V.Harish-data-science-intern-task-1

July 31, 2023

[1]:
```
#importing libraries to be used
import numpy as np # for linear algebra
import pandas as pd # data preprocessing
import matplotlib.pyplot as plt # data visualization library
import seaborn as sns # data visualization library
%matplotlib inline
import warnings
warnings.filterwarnings("ignore") # ignore warnings

from sklearn.preprocessing import MinMaxScaler # for normalization
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, Bidirectional
```

[5]:
```
df = pd.read_csv("/content/drive/MyDrive/Dataset/GOOG .csv") # data_importing
df.head(10) # fetching first 10 rows of dataset
```

[5]:

	symbol	date	close	high	low	open	\
0	GOOG	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	
1	GOOG	2016-06-15 00:00:00+00:00	718.92	722.98	717.3100	719.00	
2	GOOG	2016-06-16 00:00:00+00:00	710.36	716.65	703.2600	714.91	
3	GOOG	2016-06-17 00:00:00+00:00	691.72	708.82	688.4515	708.65	
4	GOOG	2016-06-20 00:00:00+00:00	693.71	702.48	693.4100	698.77	
5	GOOG	2016-06-21 00:00:00+00:00	695.94	702.77	692.0100	698.40	
6	GOOG	2016-06-22 00:00:00+00:00	697.46	700.86	693.0819	699.06	
7	GOOG	2016-06-23 00:00:00+00:00	701.87	701.95	687.0000	697.45	
8	GOOG	2016-06-24 00:00:00+00:00	675.22	689.40	673.4500	675.17	
9	GOOG	2016-06-27 00:00:00+00:00	668.26	672.30	663.2840	671.00	

	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	\
0	1306065	718.27	722.47	713.1200	716.48	1306065	0.0	
1	1214517	718.92	722.98	717.3100	719.00	1214517	0.0	
2	1982471	710.36	716.65	703.2600	714.91	1982471	0.0	
3	3402357	691.72	708.82	688.4515	708.65	3402357	0.0	
4	2082538	693.71	702.48	693.4100	698.77	2082538	0.0	
5	1465634	695.94	702.77	692.0100	698.40	1465634	0.0	
6	1184318	697.46	700.86	693.0819	699.06	1184318	0.0	
7	2171415	701.87	701.95	687.0000	697.45	2171415	0.0	

```
8  4449022   675.22   689.40  673.4500   675.17    4449022       0.0
9  2641085   668.26   672.30  663.2840   671.00    2641085       0.0

   splitFactor
0          1.0
1          1.0
2          1.0
3          1.0
4          1.0
5          1.0
6          1.0
7          1.0
8          1.0
9          1.0
```

[6]:
```
# shape of data
print("Shape of data:",df.shape)
```

```
Shape of data: (1258, 14)
```

[7]:
```
# statistical description of data
df.describe()
```

[7]:

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

	adjClose	adjHigh	adjLow	adjOpen	adjVolume	\
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	
mean	1216.317067	1227.430936	1204.176436	1215.260779	1.601590e+06	
std	383.333358	387.570873	378.777099	382.446995	6.960172e+05	
min	668.260000	672.300000	663.284000	671.000000	3.467530e+05	
25%	960.802500	968.757500	952.182500	959.005000	1.173522e+06	
50%	1132.460000	1143.935000	1117.915000	1131.150000	1.412588e+06	
75%	1360.595000	1374.345000	1348.557500	1361.075000	1.812156e+06	
max	2521.600000	2526.990000	2498.290000	2524.920000	6.207027e+06	

	divCash	splitFactor
count	1258.0	1258.0
mean	0.0	1.0
std	0.0	0.0

min	0.0	1.0
25%	0.0	1.0
50%	0.0	1.0
75%	0.0	1.0
max	0.0	1.0

<google.colab._quickchart_helpers.SectionTitle at 0x7b92df122c80>

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_8454858346676847654, *['close'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_8454858346676847654, *['high'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
```

```
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_8454858346676847654, *['low'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_8454858346676847654, *['open'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7b92dcff1ab0>
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_8454858346676847654, *['close'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_8454858346676847654, *['high'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_8454858346676847654, *['low'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_8454858346676847654, *['open'], **{})
```

```
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7b92dcd69d80>
```

```
import numpy as np
from google.colab import autoviz
df_8454858346676847654 = autoviz.get_df('df_8454858346676847654')

def scatter_plots(df, colname_pairs, scatter_plot_size=2.5, size=8, alpha=.6):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(len(colname_pairs) * scatter_plot_size, scatter_plot_size))
  for plot_i, (x_colname, y_colname) in enumerate(colname_pairs, start=1):
    ax = plt.subplot(1, len(colname_pairs), plot_i)
    ax.scatter(df[x_colname], df[y_colname], s=size, alpha=alpha)
    plt.xlabel(x_colname)
    plt.ylabel(y_colname)
    ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plots(df_8454858346676847654, *[[['close', 'high'], ['high',␣
↪'low'], ['low', 'open'], ['open', 'volume']]], **{})
chart
```

[8]:
```
# summary of data
df.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
```

[9]:
```
# checking null values
df.isnull().sum()
```

[9]:
```
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
```

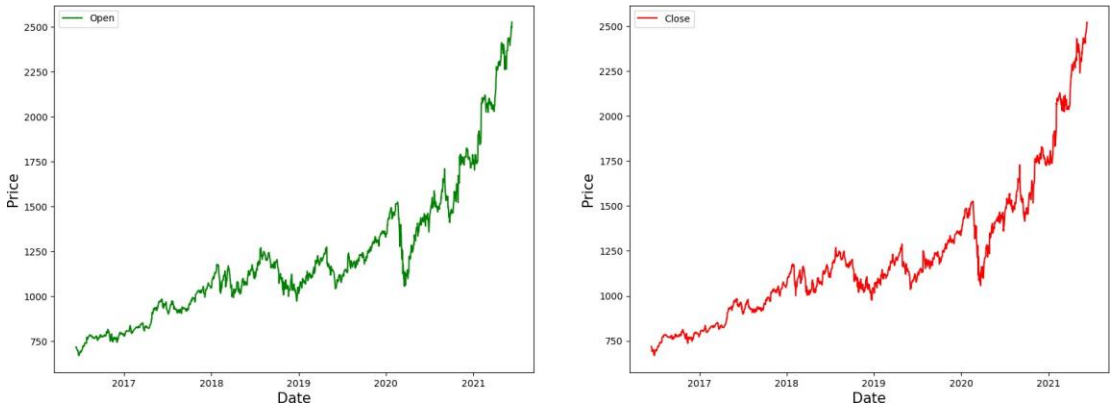
[10]:
```
df = df[['date','open','close']] # Extracting required columns
df['date'] = pd.to_datetime(df['date'].apply(lambda x: x.split()[0])) #␣
 ↪converting object dtype of date column to datetime dtype
df.set_index('date',drop=True,inplace=True) # Setting date column as index
df.head(10)
```

[10]:

date	open	close
2016-06-14	716.48	718.27
2016-06-15	719.00	718.92
2016-06-16	714.91	710.36
2016-06-17	708.65	691.72
2016-06-20	698.77	693.71
2016-06-21	698.40	695.94
2016-06-22	699.06	697.46
2016-06-23	697.45	701.87
2016-06-24	675.17	675.22
2016-06-27	671.00	668.26

[11]:
```
# plotting open and closing price on date index
fig, ax =plt.subplots(1,2,figsize=(20,7))
ax[0].plot(df['open'],label='Open',color='green')
ax[0].set_xlabel('Date',size=15)
ax[0].set_ylabel('Price',size=15)
ax[0].legend()

ax[1].plot(df['close'],label='Close',color='red')
ax[1].set_xlabel('Date',size=15)
```

```
ax[1].set_ylabel('Price',size=15)
ax[1].legend()

fig.show()
```



[12]:
```
# normalizing all the values of all columns using MinMaxScaler
MMS = MinMaxScaler()
df[df.columns] = MMS.fit_transform(df)
df.head(10)
```

[12]:

date	open	close
2016-06-14	0.024532	0.026984
2016-06-15	0.025891	0.027334
2016-06-16	0.023685	0.022716
2016-06-17	0.020308	0.012658
2016-06-20	0.014979	0.013732
2016-06-21	0.014779	0.014935
2016-06-22	0.015135	0.015755
2016-06-23	0.014267	0.018135
2016-06-24	0.002249	0.003755
2016-06-27	0.000000	0.000000

[13]:
```
# splitting the data into training and test set
training_size = round(len(df) * 0.75) # Selecting 75 % for training and 25 %
↪for testing
training_size
```

[13]: 944

[14]:
```
train_data = df[:training_size]
test_data  = df[training_size:]
```

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

[14]: ((944, 2), (314, 2))

[15]:
```
# 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))
```

[16]:
```
train_seq, train_label = create_sequence(train_data)
test_seq, test_label = create_sequence(test_data)
train_seq.shape, train_label.shape, test_seq.shape, test_label.shape
```

[16]: ((894, 50, 2), (894, 2), (264, 50, 2), (264, 2))

[17]:
```
# imported Sequential from keras.models
model = Sequential()
# importing Dense, Dropout, LSTM, Bidirectional from keras.layers
model.add(LSTM(units=50, return_sequences=True, input_shape = (train_seq.
 ↪shape[1], train_seq.shape[2])))

model.add(Dropout(0.1))
model.add(LSTM(units=50))

model.add(Dense(2))

model.compile(loss='mean_squared_error', optimizer='adam',␣
 ↪metrics=['mean_absolute_error'])

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50, 50)	10600

dropout (Dropout)	(None, 50, 50)	0
lstm_1 (LSTM)	(None, 50)	20200
dense (Dense)	(None, 2)	102

===
Total params: 30,902
Trainable params: 30,902
Non-trainable params: 0
____

[18]:
```
# fitting the model by iterating the dataset over 100 times(100 epochs)
model.fit(train_seq, train_label, epochs=100,validation_data=(test_seq,
↪test_label), verbose=1)
```

```
Epoch 1/100
28/28 [==============================] - 5s 73ms/step - loss: 0.0070 -
mean_absolute_error: 0.0597 - val_loss: 0.0155 - val_mean_absolute_error: 0.1008
Epoch 2/100
28/28 [==============================] - 1s 42ms/step - loss: 6.9591e-04 -
mean_absolute_error: 0.0209 - val_loss: 0.0064 - val_mean_absolute_error: 0.0646
Epoch 3/100
28/28 [==============================] - 1s 52ms/step - loss: 4.4913e-04 -
mean_absolute_error: 0.0154 - val_loss: 0.0039 - val_mean_absolute_error: 0.0483
Epoch 4/100
28/28 [==============================] - 1s 49ms/step - loss: 4.3055e-04 -
mean_absolute_error: 0.0150 - val_loss: 0.0055 - val_mean_absolute_error: 0.0597
Epoch 5/100
28/28 [==============================] - 1s 38ms/step - loss: 4.1487e-04 -
mean_absolute_error: 0.0150 - val_loss: 0.0040 - val_mean_absolute_error: 0.0490
Epoch 6/100
28/28 [==============================] - 1s 39ms/step - loss: 4.0929e-04 -
mean_absolute_error: 0.0148 - val_loss: 0.0055 - val_mean_absolute_error: 0.0592
Epoch 7/100
28/28 [==============================] - 1s 39ms/step - loss: 4.1738e-04 -
mean_absolute_error: 0.0149 - val_loss: 0.0065 - val_mean_absolute_error: 0.0658
Epoch 8/100
28/28 [==============================] - 1s 38ms/step - loss: 4.0575e-04 -
mean_absolute_error: 0.0148 - val_loss: 0.0031 - val_mean_absolute_error: 0.0420
Epoch 9/100
28/28 [==============================] - 1s 38ms/step - loss: 3.8307e-04 -
mean_absolute_error: 0.0145 - val_loss: 0.0036 - val_mean_absolute_error: 0.0455
Epoch 10/100
28/28 [==============================] - 1s 39ms/step - loss: 3.8019e-04 -
mean_absolute_error: 0.0143 - val_loss: 0.0044 - val_mean_absolute_error: 0.0520
Epoch 11/100
28/28 [==============================] - 1s 39ms/step - loss: 3.7979e-04 -
```

mean_absolute_error: 0.0142 - val_loss: 0.0074 - val_mean_absolute_error: 0.0723
Epoch 12/100
28/28 [==============================] - 1s 39ms/step - loss: 4.0588e-04 -
mean_absolute_error: 0.0149 - val_loss: 0.0049 - val_mean_absolute_error: 0.0552
Epoch 13/100
28/28 [==============================] - 1s 40ms/step - loss: 3.9518e-04 -
mean_absolute_error: 0.0146 - val_loss: 0.0027 - val_mean_absolute_error: 0.0381
Epoch 14/100
28/28 [==============================] - 1s 53ms/step - loss: 3.6978e-04 -
mean_absolute_error: 0.0141 - val_loss: 0.0036 - val_mean_absolute_error: 0.0463
Epoch 15/100
28/28 [==============================] - 1s 52ms/step - loss: 3.3517e-04 -
mean_absolute_error: 0.0134 - val_loss: 0.0026 - val_mean_absolute_error: 0.0375
Epoch 16/100
28/28 [==============================] - 1s 38ms/step - loss: 3.9181e-04 -
mean_absolute_error: 0.0144 - val_loss: 0.0043 - val_mean_absolute_error: 0.0521
Epoch 17/100
28/28 [==============================] - 1s 36ms/step - loss: 3.3263e-04 -
mean_absolute_error: 0.0133 - val_loss: 0.0034 - val_mean_absolute_error: 0.0451
Epoch 18/100
28/28 [==============================] - 1s 38ms/step - loss: 3.3538e-04 -
mean_absolute_error: 0.0134 - val_loss: 0.0037 - val_mean_absolute_error: 0.0482
Epoch 19/100
28/28 [==============================] - 1s 38ms/step - loss: 3.0465e-04 -
mean_absolute_error: 0.0127 - val_loss: 0.0023 - val_mean_absolute_error: 0.0357
Epoch 20/100
28/28 [==============================] - 1s 38ms/step - loss: 3.1931e-04 -
mean_absolute_error: 0.0130 - val_loss: 0.0030 - val_mean_absolute_error: 0.0422
Epoch 21/100
28/28 [==============================] - 1s 48ms/step - loss: 3.0678e-04 -
mean_absolute_error: 0.0128 - val_loss: 0.0052 - val_mean_absolute_error: 0.0599
Epoch 22/100
28/28 [==============================] - 1s 49ms/step - loss: 3.1832e-04 -
mean_absolute_error: 0.0133 - val_loss: 0.0052 - val_mean_absolute_error: 0.0596
Epoch 23/100
28/28 [==============================] - 1s 38ms/step - loss: 2.9412e-04 -
mean_absolute_error: 0.0127 - val_loss: 0.0032 - val_mean_absolute_error: 0.0444
Epoch 24/100
28/28 [==============================] - 2s 74ms/step - loss: 2.7506e-04 -
mean_absolute_error: 0.0121 - val_loss: 0.0033 - val_mean_absolute_error: 0.0447
Epoch 25/100
28/28 [==============================] - 2s 54ms/step - loss: 2.7607e-04 -
mean_absolute_error: 0.0121 - val_loss: 0.0057 - val_mean_absolute_error: 0.0621
Epoch 26/100
28/28 [==============================] - 1s 39ms/step - loss: 2.6551e-04 -
mean_absolute_error: 0.0118 - val_loss: 0.0025 - val_mean_absolute_error: 0.0372
Epoch 27/100
28/28 [==============================] - 1s 38ms/step - loss: 2.6288e-04 -

mean_absolute_error: 0.0121 - val_loss: 0.0018 - val_mean_absolute_error: 0.0311
Epoch 28/100
28/28 [==============================] - 1s 38ms/step - loss: 3.0022e-04 -
mean_absolute_error: 0.0127 - val_loss: 0.0024 - val_mean_absolute_error: 0.0363
Epoch 29/100
28/28 [==============================] - 1s 38ms/step - loss: 2.5981e-04 -
mean_absolute_error: 0.0118 - val_loss: 0.0039 - val_mean_absolute_error: 0.0494
Epoch 30/100
28/28 [==============================] - 1s 38ms/step - loss: 2.5348e-04 -
mean_absolute_error: 0.0118 - val_loss: 0.0029 - val_mean_absolute_error: 0.0403
Epoch 31/100
28/28 [==============================] - 1s 39ms/step - loss: 2.5479e-04 -
mean_absolute_error: 0.0118 - val_loss: 0.0035 - val_mean_absolute_error: 0.0465
Epoch 32/100
28/28 [==============================] - 1s 37ms/step - loss: 2.4600e-04 -
mean_absolute_error: 0.0115 - val_loss: 0.0027 - val_mean_absolute_error: 0.0396
Epoch 33/100
28/28 [==============================] - 1s 37ms/step - loss: 2.4682e-04 -
mean_absolute_error: 0.0117 - val_loss: 0.0034 - val_mean_absolute_error: 0.0449
Epoch 34/100
28/28 [==============================] - 1s 38ms/step - loss: 2.4333e-04 -
mean_absolute_error: 0.0115 - val_loss: 0.0034 - val_mean_absolute_error: 0.0453
Epoch 35/100
28/28 [==============================] - 1s 45ms/step - loss: 2.2664e-04 -
mean_absolute_error: 0.0111 - val_loss: 0.0043 - val_mean_absolute_error: 0.0523
Epoch 36/100
28/28 [==============================] - 1s 51ms/step - loss: 2.5588e-04 -
mean_absolute_error: 0.0117 - val_loss: 0.0044 - val_mean_absolute_error: 0.0534
Epoch 37/100
28/28 [==============================] - 1s 46ms/step - loss: 2.8117e-04 -
mean_absolute_error: 0.0123 - val_loss: 0.0044 - val_mean_absolute_error: 0.0538
Epoch 38/100
28/28 [==============================] - 1s 37ms/step - loss: 2.3647e-04 -
mean_absolute_error: 0.0113 - val_loss: 0.0026 - val_mean_absolute_error: 0.0388
Epoch 39/100
28/28 [==============================] - 1s 37ms/step - loss: 2.3214e-04 -
mean_absolute_error: 0.0114 - val_loss: 0.0035 - val_mean_absolute_error: 0.0457
Epoch 40/100
28/28 [==============================] - 1s 38ms/step - loss: 2.2272e-04 -
mean_absolute_error: 0.0109 - val_loss: 0.0039 - val_mean_absolute_error: 0.0497
Epoch 41/100
28/28 [==============================] - 1s 38ms/step - loss: 2.3288e-04 -
mean_absolute_error: 0.0113 - val_loss: 0.0018 - val_mean_absolute_error: 0.0310
Epoch 42/100
28/28 [==============================] - 1s 39ms/step - loss: 2.3720e-04 -
mean_absolute_error: 0.0114 - val_loss: 0.0029 - val_mean_absolute_error: 0.0420
Epoch 43/100
28/28 [==============================] - 1s 38ms/step - loss: 2.2844e-04 -

mean_absolute_error: 0.0111 - val_loss: 0.0044 - val_mean_absolute_error: 0.0547
Epoch 44/100
28/28 [==============================] - 1s 38ms/step - loss: 2.2612e-04 -
mean_absolute_error: 0.0110 - val_loss: 0.0024 - val_mean_absolute_error: 0.0371
Epoch 45/100
28/28 [==============================] - 1s 38ms/step - loss: 2.1131e-04 -
mean_absolute_error: 0.0105 - val_loss: 0.0021 - val_mean_absolute_error: 0.0342
Epoch 46/100
28/28 [==============================] - 1s 40ms/step - loss: 2.1178e-04 -
mean_absolute_error: 0.0105 - val_loss: 0.0023 - val_mean_absolute_error: 0.0362
Epoch 47/100
28/28 [==============================] - 1s 50ms/step - loss: 2.2767e-04 -
mean_absolute_error: 0.0110 - val_loss: 0.0023 - val_mean_absolute_error: 0.0363
Epoch 48/100
28/28 [==============================] - 1s 52ms/step - loss: 2.3243e-04 -
mean_absolute_error: 0.0113 - val_loss: 0.0040 - val_mean_absolute_error: 0.0505
Epoch 49/100
28/28 [==============================] - 1s 37ms/step - loss: 2.1314e-04 -
mean_absolute_error: 0.0107 - val_loss: 0.0036 - val_mean_absolute_error: 0.0467
Epoch 50/100
28/28 [==============================] - 1s 38ms/step - loss: 1.9676e-04 -
mean_absolute_error: 0.0103 - val_loss: 0.0025 - val_mean_absolute_error: 0.0386
Epoch 51/100
28/28 [==============================] - 1s 38ms/step - loss: 2.1936e-04 -
mean_absolute_error: 0.0109 - val_loss: 0.0024 - val_mean_absolute_error: 0.0371
Epoch 52/100
28/28 [==============================] - 1s 38ms/step - loss: 1.9640e-04 -
mean_absolute_error: 0.0103 - val_loss: 0.0022 - val_mean_absolute_error: 0.0345
Epoch 53/100
28/28 [==============================] - 1s 39ms/step - loss: 1.9537e-04 -
mean_absolute_error: 0.0101 - val_loss: 0.0029 - val_mean_absolute_error: 0.0423
Epoch 54/100
28/28 [==============================] - 1s 38ms/step - loss: 2.0339e-04 -
mean_absolute_error: 0.0103 - val_loss: 0.0024 - val_mean_absolute_error: 0.0386
Epoch 55/100
28/28 [==============================] - 1s 37ms/step - loss: 2.0150e-04 -
mean_absolute_error: 0.0103 - val_loss: 0.0019 - val_mean_absolute_error: 0.0319
Epoch 56/100
28/28 [==============================] - 1s 37ms/step - loss: 1.8074e-04 -
mean_absolute_error: 0.0097 - val_loss: 0.0026 - val_mean_absolute_error: 0.0403
Epoch 57/100
28/28 [==============================] - 1s 39ms/step - loss: 2.0540e-04 -
mean_absolute_error: 0.0104 - val_loss: 0.0029 - val_mean_absolute_error: 0.0435
Epoch 58/100
28/28 [==============================] - 1s 46ms/step - loss: 1.9212e-04 -
mean_absolute_error: 0.0100 - val_loss: 0.0019 - val_mean_absolute_error: 0.0325
Epoch 59/100
28/28 [==============================] - 2s 56ms/step - loss: 1.7550e-04 -

mean_absolute_error: 0.0098 - val_loss: 0.0025 - val_mean_absolute_error: 0.0388
Epoch 60/100
28/28 [==============================] - 1s 40ms/step - loss: 1.9490e-04 -
mean_absolute_error: 0.0102 - val_loss: 0.0037 - val_mean_absolute_error: 0.0507
Epoch 61/100
28/28 [==============================] - 1s 38ms/step - loss: 1.8077e-04 -
mean_absolute_error: 0.0097 - val_loss: 0.0024 - val_mean_absolute_error: 0.0382
Epoch 62/100
28/28 [==============================] - 1s 40ms/step - loss: 2.0307e-04 -
mean_absolute_error: 0.0103 - val_loss: 0.0017 - val_mean_absolute_error: 0.0306
Epoch 63/100
28/28 [==============================] - 1s 37ms/step - loss: 1.6272e-04 -
mean_absolute_error: 0.0092 - val_loss: 0.0015 - val_mean_absolute_error: 0.0292
Epoch 64/100
28/28 [==============================] - 1s 37ms/step - loss: 1.6148e-04 -
mean_absolute_error: 0.0092 - val_loss: 0.0011 - val_mean_absolute_error: 0.0243
Epoch 65/100
28/28 [==============================] - 1s 37ms/step - loss: 1.9973e-04 -
mean_absolute_error: 0.0105 - val_loss: 0.0026 - val_mean_absolute_error: 0.0397
Epoch 66/100
28/28 [==============================] - 1s 37ms/step - loss: 1.6555e-04 -
mean_absolute_error: 0.0092 - val_loss: 0.0028 - val_mean_absolute_error: 0.0431
Epoch 67/100
28/28 [==============================] - 1s 38ms/step - loss: 1.8140e-04 -
mean_absolute_error: 0.0098 - val_loss: 0.0011 - val_mean_absolute_error: 0.0243
Epoch 68/100
28/28 [==============================] - 1s 39ms/step - loss: 1.5566e-04 -
mean_absolute_error: 0.0090 - val_loss: 0.0016 - val_mean_absolute_error: 0.0297
Epoch 69/100
28/28 [==============================] - 1s 44ms/step - loss: 1.5642e-04 -
mean_absolute_error: 0.0089 - val_loss: 0.0020 - val_mean_absolute_error: 0.0342
Epoch 70/100
28/28 [==============================] - 1s 53ms/step - loss: 1.5181e-04 -
mean_absolute_error: 0.0089 - val_loss: 0.0012 - val_mean_absolute_error: 0.0250
Epoch 71/100
28/28 [==============================] - 1s 45ms/step - loss: 1.5722e-04 -
mean_absolute_error: 0.0092 - val_loss: 0.0012 - val_mean_absolute_error: 0.0253
Epoch 72/100
28/28 [==============================] - 1s 38ms/step - loss: 1.5028e-04 -
mean_absolute_error: 0.0088 - val_loss: 0.0031 - val_mean_absolute_error: 0.0448
Epoch 73/100
28/28 [==============================] - 1s 38ms/step - loss: 1.5013e-04 -
mean_absolute_error: 0.0087 - val_loss: 0.0020 - val_mean_absolute_error: 0.0336
Epoch 74/100
28/28 [==============================] - 1s 40ms/step - loss: 1.5820e-04 -
mean_absolute_error: 0.0090 - val_loss: 0.0016 - val_mean_absolute_error: 0.0295
Epoch 75/100
28/28 [==============================] - 1s 40ms/step - loss: 1.4808e-04 -

mean_absolute_error: 0.0089 - val_loss: 0.0016 - val_mean_absolute_error: 0.0297
Epoch 76/100
28/28 [==============================] - 1s 38ms/step - loss: 1.4740e-04 -
mean_absolute_error: 0.0088 - val_loss: 0.0021 - val_mean_absolute_error: 0.0354
Epoch 77/100
28/28 [==============================] - 1s 38ms/step - loss: 1.4434e-04 -
mean_absolute_error: 0.0086 - val_loss: 9.5732e-04 - val_mean_absolute_error:
0.0227
Epoch 78/100
28/28 [==============================] - 1s 38ms/step - loss: 1.5803e-04 -
mean_absolute_error: 0.0090 - val_loss: 0.0013 - val_mean_absolute_error: 0.0262
Epoch 79/100
28/28 [==============================] - 1s 39ms/step - loss: 1.5157e-04 -
mean_absolute_error: 0.0088 - val_loss: 0.0014 - val_mean_absolute_error: 0.0278
Epoch 80/100
28/28 [==============================] - 1s 42ms/step - loss: 1.4757e-04 -
mean_absolute_error: 0.0088 - val_loss: 0.0025 - val_mean_absolute_error: 0.0400
Epoch 81/100
28/28 [==============================] - 1s 51ms/step - loss: 1.4949e-04 -
mean_absolute_error: 0.0089 - val_loss: 0.0012 - val_mean_absolute_error: 0.0254
Epoch 82/100
28/28 [==============================] - 1s 51ms/step - loss: 1.3308e-04 -
mean_absolute_error: 0.0083 - val_loss: 0.0030 - val_mean_absolute_error: 0.0434
Epoch 83/100
28/28 [==============================] - 1s 38ms/step - loss: 1.3501e-04 -
mean_absolute_error: 0.0082 - val_loss: 0.0019 - val_mean_absolute_error: 0.0329
Epoch 84/100
28/28 [==============================] - 1s 39ms/step - loss: 1.3702e-04 -
mean_absolute_error: 0.0086 - val_loss: 0.0018 - val_mean_absolute_error: 0.0318
Epoch 85/100
28/28 [==============================] - 1s 38ms/step - loss: 1.3023e-04 -
mean_absolute_error: 0.0081 - val_loss: 0.0024 - val_mean_absolute_error: 0.0382
Epoch 86/100
28/28 [==============================] - 1s 38ms/step - loss: 1.2756e-04 -
mean_absolute_error: 0.0080 - val_loss: 0.0025 - val_mean_absolute_error: 0.0389
Epoch 87/100
28/28 [==============================] - 1s 39ms/step - loss: 1.3654e-04 -
mean_absolute_error: 0.0084 - val_loss: 0.0028 - val_mean_absolute_error: 0.0416
Epoch 88/100
28/28 [==============================] - 1s 38ms/step - loss: 1.4430e-04 -
mean_absolute_error: 0.0087 - val_loss: 0.0014 - val_mean_absolute_error: 0.0268
Epoch 89/100
28/28 [==============================] - 1s 41ms/step - loss: 1.3139e-04 -
mean_absolute_error: 0.0083 - val_loss: 0.0016 - val_mean_absolute_error: 0.0295
Epoch 90/100
28/28 [==============================] - 1s 38ms/step - loss: 1.2766e-04 -
mean_absolute_error: 0.0082 - val_loss: 0.0013 - val_mean_absolute_error: 0.0267
Epoch 91/100

```
28/28 [==============================] - 1s 42ms/step - loss: 1.2961e-04 -
mean_absolute_error: 0.0082 - val_loss: 0.0014 - val_mean_absolute_error: 0.0280
Epoch 92/100
28/28 [==============================] - 1s 53ms/step - loss: 1.2139e-04 -
mean_absolute_error: 0.0081 - val_loss: 0.0016 - val_mean_absolute_error: 0.0300
Epoch 93/100
28/28 [==============================] - 1s 50ms/step - loss: 1.2566e-04 -
mean_absolute_error: 0.0079 - val_loss: 0.0015 - val_mean_absolute_error: 0.0289
Epoch 94/100
28/28 [==============================] - 1s 40ms/step - loss: 1.2364e-04 -
mean_absolute_error: 0.0082 - val_loss: 0.0016 - val_mean_absolute_error: 0.0296
Epoch 95/100
28/28 [==============================] - 1s 39ms/step - loss: 1.2415e-04 -
mean_absolute_error: 0.0080 - val_loss: 0.0016 - val_mean_absolute_error: 0.0296
Epoch 96/100
28/28 [==============================] - 1s 38ms/step - loss: 1.1602e-04 -
mean_absolute_error: 0.0076 - val_loss: 0.0020 - val_mean_absolute_error: 0.0339
Epoch 97/100
28/28 [==============================] - 1s 39ms/step - loss: 1.1907e-04 -
mean_absolute_error: 0.0078 - val_loss: 0.0018 - val_mean_absolute_error: 0.0311
Epoch 98/100
28/28 [==============================] - 1s 39ms/step - loss: 1.2545e-04 -
mean_absolute_error: 0.0080 - val_loss: 0.0019 - val_mean_absolute_error: 0.0326
Epoch 99/100
28/28 [==============================] - 1s 38ms/step - loss: 1.2563e-04 -
mean_absolute_error: 0.0079 - val_loss: 0.0015 - val_mean_absolute_error: 0.0290
Epoch 100/100
28/28 [==============================] - 1s 39ms/step - loss: 1.3376e-04 -
mean_absolute_error: 0.0084 - val_loss: 0.0028 - val_mean_absolute_error: 0.0426
```

[18]:
```
<keras.callbacks.History at 0x7b92dc653970>
```

[19]:
```
# predicting the values after running the model
test_predicted = model.predict(test_seq)
test_predicted[:5]
```

```
9/9 [==============================] - 1s 8ms/step
```

[19]:
```
array([[0.3925917 , 0.3948203 ],
       [0.39278576, 0.39529413],
       [0.3889445 , 0.39180565],
       [0.3916219 , 0.3940799 ],
       [0.39539546, 0.3975677 ]], dtype=float32)
```

[20]:
```
# Inversing normalization/scaling on predicted data
test_inverse_predicted = MMS.inverse_transform(test_predicted)
test_inverse_predicted[:5]
```

[20]:
```
array([[1398.8336, 1399.9962],
       [1399.1934, 1400.8745],
       [1392.072 , 1394.4092],
       [1397.0356, 1398.624 ],
       [1404.0315, 1405.0881]], dtype=float32)
```

[21]:
```
# Merging actual and predicted data for better visualization
df_merge = pd.concat([df.iloc[-264:].copy(),
                      pd.
↪DataFrame(test_inverse_predicted,columns=['open_predicted','close_predicted'],
                                          index=df.iloc[-264:].index)], axis=1)
```

[22]:
```
# Inversing normalization/scaling
df_merge[['open','close']] = MMS.inverse_transform(df_merge[['open','close']])
df_merge.head()
```

date	open	close	open_predicted	close_predicted
2020-05-27	1417.25	1417.84	1398.833618	1399.996216
2020-05-28	1396.86	1416.73	1399.193359	1400.874512
2020-05-29	1416.94	1428.92	1392.072021	1394.409180
2020-06-01	1418.39	1431.82	1397.035645	1398.624023
2020-06-02	1430.55	1439.22	1404.031494	1405.088135

<google.colab._quickchart_helpers.SectionTitle at 0x7b92dcd87d00>

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2868927680624221977, *['open'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')
```

```
def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2868927680624221977, *['close'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2868927680624221977, *['open_predicted'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2868927680624221977, *['close_predicted'], **{})
chart
```

<google.colab._quickchart_helpers.SectionTitle at 0x7b92dcee9a80>

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_2868927680624221977, *['open'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_2868927680624221977, *['close'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()
```

```
chart = histogram(df_2868927680624221977, *['open_predicted'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_2868927680624221977, *['close_predicted'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7b92dceb1240>
```

```
import numpy as np
from google.colab import autoviz
df_2868927680624221977 = autoviz.get_df('df_2868927680624221977')

def scatter_plots(df, colname_pairs, scatter_plot_size=2.5, size=8, alpha=.6):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(len(colname_pairs) * scatter_plot_size, scatter_plot_size))
  for plot_i, (x_colname, y_colname) in enumerate(colname_pairs, start=1):
    ax = plt.subplot(1, len(colname_pairs), plot_i)
    ax.scatter(df[x_colname], df[y_colname], s=size, alpha=alpha)
    plt.xlabel(x_colname)
    plt.ylabel(y_colname)
    ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plots(df_2868927680624221977, *[[['open', 'close'], ['close',␣
↪'open_predicted'], ['open_predicted', 'close_predicted']]], **{})
chart
```

[23]:
```
# plotting the actual open and predicted open prices on date index
df_merge[["open","open_predicted"]].plot(figsize=(10,6))
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()
```

[24]:
```
# plotting the actual close and predicted close prices on date index
df_merge[["close","close_predicted"]].plot(figsize=(10,6))
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()
```

[25]:
```
# Creating a dataframe and adding 10 days to existing index

df_merge = df_merge.append(pd.DataFrame(columns=df_merge.columns,
                                        index=pd.date_range(start=df_merge.
  ↪index[-1], periods=11, freq='D', closed='right')))
df_merge['2021-06-09':'2021-06-16']
```

[25]:

	open	close	open_predicted	close_predicted
2021-06-09	2499.50	2491.40	2283.043457	2308.479004
2021-06-10	2494.01	2521.60	2288.935547	2315.539062
2021-06-11	2524.92	2513.93	2295.734131	2322.352783
2021-06-12	NaN	NaN	NaN	NaN
2021-06-13	NaN	NaN	NaN	NaN
2021-06-14	NaN	NaN	NaN	NaN
2021-06-15	NaN	NaN	NaN	NaN
2021-06-16	NaN	NaN	NaN	NaN

<google.colab._quickchart_helpers.SectionTitle at 0x7b92dcb7da50>

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
```

```
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2077258851996054484, *['open'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2077258851996054484, *['close'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2077258851996054484, *['open_predicted'], **{})
chart
```

```
import numpy as np
```

```
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def value_plot(df, y, sort_ascending=False, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  if sort_ascending:
    df = df.sort_values(y).reset_index(drop=True)
  _, ax = plt.subplots(figsize=figsize)
  df[y].plot(kind='line')
  plt.title(y)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_2077258851996054484, *['close_predicted'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7b92ca5342b0>
```

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_2077258851996054484, *['open'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()
```

```
chart = histogram(df_2077258851996054484, *['close'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_2077258851996054484, *['open_predicted'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def histogram(df, colname, num_bins=20, figsize=(2, 1)):
  from matplotlib import pyplot as plt
  _, ax = plt.subplots(figsize=figsize)
  plt.hist(df[colname], bins=num_bins, histtype='stepfilled')
  plt.ylabel('count')
  plt.title(colname)
  ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_2077258851996054484, *['close_predicted'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7b92ca303520>
```

```
import numpy as np
from google.colab import autoviz
df_2077258851996054484 = autoviz.get_df('df_2077258851996054484')

def scatter_plots(df, colname_pairs, scatter_plot_size=2.5, size=8, alpha=.6):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(len(colname_pairs) * scatter_plot_size, scatter_plot_size))
  for plot_i, (x_colname, y_colname) in enumerate(colname_pairs, start=1):
    ax = plt.subplot(1, len(colname_pairs), plot_i)
    ax.scatter(df[x_colname], df[y_colname], s=size, alpha=alpha)
```

```
    plt.xlabel(x_colname)
    plt.ylabel(y_colname)
    ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return  autoviz.MplChart.from_current_mpl_state()

chart = scatter_plots(df_2077258851996054484, *[[['open', 'close'], ['close',
 ↪'open_predicted'], ['open_predicted', 'close_predicted']]], **{})
chart
```

[26]:
```
# creating a DataFrame and filling values of open and close column
upcoming_prediction = pd.DataFrame(columns=['open','close'],index=df_merge.
 ↪index)
upcoming_prediction.index=pd.to_datetime(upcoming_prediction.index)
```

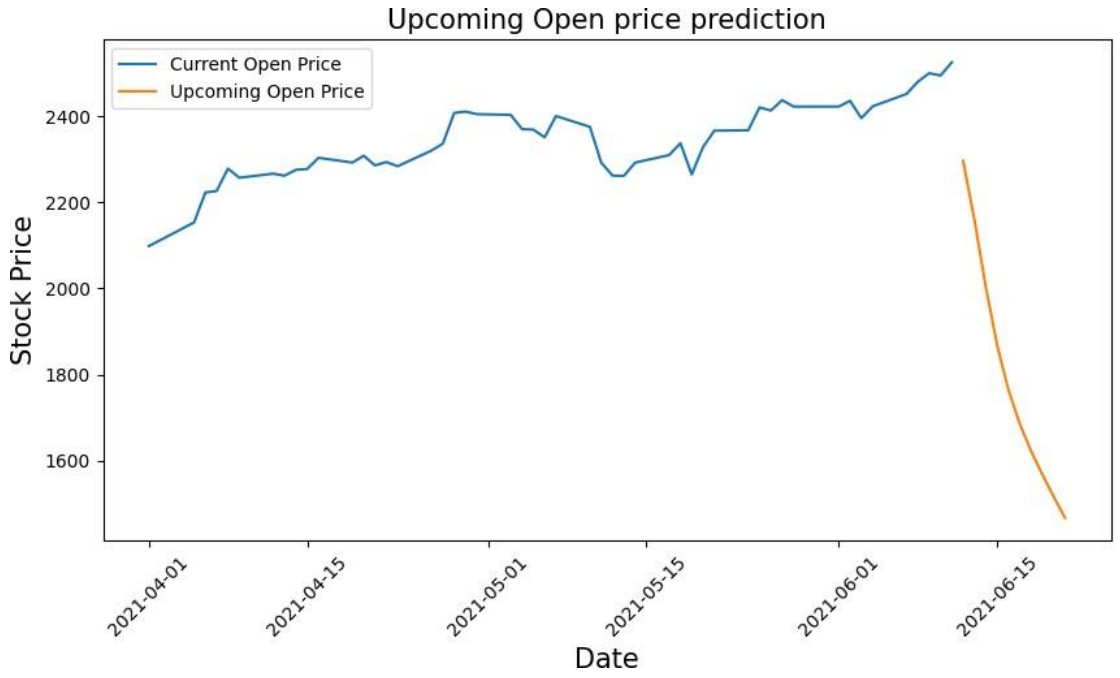
[27]:
```
curr_seq = test_seq[-1:]

for i in range(-10,0):
  up_pred = model.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(test_seq[-1:].shape)
```

```
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 50ms/step
```

[28]:
```
#  inversing  Normalization/scaling
upcoming_prediction[['open','close']] = MMS.
 ↪inverse_transform(upcoming_prediction[['open','close']])
```

[29]:
```
# plotting Upcoming Open price on date index
fig,ax=plt.subplots(figsize=(10,5))
ax.plot(df_merge.loc['2021-04-01':,'open'],label='Current Open Price')
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()
fig.show()
```



[30]:
```
# plotting Upcoming Close price on date index
fig,ax=plt.subplots(figsize=(10,5))
ax.plot(df_merge.loc["2021-04-01":,"close"],label="Current close Price")
ax.plot(upcoming_prediction.loc["2021-04-01":,"close"],label="Upcoming close␣
↪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 close price prediction",size=15)
ax.legend()
fig.show()
```


Upcoming close price prediction
Current close Price
Upcoming close Price
Stock Price
2400
2200
2000
1800
1600
2021-04-01
2021-04-15
2021-05-01
2021-05-15
2021-06-01
2021-06-15
Date

[ ]: