2494.495	2468.2400	2479.90	1253253	2482.85	2494.495	2468.2
1255	2021-06-09	2491.40	2505.000	2487.3300	2499.50	1006337	2491.40	2505.000	2487.3
1256	2021-06-10	2521.60	2523.260	2494.0000	2494.01	1561733	2521.60	2523.260	2494.0
1257	2021-06-11	2513.93	2526.990	2498.2900	2524.92	1262309	2513.93	2526.990	2498.2

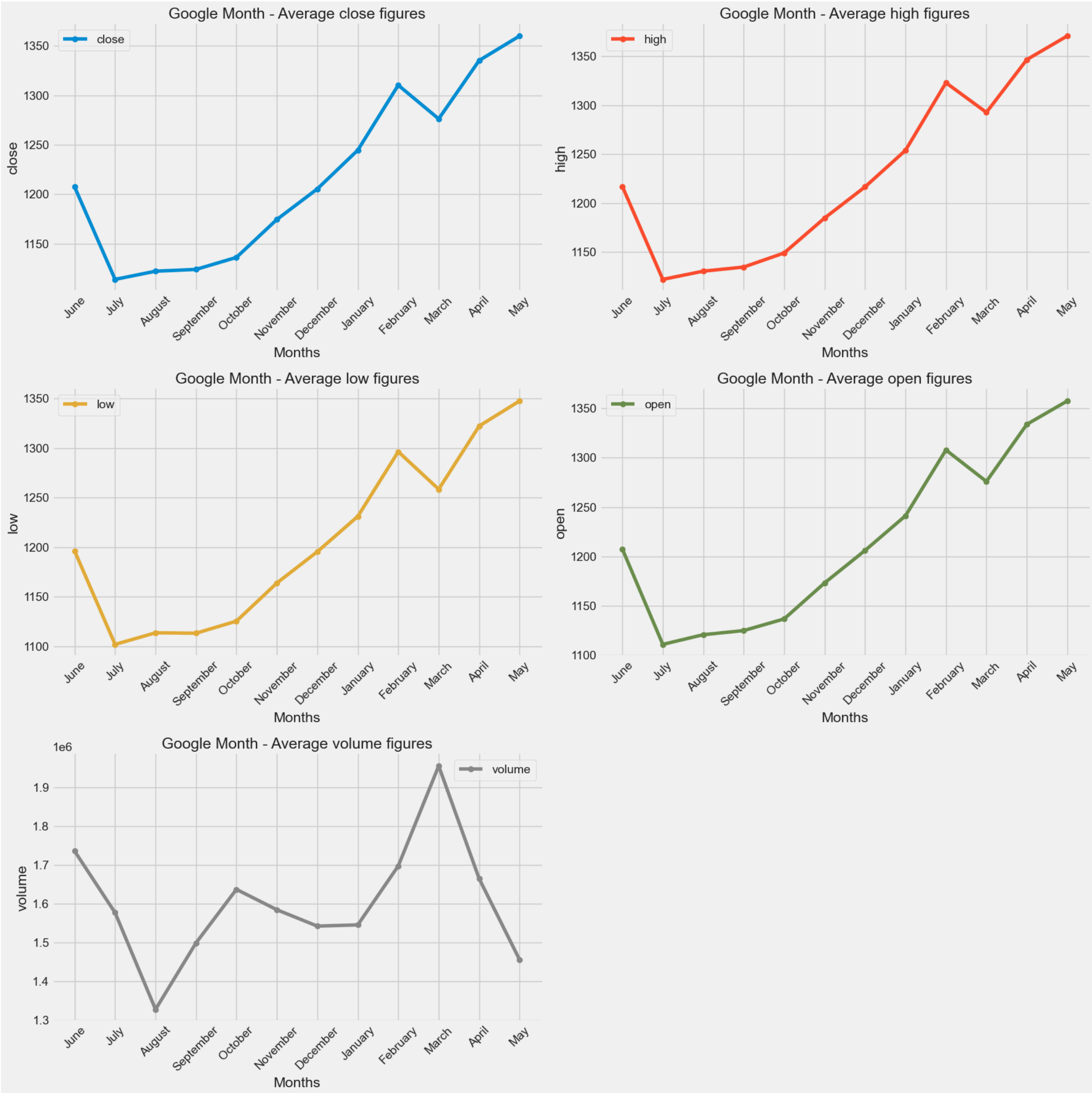
1258 rows × 13 columns

Visualization :---

In [9]:

```
# Convert 'date' column to datetime format
stock_data['date'] = pd.to_datetime(stock_data['date'])

# Set 'date' as the DateTime index
stock_data.set_index('date', inplace=True)

plt.rcParams['font.size'] = 14
plt.rcParams['figure.dpi'] = 100
plt.rcParams['figure.figsize'] = (20, 10)
colors = plt.rcParams["axes.prop_cycle"]()
a1 = 3  # number of rows
a2 = 2  # number of columns
a3 = 1  # initialize plot counter

# Set the figure size of the plot
fig = plt.figure(figsize=(18, 18))

# Specify the columns to plot
columns_to_plot = ['close', 'high', 'low', 'open', 'volume']
```

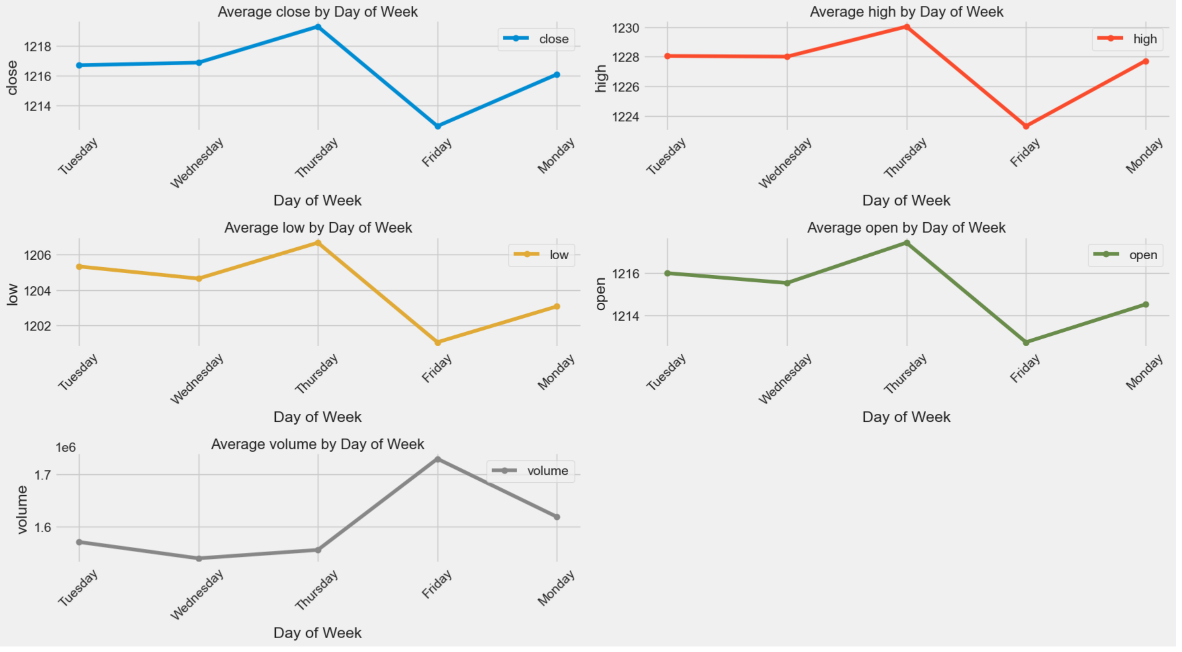
```
# Loop through each column to generate a subplot
for column in columns_to_plot:
    color = next(colors)["color"]
    # Generate a subplot with the given dimensions
    plt.subplot(a1, a2, a3)
    # Plot the data in a line graph, with different colors for each line
    plt.plot(stock_data.groupby(stock_data.index.month_name(), sort=False).mean(

# Remove the top and right borders
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    # Rotate the x-tick labels by 45 degrees
    plt.xticks(rotation=45)
    # Set the title, x-axis label, y-axis label, and legend
    plt.title(f"Google Month - Average {column} figures", fontsize=18)
    plt.xlabel('Months')
    plt.ylabel(column)
    plt.legend([column])
    # Increment the subplot counter
    a3 = a3 + 1

# Adjust the layout of the plot
plt.tight_layout()
# Show the plot
plt.show()
```



In [10]:

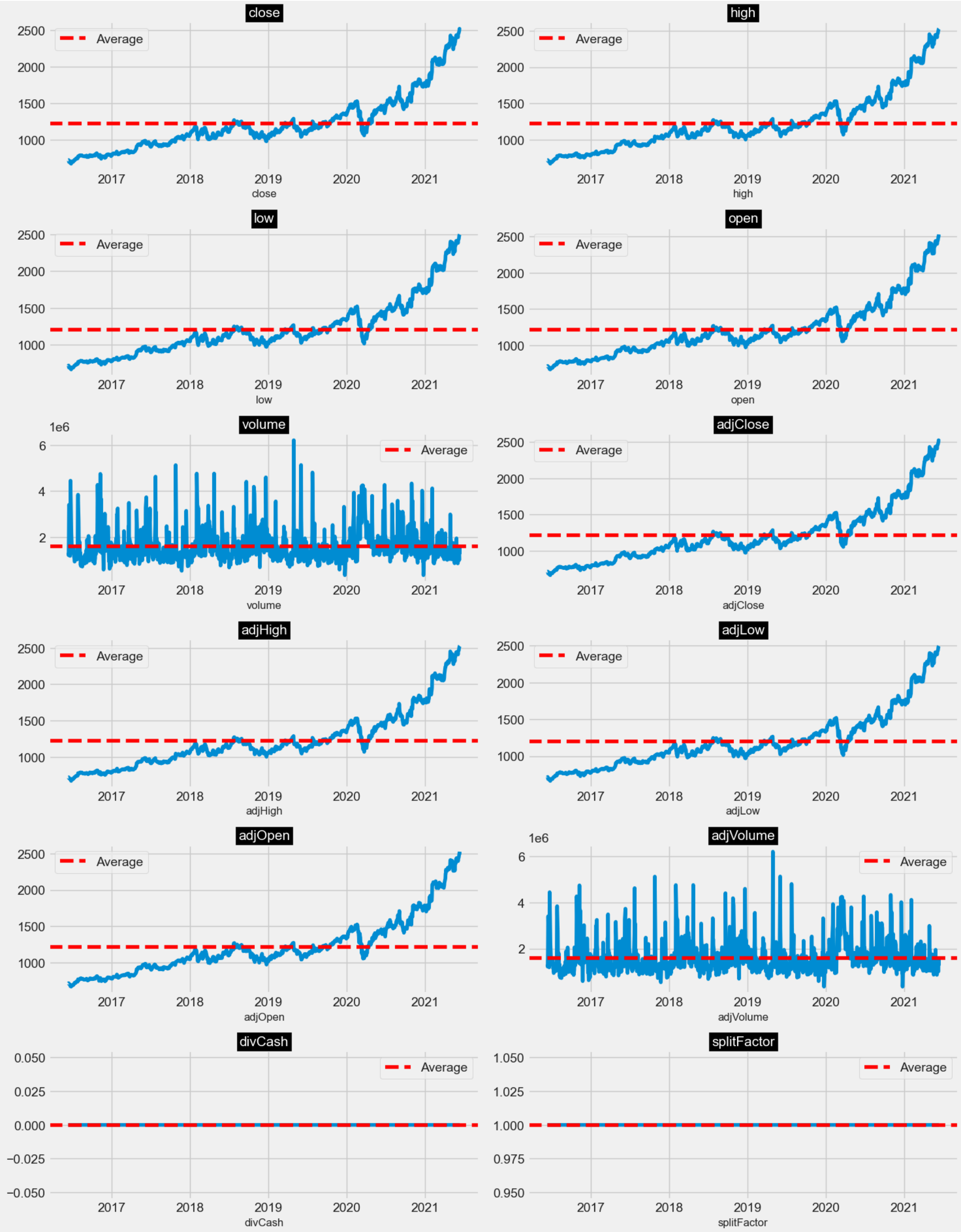
```
plt.rcParams['font.size'] = 14
plt.rcParams['figure.dpi'] = 100
plt.rcParams['figure.figsize'] = (18, 10)
colors = plt.rcParams["axes.prop_cycle"]()
b1 = 3  # number of rows
b2 = 2  # number of columns
b3 = 1  # initialize plot counter

# Set the figure size of the plot
fig = plt.figure()

# Specify the columns to plot
columns_to_plot = ['close', 'high', 'low', 'open', 'volume']

# Loop through each column to generate a subplot
for column in columns_to_plot:
    color = next(colors)["color"]
    # Generate a subplot with the given dimensions
    plt.subplot(b1, b2, b3)
    # Plot the data in a line graph, with different colors for each line
    plt.plot(stock_data.groupby(stock_data.index.day_name(), sort=False)[column]
    # Remove the top and right borders
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
```

```
    # Rotate the x-tick labels by 45 degrees
    plt.xticks(rotation=45)
    # Set the title, x-axis label, y-axis label, and legend
    plt.title(f"Average {column} by Day of Week", fontsize=16)
    plt.xlabel('Day of Week')
    plt.ylabel(column)
    plt.legend([column])
    # Increment the subplot counter
    b3 += 1

# Adjust the layout of the plot
plt.tight_layout()
# Show the plot
plt.show()
```



In [11]:

```
plt.figure(figsize=(15, 25))
for idx, column in enumerate(stock_data):
    plt.subplot(8, 2, idx + 1)
    plt.plot(stock_data.index.values, stock_data[column])
    #Adding a horizontal line for the average of the column
    plt.axhline(stock_data[column].mean(), color='red', linestyle='--', label='A
    plt.title(column, backgroundcolor='black', color='white', fontsize=15)
    plt.xlabel(column, size=12)
    plt.legend()
plt.tight_layout()
plt.show()
```



Moving Average Plot

In [12]:

```
stock_data1 = pd.read_csv(r"C:\Users\chitt\Downloads\GOOG.csv")
stock_data1= pd.DataFrame(stock_data1)
stock_data1 = stock_data1.drop(['symbol'],axis=1)
stock_data1['date']= stock_data1['date'].str.split(" ", n = 1, expand = True)[0]
###The selected date part is then converted to a datetime format using
stock_data1['date']= pd.to_datetime(stock_data1['date'])

stock_data1
```

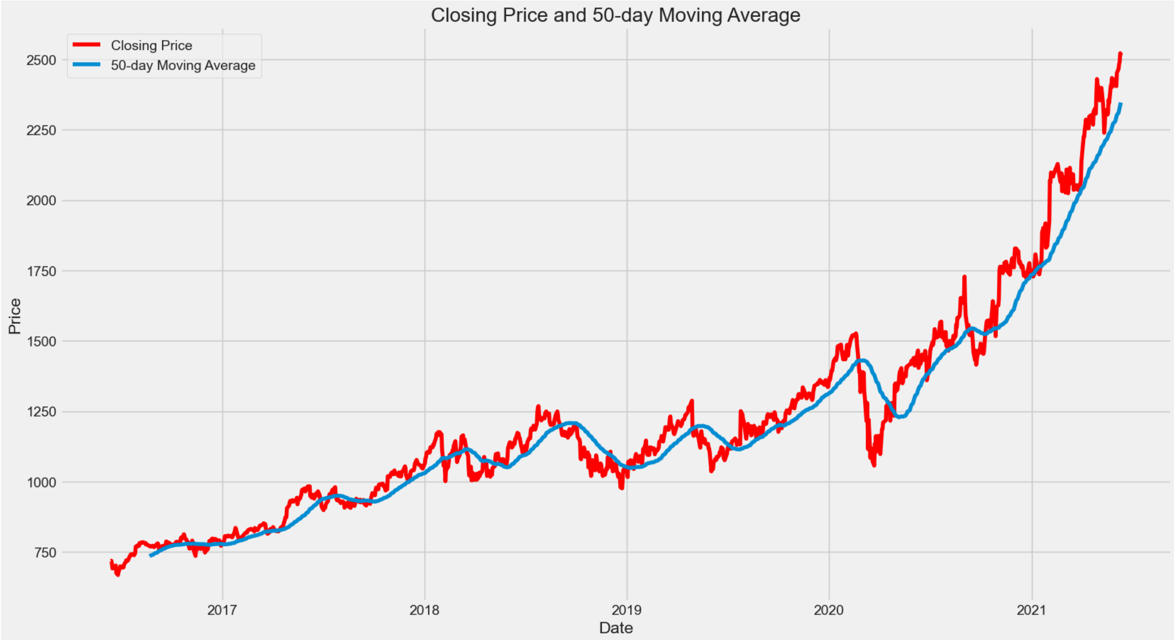
Out[12]:

	date	close	high	low	open	volume	adjClose	adjHigh	adjl
0	2016-06-14	718.27	722.470	713.1200	716.48	1306065	718.27	722.470	713.1
1	2016-06-15	718.92	722.980	717.3100	719.00	1214517	718.92	722.980	717.3
2	2016-06-16	710.36	716.650	703.2600	714.91	1982471	710.36	716.650	703.2
3	2016-06-17	691.72	708.820	688.4515	708.65	3402357	691.72	708.820	688.4
4	2016-06-20	693.71	702.480	693.4100	698.77	2082538	693.71	702.480	693.4
...	...	...	...	...	...	...	...	...	
1253	2021-06-07	2466.09	2468.000	2441.0725	2451.32	1192453	2466.09	2468.000	2441.0
1254	2021-06-08	2482.85	2494.495	2468.2400	2479.90	1253253	2482.85	2494.495	2468.2
1255	2021-06-09	2491.40	2505.000	2487.3300	2499.50	1006337	2491.40	2505.000	2487.3
1256	2021-06-10	2521.60	2523.260	2494.0000	2494.01	1561733	2521.60	2523.260	2494.0
1257	2021-06-11	2513.93	2526.990	2498.2900	2524.92	1262309	2513.93	2526.990	2498.2

1258 rows × 13 columns

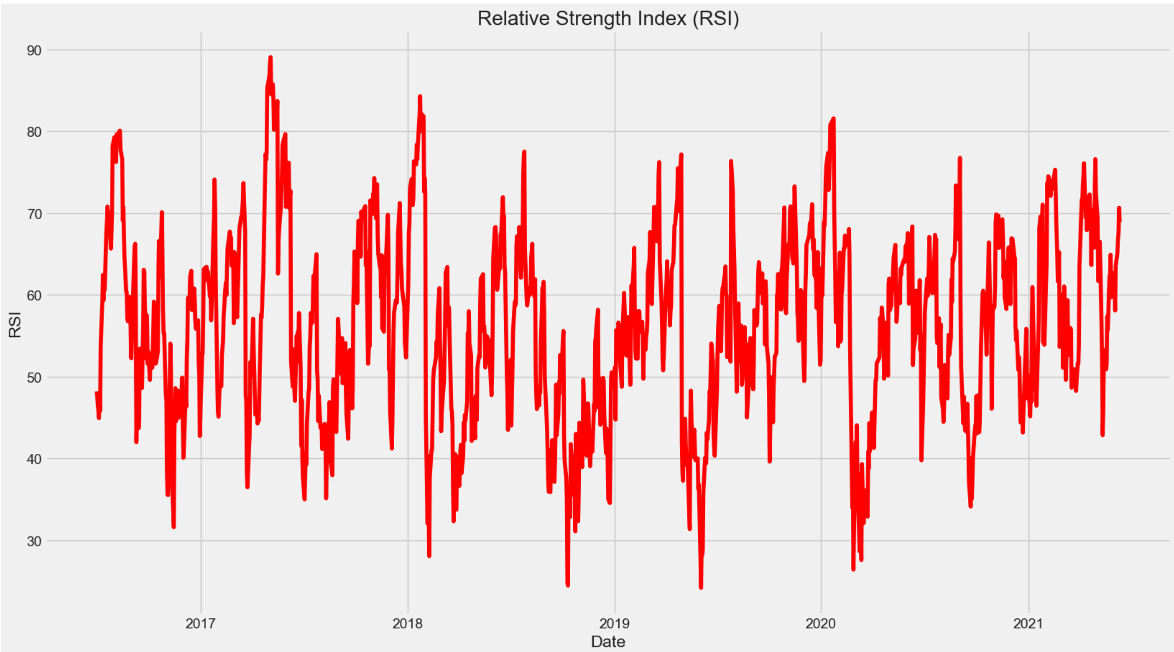
In [19]:

```
rolling_avg = stock_data1['close'].rolling(window=50).mean()

plt.plot(stock_data1['date'], stock_data1['close'], label='Closing Price',color=
plt.plot(stock_data1['date'], rolling_avg, label='50-day Moving Average')
##plt.plot(stock_data1['date'], rolling_avg, label='20-day Moving Average')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Closing Price and 50-day Moving Average')
plt.legend()
plt.show()
```



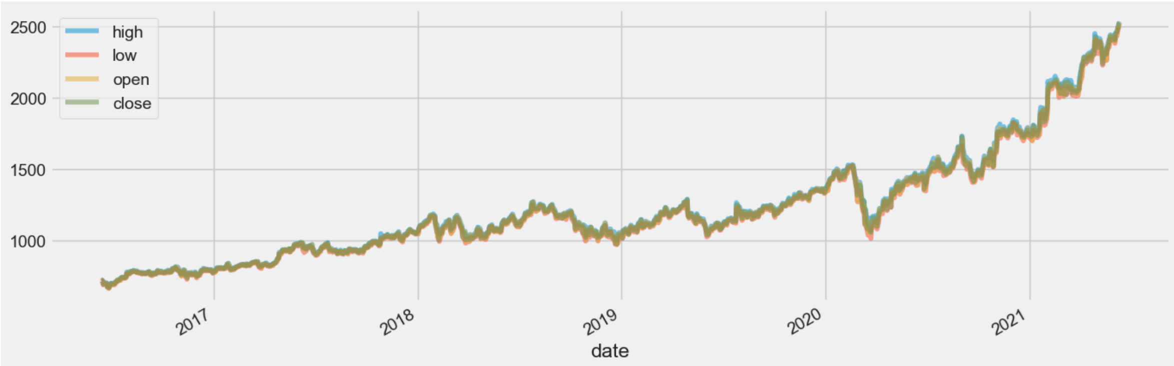
Relative Strength Index(RSI)

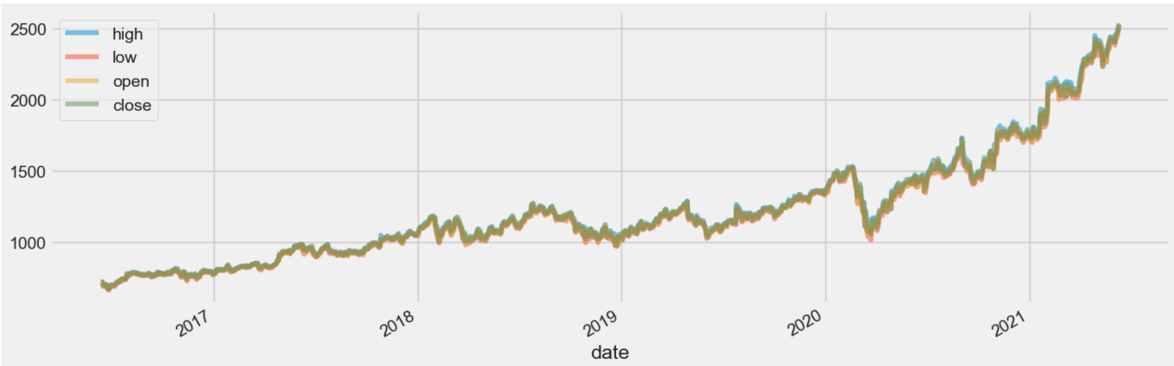
In [23]:
```
rsi = ta.momentum.RSIIndicator(stock_data1['close']).rsi()

plt.plot(stock_data1['date'], rsi, color='r')  # Change the color to red
plt.xlabel('Date')
plt.ylabel('RSI')
plt.title('Relative Strength Index (RSI)')
plt.show()
```



Let's plot four of the indicators in the table and differentiate their corresponding curves by colours

In [26]:
```
stock_data[['high','low','open','close']].plot(figsize = (15, 5), alpha = 0.5)
plt.show()
###he alpha parameter adjusts the transparency of the lines, with 0.5 indicating
```





In [27]:
```
stock_data
```

Out[27]:

date	close	high	low	open	volume	adjClose	adjHigh	adjLow	a
2016-06-14	718.27	722.470	713.1200	716.48	1306065	718.27	722.470	713.1200	
2016-06-15	718.92	722.980	717.3100	719.00	1214517	718.92	722.980	717.3100	
2016-06-16	710.36	716.650	703.2600	714.91	1982471	710.36	716.650	703.2600	
2016-06-17	691.72	708.820	688.4515	708.65	3402357	691.72	708.820	688.4515	
2016-06-20	693.71	702.480	693.4100	698.77	2082538	693.71	702.480	693.4100	
...	...	...	...	...	...	...	...	...	
2021-06-07	2466.09	2468.000	2441.0725	2451.32	1192453	2466.09	2468.000	2441.0725	
2021-06-08	2482.85	2494.495	2468.2400	2479.90	1253253	2482.85	2494.495	2468.2400	
2021-06-09	2491.40	2505.000	2487.3300	2499.50	1006337	2491.40	2505.000	2487.3300	
2021-06-10	2521.60	2523.260	2494.0000	2494.01	1561733	2521.60	2523.260	2494.0000	
2021-06-11	2513.93	2526.990	2498.2900	2524.92	1262309	2513.93	2526.990	2498.2900	

1258 rows × 12 columns

Modeling

In [28]:
```
stock_data = stock_data[['high','low','open','close']] # Extracting required col
```

In [29]:
```
from sklearn.preprocessing import MinMaxScaler
MMS = MinMaxScaler()
stock_data[stock_data.columns] = MMS.fit_transform(stock_data)
```

In [30]:
```
stock_data.shape
```

Out[30]: (1258, 4)

In [31]:
```
training_size = round(len(stock_data) * 0.80) # Selecting 80 % for training and
training_size
```

Out[31]: 1006

In [32]:
```
train_data = stock_data[:training_size]
test_data  = stock_data[training_size:]
```

```
train_data.shape, test_data.shape
```

Out[32]: ((1006, 4), (252, 4))

In [33]:
```
# Function to create sequence of data for training and testing

def create_sequence(dataset):
  sequences = []
  labels = []

  start_idx = 0

  for stop_idx in range(50,len(dataset)): # Selecting 50 rows at a time
    sequences.append(dataset.iloc[start_idx:stop_idx])
    labels.append(dataset.iloc[stop_idx])
    start_idx += 1
  return (np.array(sequences),np.array(labels))
```

In [34]:
```
X_train, y_train = create_sequence(train_data)
X_test, y_test = create_sequence(test_data)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[34]: ((956, 50, 4), (956, 4), (202, 50, 4), (202, 4))

In [35]:
```
from tensorflow.keras.layers import LSTM, Dropout, Dense
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.s
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 4))

regressor.compile(optimizer = 'adam', loss = 'mean_squared_error',metrics=['mean

print(regressor.summary())
```

Model: "sequential"

Layer (type)	Output Shape
lstm (LSTM)	(None, 50, 50)
dropout (Dropout)	(None, 50, 50)
lstm_1 (LSTM)	(None, 50, 50)
dropout_1 (Dropout)	(None, 50, 50)
lstm_2 (LSTM)	(None, 50, 50)
dropout_2 (Dropout)	(None, 50, 50)
lstm_3 (LSTM)	(None, 50)
dropout_3 (Dropout)	(None, 50)
dense (Dense)	(None, 4)

Total params: 71,804 (280.48 KB)

Trainable params: 71,804 (280.48 KB)

Non-trainable params: 0 (0.00 B)

None

In [36]:
```
#model.fit(train_seq, train_label, epochs=80,validation_data=(test_seq, test_lab
regressor.fit(X_train, y_train, epochs = 15,validation_data=(X_test, y_test), ba
```

```
Epoch 1/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - loss: 0.0166 - mean_absolute_error: 0.0
974 - val_loss: 0.0953 - val_mean_absolute_error: 0.2816
Epoch 2/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0028 - mean_absolute_error: 0.0
408 - val_loss: 0.0339 - val_mean_absolute_error: 0.1551
Epoch 3/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0024 - mean_absolute_error: 0.0
356 - val_loss: 0.0510 - val_mean_absolute_error: 0.2010
Epoch 4/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0023 - mean_absolute_error: 0.0
359 - val_loss: 0.0288 - val_mean_absolute_error: 0.1437
Epoch 5/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - loss: 0.0020 - mean_absolute_error: 0.0
324 - val_loss: 0.0293 - val_mean_absolute_error: 0.1466
Epoch 6/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - loss: 0.0017 - mean_absolute_error: 0.0
294 - val_loss: 0.0255 - val_mean_absolute_error: 0.1369
Epoch 7/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - loss: 0.0017 - mean_absolute_error: 0.0
299 - val_loss: 0.0298 - val_mean_absolute_error: 0.1520
Epoch 8/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - loss: 0.0018 - mean_absolute_error: 0.0
300 - val_loss: 0.0135 - val_mean_absolute_error: 0.0962
Epoch 9/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - loss: 0.0015 - mean_absolute_error: 0.0
281 - val_loss: 0.0206 - val_mean_absolute_error: 0.1230
Epoch 10/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0013 - mean_absolute_error: 0.0
260 - val_loss: 0.0244 - val_mean_absolute_error: 0.1362
Epoch 11/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - loss: 0.0014 - mean_absolute_error: 0.0
273 - val_loss: 0.0139 - val_mean_absolute_error: 0.0967
Epoch 12/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 0.0013 - mean_absolute_error: 0.0
264 - val_loss: 0.0192 - val_mean_absolute_error: 0.1177
Epoch 13/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0012 - mean_absolute_error: 0.0
256 - val_loss: 0.0153 - val_mean_absolute_error: 0.1035
Epoch 14/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - loss: 9.4977e-04 - mean_absolute_error:
0.0223 - val_loss: 0.0258 - val_mean_absolute_error: 0.1381
Epoch 15/15
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0011 - mean_absolute_error: 0.0
241 - val_loss: 0.0329 - val_mean_absolute_error: 0.1584
```

Out[36]:
```
<keras.src.callbacks.history.History at 0x1cef9a99690>
```

In [37]:
```python
test_predicted = regressor.predict(X_test)
test_predicted[:5]
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step
```

Out[37]:
```
array([[0.39652416, 0.3845749 , 0.39793396, 0.39665145],
       [0.4025369 , 0.39070198, 0.4038898 , 0.40232503],
       [0.40917635, 0.3975008 , 0.41050863, 0.40872633],
       [0.41636482, 0.40488136, 0.41771558, 0.4157679 ],
       [0.42398497, 0.4127118 , 0.42540073, 0.4233332 ]], dtype=float32)
```

In [38]:
```python
test_inverse_predicted = MMS.inverse_transform(test_predicted) # Inversing scali
test_inverse_predicted[:5]
```

Out[38]:
```
array([[1407.7294, 1368.9812, 1408.7378, 1403.39  ],
       [1418.8811, 1380.2245, 1419.7793, 1413.9052],
       [1431.1953, 1392.7003, 1432.0502, 1425.7689],
       [1444.5277, 1406.2437, 1445.4113, 1438.8193],
       [1458.6606, 1420.6125, 1459.6589, 1452.8403]], dtype=float32)
```

In [39]:
```
# Merging actual and predicted data for better visualization

merge_data = pd.concat([stock_data.iloc[-202:].copy(),pd.DataFrame(test_inverse_
```

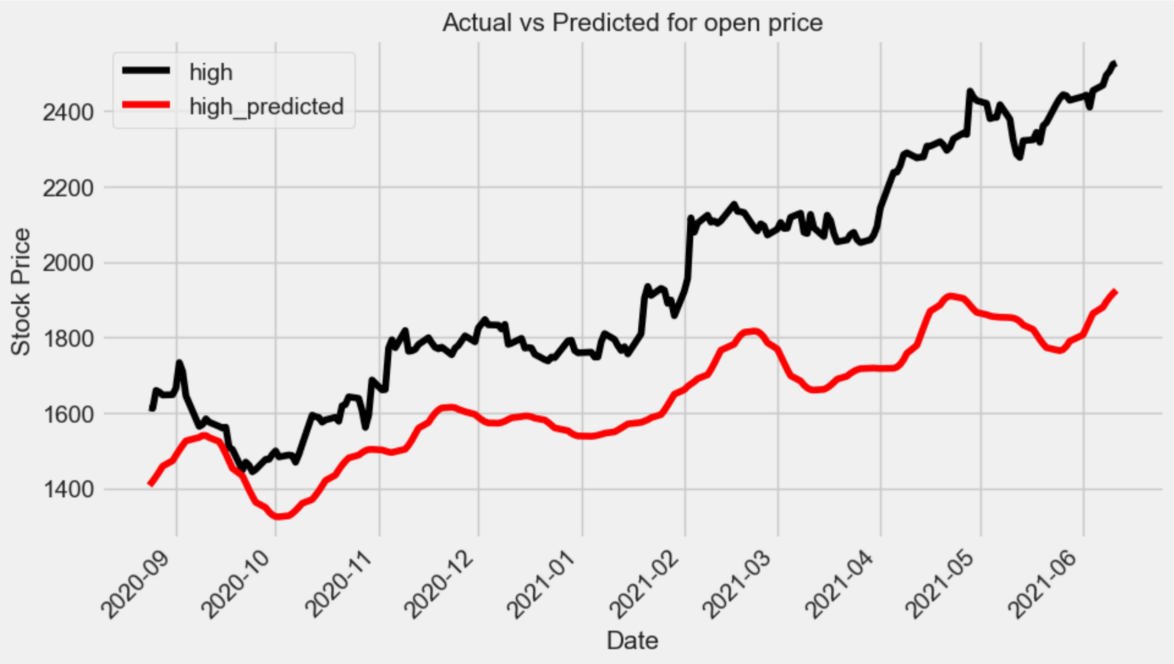
In [40]:
```
merge_data[['high','low','open','close']] = MMS.inverse_transform(merge_data[['h
```

In [41]:
```
merge_data.head()
```

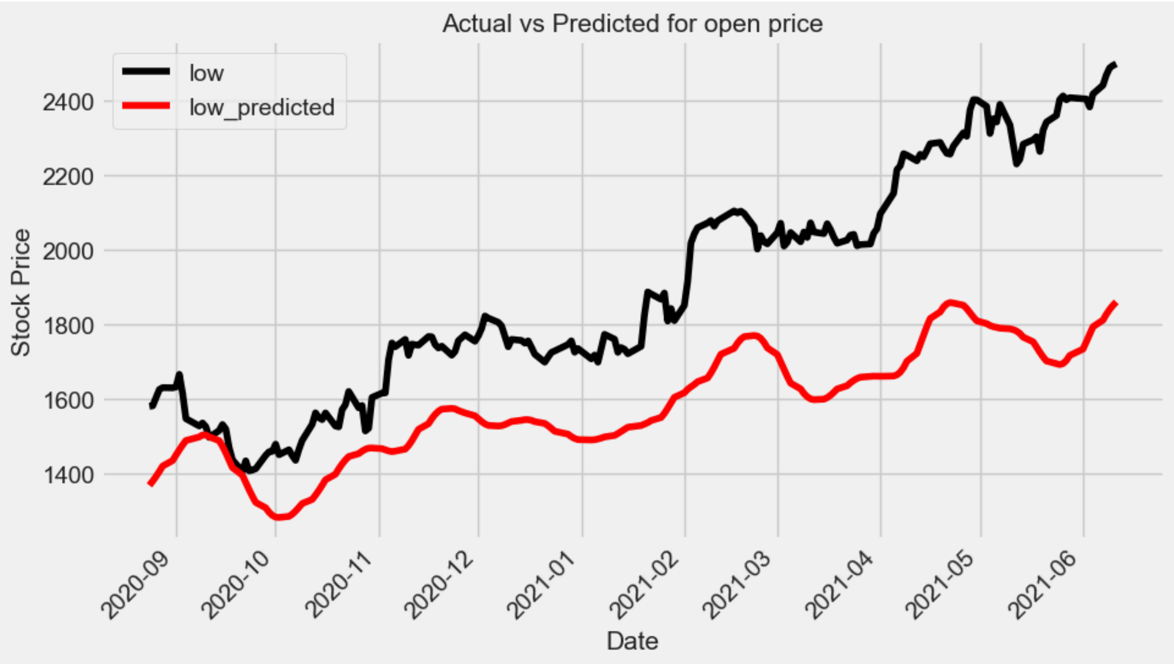
Out[41]:

date	high	low	open	close	high_predicted	low_predicted	open_predic
2020-08-24	1614.1700	1580.57	1593.98	1588.20	1407.729370	1368.981201	1408.737
2020-08-25	1611.6200	1582.07	1582.07	1608.22	1418.881104	1380.224487	1419.779
2020-08-26	1659.2200	1603.60	1608.00	1652.38	1431.195312	1392.700317	1432.050
2020-08-27	1655.0000	1625.75	1653.68	1634.33	1444.527710	1406.243652	1445.411
2020-08-28	1647.1699	1630.75	1633.49	1644.41	1458.660645	1420.612549	1459.658

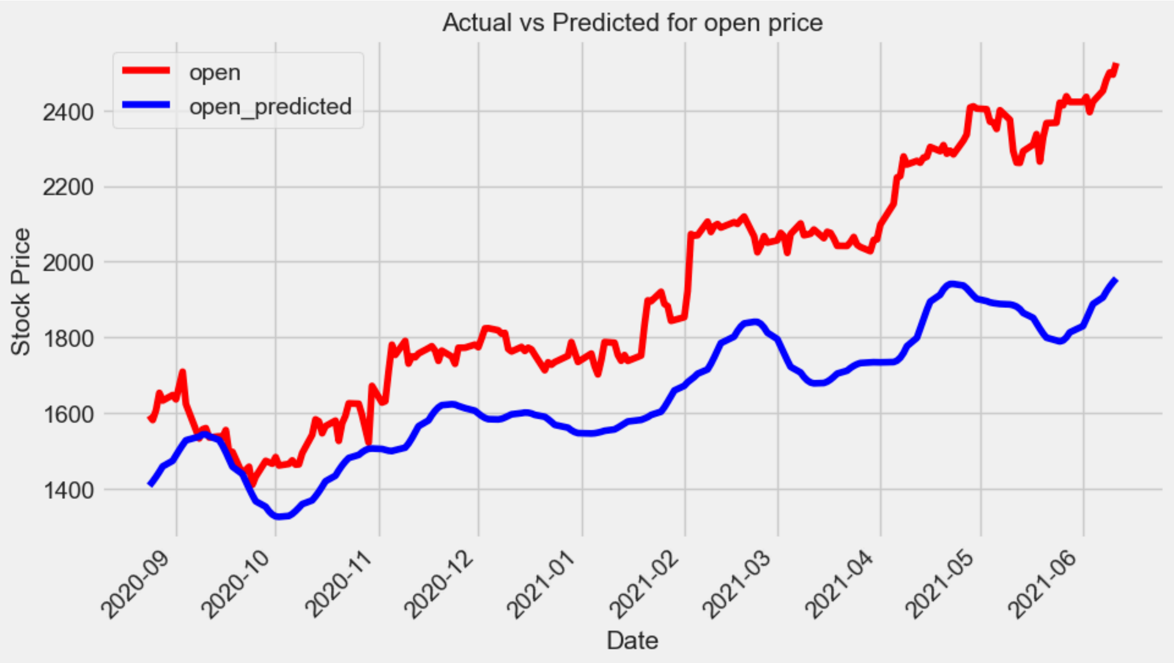
In [42]:
```
merge_data[['high','high_predicted']].plot(figsize=(10,6),color=['black', 'red']
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for open price',size=15)
plt.show()
```



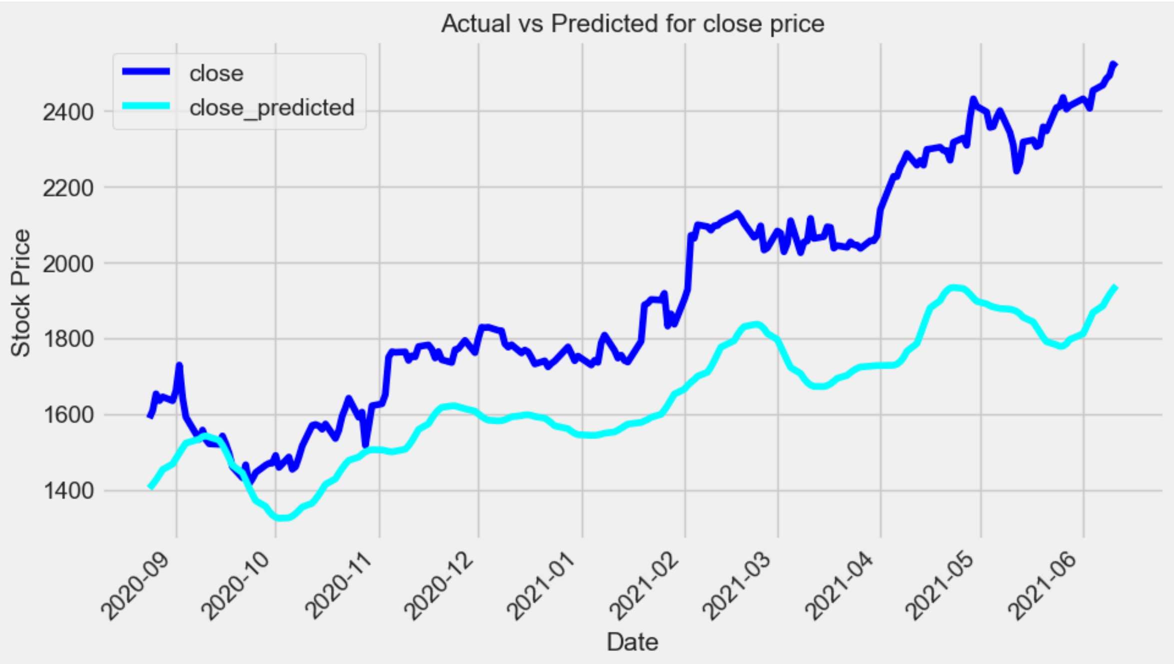
In [43]:
```
merge_data[['low','low_predicted']].plot(figsize=(10,6),color=['black', 'red'])
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for open price',size=15)
plt.show()
```



In [44]:
```
merge_data[['open','open_predicted']].plot(figsize=(10,6),color=['red', 'blue'])
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for open price',size=15)
plt.show()
```



In [45]:
```
merge_data[['close','close_predicted']].plot(figsize=(10,6),color=['blue', 'cyan
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for close price',size=15)
plt.show()
```



In [51]:
```
# Creating a dataframe and adding 15 days to existing index
import pandas as pd
```

In [55]:
```
import pandas as pd

# Adjust `start` to begin from the next day after the last index of merge_data
merge_data_2 = pd.concat([
    merge_data,
    pd.DataFrame(
        columns=merge_data.columns,
        index=pd.date_range(start=merge_data.index[-1] + pd.Timedelta(days=1), p
```

```
    )
])
```

In [56]:
```
merge_data_2['2021-06-09':'2021-06-21']
```

Out[56]:

	high	low	open	close	high_predicted	low_predicted	open_predicte
2021-06-09	2505.00	2487.33	2499.50	2491.40	1905.879272	1840.061768	1934.22045
2021-06-10	2523.26	2494.00	2494.01	2521.60	1915.977417	1851.076904	1945.67492
2021-06-11	2526.99	2498.29	2524.92	2513.93	1924.614746	1860.408936	1955.72766
2021-06-12	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-13	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-14	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-15	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-16	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-17	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-18	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-19	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-20	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2021-06-21	NaN	NaN	NaN	NaN	NaN	NaN	Nal

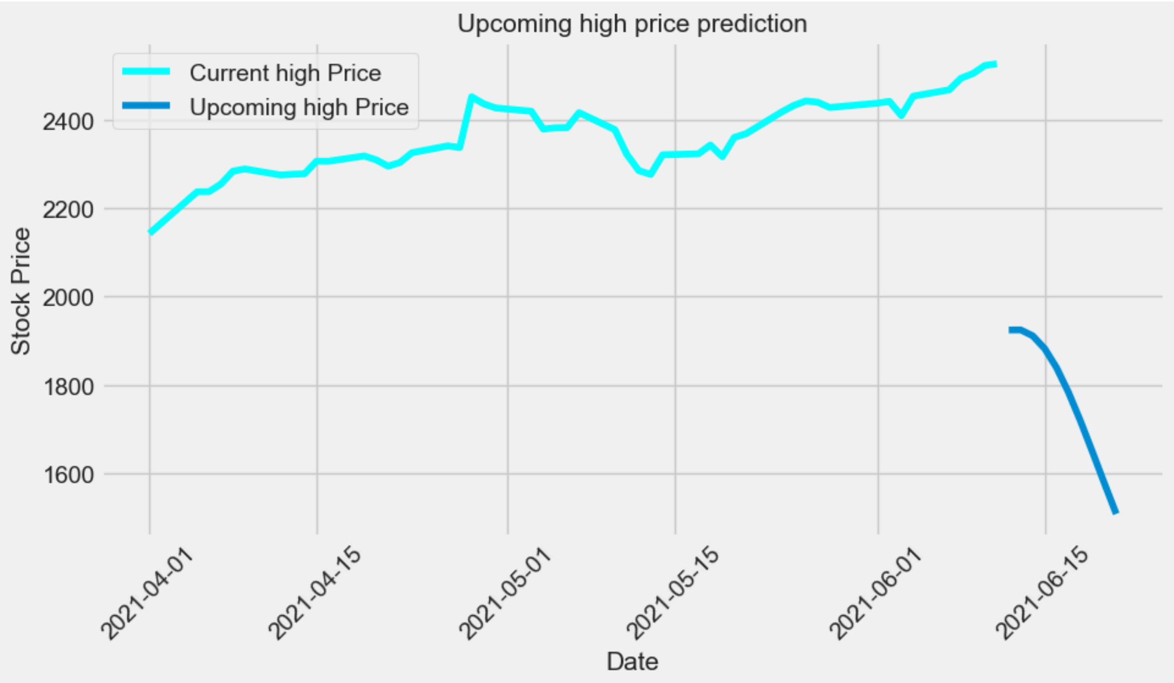
In [57]:
```
upcoming_prediction = pd.DataFrame(columns=['high','low','open','close'],index=m
upcoming_prediction.index=pd.to_datetime(upcoming_prediction.index)
```

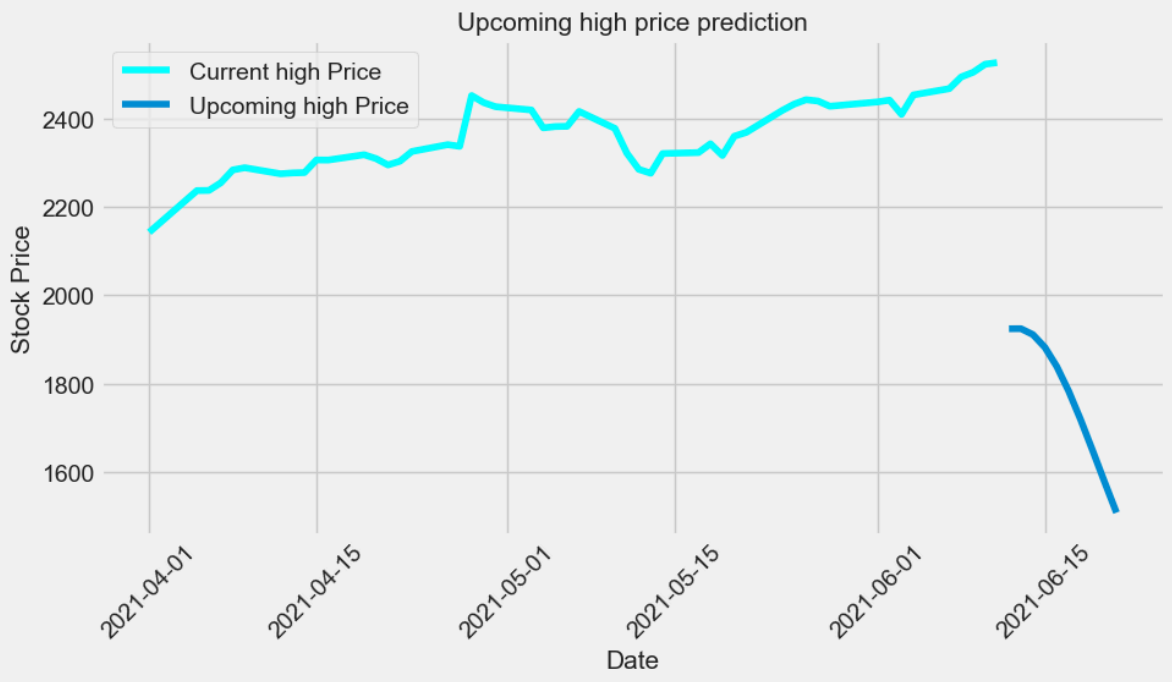
In [58]:
```
curr_seq = X_test[-1:]

for i in range(-10,0):
  up_pred = regressor.predict(curr_seq)
  upcoming_prediction.iloc[i] = up_pred
  curr_seq = np.append(curr_seq[0][1:],up_pred,axis=0)
  curr_seq = curr_seq.reshape(X_test[-1:].shape)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```

In [59]:
```
upcoming_prediction[['high','low','open','close']] = MMS.inverse_transform(upcom
```

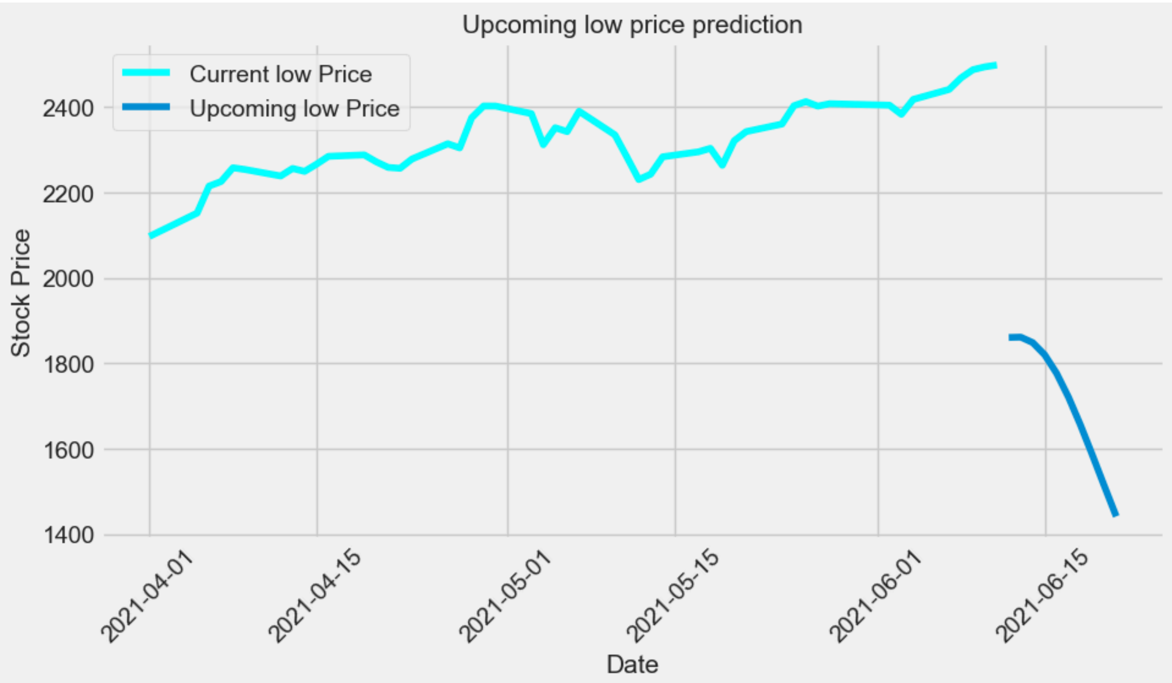
In [61]:
```
fg,ax=plt.subplots(figsize=(10,5))
ax.plot(merge_data_2.loc['2021-04-01':,'high'],label='Current high Price',color=
ax.plot(upcoming_prediction.loc['2021-04-01':,'high'],label='Upcoming high Price
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming high price prediction',size=15)
ax.legend()
fg.show()
plt.show()
```

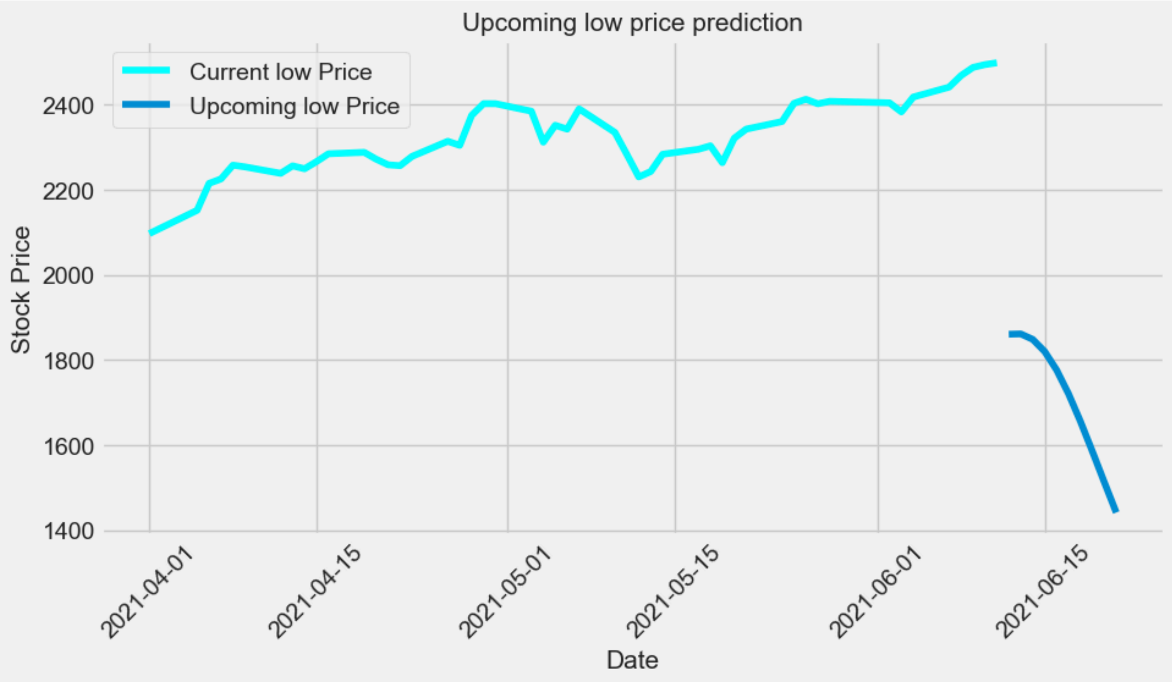




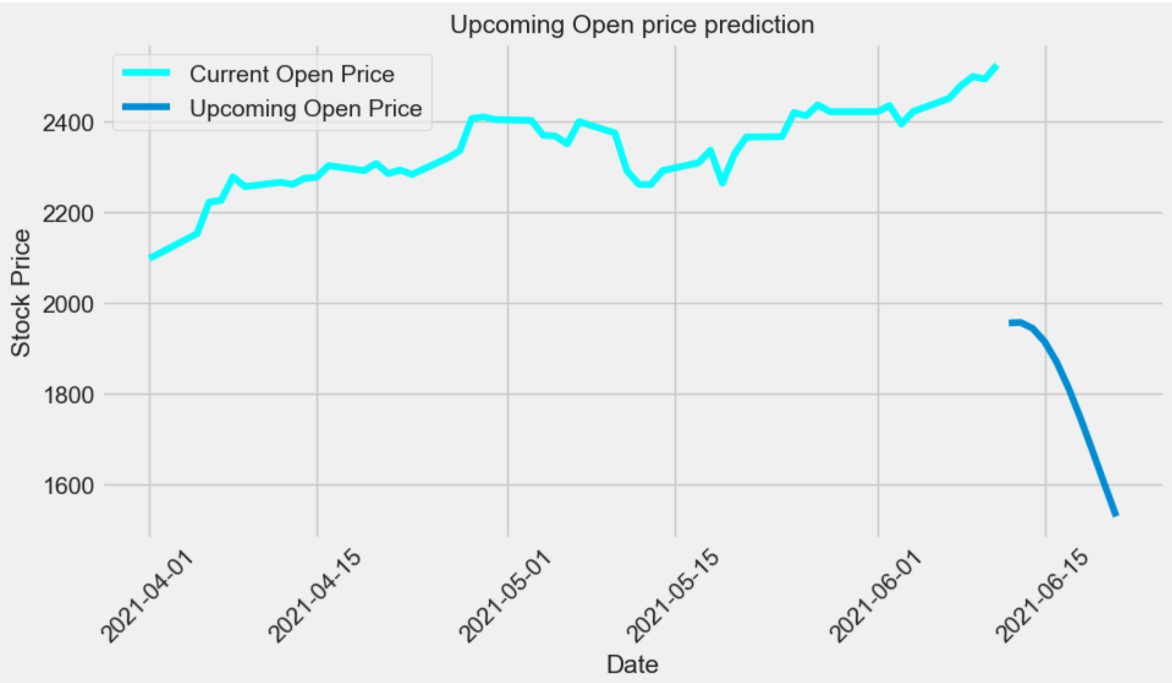
In [63]:

```
fg,ax=plt.subplots(figsize=(10,5))
ax.plot(merge_data_2.loc['2021-04-01':,'low'],label='Current low Price',color='c
ax.plot(upcoming_prediction.loc['2021-04-01':,'low'],label='Upcoming low Price')
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming low price prediction',size=15)
ax.legend()
fg.show()
plt.show()
```



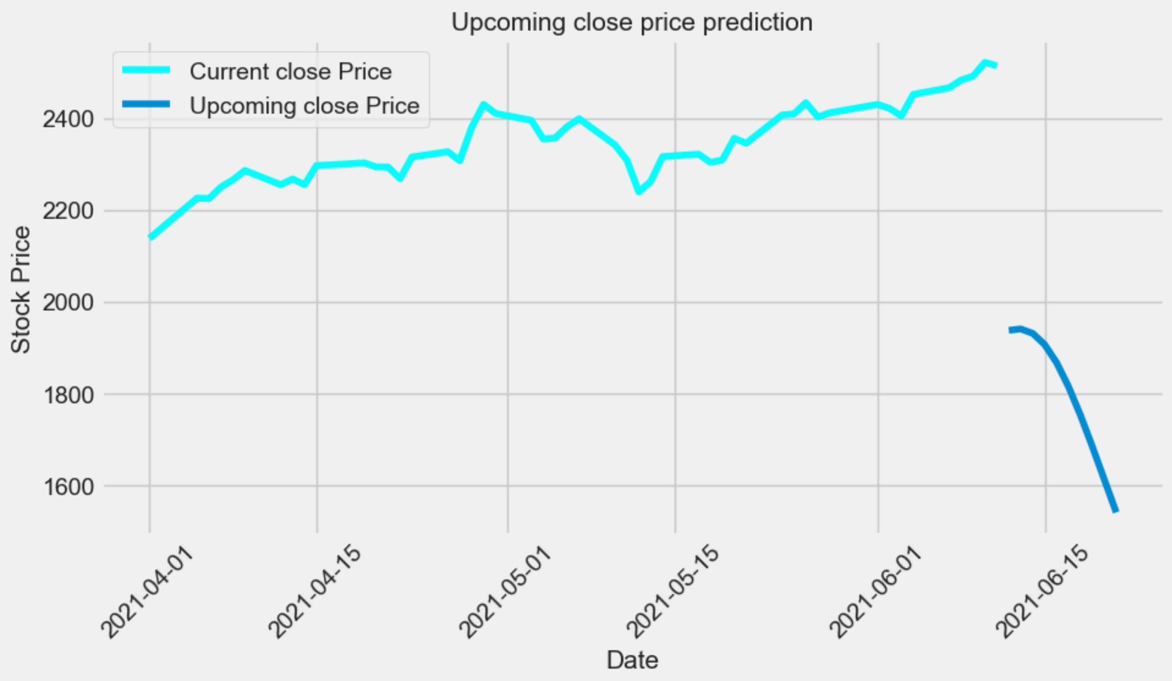


In [64]:
```
fg,ax=plt.subplots(figsize=(10,5))
ax.plot(merge_data_2.loc['2021-04-01':,'open'],label='Current Open Price',color=
ax.plot(upcoming_prediction.loc['2021-04-01':,'open'],label='Upcoming Open Price
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming Open price prediction',size=15)
ax.legend()
fg.show()
plt.show()
```



In [65]:
```
fg,ax=plt.subplots(figsize=(10,5))
ax.plot(merge_data_2.loc['2021-04-01':,'close'],label='Current close Price',colo
ax.plot(upcoming_prediction.loc['2021-04-01':,'close'],label='Upcoming close Pri
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming close price prediction',size=15)
```

```
ax.legend()
fg.show()
plt.show()
```



Completed

Thankyou so much for your atention.

In [ ]: