stock-market-prediction-task-2

October 10, 2023

1 Task 2 : Stock Market Prediction

2 Stock Market Prediction helps you determine the future value of company stock and other financial instruments traded on an exchange

[1]:
```
#Import the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
import plotly.graph_objs as go
from plotly.offline import plot
```

[3]:
```
from  plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
```

[4]:
```
tesla = pd.read_csv(r"C:\Users\hp\Desktop\datapreprocessing\tesla.csv")
```

[5]:
```
tesla.head()
```

[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	29-06-2010	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	30-06-2010	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	01-07-2010	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	02-07-2010	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	06-07-2010	20.000000	20.00	15.830000	16.110001	16.110001	6866900

[6]:
```
tesla.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2193 entries, 0 to 2192
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       2193 non-null   object
```

```
 1   Open       2193 non-null   float64
 2   High       2193 non-null   float64
 3   Low        2193 non-null   float64
 4   Close      2193 non-null   float64
 5   Adj Close  2193 non-null   float64
 6   Volume     2193 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 120.1+ KB
```

[7]:
```
tesla['Date'] = pd.to_datetime(tesla['Date'])
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_5280\3702129700.py:1: UserWarning:

Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was
specified. This may lead to inconsistently parsed dates! Specify a format to
ensure consistent parsing.
```

[8]:
```
print(f'Dataframe contains stock prices between {tesla.Date.min()} {tesla.Date.
 ↪max()}')
print(f'Total days = {(tesla.Date.max() - tesla.Date.min()).days} days')
```

```
Dataframe contains stock prices between 2010-01-07 00:00:00 2019-12-03 00:00:00
Total days = 3617 days
```

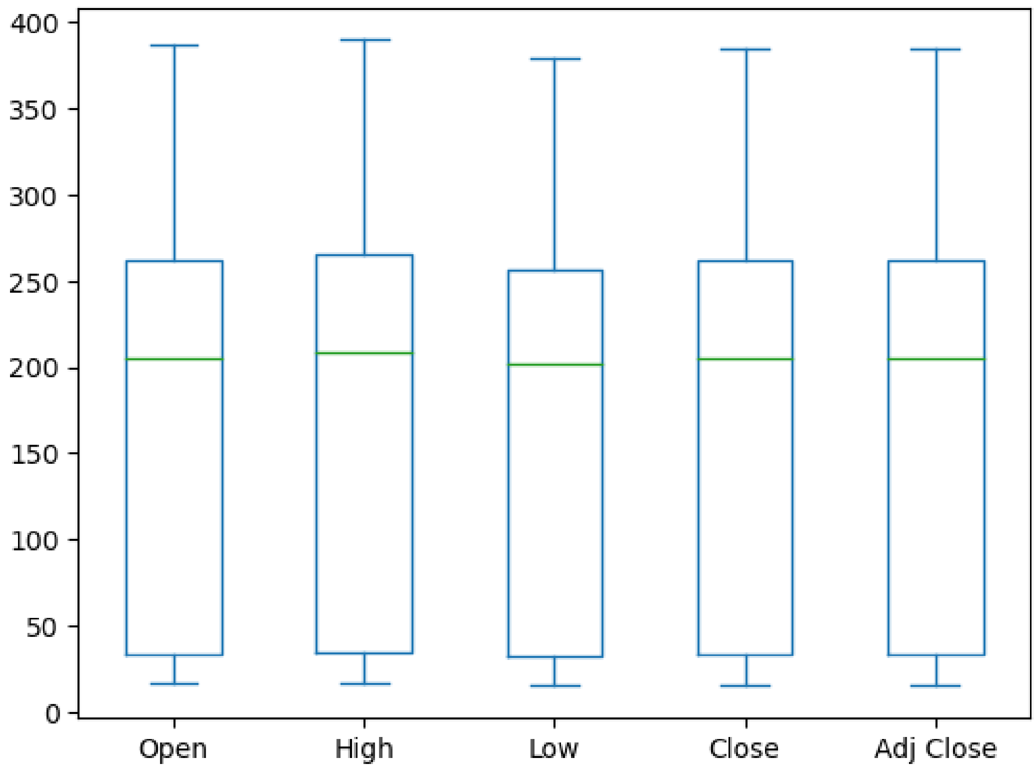
[9]:
```
tesla.describe()
```

[9]:
```
              Open         High          Low        Close    Adj Close  \
count  2193.000000  2193.000000  2193.000000  2193.000000  2193.000000
mean    175.652882   178.710262   172.412075   175.648555   175.648555
std     115.580903   117.370092   113.654794   115.580771   115.580771
min      16.139999    16.629999    14.980000    15.800000    15.800000
25%      33.110001    33.910000    32.459999    33.160000    33.160000
50%     204.990005   208.160004   201.669998   204.990005   204.990005
75%     262.000000   265.329987   256.209991   261.739990   261.739990
max     386.690002   389.609985   379.350006   385.000000   385.000000

             Volume
count  2.193000e+03
mean   5.077449e+06
std    4.545398e+06
min    1.185000e+05
25%    1.577800e+06
50%    4.171700e+06
75%    6.885600e+06
max    3.716390e+07
```

[10]:
```
tesla[['Open','High','Low','Close','Adj Close']].plot(kind='box')
```

[10]: <Axes: >



[11]:
```
# Setting the layout for our plot
layout = go.Layout(
    title='Stock Prices of Tesla',
    xaxis=dict(
        titlefont=dict(
            family='Courier New, monospace',
            size=18,
            color='#7f7f7f'
        )
    ),
    yaxis=dict(
        title='Price',
        titlefont=dict(
            family='Courier New, monosapce',
            size=18,
            color='#7f7f7f'
        )
    )
)
```

```
tesla_data = [{'x':tesla['Date'], 'y':tesla['Close']}]
plot = go.Figure(data=tesla_data, layout=layout)
```

[12]:
```
#plot(plot #plotting offline)
iplot(plot)
```

[13]:
```
# Building the regression model
from sklearn.model_selection import train_test_split

# for preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

#for model evaluation
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
```

[14]:
```
#Split the data into train and test sets
X = np.array(tesla.index).reshape(-1,1)
Y = tesla['Close']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
↪random_state=101)
```

[15]:
```
# Feature scaling
scaler = StandardScaler().fit(X_train)
```

[16]:
```
from sklearn.linear_model import LinearRegression
```

[17]:
```
#Creating a linear model
lm = LinearRegression()
lm.fit(X_train, Y_train)
```

[17]: LinearRegression()

[18]:
```
#Plot actual and predicted values for train dataset
trace0 = go.Scatter(
    x = X_train.T[0],
    y = Y_train,
    mode = 'markers',
    name = 'Actual'
)
trace1 = go.Scatter(
    x = X_train.T[0],
    y = lm.predict(X_train).T,
    mode = 'lines',
    name = 'Predicted'
)
```

```
tesla_data = [trace0,trace1]
layout.xaxis.title.text = 'Day'
plot2 = go.Figure(data=tesla_data, layout=layout)
```

[19]:
```
iplot(plot2)
```

[20]:
```
#Calculate scores for model evaluation
scores = f'''
{'Metric'.ljust(10)}{'Train'.center(20)}{'Test'.center(20)}
{'r2_score'.ljust(10)}{r2_score(Y_train, lm.
 ↪predict(X_train))}\t{r2_score(Y_test, lm.predict(X_test))}
{'MSE'.ljust(10)}{mse(Y_train, lm.predict(X_train))}\t{mse(Y_test, lm.
 ↪predict(X_test))}
'''
print(scores)
```

```
Metric              Train                 Test
r2_score   0.8658871776828707    0.8610649253244574
MSE        1821.3833862936174    1780.987539418845
```

3 LONG SHORT TERM MEMORY

[21]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense,LSTM,Dropout
```

[22]:
```
data = pd.read_csv(r"C:\Users\hp\Desktop\EDA\Google_train_data.csv")
```

[23]:
```
data.head()
```

[23]:

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

[24]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 6 columns):
```

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1258 non-null   object
 1   Open    1258 non-null   float64
 2   High    1258 non-null   float64
 3   Low     1258 non-null   float64
 4   Close   1258 non-null   object
 5   Volume  1258 non-null   object
dtypes: float64(3), object(3)
memory usage: 59.1+ KB
```

[25]:
```
data["Close"]=pd.to_numeric(data.Close,errors='coerce')
data = data.dropna()
trainData = data.iloc[:,4:5].values
```

[26]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1149 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1149 non-null   object
 1   Open    1149 non-null   float64
 2   High    1149 non-null   float64
 3   Low     1149 non-null   float64
 4   Close   1149 non-null   float64
 5   Volume  1149 non-null   object
dtypes: float64(4), object(2)
memory usage: 62.8+ KB
```

[27]:
```
sc = MinMaxScaler(feature_range=(0,1))
trainData = sc.fit_transform(trainData)
trainData.shape
```

[27]:
```
(1149, 1)
```

[28]:
```
X_train = []
y_train = []

for i in range (60,1149): #60 : timestep // 1149 : Length of the data
    X_train.append(trainData[i-60:i,0])
    y_train.append(trainData[i,0])

X_train,y_train = np.array(X_train),np.array(y_train)
```

[29]:
```
X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1)) #adding the␣
↪batch_size axis
X_train.shape
```

[29]: (1089, 60, 1)

[30]:
```
model = Sequential()

model.add(LSTM(units=100, return_sequences = True, input_shape =(X_train.
↪shape[1],1)))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = False))
model.add(Dropout(0.2))

model.add(Dense(units =1))
model.compile(optimizer='adam',loss="mean_squared_error")
```

[ ]:
```
hist = model.fit(X_train, y_train, epochs = 20, batch_size = 32, verbose=2)
```

```
Epoch 1/20
35/35 - 20s - loss: 0.0307 - 20s/epoch - 578ms/step
Epoch 2/20
35/35 - 10s - loss: 0.0083 - 10s/epoch - 283ms/step
Epoch 3/20
35/35 - 10s - loss: 0.0152 - 10s/epoch - 281ms/step
Epoch 4/20
35/35 - 10s - loss: 0.0071 - 10s/epoch - 292ms/step
Epoch 5/20
35/35 - 10s - loss: 0.0063 - 10s/epoch - 285ms/step
Epoch 6/20
35/35 - 10s - loss: 0.0060 - 10s/epoch - 294ms/step
Epoch 7/20
35/35 - 10s - loss: 0.0060 - 10s/epoch - 278ms/step
Epoch 8/20
35/35 - 10s - loss: 0.0057 - 10s/epoch - 285ms/step
Epoch 9/20
35/35 - 10s - loss: 0.0051 - 10s/epoch - 296ms/step
Epoch 10/20
35/35 - 10s - loss: 0.0047 - 10s/epoch - 293ms/step
Epoch 11/20
35/35 - 10s - loss: 0.0051 - 10s/epoch - 273ms/step
Epoch 12/20
```

[ ]:
```
plt.plot(hist.history['loss'])
plt.title('Training model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```

[ ]:
```
testData = pd.read_csv(r"C:\Users\hp\Desktop\EDA\Google_test_data.csv")
```

[ ]:
```
testData["Close"]=pd.to_numeric(testData.Close,errors='coerce')
testData = testData.dropna()
testData = testData.iloc[:,4:5].values
```

[ ]:
```
#input array for the model
inputClosing = testData.iloc[:,0:].values
inputClosing_scaled = sc.transform(inputClosing)
inputClosing_scaled.shape
X_test = []
length = len(testData)
timestep = 60
for i in range(timestep,length):
    X_test.append(inputClosing_scaled[i-timestep:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
X_test.shape
```

[ ]:
```
y_pred = model.predict(X_test)
```

[ ]:
```
predicted_price = sc.inverse_transform(y_pred)
```

[ ]:
```
plt.plot(y_test, color = 'red', label = 'Actual Stock Price')
plt.plot(predicted_price, color = 'green', label = 'Predicted Stock Price')
plt.title('Google stock price prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

[ ]:

[ ]:

[ ]: