Name - ARYAN BAJAJ

Task 7 - Stock Market Prediction using Numerical and Textual Analysis (Level - Advanced)

In [1]:

```
import yfinance as yf
import math
import pandas_datareader as web
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
```

In [2]:

```
stock = "WIPRO.BO"
```

In [3]:
```
Stock = yf.Ticker(stock)
print(Stock.info.keys())
```

```
dict_keys(['ebitdaMargins', 'profitMargins', 'grossMargins', 'operatingCashflow', 'revenueGrowth', 'operatingMargins',
'ebitda', 'targetLowPrice', 'recommendationKey', 'grossProfits', 'freeCashflow', 'targetMedianPrice', 'currentPrice',
'earningsGrowth', 'currentRatio', 'returnOnAssets', 'numberOfAnalystOpinions', 'targetMeanPrice', 'debtToEquity', 'retu
rnOnEquity', 'targetHighPrice', 'totalCash', 'totalDebt', 'totalRevenue', 'totalCashPerShare', 'financialCurrency', 'ma
xAge', 'revenuePerShare', 'quickRatio', 'recommendationMean', 'exchange', 'shortName', 'longName', 'exchangeTimezoneNam
e', 'exchangeTimezoneShortName', 'isEsgPopulated', 'gmtOffSetMilliseconds', 'quoteType', 'symbol', 'market', 'previousC
lose', 'regularMarketOpen', 'twoHundredDayAverage', 'trailingAnnualDividendYield', 'payoutRatio', 'volume24Hr', 'regula
rMarketDayHigh', 'navPrice', 'averageDailyVolume10Day', 'totalAssets', 'regularMarketPreviousClose', 'fiftyDayAverage',
'trailingAnnualDividendRate', 'open', 'toCurrency', 'averageVolume10days', 'expireDate', 'yield', 'algorithm', 'dividen
dRate', 'exDividendDate', 'beta', 'circulatingSupply', 'startDate', 'regularMarketDayLow', 'priceHint', 'currency', 'tr
ailingPE', 'regularMarketVolume', 'lastMarket', 'maxSupply', 'openInterest', 'marketCap', 'volumeAllCurrencies', 'strik
ePrice', 'averageVolume', 'priceToSalesTrailing12Months', 'dayLow', 'ask', 'ytdReturn', 'askSize', 'volume', 'fiftyTwoW
eekHigh', 'forwardPE', 'fromCurrency', 'fiveYearAvgDividendYield', 'fiftyTwoWeekLow', 'bid', 'tradeable', 'dividendYiel
d', 'bidSize', 'dayHigh', 'regularMarketPrice', 'logo_url'])
```

In [4]:
```
# Current Share Price
Stock.info['currentPrice']
```

Out[4]:
```
598.1
```

In [5]:
```
# Price Earnings Ratio
Stock.info['trailingPE']
```

Out[5]:
```
35.426167
```

In [6]:
```
# Company Beta
Stock.info['exchange']
```

Out[6]:
```
'BSE'
```

In [7]:
```
hist = Stock.history(period="6mo")
```

In [8]:
```
hist.head(2)
```

Out[8]:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2021-02-08	430.0	437.5	426.549988	435.250000	690837	0	0
2021-02-09	440.0	451.5	435.250000	438.950012	963047	0	0

In [9]:
```
hist.tail(2)
```

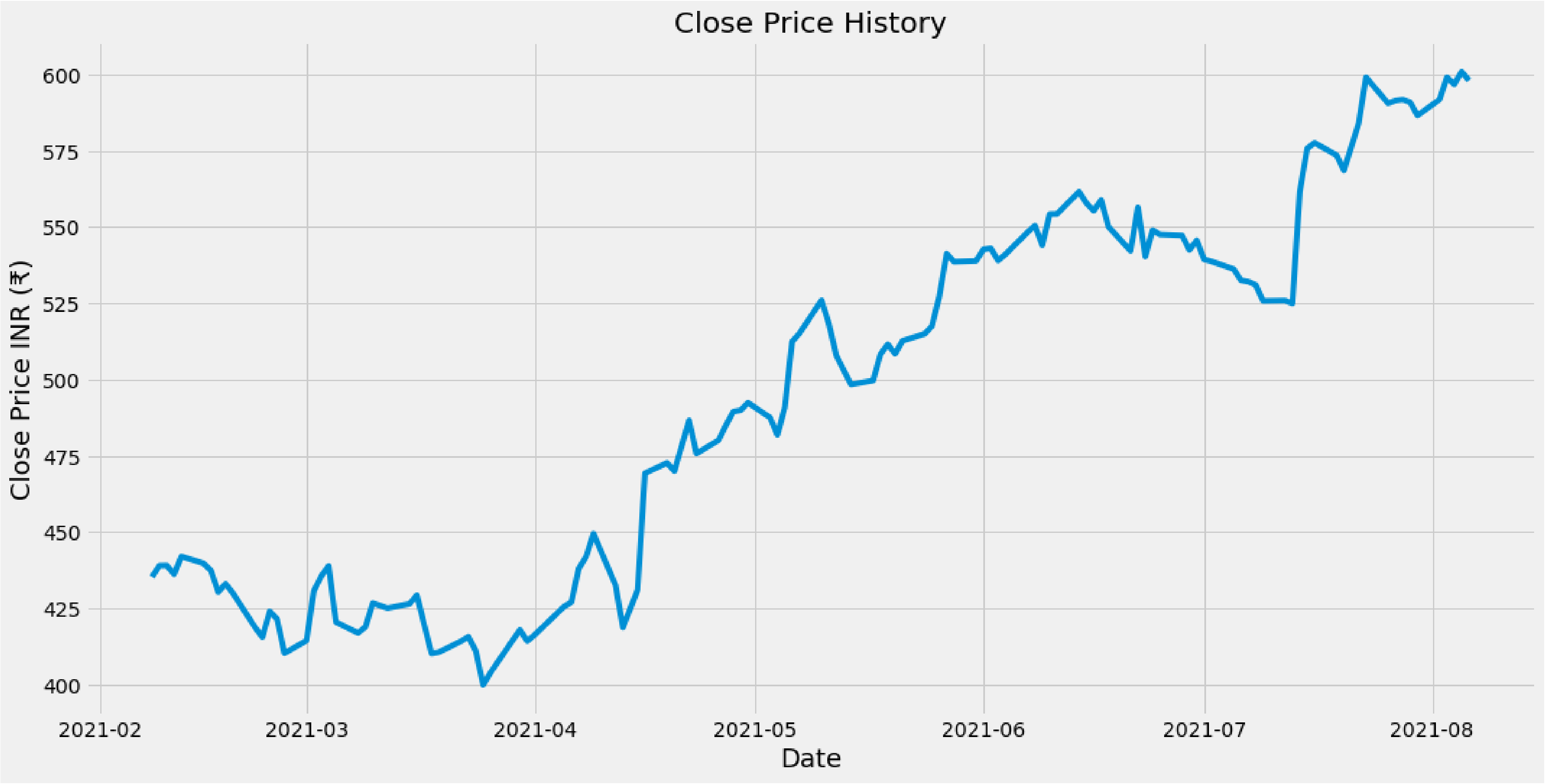
Out[9]:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2021-08-05	597.000000	614.549988	597.000000	601.000000	798154	0	0
2021-08-06	602.549988	606.450012	596.150024	598.099976	253604	0	0

In [10]:
```
hist.shape
```

Out[10]:
```
(123, 7)
```

In [11]:
```
#Visualize the Closing Price History
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(hist['Close'])
plt.xlabel('Date',fontsize=18)
plt.ylabel('Close Price INR (₹)', fontsize=18)
plt.show()
```



In [12]:
```
data = hist.filter(['Close'])
```

In [13]:
```
data.head(2)
```

Out[13]:

	Close
Date	
2021-02-08	435.250000
2021-02-09	438.950012

In [14]:
```
dataset=data.values
```

In [15]:
```
training_data_len = math.ceil(len(dataset) * .8)
```

In [16]:
```
training_data_len
```

Out[16]:
```
99
```

In [17]:
```
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
scaled_data
```

Out[17]:
```
array([[0.17619289],
       [0.19458261],
       [0.19483106],
       [0.18041758],
       [0.20949315],
       [0.19831025],
       [0.1866303 ],
       [0.15159045],
       [0.16500999],
       [0.14960241],
       [0.0939365 ],
       [0.07803187],
       [0.12002994],
       [0.10735589],
       [0.05168998],
       [0.07231619],
       [0.15357864],
       [0.17718699],
       [0.19408557],
```

In [18]:
```
train_data = scaled_data[0:training_data_len,:]
x_train = []
y_train = []
```

In [19]:

```
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i,0])
    y_train.append(train_data[i,0])
    if i<= 60:
        print(x_train)
        print(y_train)
        print()
```

```
[array([0.17619289, 0.19458261, 0.19483106, 0.18041758, 0.20949315,
       0.19831025, 0.1866303 , 0.15159045, 0.16500999, 0.14960241,
       0.0939365 , 0.07803187, 0.12002994, 0.10735589, 0.05168998,
       0.07231619, 0.15357864, 0.17718699, 0.19408557, 0.10238571,
       0.08474164, 0.09443339, 0.13344933, 0.12524856, 0.13220678,
       0.14637182, 0.09816103, 0.05119293, 0.05342942, 0.07132209,
       0.07877736, 0.05541761, 0.        , 0.02012931, 0.08996026,
       0.07157069, 0.08151105, 0.12773365, 0.13494047, 0.18986088,
       0.20924455, 0.24676949, 0.1625249 , 0.0936879 , 0.15506963,
       0.34517897, 0.36207754, 0.34865815, 0.4314116 , 0.37723668,
       0.39910543, 0.42345928, 0.44557664, 0.44781312, 0.46023857,
       0.43588473, 0.40780325, 0.45228625, 0.55989074, 0.57331029])]
[0.6272366031328751]
```

In [20]:

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

In [21]:

```
x_train = np.reshape(x_train,(x_train.shape[0], x_train.shape[1],1))
x_train.shape
```

Out[21]:

```
(39, 60, 1)
```

In [22]:

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape = (x_train.shape[1],1)))
model.add(LSTM(50, return_sequences = False))
model.add(Dense(25))
model.add(Dense(1))
```

In [23]:

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

In [24]:

```
model.fit(x_train, y_train, batch_size = 1, epochs = 1)
```

```
39/39 [==============================] - 23s 22ms/step - loss: 0.1149
```

Out[24]:

```
<keras.callbacks.History at 0x23d8a74d340>
```

In [25]:

```
test_data = scaled_data[training_data_len - 60:, :]
x_test = []
y_test = dataset[training_data_len:, :]

for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])
```

In [26]:

```
x_test = np.array(x_test)
```

In [27]:

```
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

In [28]:

```
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

In [29]:

```
rmse = np.sqrt(np.mean(predictions - y_test)**2)
rmse
```

Out[29]:

```
1.1559829711914062
```

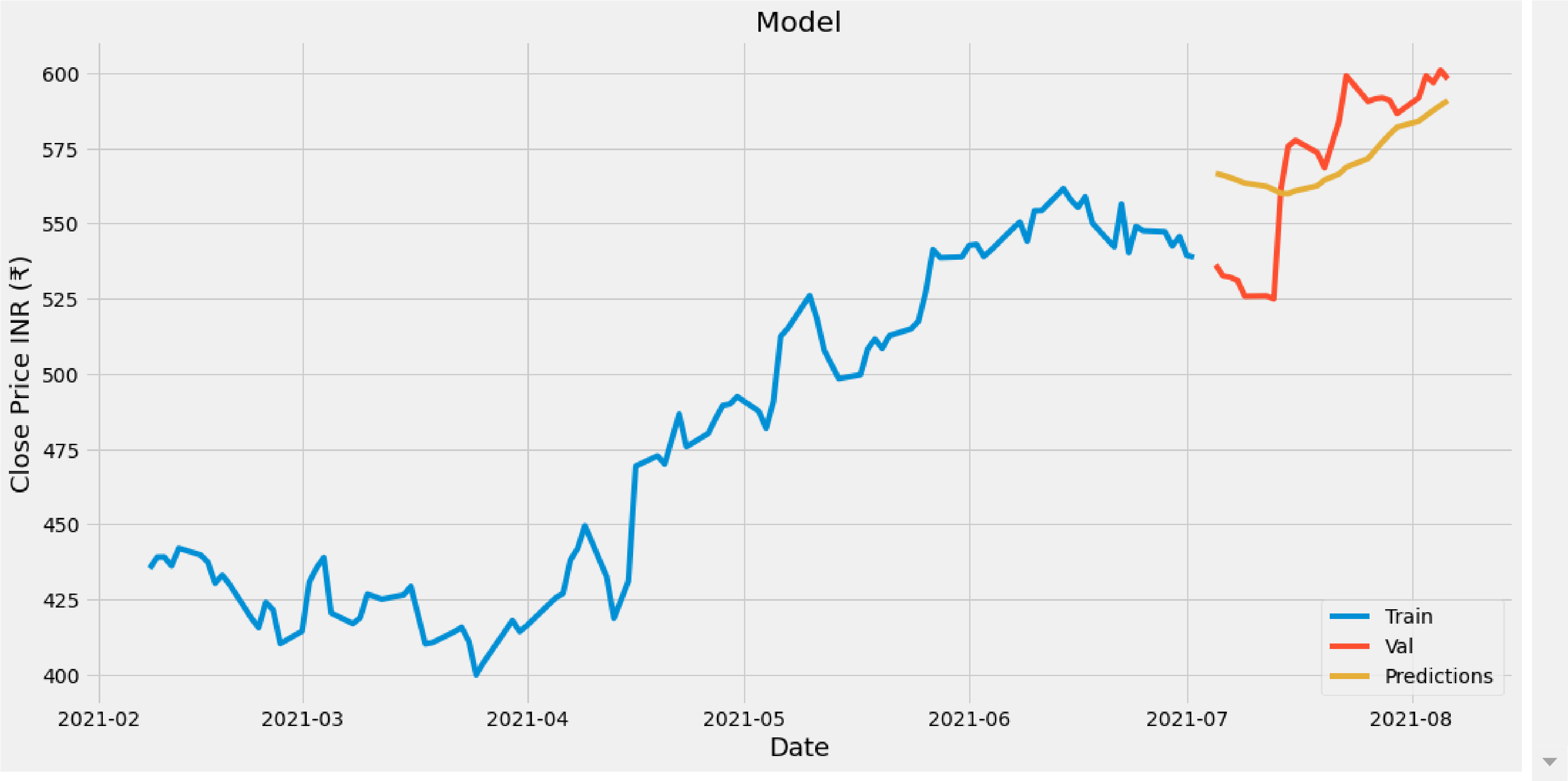
In [30]:

```
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions

plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date', fontsize = 18)
plt.ylabel('Close Price INR (₹)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])
plt.legend(['Train','Val','Predictions'], loc='lower right')
plt.show()
```

```
<ipython-input-30-66268c3d5854>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  valid['Predictions'] = predictions
```


Model
Close Price INR (₹)
600
575
550
525
500
475
450
425
400
2021-02
2021-03
2021-04
2021-05
2021-06
2021-07
2021-08
Date
Train
Val
Predictions

In [31]:
```
valid.tail(5)
```

Out[31]:

Date	Close	Predictions
2021-08-02	591.849976	584.040161
2021-08-03	599.150024	585.844727
2021-08-04	596.849976	587.638733
2021-08-05	601.000000	589.289307
2021-08-06	598.099976	590.895874

In [32]:
```
valid['Returns on Actual Closing Price(₹)'] = ((valid['Close'] - valid['Close'].shift(1))/(valid['Close'].shift(1)))
valid['Returns on Predicted Closing Price(₹)'] = ((valid['Predictions'] - valid['Predictions'].shift(1))/(valid['Predict
```

```
<ipython-input-32-2d7f63499899>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  valid['Returns on Actual Closing Price(₹)'] = ((valid['Close'] - valid['Close'].shift(1))/(valid['Close'].shift(1)))
<ipython-input-32-2d7f63499899>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  valid['Returns on Predicted Closing Price(₹)'] = ((valid['Predictions'] - valid['Predictions'].shift(1))/(valid['Pred
ictions'].shift(1)))
```

In [33]:
```
valid['Diff_Actual_vs_Pred Closing Price(₹)'] = (valid['Close'] - valid['Predictions'])
valid.tail(5)
```

```
<ipython-input-33-f33b39b6844b>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  valid['Diff_Actual_vs_Pred Closing Price(₹)'] = (valid['Close'] - valid['Predictions'])
```

Out[33]:

	Close	Predictions	Returns on Actual Closing Price(₹)	Returns on Predicted Closing Price(₹)	Diff_Actual_vs_Pred Closing Price(₹)
Date					
2021-08-02	591.849976	584.040161	0.008950	0.003407	7.809814
2021-08-03	599.150024	585.844727	0.012334	0.003090	13.305298
2021-08-04	596.849976	587.638733	-0.003839	0.003062	9.211243
2021-08-05	601.000000	589.289307	0.006953	0.002809	11.710693
2021-08-06	598.099976	590.895874	-0.004825	0.002726	7.204102

In [34]:

```python
quote = hist
new_df = quote.filter(['Close'])
last_60_days = new_df[-60:].values
last_60_days_scaled = scaler.transform(last_60_days)
X_test = []
X_test.append(last_60_days_scaled)
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
pred_price = model.predict(X_test)
pred_price = scaler.inverse_transform(pred_price)
print('Tomorrow(s) Predicted price for', stock, 'will be :', pred_price)
```

```
Tomorrow(s) Predicted price for WIPRO.BO will be : [[592.3306]]
```