Mini Project Report
On
Stock Market Analysis using Python

Submitted by

ANIKET KASHYAP 21052050
CHETAN DEV MASKARA 21052151
HARSHIL GAUTAM 21052155

School of Computer Engineering
KIIT - DU

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Table of Contents

Chapter 1

Introduction

Stock market analysis involves evaluating the performance of individual stocks, entire sectors, or even entire markets to make informed investment decisions. It encompasses various techniques and methodologies to assess the past performance, current status, and potential future trends of stocks or markets.

Python is a powerful programming language commonly used for stock market analysis due to its extensive libraries for data manipulation, statistical analysis, and visualization. Here's how Python can be used for stock market analysis:

Data Collection: Python can be used to collect stock market data from various sources such as financial websites, APIs (Application Programming Interfaces), and databases. Libraries like Pandas, BeautifulSoup, and requests are commonly used for web scraping and data retrieval.

Data Cleaning and Preprocessing: Once the data is collected, Python can be used to clean and preprocess it by removing missing values, handling outliers, and formatting the data into a suitable format for analysis. Pandas is particularly useful for data manipulation and preprocessing tasks.

Analysis Techniques: Python provides a wide range of libraries and tools for performing both fundamental and technical analysis. For fundamental analysis, libraries like Pandas, NumPy, and Statsmodels can be used to calculate financial ratios, perform regression analysis, and conduct valuation modeling. For technical analysis, libraries like TA-Lib and Matplotlib can be used to plot charts, calculate technical indicators, and identify trading signals.

Visualization: Python offers powerful visualization libraries such as Matplotlib, Seaborn, and Plotly, which can be used to create interactive charts, graphs, and dashboards to visualize stock market data and analysis results.

Chapter 2

Problem Statement & Objectives

Problem Statement: Develop a Python-based stock market analysis tool to provide investors with actionable insights and decision support in navigating the dynamic and complex stock market landscape. The tool should offer comprehensive analysis of historical and real-time stock market data to identify trends, patterns, and potential investment opportunities, while also assessing risks and assisting in portfolio management.

Objective: The objective of this project is to create a robust and user-friendly Python application capable of:

1. Gathering and processing vast amounts of stock market data from various reliable sources.
2. Performing comprehensive analysis, including but not limited to: trend analysis, volatility assessment, correlation studies, and fundamental analysis.
3. Implementing advanced statistical and machine learning techniques to forecast stock prices and identify potential market movements.
4. Visualizing data and analysis results in clear and intuitive formats, such as interactive charts, graphs, and dashboards.
5. Providing personalized recommendations and insights tailored to individual investor preferences and risk profiles.
6. Supporting backtesting of trading strategies and evaluating their performance.
7. Ensuring scalability, reliability, and efficiency in handling large datasets and real-time data feeds.
8. Facilitating seamless integration with brokerage platforms and other financial tools for streamlined decision-making and execution.
9. Continuously updating and enhancing the tool with the latest advancements in data analysis, machine learning, and financial research to stay ahead in the ever-evolving stock market landscape.
10. Ultimately, empowering investors with the knowledge and tools necessary to make informed investment decisions and achieve their financial goals.

Chapter 3
Python Packages Used Details

Name	Functions Used	Explanation
Pandas	read_csv	Used to read data from csv files and store in tabular format
	isna()	used to detect if there are any null values
	shape	when applied to a pandas DataFrame or NumPy array, returns a tuple representing the dimensions of the data structure
	info	Gives info about the dataframe and each attribute
Numpy	info	Gives info about the dataframe and each attribute
	desctibe	Describes the entire dataframes
	drop	Used to drop a certain attribute
MATPLOTLIB	Plot	Used for plotting the graph between different attributes and datasets
	Figure	Sets the figure size
	Title	Used to mention the title of graph
	Xlabel	Used to mention the xaxis
	Ylabel	Used to mention the yaxis
	show	Used to display the graph
	bar	Used to plot a bar graph
	pie	Used to plot a pie chart

Chapter 4

Source code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


ITC=pd.read_csv("ITC.csv")
TITAN=pd.read_csv("TITAN.csv")
WIPRO=pd.read_csv("WIPRO.csv")


#viewing first 15 data from datset
print(tata_motors.head(15))
          Date Symbol Series  Prev Close    Open    High     Low    Last  \
0   2000-01-03  TELCO     EQ      201.60  207.40  217.25  207.40  217.00
1   2000-01-04  TELCO     EQ      216.75  217.00  219.00  206.00  211.90
2   2000-01-05  TELCO     EQ      208.20  194.00  217.80  194.00  213.10
3   2000-01-06  TELCO     EQ      213.25  215.00  229.90  215.00  222.00
4   2000-01-07  TELCO     EQ      222.10  224.00  239.90  223.10  239.90
5   2000-01-10  TELCO     EQ      239.90  259.00  259.10  248.00  259.10
6   2000-01-11  TELCO     EQ      258.85  260.00  260.00  238.15  238.15
7   2000-01-12  TELCO     EQ      238.15  240.00  246.00  229.00  242.45
8   2000-01-13  TELCO     EQ      240.25  243.00  245.00  231.60  233.00
9   2000-01-14  TELCO     EQ      235.45  234.95  243.00  233.00  239.80
10  2000-01-17  TELCO     EQ      239.05  242.95  258.10  242.95  251.00
11  2000-01-18  TELCO     EQ      250.65  251.00  252.50  245.05  250.00
12  2000-01-19  TELCO     EQ      248.00  251.00  257.90  242.60  249.50
13  2000-01-20  TELCO     EQ      250.50  250.00  250.00  233.60  234.00
14  2000-01-21  TELCO     EQ      235.70  230.10  233.80  225.50  231.10

     Close    VWAP   Volume      Turnover  Trades  Deliverable Volume  \
0   216.75  214.28   676126  1.448775e+13     NaN                 NaN
1   208.20  209.50   679215  1.422962e+13     NaN                 NaN
2   213.25  210.33  1120951  2.357684e+13     NaN                 NaN
3   222.10  225.29  1968998  4.435932e+13     NaN                 NaN
4   239.90  236.32  2199431  5.197636e+13     NaN                 NaN
5   258.85  257.20  1720542  4.425303e+13     NaN                 NaN
6   238.15  246.63  1207916  2.979082e+13     NaN                 NaN
...
11         NaN
12         NaN
13         NaN
14         NaN
```

```
checking the size of data
ITC.shape
```

```
(5306, 15)
```

```
#Viewing Datatypes of all columns
print("ITC")
ITC.info()
```

```
ITC
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5306 entries, 0 to 5305
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Date               5306 non-null   object 
 1   Symbol             5306 non-null   object 
 2   Series             5306 non-null   object 
 3   Prev Close         5306 non-null   float64
 4   Open               5306 non-null   float64
 5   High               5306 non-null   float64
 6   Low                5306 non-null   float64
 7   Last               5306 non-null   float64
 8   Close              5306 non-null   float64
 9   VWAP               5306 non-null   float64
 10  Volume             5306 non-null   int64  
 11  Turnover           5306 non-null   float64
 12  Trades             2456 non-null   float64
 13  Deliverable Volume  4792 non-null   float64
 14  %Deliverble        4792 non-null   float64
dtypes: float64(11), int64(1), object(3)
memory usage: 621.9+ KB
WIPRO
<class 'pandas.core.frame.DataFrame'>
...
 13  Deliverable Volume  4792 non-null   float64
 14  %Deliverble        4792 non-null   float64
dtypes: float64(11), int64(1), object(3)
memory usage:  memory usage: 621.9+ KB                                  621.9+ KB
```

```
#Checking for Null Values
print("ITC")
ITC.isna().sum()
```

```
Date                    0
Symbol                  0
Series                  0
Prev Close              0
Open                    0
High                    0
Low                     0
Last                    0
Close                   0
VWAP                    0
Volume                  0
Turnover                0
Trades               2850
Deliverable Volume    514
%Deliverble           514
dtype: int64
```

=

```
#Description of Data in the Dataframe and rounding its values up to three decimal places
```

```
ITC.describe().round(3)
```

	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
count	5306.000	5306.000	5306.000	5306.000	5306.000	5306.000	5306.000	5.306000e+03	5.306000e+03	2456.000	4.792000e+03	4792.000
mean	420.359	420.632	426.629	414.242	420.250	420.274	420.523	7.173165e+06	1.799399e+14	94563.758	4.571758e+06	0.592
std	328.168	328.240	333.333	323.333	328.108	328.165	328.436	9.613497e+06	2.155242e+14	59571.722	5.040517e+06	0.128
min	115.450	115.000	116.200	114.400	115.500	115.450	115.390	6.797000e+03	5.340132e+11	1425.000	6.120000e+03	0.098
25%	201.650	202.000	204.500	198.500	201.500	201.650	201.705	1.038020e+06	5.423007e+13	56465.250	1.614865e+06	0.516
50%	280.050	280.200	283.000	277.025	279.900	280.025	280.055	5.122630e+06	1.222452e+14	80494.500	3.569193e+06	0.612
75%	631.875	630.975	640.000	623.500	631.150	631.362	631.728	8.788539e+06	2.315038e+14	113228.000	6.023117e+06	0.686
max	1940.100	1946.000	1964.800	1916.300	1940.000	1940.100	1933.790	1.494797e+08	4.254992e+15	667965.000	8.629348e+07	0.997

```
TITAN.describe().round(3)
```

	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
count	5306.000	5306.000	5306.000	5306.000	5306.000	5306.000	5306.000	5.306000e+03	5.306000e+03	2456.00	4.792000e+03	4792.000
mean	709.231	709.990	723.313	696.431	709.449	709.484	710.415	1.172596e+06	7.796053e+13	47241.86	4.408763e+05	0.368
std	785.171	785.435	799.085	772.184	784.996	785.206	786.267	1.763858e+06	1.378542e+14	43331.44	7.022005e+05	0.165
min	27.500	27.000	28.800	27.000	27.750	27.500	27.880	2.000000e+02	8.125000e+08	993.00	3.600000e+02	0.024
25%	192.262	192.625	198.800	189.500	192.838	193.413	194.245	9.867325e+04	4.299007e+12	18126.50	4.992600e+04	0.249
50%	396.150	398.250	404.525	391.125	397.025	396.350	398.095	5.491360e+05	2.785515e+13	35076.50	2.012115e+05	0.352
75%	1017.725	1018.925	1046.500	991.188	1020.000	1019.000	1016.790	1.630698e+06	8.382494e+13	63520.25	6.247118e+05	0.473
max	4714.600	4730.000	4754.950	4559.900	4734.000	4714.600	4647.540	3.327661e+07	2.451345e+15	536406.00	2.169911e+07	1.000

```
WIPRO.describe().round(3)
```

	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
count	5306.000	5306.000	5306.000	5306.000	5306.000	5306.000	5306.000	5.306000e+03	5.306000e+03	2456.000	4.797000e+03	4797.000
mean	754.857	756.800	772.478	738.151	754.522	754.475	755.166	2.236123e+06	1.024537e+14	48112.936	9.957321e+05	0.459
std	794.522	803.372	825.260	768.613	794.383	794.159	795.826	4.827183e+06	1.741585e+14	46662.502	1.324731e+06	0.185
min	162.350	163.100	171.350	159.400	161.800	162.350	166.440	1.003000e+03	4.355942e+11	692.000	8.242000e+03	0.036
25%	380.525	380.575	386.900	374.475	380.612	380.525	380.480	6.937025e+05	4.319006e+13	25021.000	3.023500e+05	0.332
50%	503.400	504.000	510.800	494.500	503.800	503.250	503.450	1.152270e+06	6.579357e+13	36659.500	6.432110e+05	0.483
75%	679.225	680.000	690.888	670.812	679.925	678.688	680.795	2.004576e+06	1.075259e+14	55772.750	1.197829e+06	0.597
max	9587.450	10350.000	10350.000	8928.350	9640.000	9587.450	9607.140	1.303677e+08	5.075003e+15	793471.000	3.554140e+07	0.939

```
#Converting the “Date” column dtype from object to date
```

```
TITAN["Date"]=pd.to_datetime(tata_motors["Date"])
ITC["Date"]=pd.to_datetime(tata_steel["Date"])
WIPRO["Date"]=pd.to_datetime(tcs["Date"])
```

```
TITAN.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5306 entries, 0 to 5305
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype         
---  ------              --------------  -----         
 0   Date                5306 non-null   datetime64[ns]
 1   Symbol              5306 non-null   object        
 2   Series              5306 non-null   object        
 3   Prev Close          5306 non-null   float64       
 4   Open                5306 non-null   float64       
 5   High                5306 non-null   float64       
 6   Low                 5306 non-null   float64       
 7   Last                5306 non-null   float64       
 8   Close               5306 non-null   float64       
 9   VWAP                5306 non-null   float64       
 10  Volume              5306 non-null   int64         
 11  Turnover            5306 non-null   float64       
 12  Trades              2456 non-null   float64       
 13  Deliverable Volume  4792 non-null   float64       
 14  %Deliverble         4792 non-null   float64       
dtypes: datetime64[ns](1), float64(11), int64(1), object(2)
memory usage: 621.9+ KB
```

```
ITC.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5306 entries, 0 to 5305
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype         
---  ------              --------------  -----         
 0   Date                5306 non-null   datetime64[ns]
 1   Symbol              5306 non-null   object        
 2   Series              5306 non-null   object        
 3   Prev Close          5306 non-null   float64       
 4   Open                5306 non-null   float64       
 5   High                5306 non-null   float64       
 6   Low                 5306 non-null   float64       
 7   Last                5306 non-null   float64       
 8   Close               5306 non-null   float64       
 9   VWAP                5306 non-null   float64       
 10  Volume              5306 non-null   int64         
 11  Turnover            5306 non-null   float64       
 12  Trades              2456 non-null   float64       
 13  Deliverable Volume  4792 non-null   float64       
 14  %Deliverble         4792 non-null   float64       
dtypes: datetime64[ns](1), float64(11), int64(1), object(2)
memory usage: 621.9+ KB
```

```
WIPRO.info()

 <class 'pandas.core.frame.DataFrame'>
 RangeIndex: 5306 entries, 0 to 5305
 Data columns (total 15 columns):
  #   Column              Non-Null Count  Dtype
 ---  ------              --------------  -----
  0   Date                5306 non-null   datetime64[ns]
  1   Symbol              5306 non-null   object
  2   Series              5306 non-null   object
  3   Prev Close          5306 non-null   float64
  4   Open                5306 non-null   float64
  5   High                5306 non-null   float64
  6   Low                 5306 non-null   float64
  7   Last                5306 non-null   float64
  8   Close               5306 non-null   float64
  9   VWAP                5306 non-null   float64
  10  Volume              5306 non-null   int64
  11  Turnover            5306 non-null   float64
  12  Trades              2456 non-null   float64
  13  Deliverable Volume  4797 non-null   float64
  14  %Deliverble         4797 non-null   float64
 dtypes: datetime64[ns](1), float64(11), int64(1), object(2)
 memory usage: 621.9+ KB

#Dropping columns Trades, Deliverable Volume, and %Deliverable

ITC=ITC.drop(['Trades'], axis=1)
TITAN=TITAN.drop(['Deliverable Volume'], axis=1)
WIPRO=WIPRO.drop(['%Deliverble'], axis=1)


#Adding 3 more new columns to each of the Dataset

ITC['Month']=ITC["Date"].dt.month

ITC['Year']=ITC["Date"].dt.year

ITC['Day']=ITC["Date"].dt.day


WIPRO['Month']=WIPRO["Date"].dt.month

WIPRO['Year']=WIPRO["Date"].dt.year

WIPRO['Day']=WIPRO["Date"].dt.day


TITAN['Day']=TITAN['Date'].dt.day

TITAN['Year']=TITAN['Date'].dt.year
```

```
TITAN['Month']=TITAN['Date'].dt.month


#Price Comparision
plt.figure(figsize=(20,7))
plt.plot(ITC['Date'],ITC['Open'],color='blue',label='ITC')
plt.plot(TITAN['Date'],TITAN['Open'],color='grey',label='TITAN')
plt.plot(WIPRO['Date'],WIPRO['Open'],color='orange',label='WIPRO')

plt.title("Relation between ITC, Titan and Wipro Price")
plt.xlabel("Year")
plt.ylabel("Price")
plt.legend(title="")
plt.show()



#Volume Comparision
plt.figure(figsize=(20,7))
plt.plot(ITC['Date'],ITC['Volume'],color='blue',label='ITC')
plt.plot(TITAN['Date'],TITAN['Volume'],color='grey',label='TITAN')
plt.plot(WIPRO['Date'],WIPRO['Volume'],color='orange',label='WIPRO')


plt.title("Relation between ITC, Titan and Wipro Volume")
plt.xlabel("Year")
plt.ylabel("Volume")
plt.legend(title="")
```

```
plt.show()



#ITC ROI

sumTM=0 #total amount invested in ITC

s1=0 #number of shares owned by ITC

#calcuating total amount invested and number of shares owned in ITC

for i in range(len(ITC)):

    if ITC.loc[i,'Day']==30:

        sumTM+=ITC.loc[i,'Open']

        s1+=1

#displaying basic results

print("Total Invested in ITC = Rs",round(sumTM,2))

print("Shares Owned of ITC =",s1)

print("Average Investmentment of 1 share = Rs",round((sumTM/s1),2))



tm_end=298.2 #last open price of ITC on 2021-04-30


#calculating investment results

result1=round((tm_end*s1)-sumTM,2)

roiITC=round((result1/sumTM)*100,2)


#displaying investment results

print("Investment Result:")


if result1<0:

    print("Net Unrealised Loss = Rs",result1)
```

```
else:
    print("Net Unrealised Profit = Rs",result1)

print("ITC ROI from 2000-1-3 to 2021-04-30 =",roiTM,"%")

#TITAN ROI
sumTM=0 #total amount invested in TITAN
s2=0 #number of shares owned by TITAN
#calcuating total amount invested and number of shares owned in TITAN
for i in range(len(TITAN)):
    if TITAN.loc[i,'Day']==30:
        sumTM+=TITAN.loc[i,'Open']
        s2+=1
#displaying basic results
print("Total Invested in TITAN = Rs",round(sumTM,2))
print("Shares Owned of ITC =",s2)
print("Average Investmentment of 1 share = Rs",round((sumTM/s2),2))

tm_end=298.2 #last open price of TITAN on 2021-04-30

#calculating investment results
result2=round((tm_end*s2)-sumTM,2)
roiTITAN=round((result1/sumTM)*100,2)

#displaying investment results
print("Investment Result:")
```

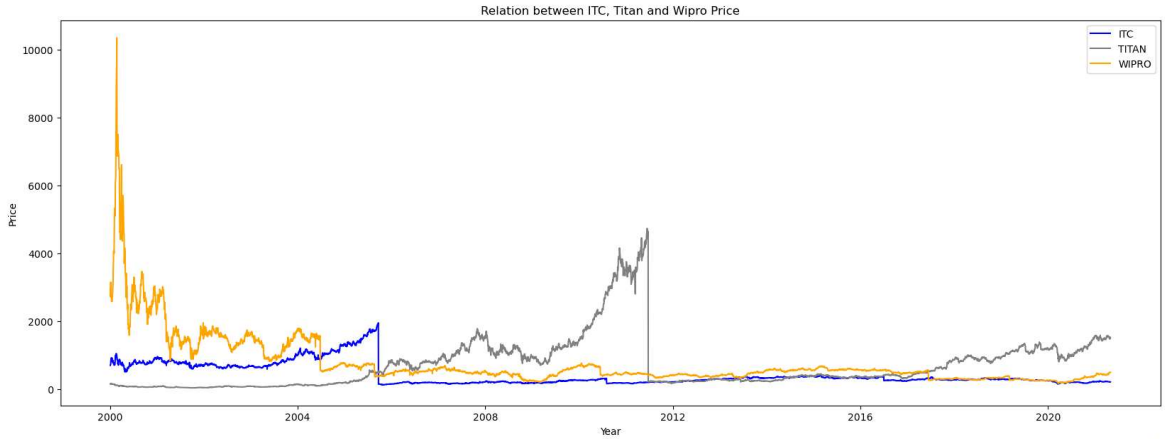
```
if result2<0:
    print("Net Unrealised Loss = Rs",result1)
else:
    print("Net Unrealised Profit = Rs",result1)

print("TITAN ROI from 2000-1-3 to 2021-04-30 =",roiTM,"%")

#WIPRO ROI
sumTM=0 #total amount invested in WIPRO
s3=0 #number of shares owned by WIPRO
#calcuating total amount invested and number of shares owned in WIPRO
for i in range(len(WIPRO)):
    if WIPRO.loc[i,'Day']==30:
        sumTM+=WIPRO.loc[i,'Open']
        s3+=1
#displaying basic results
print("Total Invested in WIPRO = Rs",round(sumTM,2))
print("Shares Owned of WIPRO =",s3)
print("Average Investmentment of 1 share = Rs",round((sumTM/s3),2))

tm_end=298.2 #last open price of WIPRO on 2021-04-30
#calculating investment results
result3=round((tm_end*s3)-sumTM,2)
roiWIPRO=round((result1/sumTM)*100,2)
```

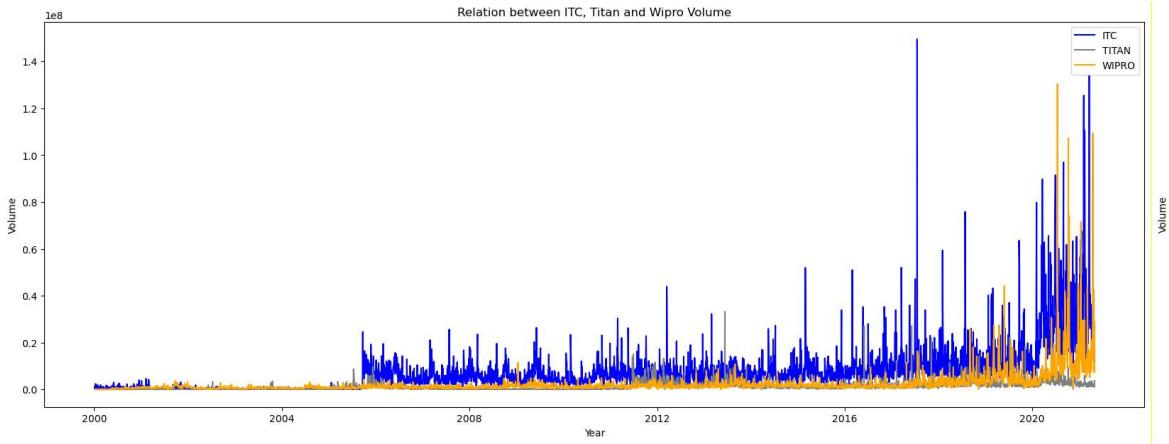
```
#displaying investment results

print("Investment Result:")

if result3<0:

    print("Net Unrealised Loss = Rs",result1)

else:

    print("Net Unrealised Profit = Rs",result1)


print("WIPRO ROI from 2000-1-3 to 2021-04-30 =",roiTM,"%")



#Plotting ROI on Bar Graph
plt.figure(figsize=(5,7))

stock=['ITC','TITAN','WIPRO']

ROI=[roiITC,roiTITAN,roiWIPRO]

col=['Blue','Grey','Orange']



plt.bar(stock,ROI,color=col)



plt.title("ROI")

plt.xlabel("Stocks")

plt.ylabel("Percentage")

#Plotting Profit/Loss Amount on Bar Graph

plt.figure(figsize=(5,7))
```

```
stock=['ITC','TITAN','WIPRO']

amt=[result1,result2,result3]

col=['Red','Green','Black']



plt.bar(stock,amt,color=col)



plt.title("Profit/Loss")

plt.xlabel("Stocks")

plt.ylabel("Amount")

#Displaying Number of shares owned.

plt.figure(figsize=(5,7))

stock=['Tata Motors','Tata Steel','TCS']

shares=[s1,s2,s3]

col=['Blue','Red','Orange']



plt.pie(shares,labels=stock,autopct="%1.2f%%",colors=col)

plt.legend(title="",loc="upper left")

plt.title("Portfolio Allocation")
```

Chapter 5

Implementation Result:

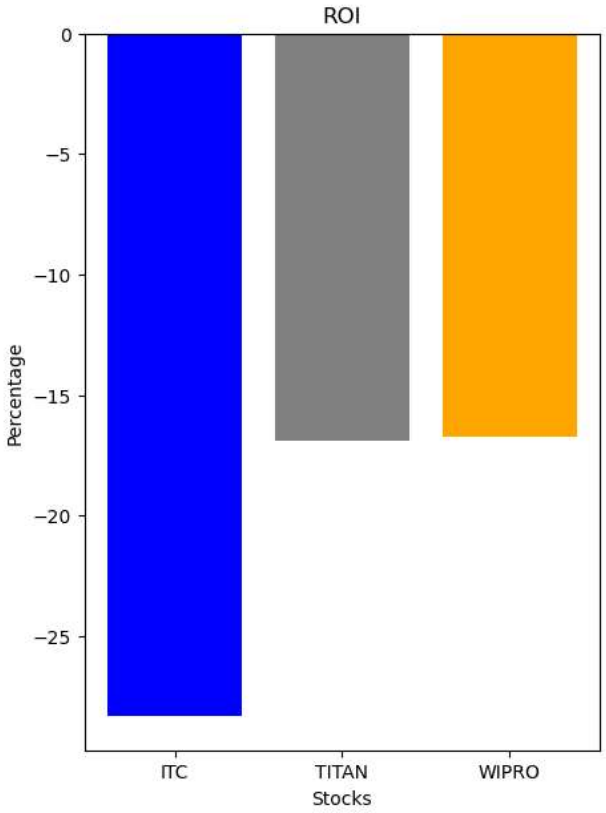


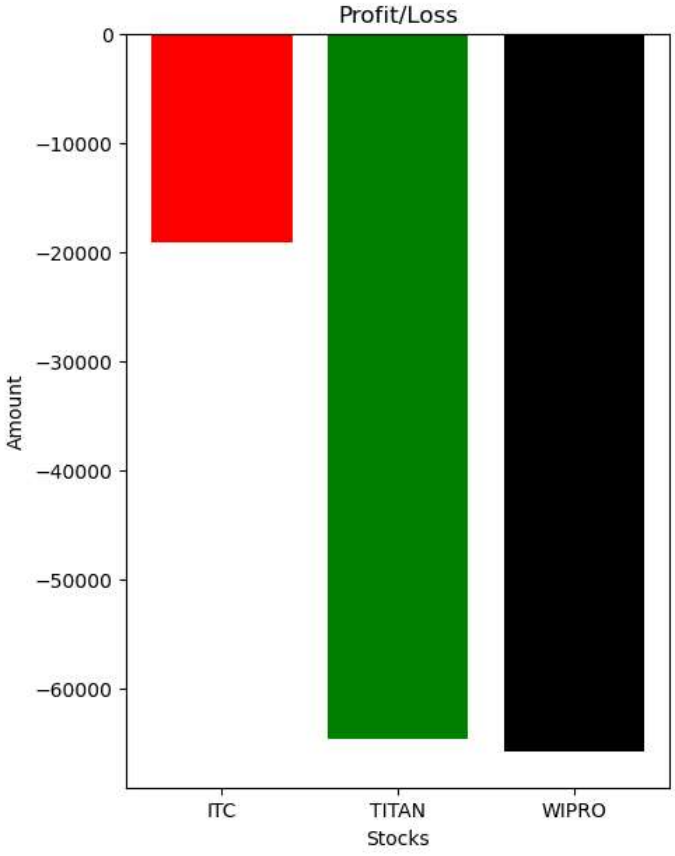


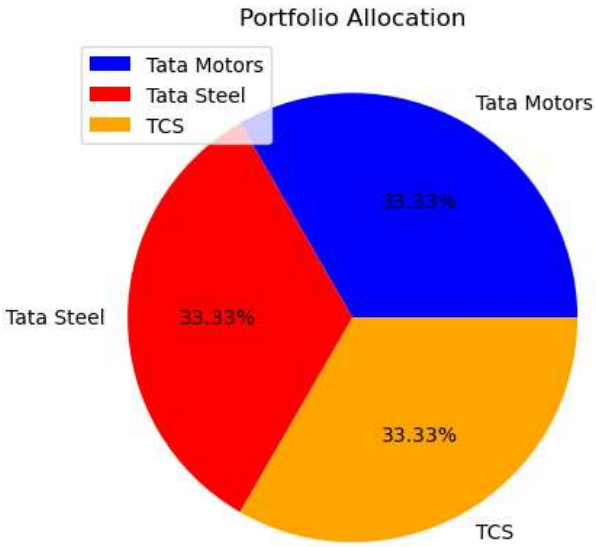
Total Invested in ITC = Rs 67403.4
Shares Owned of ITC = 162
Average Investmentment of 1 share = Rs 416.07
Investment Result:
Net Unrealised Loss = Rs -19095.0
ITC ROI from 2000-1-3 to 2021-04-30 = -57.66

Total Invested in TITAN = Rs 112904.9
Shares Owned of ITC = 162
Average Investmentment of 1 share = Rs 696.94
Investment Result:
Net Unrealised Loss = Rs -19095.0
TITAN ROI from 2000-1-3 to 2021-04-30 = -57.66

Total Invested in WIPRO = Rs 114104.7
Shares Owned of WIPRO = 162
Average Investmentment of 1 share = Rs 704.35
Investment Result:
Net Unrealised Loss = Rs -19095.0
WIPRO ROI from 2000-1-3 to 2021-04-30 = -57.66 %


ROI
0
−5
−10
−15
−20
−25
Percentage
ITC
TITAN
WIPRO
Stocks


Profit/Loss
0
−10000
−20000
−30000
−40000
−50000
−60000
Amount
ITC
TITAN
WIPRO
Stocks


Portfolio Allocation
Tata Motors
Tata Steel
TCS
Tata Motors
33.33%
Tata Steel
33.33%
33.33%
TCS

Chapter 6

Conclusion:

The project involved analyzing the stock market data for three companies: ITC, TITAN, and WIPRO. Various tasks were performed, including data loading, data exploration, data manipulation, visualization, and financial analysis.

Data Loading and Exploration:

The stock market data for each company was loaded into pandas DataFrames from CSV files.
Basic exploratory data analysis was conducted to understand the structure and contents of the datasets. This included checking the size, data types, presence of null values, and descriptive statistics of the data.
Data Manipulation:

Necessary data manipulations were performed, such as converting the date column to datetime format and adding new columns for year, month, and day to facilitate further analysis.
Financial calculations were carried out to determine the total investment, number of shares owned, and ROI (Return on Investment) for each company.
Visualization:

Data visualization techniques, including line plots and bar graphs, were used to visualize the stock prices, trading volumes, ROI, and profit/loss amounts.
The visualizations provided insights into the trends, patterns, and performance of the companies over time.
Financial Analysis:

Financial analysis was conducted to evaluate the investment results and ROI for each company.
The analysis helped in assessing the profitability and performance of the investments made in the respective stocks.
Portfolio Allocation:

The number of shares owned for each company was visualized using a pie chart, providing a clear representation of the portfolio allocation.

REFRENCES:

https://www.kaggle.com/datasets/rohanrao/nifty50-stock-market-data