NAME: RADHIKA GUPTA

REG. NO: 22MCA1119

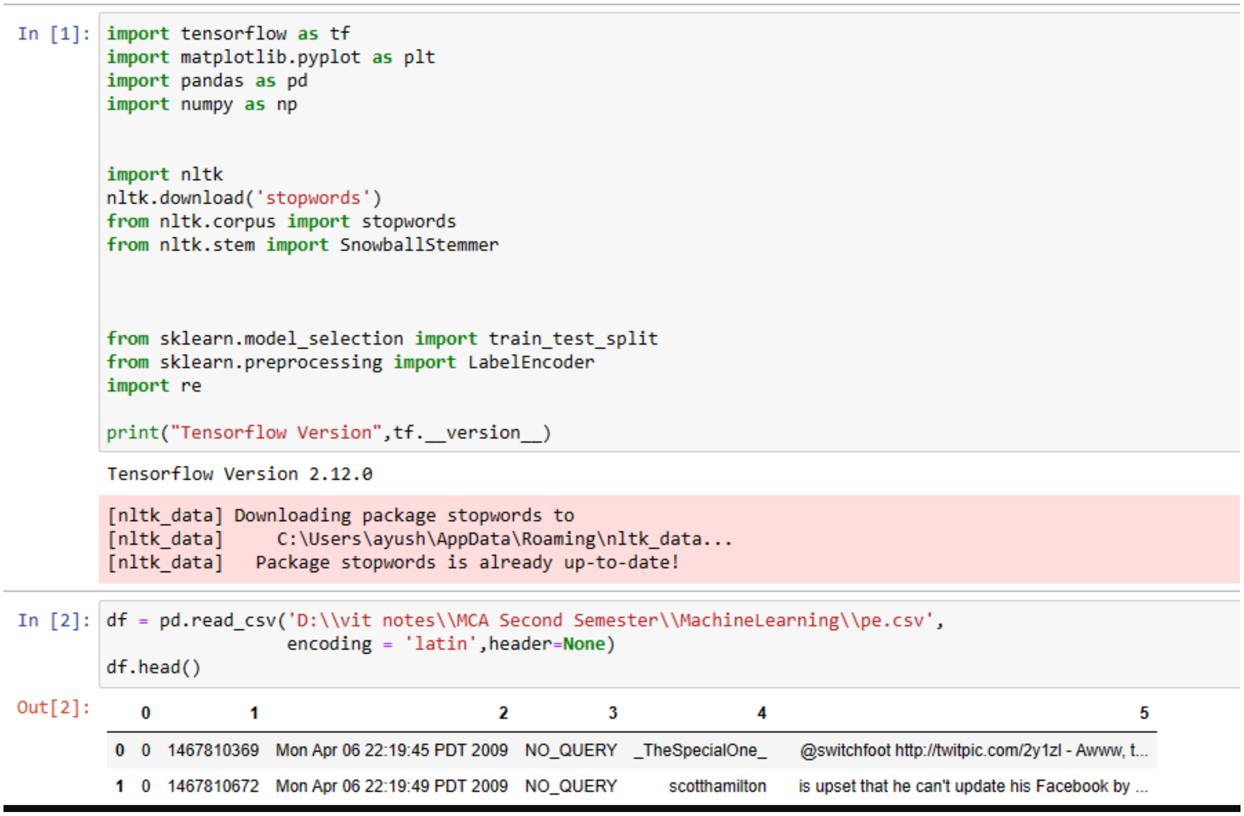
ITA-6016 Machine Learning

Digital Assignment –Lab-6

SUBMITTED TO: Dr Dominic Savio M

LSTM:

CODE OF THE PROGRAM AND OUTPUT:



In [1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer



from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import re

print("Tensorflow Version",tf.__version__)
```

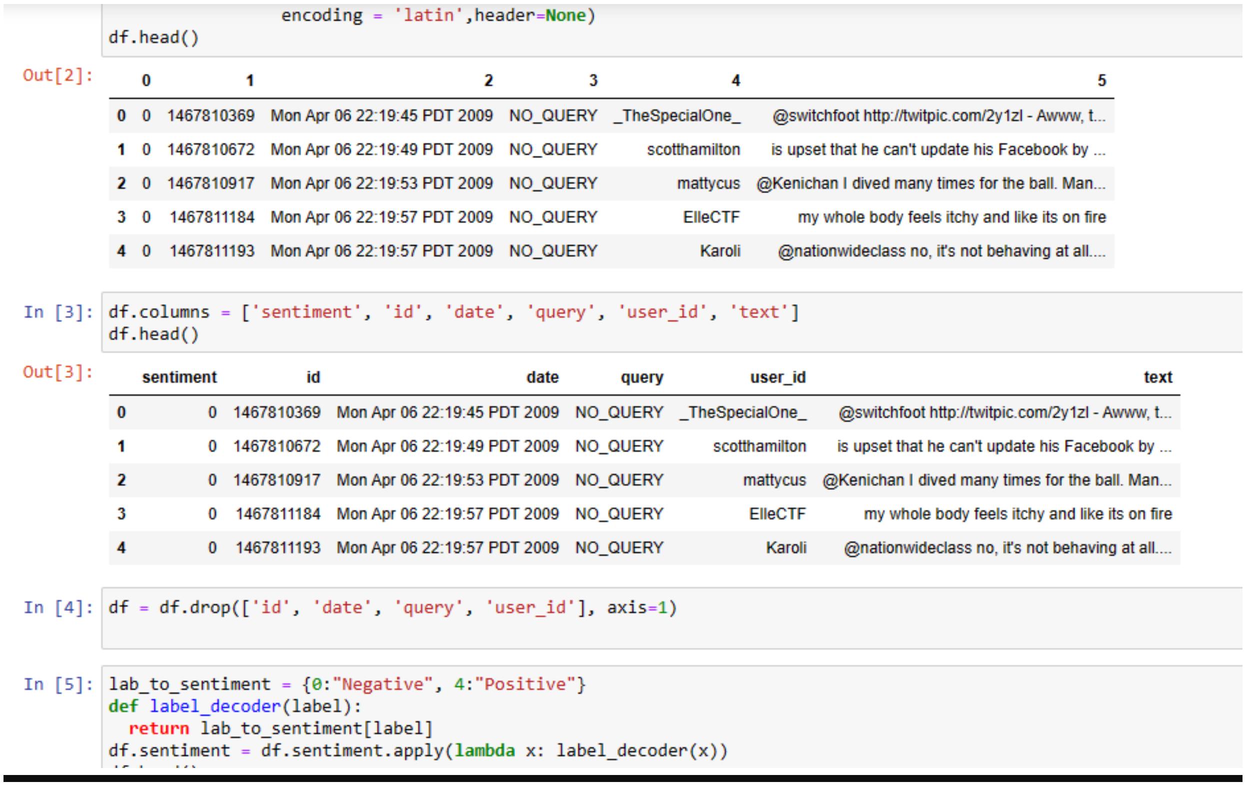
```
Tensorflow Version 2.12.0
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [2]:
```
df = pd.read_csv('D:\\vit notes\\MCA Second Semester\\MachineLearning\\pe.csv',
                 encoding = 'latin',header=None)
df.head()
```

Out[2]:

	0	1	2	3	4	5
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...



```
                 encoding = 'latin',header=None)
df.head()
```

Out[2]:

```
   0  1           2                             3         4                5
0  0  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY  _TheSpecialOne_  @switchfoot http://twitpic.com/2y1zl - Awww, t...
1  0  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY     scotthamilton  is upset that he can't update his Facebook by ...
2  0  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY          mattycus  @Kenichan I dived many times for the ball. Man...
3  0  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY           ElleCTF    my whole body feels itchy and like its on fire
4  0  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY            Karoli  @nationwideclass no, it's not behaving at all....
```

In [3]:

```python
df.columns = ['sentiment', 'id', 'date', 'query', 'user_id', 'text']
df.head()
```

Out[3]:

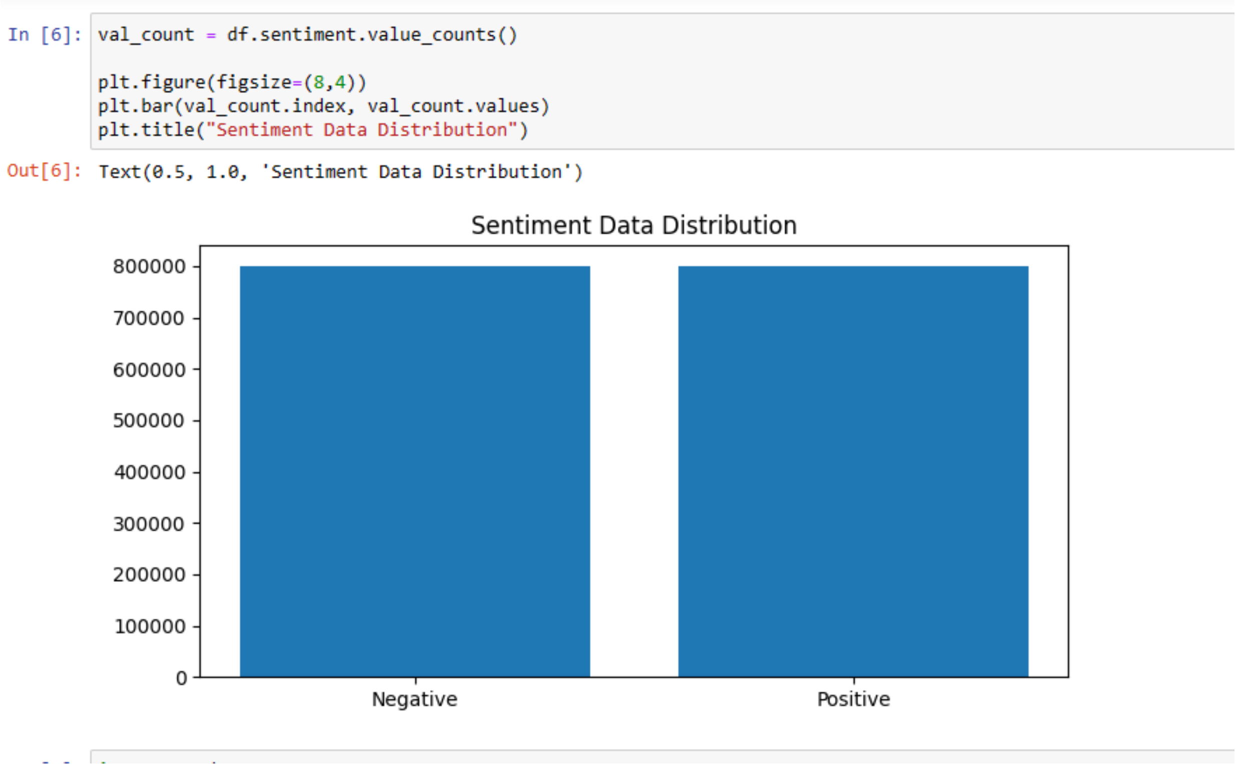
```
   sentiment  id          date                          query     user_id          text
0          0  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY  _TheSpecialOne_  @switchfoot http://twitpic.com/2y1zl - Awww, t...
1          0  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY     scotthamilton  is upset that he can't update his Facebook by ...
2          0  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY          mattycus  @Kenichan I dived many times for the ball. Man...
3          0  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY           ElleCTF    my whole body feels itchy and like its on fire
4          0  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY            Karoli  @nationwideclass no, it's not behaving at all....
```

In [4]:

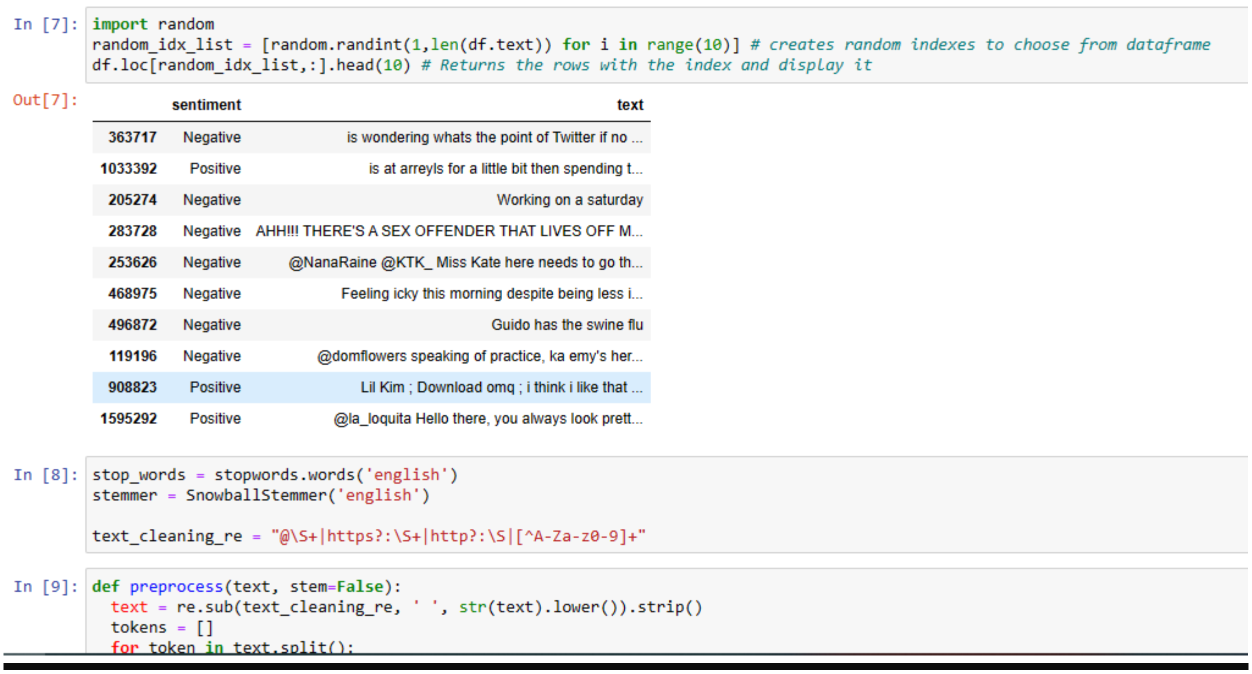
```python
df = df.drop(['id', 'date', 'query', 'user_id'], axis=1)
```

In [5]:

```python
lab_to_sentiment = {0:"Negative", 4:"Positive"}
def label_decoder(label):
  return lab_to_sentiment[label]
df.sentiment = df.sentiment.apply(lambda x: label_decoder(x))
df.head()
```



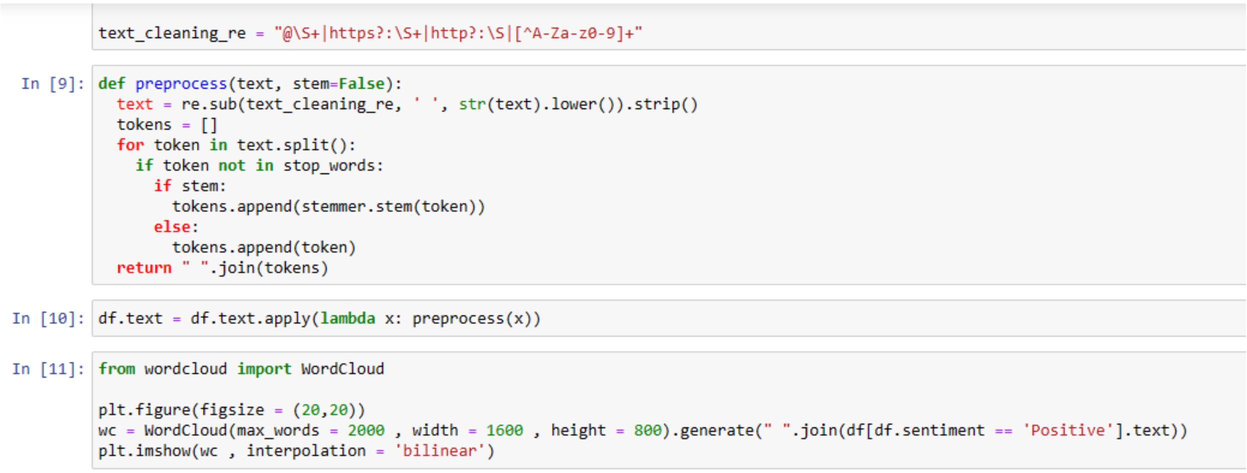
```
In [6]: val_count = df.sentiment.value_counts()

        plt.figure(figsize=(8,4))
        plt.bar(val_count.index, val_count.values)
        plt.title("Sentiment Data Distribution")

Out[6]: Text(0.5, 1.0, 'Sentiment Data Distribution')
```



```
In [7]: import random
        random_idx_list = [random.randint(1,len(df.text)) for i in range(10)] # creates random indexes to choose from dataframe
        df.loc[random_idx_list,:].head(10) # Returns the rows with the index and display it

Out[7]:
```

	sentiment	text
363717	Negative	is wondering whats the point of Twitter if no ...
1033392	Positive	is at arreyls for a little bit then spending t...
205274	Negative	Working on a saturday
283728	Negative	AHH!!! THERE'S A SEX OFFENDER THAT LIVES OFF M...
253626	Negative	@NanaRaine @KTK_ Miss Kate here needs to go th...
468975	Negative	Feeling icky this morning despite being less i...
496872	Negative	Guido has the swine flu
119196	Negative	@domflowers speaking of practice, ka emy's her...
908823	Positive	Lil Kim ; Download omq ; i think i like that ...
1595292	Positive	@la_loquita Hello there, you always look prett...

```
In [8]: stop_words = stopwords.words('english')
        stemmer = SnowballStemmer('english')

        text_cleaning_re = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"

In [9]: def preprocess(text, stem=False):
          text = re.sub(text_cleaning_re, ' ', str(text).lower()).strip()
          tokens = []
          for token in text.split():
```

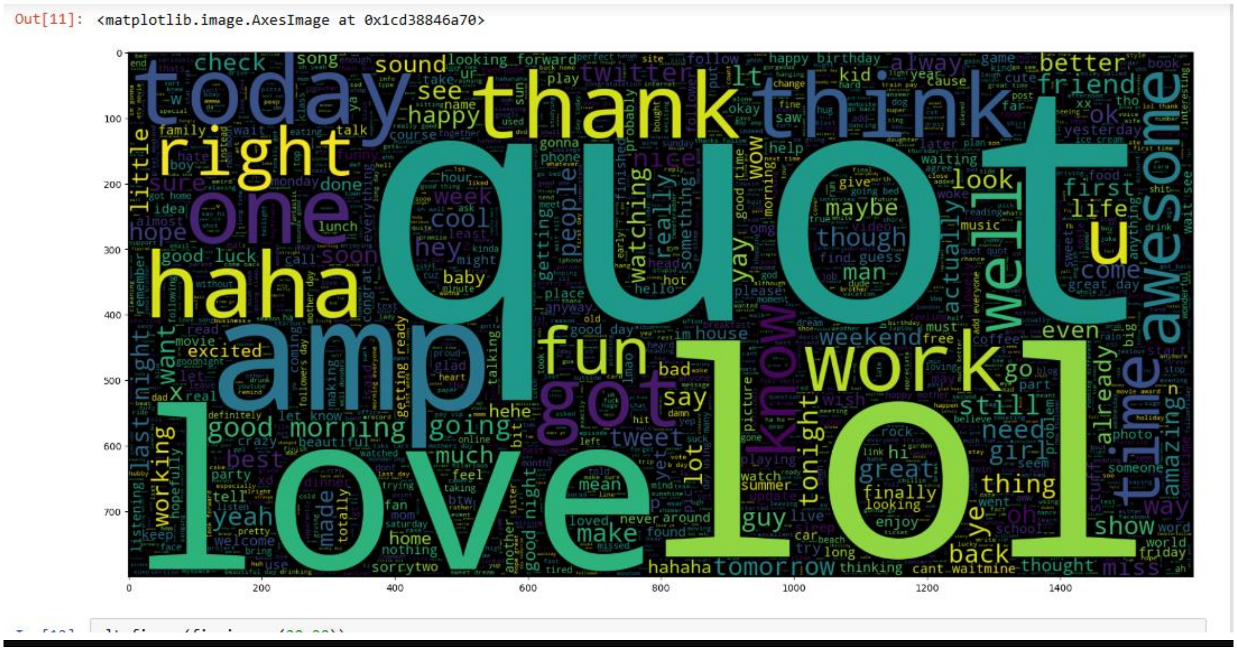


```
text_cleaning_re = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"
```

```
In [9]: def preprocess(text, stem=False):
          text = re.sub(text_cleaning_re, ' ', str(text).lower()).strip()
          tokens = []
          for token in text.split():
            if token not in stop_words:
              if stem:
                tokens.append(stemmer.stem(token))
              else:
                tokens.append(token)
          return " ".join(tokens)
```

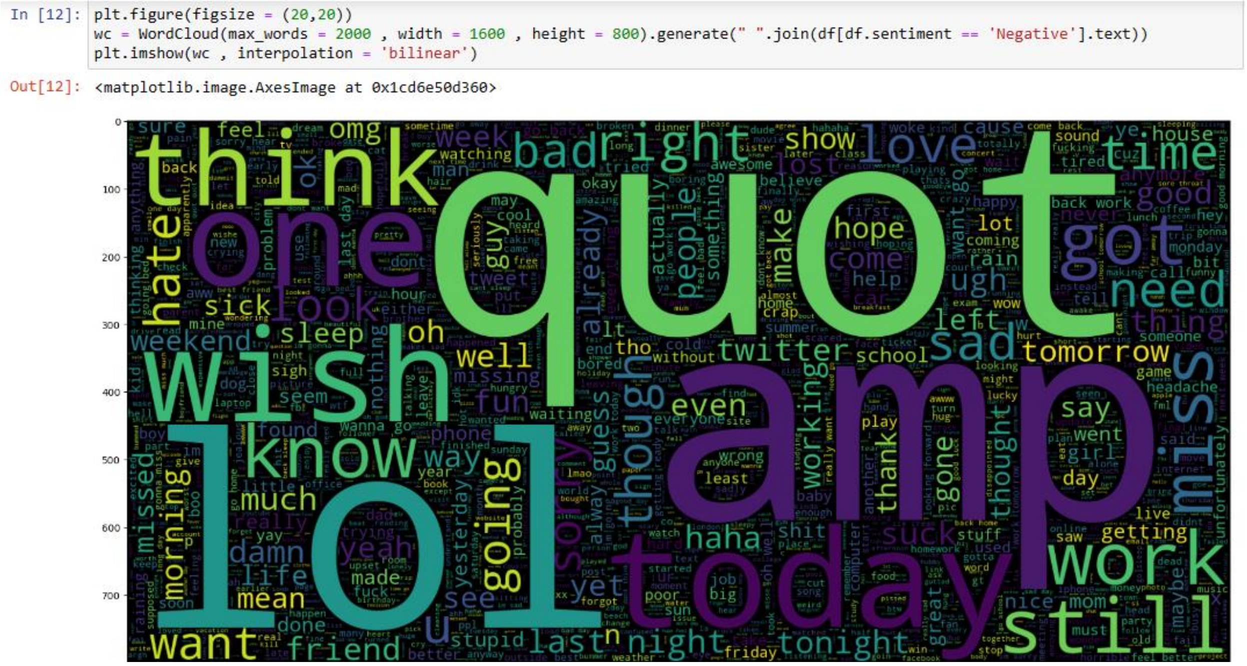
```
In [10]: df.text = df.text.apply(lambda x: preprocess(x))
```

```
In [11]: from wordcloud import WordCloud

         plt.figure(figsize = (20,20))
         wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.sentiment == 'Positive'].text))
         plt.imshow(wc , interpolation = 'bilinear')
```

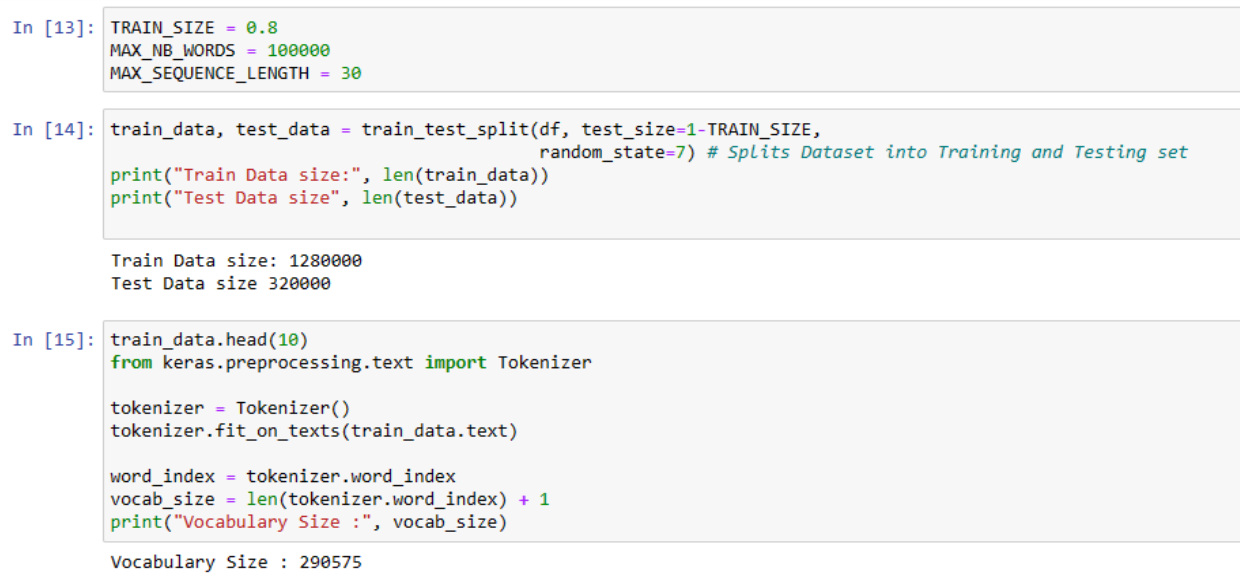


```
Out[11]: <matplotlib.image.AxesImage at 0x1cd38846a70>
```

```
In [12]: plt.figure(figsize = (20,20))
```



```
In [12]: plt.figure(figsize = (20,20))
         wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.sentiment == 'Negative'].text))
         plt.imshow(wc , interpolation = 'bilinear')

Out[12]: <matplotlib.image.AxesImage at 0x1cd6e50d360>
```



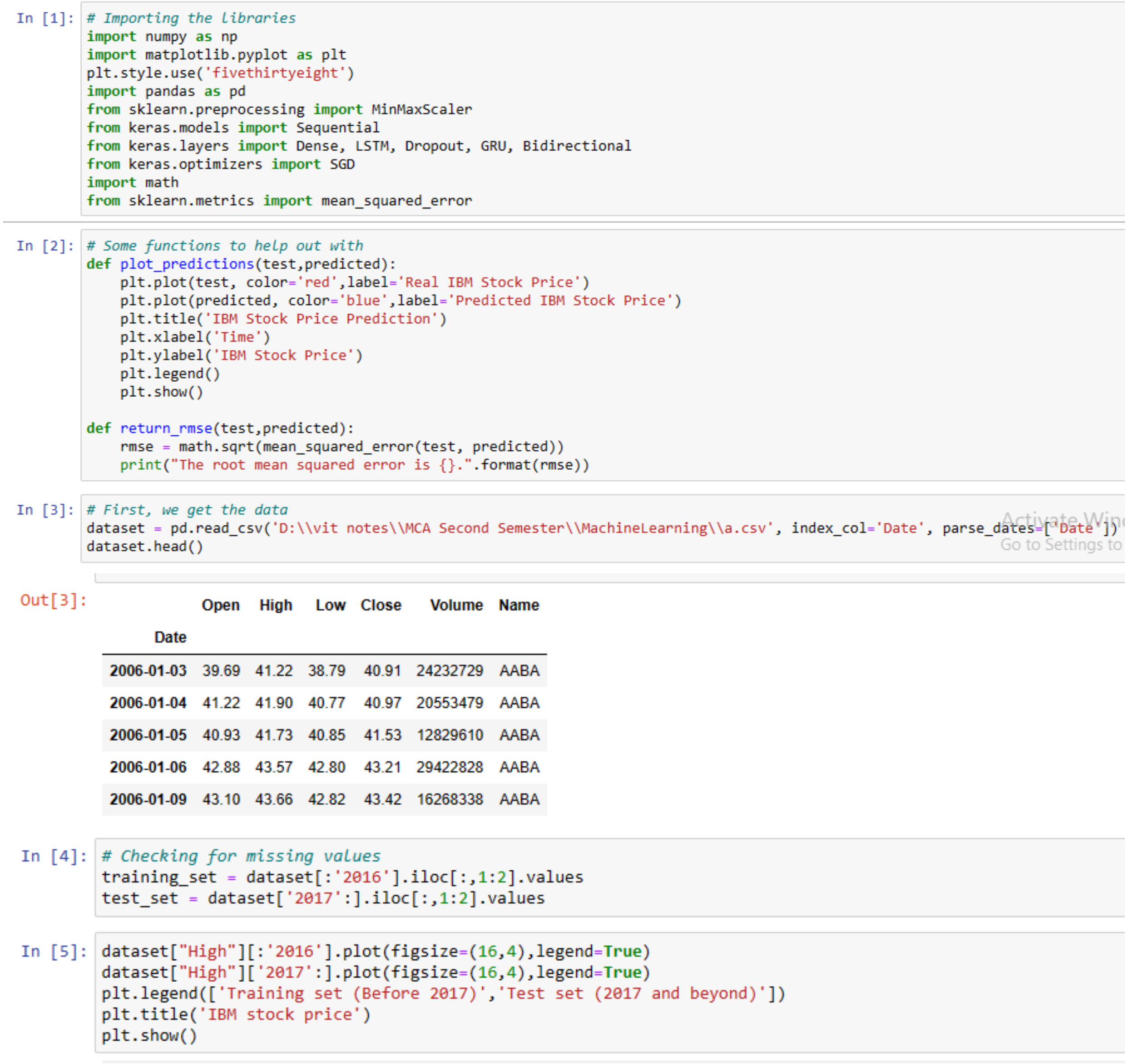
```
In [13]: TRAIN_SIZE = 0.8
         MAX_NB_WORDS = 100000
         MAX_SEQUENCE_LENGTH = 30
```

```
In [14]: train_data, test_data = train_test_split(df, test_size=1-TRAIN_SIZE,
                                                  random_state=7) # Splits Dataset into Training and Testing set
         print("Train Data size:", len(train_data))
         print("Test Data size", len(test_data))

         Train Data size: 1280000
         Test Data size 320000
```

```
In [15]: train_data.head(10)
         from keras.preprocessing.text import Tokenizer

         tokenizer = Tokenizer()
         tokenizer.fit_on_texts(train_data.text)

         word_index = tokenizer.word_index
         vocab_size = len(tokenizer.word_index) + 1
         print("Vocabulary Size :", vocab_size)

         Vocabulary Size : 290575
```

<u>RNN:</u>

CODE OF THE PROGRAM AND OUTPUT:



In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```

In [2]:
```
# Some functions to help out with
def plot_predictions(test,predicted):
    plt.plot(test, color='red',label='Real IBM Stock Price')
    plt.plot(predicted, color='blue',label='Predicted IBM Stock Price')
    plt.title('IBM Stock Price Prediction')
    plt.xlabel('Time')
    plt.ylabel('IBM Stock Price')
    plt.legend()
    plt.show()

def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
```

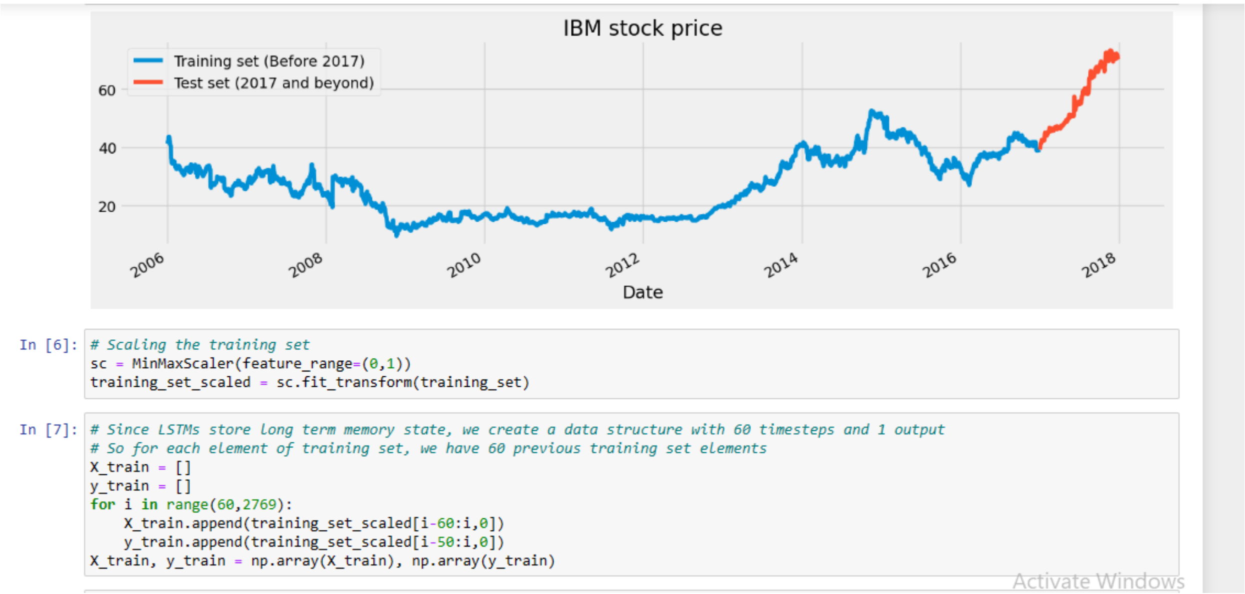
In [3]:
```
# First, we get the data
dataset = pd.read_csv('D:\\vit notes\\MCA Second Semester\\MachineLearning\\a.csv', index_col='Date', parse_dates=['Date'])
dataset.head()
```

Out[3]:

Date	Open	High	Low	Close	Volume	Name
2006-01-03	39.69	41.22	38.79	40.91	24232729	AABA
2006-01-04	41.22	41.90	40.77	40.97	20553479	AABA
2006-01-05	40.93	41.73	40.85	41.53	12829610	AABA
2006-01-06	42.88	43.57	42.80	43.21	29422828	AABA
2006-01-09	43.10	43.66	42.82	43.42	16268338	AABA

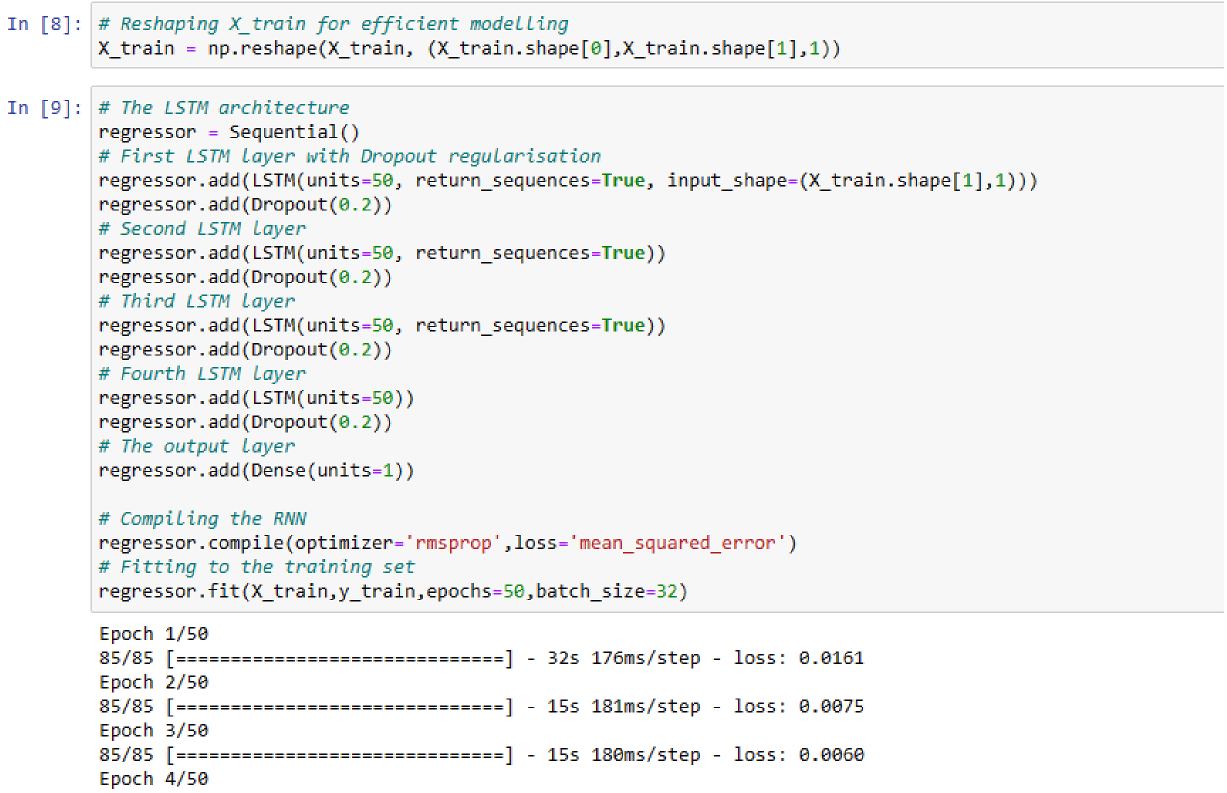
In [4]:
```
# Checking for missing values
training_set = dataset[:'2016'].iloc[:,1:2].values
test_set = dataset['2017':].iloc[:,1:2].values
```

In [5]:
```
dataset["High"][:'2016'].plot(figsize=(16,4),legend=True)
dataset["High"]['2017':].plot(figsize=(16,4),legend=True)
plt.legend(['Training set (Before 2017)','Test set (2017 and beyond)'])
plt.title('IBM stock price')
plt.show()
```



In [6]:
```
# Scaling the training set
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

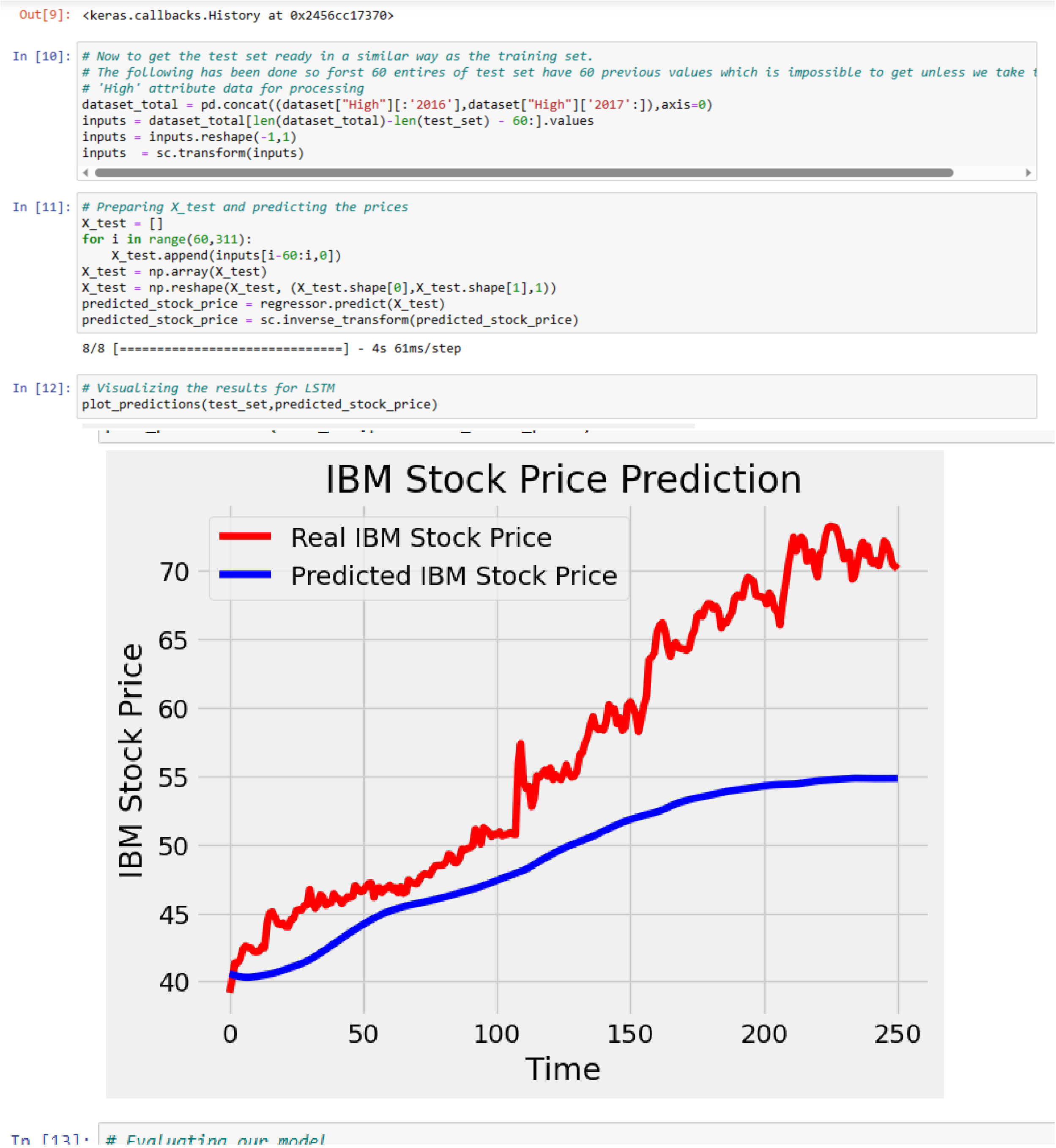
In [7]:
```
# Since LSTMs store long term memory state, we create a data structure with 60 timesteps and 1 output
# So for each element of training set, we have 60 previous training set elements
X_train = []
y_train = []
for i in range(60,2769):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i-50:i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
```



In [8]:
```
# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

In [9]:
```
# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],1)))
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Fourth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# The output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
# Fitting to the training set
regressor.fit(X_train,y_train,epochs=50,batch_size=32)
```

```
Epoch 1/50
85/85 [==============================] - 32s 176ms/step - loss: 0.0161
Epoch 2/50
85/85 [==============================] - 15s 181ms/step - loss: 0.0075
Epoch 3/50
85/85 [==============================] - 15s 180ms/step - loss: 0.0060
Epoch 4/50
```



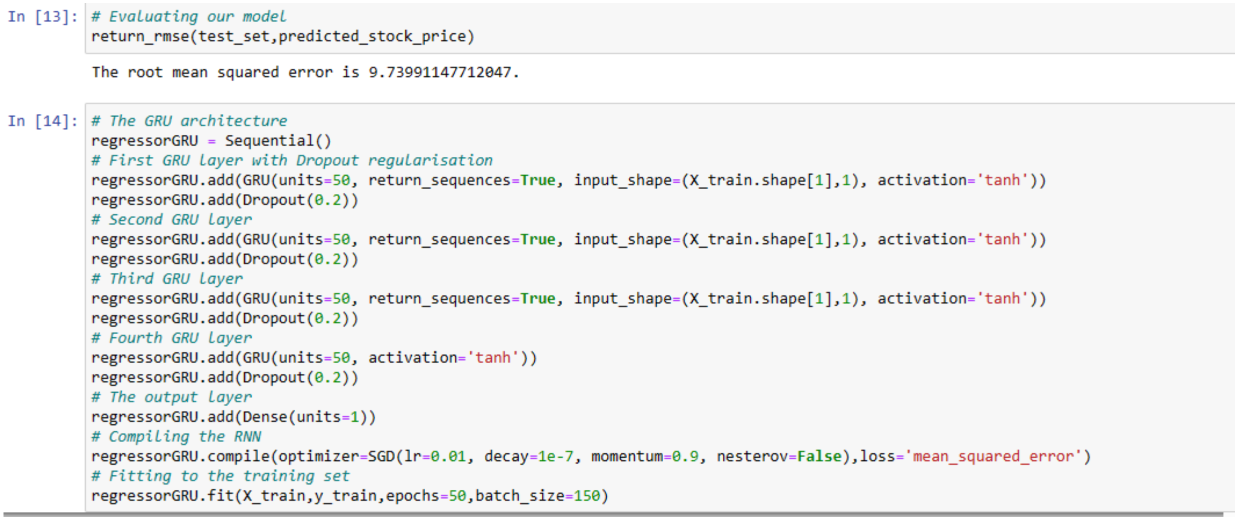
```
Out[9]: <keras.callbacks.History at 0x2456cc17370>
```

```
In [10]: # Now to get the test set ready in a similar way as the training set.
         # The following has been done so forst 60 entires of test set have 60 previous values which is impossible to get unless we take t
         # 'High' attribute data for processing
         dataset_total = pd.concat((dataset["High"][:'2016'],dataset["High"]['2017':]),axis=0)
         inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
         inputs = inputs.reshape(-1,1)
         inputs  = sc.transform(inputs)
```

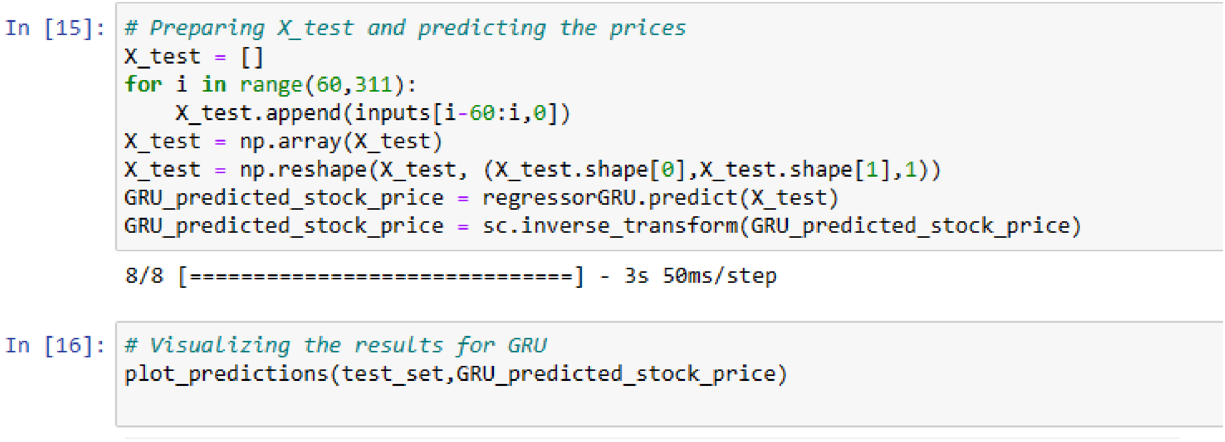
```
In [11]: # Preparing X_test and predicting the prices
         X_test = []
         for i in range(60,311):
             X_test.append(inputs[i-60:i,0])
         X_test = np.array(X_test)
         X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
         predicted_stock_price = regressor.predict(X_test)
         predicted_stock_price = sc.inverse_transform(predicted_stock_price)

         8/8 [==============================] - 4s 61ms/step
```

```
In [12]: # Visualizing the results for LSTM
         plot_predictions(test_set,predicted_stock_price)
```

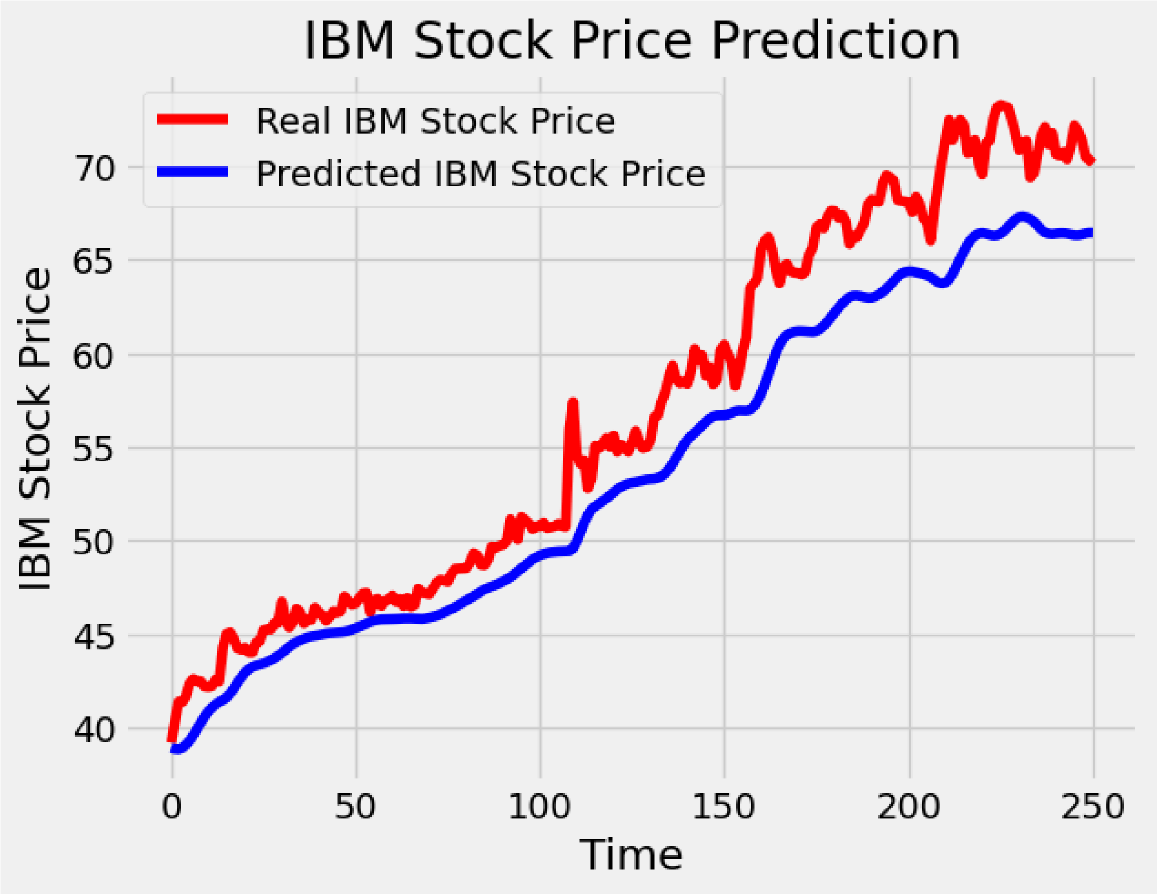
```
In [13]: # Evaluating our model
```

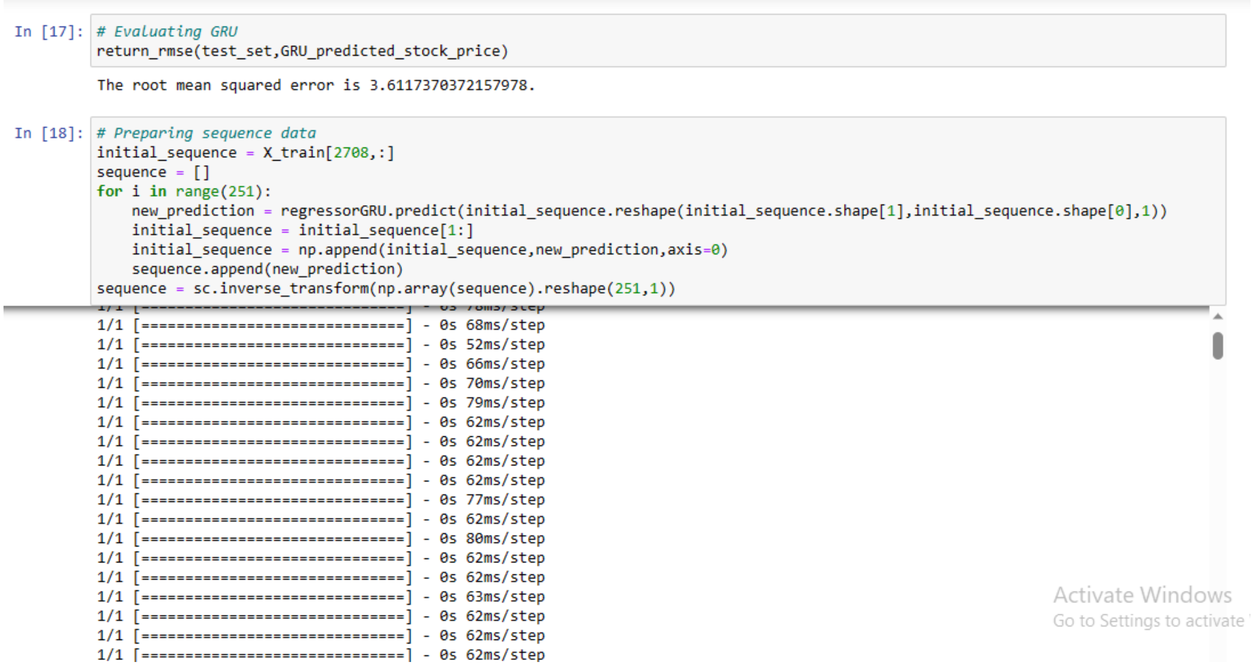


```
In [13]: # Evaluating our model
         return_rmse(test_set,predicted_stock_price)

         The root mean squared error is 9.73991147712047.

In [14]: # The GRU architecture
         regressorGRU = Sequential()
         # First GRU layer with Dropout regularisation
         regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
         regressorGRU.add(Dropout(0.2))
         # Second GRU layer
         regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
         regressorGRU.add(Dropout(0.2))
         # Third GRU layer
         regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
         regressorGRU.add(Dropout(0.2))
         # Fourth GRU layer
         regressorGRU.add(GRU(units=50, activation='tanh'))
         regressorGRU.add(Dropout(0.2))
         # The output layer
         regressorGRU.add(Dense(units=1))
         # Compiling the RNN
         regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9, nesterov=False),loss='mean_squared_error')
         # Fitting to the training set
         regressorGRU.fit(X_train,y_train,epochs=50,batch_size=150)
```



```
In [15]: # Preparing X_test and predicting the prices
         X_test = []
         for i in range(60,311):
             X_test.append(inputs[i-60:i,0])
         X_test = np.array(X_test)
         X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
         GRU_predicted_stock_price = regressorGRU.predict(X_test)
         GRU_predicted_stock_price = sc.inverse_transform(GRU_predicted_stock_price)

         8/8 [==============================] - 3s 50ms/step

In [16]: # Visualizing the results for GRU
         plot_predictions(test_set,GRU_predicted_stock_price)
```

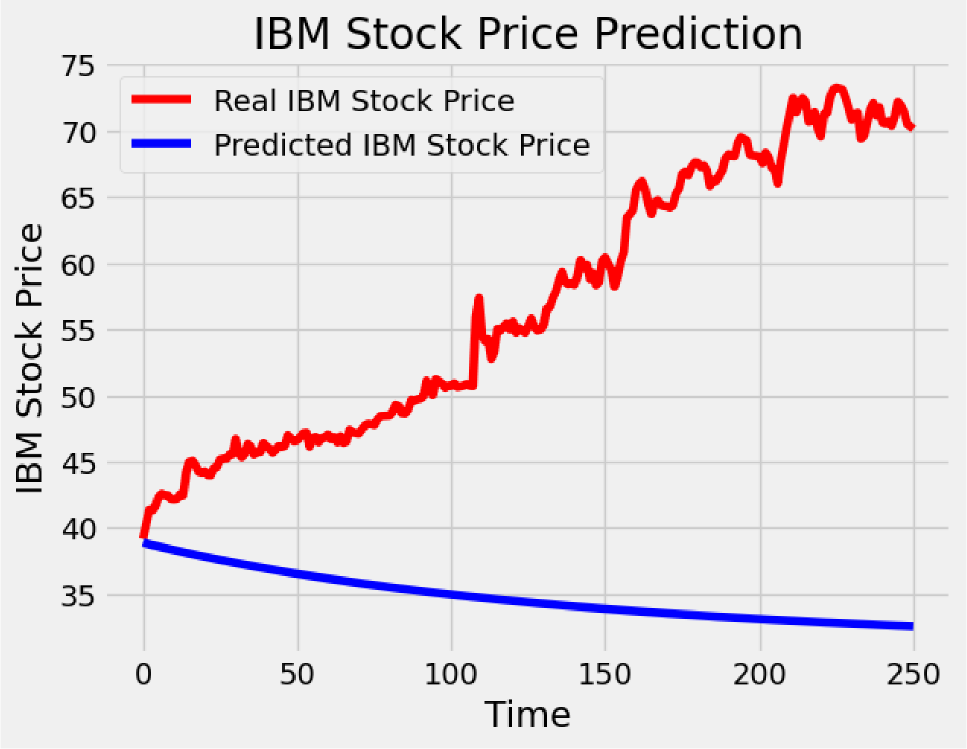




```
In [17]: # Evaluating GRU
         return_rmse(test_set,GRU_predicted_stock_price)

         The root mean squared error is 3.6117370372157978.

In [18]: # Preparing sequence data
         initial_sequence = X_train[2708,:]
         sequence = []
         for i in range(251):
             new_prediction = regressorGRU.predict(initial_sequence.reshape(initial_sequence.shape[1],initial_sequence.shape[0],1))
             initial_sequence = initial_sequence[1:]
             initial_sequence = np.append(initial_sequence,new_prediction,axis=0)
             sequence.append(new_prediction)
         sequence = sc.inverse_transform(np.array(sequence).reshape(251,1))
```

```
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 79ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 62ms/step
```

```
In [19]: # Visualizing the sequence
         plot_predictions(test_set,sequence)
```



```
In [20]: # Evaluating the sequence
         return_rmse(test_set,sequence)

         The root mean squared error is 25.132233844083935.

In [ ]: 
```