Top of Form

DAY\_RANGE = 90

TIME = '16:00:00'

# Importing the training set

metro\_traffic = pd.read\_csv('Metro\_Interstate\_Traffic\_Volume.csv')

# Converting datetime feature to datetime

metro\_traffic['date\_time'] = pd.to\_datetime(metro\_traffic.date\_time)

# Segrigating datetime to date & time feature

metro\_traffic['date'] = metro\_traffic['date\_time'].dt.date

metro\_traffic['time'] = metro\_traffic['date\_time'].dt.time

# Converting time to string

metro\_traffic['time'] = metro\_traffic['time'].astype(str)

# Converting time to string

metro\_traffic['date'] = metro\_traffic['date'].astype(str)

# filtering only 9:00 oclock records

dataset=metro\_traffic[metro\_traffic['time'] == TIME]

dataset = dataset.drop\_duplicates(subset=['date'], keep='first', inplace=False)

# divide the dataset into test and training dataset

dataset\_train = dataset[dataset['date'] <= '2018-09-01']

dataset\_test = dataset[dataset['date'] >= '2018-09-01']

training\_set = dataset\_train.iloc[:, 8:9].values

# Feature Scaling

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range = (0, 1))

training\_set\_scaled = sc.fit\_transform(training\_set)

# Creating a data structure with 60 timesteps and 1 output

X\_train = []

y\_train = []

for i in range(DAY\_RANGE, len(dataset\_train)):

X\_train.append(training\_set\_scaled[i-DAY\_RANGE:i, 0])

y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshaping

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

training\_set = dataset\_train.iloc[:,8:9].values

# Part 2 - Building the RNN

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

# Initialising the RNN

# Initialising the RNN

regressor = Sequential()

# Adding the input layer and the LSTM layer

regressor.add(LSTM(units = 95, return\_sequences = True, input\_shape = (None, 1)))

regressor.add(Dropout(.055))

# Adding a second LSTM layer

#regressor.add(LSTM(units = 90, return\_sequences = True))

# Adding a third LSTM layer

regressor.add(LSTM(units = 95, return\_sequences = True))

regressor.add(Dropout(.055))

# Adding a fourth LSTM layer

regressor.add(LSTM(units = 95))

regressor.add(Dropout(.055))

# Adding the output layer

regressor.add(Dense(units = 1))

regressor.add(Dropout(.055))

# Compiling the RNN

regressor.compile(optimizer = 'rmsprop', loss = 'mean\_squared\_error')

# Fitting the RNN to the Training set

regressor.fit(X\_train, y\_train, epochs = 100, batch\_size = 32)



# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

DAY\_RANGE = 60

TIME = '16:00:00'

# Importing the training set

metro\_traffic = pd.read\_csv('Metro\_Interstate\_Traffic\_Volume.csv')

# Converting datetime feature to datetime

metro\_traffic['date\_time'] = pd.to\_datetime(metro\_traffic.date\_time)

# Segrigating datetime to date & time feature

metro\_traffic['date'] = metro\_traffic['date\_time'].dt.date

metro\_traffic['time'] = metro\_traffic['date\_time'].dt.time

# Converting time to string

metro\_traffic['time'] = metro\_traffic['time'].astype(str)

# Converting time to string

metro\_traffic['date'] = metro\_traffic['date'].astype(str)

# filtering only 9:00 oclock records

dataset=metro\_traffic[metro\_traffic['time'] == TIME]

dataset = dataset.drop\_duplicates(subset=['date'], keep='first', inplace=False)

# divide the dataset into test and training dataset

dataset\_train = dataset[dataset['date'] <= '2018-09-01']

dataset\_test = dataset[dataset['date'] >= '2018-09-01']

training\_set = dataset\_train.iloc[:, 8:9].values

# Feature Scaling

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range = (0, 1))

training\_set\_scaled = sc.fit\_transform(training\_set)

# Creating a data structure with 60 timesteps and 1 output

X\_train = []

y\_train = []

for i in range(DAY\_RANGE, len(dataset\_train)):

X\_train.append(training\_set\_scaled[i-DAY\_RANGE:i, 0])

y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Reshaping

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

training\_set = dataset\_train.iloc[:,8:9].values

# Part 2 - Building the RNN

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

# Initialising the RNN

regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 50))

regressor.add(Dropout(0.2))

# Adding the output layer

regressor.add(Dense(units = 1))

# Compiling the RNN

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

# Fitting the RNN to the Training set

regressor.fit(X\_train, y\_train, epochs = 100, batch\_size = 32)

# Part 3 - Making the predictions and visualising the results

# Getting the real stock price of 2017

#dataset\_test = pd.read\_csv('Google\_Stock\_Price\_Test.csv')

real\_stock\_price = dataset\_test.iloc[:, 8:9].values

# Getting the predicted stock price of 2017

dataset\_total = dataset.iloc[:, 8:9].values

inputs = dataset\_total[len(dataset\_total) - len(dataset\_test) - DAY\_RANGE:]

inputs = inputs.reshape(-1,1)

inputs = sc.transform(inputs)

X\_test = []

for i in range(DAY\_RANGE, len(inputs)):

X\_test.append(inputs[i-DAY\_RANGE: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)



FINAL one -

