<u>Stock Price Prediction Using Machine Learning</u>

By

21781A33C4 (R Pavithra)

21781A33D0 (S Soniya)

21781A33D1 (Sanjana K)

21781A33E9 (V Naga Shanthi)

Guided by

Prof. Ms Raasha

A Dissertation Submitted to

SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY, An Autonomous Institution affiliated to 'JNTU Ananthapur' in Partial Fulfilment of the Bachelor of Technology (***Computer Engineering***) with Specialization in ***Artificial Intelligence and Machine Learning.***

May 2024

SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY

R.V.S. Nagar Tirupathi Road, Andhra Pradesh– 517127

DATA PREPROCESSING:

Importing The Libraries

```
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
import sklearn
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import math
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
import numpy as np
import pandas as pd
from itertools import product
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings('ignore')
```

Read The Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.
In pandas, we have a function called read_csv() to read the dataset. As a parameter, we have to give the directory of the CSV file.

Data Loading

```
amd = pd.read_csv('Data/AMD (1980-2023).csv')
asus = pd.read_csv('Data/ASUS (2000-2023).csv')
intel = pd.read_csv('Data/Intel (1980-2023).csv')
msi = pd.read_csv('Data/MSI (1962-2023).csv')
nvidia = pd.read_csv('Data/NVIDIA (1999-2023).csv')
```

We can print the first 5 rows of the datasets using the .head() method as shown in the below screenshots.

In [4]:
```
amd.head()
```
Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-03-18	0.0	3.125000	2.937500	3.031250	3.031250	727200
1	1980-03-19	0.0	3.083333	3.020833	3.041667	3.041667	295200
2	1980-03-20	0.0	3.062500	3.010417	3.010417	3.010417	159600
3	1980-03-21	0.0	3.020833	2.906250	2.916667	2.916667	130800
4	1980-03-24	0.0	2.916667	2.635417	2.666667	2.666667	436800

In [4]:
```
asus.head()
```
Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2000-01-05	438.747223	446.535675	436.151154	438.747223	93.584663	6.106176e+09
1	2000-01-06	440.045380	447.833862	436.151154	437.449310	93.307838	6.545984e+09
2	2000-01-07	432.256927	433.555084	425.766632	428.362701	91.369652	4.764317e+09
3	2000-01-10	434.853271	454.324158	434.853271	450.429901	96.076584	1.199988e+10
4	2000-01-11	463.410767	463.410767	442.641449	443.939606	94.692215	1.423350e+10

In [5]:
```
intel.head()
```
Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-03-18	0.325521	0.328125	0.322917	0.322917	0.184470	17068800
1	1980-03-19	0.330729	0.335938	0.330729	0.330729	0.188933	18508800
2	1980-03-20	0.330729	0.334635	0.329427	0.329427	0.188189	11174400
3	1980-03-21	0.322917	0.322917	0.317708	0.317708	0.181494	12172800
4	1980-03-24	0.316406	0.316406	0.311198	0.311198	0.177775	8966400

In [6]:
```
msi.head()
```
Out[6]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1962-01-03	0.0	1.444702	1.427952	1.436327	0.632343	77611
1	1962-01-04	0.0	1.438421	1.411202	1.423765	0.626812	59701
2	1962-01-05	0.0	1.432140	1.394452	1.415390	0.623125	107462
3	1962-01-08	0.0	1.432140	1.390264	1.390264	0.612063	89551
4	1962-01-09	0.0	1.402827	1.356764	1.356764	0.597315	83581

In [7]:
```
nvidia.head()
```
Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1999-01-25	0.442708	0.458333	0.410156	0.453125	0.415786	51048000
1	1999-01-26	0.458333	0.467448	0.411458	0.417969	0.383527	34320000
2	1999-01-27	0.419271	0.429688	0.395833	0.416667	0.382332	24436800
3	1999-01-28	0.416667	0.419271	0.412760	0.415365	0.381137	22752000
4	1999-01-29	0.415365	0.416667	0.395833	0.395833	0.363215	24403200

Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Checking for missing values
- Data manipulation
- Resampling the data
- Merging and splitting data into test and train variables

Note: These are the general steps of pre-processing the data before using it for training models. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Checking For Missing Values

For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

In [3]:
```
amd.isnull().sum()
```
Out[3]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [12]:
```
asus.isnull().sum()
```
Out[12]:
```
Date           0
Open         123
High         123
Low          123
Close        123
Adj Close    123
Volume       123
dtype: int64
```

In [11]:
```
intel.isnull().sum()
```
Out[11]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [10]:
```
msi.isnull().sum()
```
Out[10]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [9]:
```
nvidia.isnull().sum()
```
Out[9]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Data Manipulation

Since we found null values in the dataset related to ASUS company we are going to drop the null values but we can also fill those null values using mean/median/mode of the respective volumns.

```
asus = asus.dropna()
```

To convert the date strings to time stamps we are going to perform below steps.
First let's check the type of data of the column "Year" using the type() function.

```
type(df['Year'][0])
```

```
numpy.int64
```

We will convert the data of "Year" column using the pandas "to_datetime" function. We can change the 'Date' columns of all the datasets using a for loop to reduce Lines of code..

```
data_list = [amd,asus,intel,msi,nvidia]

for data in data_list:
    data['Date'] = pd.to_datetime(data['Date'])
```

Resampling The Data

Now we are going to add new columns to all the loaded datasets so that when we merge our data it will be easier to find which rows of data belong to which company and also so that our model will be able to differentiate the rows of data based on the value of the new column named "Company". The new column "Company" will be a categorical value ranging from 0 to 4 where each number denotes a company as shown below. We are going to process the datasets as mentioned above using a for loop.
We will also three new columns of data namely "Year", "Day" and "Month" which are derived from the "Date" column. We will use these

three columns for training not the "Date" columns but we use the "Date" column for analysis in later stages.

```
data_list = [amd,asus,intel,msi,nvidia]

# Below names list will be used to add a new 'Company' column to every dataset
# 0: AMD
# 1: ASUS
# 2: INTEL
# 3: MSI
# 4: NVIDIA
names = [0,1,2,3,4]
index = 0
for data in data_list:
    dates = data['Date']
    data['Company'] = np.repeat(names[index],len(data))
    data['Year'] = dates.dt.year
    data['Month'] = dates.dt.month
    data['Day'] = dates.dt.day
    index+=1
```

Merging And Splitting Data Into Test And Train Variables

Now we are going to merge our datasets, simultaneously we are also going to split the data to test and train variables using the below piece of code with a train to test ratio of 80 to 20.

First we are going to take 2 lists of train and test in which we are going to append train and test splits of every dataset into these 2 lists. In the output each line represents the data of companies in order as stored in the data_list in below code.

```
data_list = [amd,asus,intel,msi,nvidia]
test_data = []
train_data = []
for data in data_list:
    train = data[:int(0.8*len(data))]
    test = data[int(0.8*len(data)):]
    train_data.append(train)
    test_data.append(test)
    print(test.shape,train.shape)
```

```
(2172, 11) (8687, 11)
(1138, 11) (4548, 11)
(2172, 11) (8687, 11)
(3085, 11) (12339, 11)
(1219, 11) (4875, 11)
```

Next we are going to merge the data of each variable of train_data and test_data using the pandas.concat() function as shown below.

```
train_data = pd.concat(train_data)
test_data = pd.concat(test_data)
print(train_data.shape)
print(test_data.shape)
```

```
(39136, 11)
(9786, 11)
```

Finally we are going to split the train_data and test_data variables into x_train, y_train, x_test and y_test using the below piece of code.

```
x_train = train_data[['Open', 'High', 'Low', 'Volume', 'Year','Month', 'Day','Company']]
x_test = test_data[['Open', 'High', 'Low', 'Volume', 'Year','Month', 'Day','Company']]
y_train = train_data['Close']
y_test = test_data['Close']

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(39136, 8)
(9786, 8)
(39136,)
(9786,)
```

Exploratory Data Analysis

Descriptive Statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

In [119]:
```
amd.describe(include='all')
```

Out[119]:

	Date	Open	High	Low	Close	Adj Close	Volume
count	10859	10859.000000	10859.000000	10859.000000	10859.000000	10859.000000	1.085900e+04
unique	10859	NaN	NaN	NaN	NaN	NaN	NaN
top	1980-03-18	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	16.350958	17.010710	16.280518	16.648080	16.648080	1.819320e+07
std	NaN	22.396174	22.668726	21.705814	22.196357	22.196357	2.794779e+07
min	NaN	0.000000	1.690000	1.610000	1.620000	1.620000	0.000000e+00
25%	NaN	4.937500	5.405000	5.120000	5.265000	5.265000	1.216500e+06
50%	NaN	9.810000	10.000000	9.562500	9.760000	9.760000	6.745600e+06
75%	NaN	16.000000	16.250000	15.687500	16.000000	16.000000	2.242935e+07
max	NaN	163.279999	164.460007	156.100006	161.910004	161.910004	3.250584e+08

In [120]:
```
asus.describe(include='all')
```

Out[120]:

	Date	Open	High	Low	Close	Adj Close	Volume
count	5809	5686.000000	5686.000000	5686.000000	5686.000000	5686.000000	5.686000e+03
unique	5809	NaN	NaN	NaN	NaN	NaN	NaN
top	2000-01-05	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	290.387916	293.572143	286.947741	290.132960	133.244407	1.027367e+09
std	NaN	76.336647	77.122349	75.308642	75.978864	67.614911	2.186379e+09
min	NaN	127.106941	130.196335	127.106941	130.196335	30.318747	0.000000e+00
25%	NaN	234.106556	236.500000	231.500000	234.000000	79.519874	1.696933e+06
50%	NaN	277.500000	280.000000	275.000000	277.500000	125.252792	3.201000e+06
75%	NaN	331.005699	335.315742	327.000000	331.007599	170.654297	1.125660e+09
max	NaN	567.667419	575.104126	547.836243	565.188538	330.402832	2.833812e+10

In [121]:
```
intel.describe(include='all')
```
Out[121]:

	Date	Open	High	Low	Close	Adj Close	Volume
count	10859	10859.000000	10859.000000	10859.000000	10859.000000	10859.000000	1.085900e+04
unique	10859	NaN	NaN	NaN	NaN	NaN	NaN
top	1980-03-18	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	19.834106	20.105102	19.565597	19.833531	14.636382	5.056303e+07
std	NaN	17.514871	17.756945	17.278878	17.514300	14.829796	3.487815e+07
min	NaN	0.218750	0.218750	0.216146	0.216146	0.123476	0.000000e+00
25%	NaN	1.328125	1.343750	1.304688	1.328125	0.758707	2.708505e+07
50%	NaN	20.277344	20.562500	20.010000	20.280001	12.665797	4.460160e+07
75%	NaN	29.980000	30.425000	29.520000	29.950001	19.808317	6.477205e+07
max	NaN	75.625000	75.828125	73.625000	74.875000	63.608189	5.677088e+08

In [123]:
```
msi.describe(include='all')
```
Out[123]:

	Date	Open	High	Low	Close	Adj Close	Volume
count	15424	15424.000000	15424.000000	15424.000000	15424.000000	15424.000000	1.542400e+04
unique	15424	NaN	NaN	NaN	NaN	NaN	NaN
top	1962-01-03	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	44.977492	46.412619	45.224755	45.825400	37.652783	1.997183e+06
std	NaN	54.883169	54.841002	53.578205	54.222402	51.144917	2.347513e+06
min	NaN	0.000000	0.866821	0.808196	0.845884	0.376771	0.000000e+00
25%	NaN	3.768789	5.175804	5.025052	5.100428	2.625478	5.059500e+05
50%	NaN	23.517244	23.856436	23.230058	23.554932	16.259139	1.294556e+06
75%	NaN	67.235199	67.988953	66.486078	67.235199	52.269761	2.627212e+06
max	NaN	286.209991	287.420013	283.739990	286.140015	286.140015	4.717163e+07

In [124]:
```
nvidia.describe(include='all')
```
Out[124]:

	Date	Open	High	Low	Close	Adj Close	Volume
count	6094	6094.000000	6094.000000	6094.000000	6094.000000	6094.000000	6.094000e+03
unique	6094	NaN	NaN	NaN	NaN	NaN	NaN
top	1999-01-25	NaN	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	30.987375	31.585369	30.369141	31.003835	30.743154	6.134634e+07
std	NaN	59.862014	61.089822	58.564768	59.881405	59.882440	4.399760e+07
min	NaN	0.348958	0.355469	0.333333	0.341146	0.313034	1.968000e+06
25%	NaN	2.671094	2.750000	2.598027	2.670208	2.450174	3.440110e+07
50%	NaN	4.285000	4.377500	4.210000	4.290000	3.946429	5.151250e+07
75%	NaN	26.690000	27.198125	26.404999	26.818125	26.440031	7.462690e+07
max	NaN	335.170013	346.470001	320.359985	333.760010	333.350800	9.230856e+08

Visual Analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we don’t have much need to perform univariate analysis to understand the data as most of the columns provided are continuous.

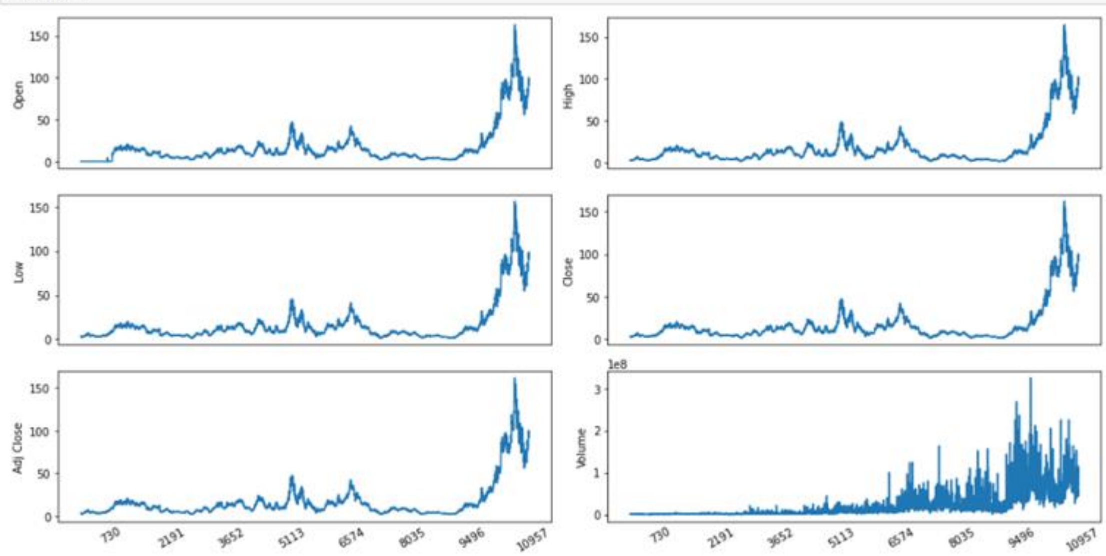
Bivariate analysis

To find the relation between two features we use bivariate analysis.

```
In [126]: import matplotlib.dates as mdates
          # Plot line charts
          df_plot = amd.drop(columns=['Date'])

          ncols = 2
          nrows = int(round(df_plot.shape[1] / ncols, 0))

          fig, ax = plt.subplots(nrows=nrows, ncols=ncols, sharex=True, figsize=(14, 7))
          for i, ax in enumerate(fig.axes):
                  sns.lineplot(data = df_plot.iloc[:, i], ax=ax)
                  ax.tick_params(axis="x", rotation=30, labelsize=10, length=0)
                  ax.xaxis.set_major_locator(mdates.AutoDateLocator())
          fig.tight_layout()
          plt.show()
```



In the line plots visualised below following observations can be observed:

- There is sudden increase in the prices of stocks of AMD in the recent years and the price of the stock didn’t retain for a long period but it soon fell to a lower price.
- The volume of the stocks being traded have increased tremendously in the recent years.

Similar to this, analysis on other companies can be done.