STOCK PRICE PREDICTION

AIM:

The aim is to create a model to get the prediction of the **"New York Stock Exchange"**

Here we have the value of 2016 whole year, 80% of the value will be used to train the model, and the remaining 20% test the model.

Dataset consists of the following files:

prices.csv: raw, as-is daily prices. Most of the data spans from 2010 to the end of 2016, for companies new on the stock market date range, is shorter. There have been approx. 140 stock splits in that time, this set doesn't account for that.

prices-split-adjusted.csv: same as prices, but there have been added adjustments for splits.

securities.csv: a general description of each company with the division on sectors

fundamentals.csv: metrics extracted from annual SEC 10K fillings (2012-2016), should be enough to derive most of the popular fundamental indicators

Import Libraries

```
import numpy as np

import pandas as pd

import os

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 15, 6
```

Check the current working directory

```
display (os.getcwd())
```

Change the current working directory

```
os.chdir ('C:\\Noble\\Training\\Acmegrade\\Machine Learning\\Projects\\PRJ Stock Price Prediction\\')
display (os.getcwd())
```

Read and display the data set

```
# calling the file in nyse named prices.csv

df =pd.read_csv("prices.csv", header=0)
display (df)
```

Data Description

The table consists of data of New York Stock Exchange consist of 851264 entries

Column1—Date: The date here is from 05-01-2016 to 30-12-2016 i.e of the year 2016 with many company performances

Column2—Symbols: Symbols are code for 501 companies i.e each symbol is allotted to each corporation. Other way we consider it as company names

Column3—open: Open is the stock opening price of the date.

Column4—close: Close is the stock closing price of the date

Column5—low: Low is the lowest price of the date

Column6—high: High is the highest price of the date.

Display the shape

```
print(df.shape)
```

Display the column names

```
print(df.columns)
```

Value count (Number of records by company)

```
df.symbol.value_counts()
```

Unique Values

```
df.symbol.unique()
```

Number of Unique Values

```
display(df.symbol.unique().shape)
```

First 20 Unique values

```
df.symbol.unique()[0:20]
```

Number of Records

```
print(len(df.symbol.values))
```

Display Info

```
df.info()
```

Describe the column

```
# Gives the details of each column of the dataset like mean, max, etc
df.describe()
```

Check for NULL Values

```
# checking whether there is any null value in the dataset
# .sum() will give the total no. of null value column-wise

df.isnull().sum()
```

Get all Unique Dates

```
df.date.unique()
```

Display Date as Data Frame

```
pd.DataFrame(df.date.unique())
```

Check for Duplicate Records

```
df.duplicated().sum()
```

Load the File

#Calling the file in nyse named securities.csv, It has the company details

comp_info = pd.read_csv('securities.csv')

comp_info

Data description

Table consists of 504 entries of descriptions of companies.

column1-- Ticker Symbol It is the symbol of companies, same as the symbol in df

column2-- Security It is the name of a corporation like amazon.com Inc

column4-- GICS Sector It is the type of the company. eg Adobe Systems Inc is Information Technology

column5-- GICS Sub Industry It is the sub department of the type of company. eg Adobe Inc is Application Software

column6-- Address of headquarters It is company head base(main office) location

Check for Unique count

comp_info["Ticker symbol"].nunique()

Check for info

comp_info.info()

Check for Null Values

comp_info.isnull().sum()

Describe the Details

```
comp_info.describe()
```

Search for Particular company details

```
# for locating specific data here.... in security column of string that starts with "Face"

comp_info.loc[comp_info.Security.str.startswith('Face') , :]
```

Example – Search for another company - Acc

```
comp_info.loc[comp_info.Security.str.startswith('Acc') , :]
```

Select any 6 companies using the above method for visualizations on respective opening and closing stock prices.

- Yahoo Inc,
- Xerox Corp,
- Microsoft Corp
- Facebook
- Adobe Systems Inc
- Goldman Sachs Group

```
# here we locate Ticker symbol of company with security like Yahoo, Xerox, Adobe etc, then taking their no. of times entry and thier "Ticker symbol"

In the result, left side 6.181,212 etc the index values corresponds to the selected companies from  comp info

comp_plot = comp_info.loc[(comp_info["Security"] == 'Yahoo Inc.') | (comp_info["Security"] == 'Xerox Corp.') | (comp_info["Security"] == 'Adobe Systems Inc')
```

```
        | (comp_info["Security"] == 'Microsoft Corp.') | (comp_info["Security"]
== 'Adobe Systems Inc')
        | (comp_info["Security"] == 'Facebook') | (comp_info["Security"] ==
'Goldman Sachs Group') , ["Ticker symbol"] ]["Ticker symbol"]
print(comp_plot)
```

For Loop to display 6 Company names

```
for i in comp_plot:
    print (i)
```

Create a function to plot Graphs for selected companies opening stock and closing stock against time

```
def plotter(code):
    # Function used to create graphs for 6 companies
    global closing_stock ,opening_stock
    #creating plot of all 6 company for opening and closing stock  total 12 graphs
    # Below statement create 2X2 empty chart
    f, axs = plt.subplots(2,2,figsize=(15,8))
    # total 12 graphs
    # creating plot opening prize of particular company
    plt.subplot(212)
    #taking name of the company as code, get all records related to one
company
    company = df[df['symbol']==code]
    #taking the values of one company and taking its open column values to 1D
array
    company = company.open.values.astype('float32')
    #reshaping the open stock value from 1D  to 2D .
    company = company.reshape(-1, 1)
```

```
    # putting the value of company in opening_stock
    opening_stock = company
    # plotting the data with green graph between "Time" and "prices vs time"

    plt.grid(True)# enalbling the grid in graph
    plt.xlabel('Time') # setting X axis as time
    # setting Y axis as company name + open stock prices
    plt.ylabel(code + " open stock prices")
    plt.title('prices Vs Time') # setting title
    plt.plot(company , 'g') # calling the graph with green graph line

    # creating plot closing prize of particular company
    plt.subplot(211)
    #taking name of the company as code
    company_close = df[df['symbol']==code]
    #taking the values of one company and taking its close column values
    company_close = company_close.close.values.astype('float32')
    #reshaping the open column value in 1D and calling it closing_stock
    # -1 for unknown dimension
    company_close = company_close.reshape(-1, 1)
    # putting company_close value in closing_stock
    closing_stock = company_close
    # plotting the data graph between "Time" and "prices vs time"
    plt.xlabel('Time') # setting x axis as time
    plt.ylabel(code + " close stock prices")# setting y axis as company name +
open stock prices
    plt.title('prices Vs Time') # setting title as price vs time
```

```
    plt.grid(True) # enabling the grid in graph
    plt.plot(company_close , 'b') #creating the data graph in blue graph line
    plt.show() # calling the graph
```

Calling the graphs through the function

```
for i in comp_plot:
    plotter(i)
```

Plot Summary

Adobe report

As we can see, that close and open started around 38 points and shows an increasing growth with time. The open and close graph closes at a very high position (around 105) compared to the start.

Facebook report

As the graph shows, open and close started around 30 points. And from there, it shows a bit downfall time 150. But then it shows a massive uprise and ends at the120 points for both open and close.

Goldman Sachs

As we see in graphs of open and close stocks, we see that Goldman has taken many up and downs almost the whole journey. It starts from high points than others around 175 points and faced major downfall but at last, it reached higher to 240 points.

Microsoft

As the graphs explain open and close stock for Microsoft started from 30 points. And from there they show total elevation to 65 points. It is a great option for investment

Xerox

As the graphs suggest for Xerox, it started from very low points of 9 and from there it struggled to progress. There was a major fall after some time and in the end, the stock prices are still nearly the same.

Yahoo

As we see in the graphs, Yahoo prices remained the same for some time. But then they show effective progress. And at the end, they go to around 40 points.

Extract closing stock of one company, here YHOO

```
stocks= np.array (df[df.symbol.isin (['YHOO'])].close)
print(stocks)
```

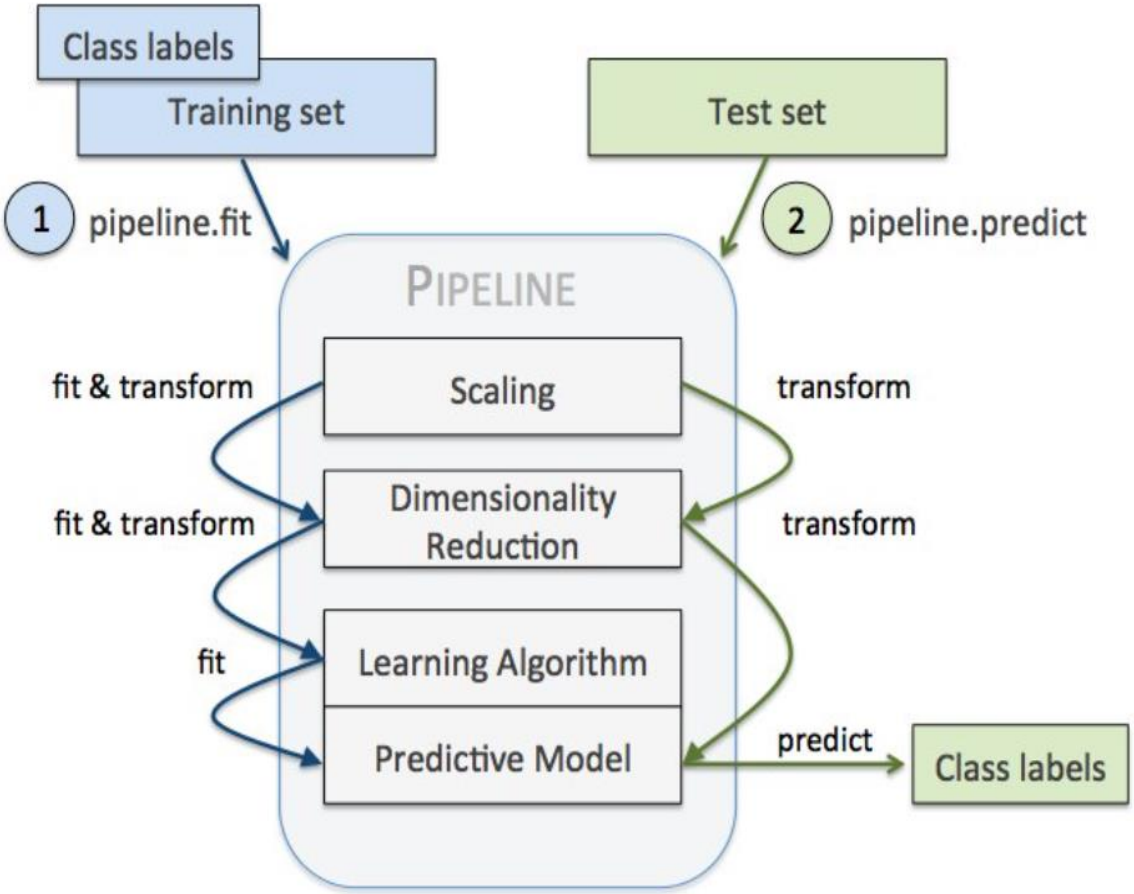
Display Shape

```
display (stocks.shape)
```

Convert stock to 2D array

```
stocks = stocks.reshape(len(stocks) , 1)
print (stocks.shape)
print(stocks)
```

Remaining Steps to be Followed



Min Max Scaler

Feature scaling the vector for better model performance.

```
from sklearn.preprocessing import MinMaxScaler
#scaling features between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
stocks = scaler.fit_transform(stocks)
display (stocks)
```

Train Test Split (80 % Training and 20 % Test Data)

```
Total Number of Records
print (stocks.shape)

# 80 % Training record count
train = int(len(stocks) * 0.80)
print (train)

# 20 % record count    (Total – 80%)
test = len(stocks) - train
print (test)
```

Training Data

```
#divinding the values of stocks data to train from 0 to 1409 i.e 80% data
train = stocks[0:train]
display (train.shape)
print(train)
```

Test Data

```
#divinding the values of stocks data to test from train ending to stock data ending i.e rest 20% data
test = stocks[len(train) : ]
display(test.shape)
display (test)
```

Function to prepare the data, use two previous values to predict for third data

```
#creating function to create trainX,testX and target(trainY, testY)
def process_data(data , n_features):
    dataX, dataY = [], [] # creating data for dataset and dividing inta X,Y
    for i in range(len(data)-n_features):
        # taking i range from total size- 3
        a = data[i:(i+n_features), 0]
        # Here a is value of data from i to i+ n_features, ie two values and put it in dataX
        dataX.append(a) #putting a in dataX
        #here dataY takes the value of data of i + n_features
        dataY.append(data[i + n_features, 0])
        # putting i+ n_features in  dataY
    return np.array(dataX), np.array(dataY)
# returning dataX and dataY in array
```

Call the function to prepare training data

n_features:

This is the number of previous period's data required to do the current prediction. In this case, use two previous records data to predict the current value.

```
n_features = 2
# Here we create train X, Train Y and test X, Test Y data where trainX, testX has two value is each block
```

```
trainX, trainY = process_data(train, n_features)
print(trainX.shape , trainY.shape)
```

Call the function to prepare test data - n_features = 2

```
testX, testY = process_data(test, n_features)
print (testX.shape , testY.shape)
```

Call the function to prepare full data - n_features = 2

```
stocksX, stocksY = process_data(stocks, n_features)
print (stocksX.shape , stocksY.shape)
```

Display the First 10 Records from Train X

```
display (trainX[:10])
```

Display the First 10 Records from TrainY

```
display (trainY[:10])
```

Reshape to 3D array to use for LSTM /Deep Learning model

```
# reshaping trainX and testX to use in deep learning model
trainX = trainX.reshape(trainX.shape[0] , 1 ,trainX.shape[1])
display (trainX.shape)
```

Reshape to 3D array test X

```
testX = testX.reshape(testX.shape[0] , 1 ,testX.shape[1])
display (testX.shape)
```

Reshape to 3D array full data set

```
stocksX= stocksX.reshape(stocksX.shape[0] , 1 ,stocksX.shape[1])
display (stocksX.shape)
```

Import Libraries

```
# helps us do mathematical operations
import math
# for setting layers one by one neural layer in model
from keras.models import Sequential
# types of layers
from keras.layers import Dense , BatchNormalization , Dropout , Activation
# types of RNN
from keras.layers import LSTM , GRU
#It puts the data in between given range to set data before putting layer
from sklearn.preprocessing import MinMaxScaler
# In this method the errors in column is squared and then mean is found
from sklearn.metrics import mean_squared_error
# Optimizers used
from keras.optimizers import Adam , SGD , RMSprop
```

keras

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

RNN

recurrent neural network processes sequences — whether daily stock prices, sentences, or sensor measurements — one element at a time while retaining a memory (called a state) of what has come previously in the sequence.Recurrent means the output at the current time step becomes the input to the next time step. At each element of the sequence, the model considers not just the current input, but what it remembers about the preceding elements.

Optimizers

Optimization algorithms helps us to minimize (or maximize) an Objective function (another name for Error function) E(x) which is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values(Y) from the set of predictors(X) used in the model.

Create Check Points

```
#Checkpointing the model when required and using other call-backs.

filepath="stock_weights1.hdf5"

from keras.callbacks import ReduceLROnPlateau , ModelCheckpoint

# ReduceLROnPlateau- This reduce the learning rate when the matrix stop improving or  too close to reduce overfitting

lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.1, epsilon=0.0001, patience=1, verbose=1)

#This check point will stop processing, if the model is not improving.

checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='max')
```

Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. You can pass a list of callbacks (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes. The relevant methods of the callbacks will then be called at each stage of the training.

ReduceLROnPlateau

Reduce learning rate when a metric has stopped improving.Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

ModelCheckpoint

Save the model after every epoch.filepath can contain named formatting options, which will be filled with the values of epoch and keys in logs (passed in on_epoch_end).

Create the model – Fully connected Layer

```
# creating model for training data using sequential to give series wise output between layers

model = Sequential()

# GRU or Gated Recurrent Unit used for matrix manipulation within Recurrent layer

#This is the input Layer

model.add(GRU(256 , input_shape = (1 , n_features) , return_sequences=True))

#dropout is used to remove overfitting data on each layer of neural network

model.add(Dropout(0.4))
```

```
#Long Short Term Memory is a type of RNN specially used for time series problems
model.add(LSTM(256))
#dropout is used to remove overfitting data on each layer of neural network
model.add(Dropout(0.4))
#Dense layer are fully connected neural networks
model.add(Dense(64 ,  activation = 'relu'))
#This is the output Layer, Output is only one neuron
model.add(Dense(1))
#for getting the details of our models
print(model.summary())
```

Compile the Model

```
# Selecting the loss measurement metrics and optimizer for our model, to find out mean square error
model.compile(loss='mean_squared_error', optimizer=Adam(lr = 0.0005) , metrics = ['mean_squared_error'])
```

Fit the Model

```
# fitting the data i.e training the trainX, to relate to trainY
# epochs is the times each data in send to fit
# batch size is the size of information send at a time
# validation_data is the validation or data used for testing
history = model.fit(trainX, trainY, epochs=100 , batch_size = 128 ,
        callbacks = [checkpoint , lr_reduce] , validation_data = (testX,testY))
```

Predict the value for testX, display top 10 records

```
test_pred = model.predict(testX)
display (test_pred [:10])
```

Inverse Transform the data – Convert the data to Original form

```
test_pred = scaler.inverse_transform(test_pred)
display (test_pred [:10])
```

Reshape and Display Original Data

```
testY = testY.reshape(testY.shape[0] , 1)
#Converting reshaped list in 1D array so that it will be efficient in plotting
testY = scaler.inverse_transform(testY)
# taking testY from 1 to 10
display (testY[:10])
```

Display the accuracy

```
from sklearn.metrics import r2_score
r2_score(testY,test_pred)
```

Plot the Graph Actual and Predicted Data

```
# Ploting the graph of stock prices with time
print("Red - Predicted Stock Prices  ,  Blue - Actual Stock Prices")
plt.rcParams["figure.figsize"] = (15,7)
# testY is the blue line
plt.plot(testY , 'b')
```

```
# pred is the red line
plt.plot(test_pred , 'r')
# Setting x axis as time
plt.xlabel('Time')
# Setting y axis as stock prices
plt.ylabel('Stock Prices')
# setting title
plt.title('Check the accuracy of the model with time')
# enabling grids in graph
plt.grid(True)
# it call the graph with labels, titles, lines
plt.show()
```

Prediction Training Data and Check Accuracy

```
train_pred = model.predict(trainX)
train_pred = scaler.inverse_transform(train_pred)
trainY = trainY.reshape(trainY.shape[0] , 1)
trainY = scaler.inverse_transform(trainY)
print ('Display Accuracy Training Data')
display (r2_score(trainY,train_pred))
```

Plot the Graph

```
# Ploting the graph of stock prices with time - Training Data
print("Red - Predicted Stock Prices  ,  Blue - Actual Stock Prices")
plt.rcParams["figure.figsize"] = (15,7)
```

```
plt.plot(trainY  , 'b')
plt.plot(train_pred, 'r')
plt.xlabel('Time')
plt.ylabel('Stock Prices')
plt.title('Check the accuracy of the model with time')
plt.grid(True)
plt.show()
```

Prediction Full Data

```
stocks_pred = model.predict(stocksX)
stocks_pred = scaler.inverse_transform(stocks_pred)
stocksY = stocksY.reshape(stocksY.shape[0] , 1)
stocksY = scaler.inverse_transform(stocksY)
print ('Display Accuracy Training Data')
display (r2_score(stocksY,stocks_pred))
```

Plot the Graph

```
plt.rcParams["figure.figsize"] = (15,7)
plt.plot(stocksY  , 'b')
plt.plot(stocks_pred, 'r')
plt.xlabel('Time')
plt.ylabel('Stock Prices')
plt.title('Check the accuracy of the model with time')
plt.grid(True)
plt.show()
```

Concat original and prediction data

```
# Extract the data related to company - YHOO
results= df[df.symbol.isin (['YHOO'])]
# Update the data frame starting with 2nd records , since first prediction is for 2nd record
results= results [2:]
# Reset the index 0, 1,2 etc
results = results.reset_index(drop=True)
# Convert Predicted Value to Data Frame
df_stocks_pred= pd.DataFrame(stocks_pred, columns = ['Close_Prediction'])
# Concat Original and prediction data
results= pd.concat([results,df_stocks_pred],axis =1)
results.to_excel('results.xlsx')
display(results)
```