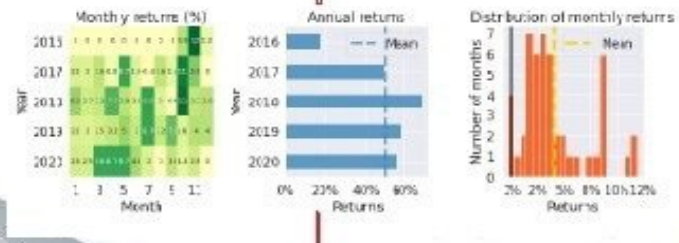
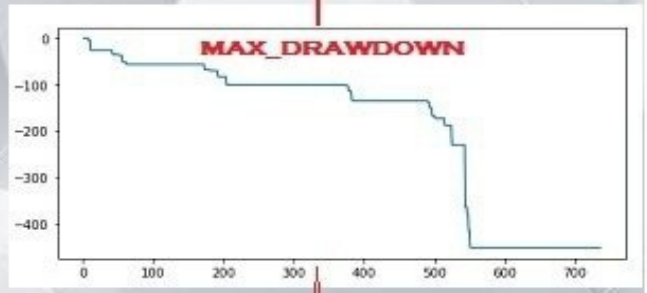
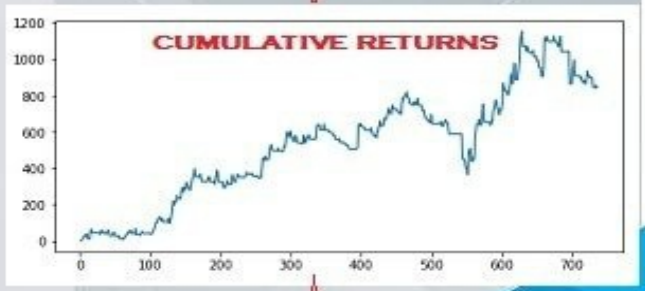
TRADING FOR PROFITS



PYTHON FOR TRADING ON TECHNICALS





A STEP TOWARDS SYSTEMATIC TRADING.

BUILD YOUR OWN TRADING STRATEGIES.

ANJANA GUPTA

STRATEGY DEVELOPMENT

PYTHON FOR TRADING ON TECHNICAL

A step towards systematic trading.

Anjana Gupta

CONTENTS

Disclaimer

This book is educational in nature. Various Derivative contracts traded in Indian market and data used in various examples are for illustrative and educational purpose only. Example/ data used may or may not be based on Historical/factual data. We are not rendering legal/professional services or any kind of advice for trading. Past performance or data or example discussed here does not guarantee of future results. We disclaim any liability/loss/risk resulting directly or indirectly from use or application of any content of this book.

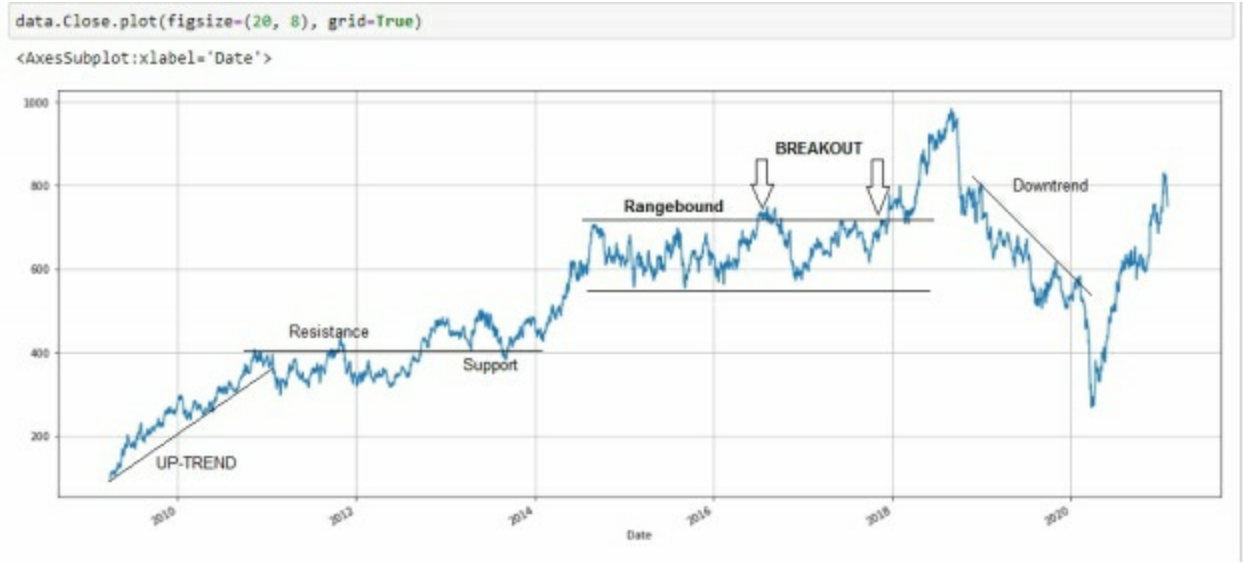
PYTHON FOR TRADING ON TECHNICAL-

Technical analysis today has its origins in the theories first proposed by Charles Dow in eighteenth century. Dow published his ideas in series of editorials he wrote for wall street Journal. Dow Theory is all about trends. A trend is really nothing more than a somewhat uniform change in price levels over time. For an Uptrend prices start low and through a series of advances and pullbacks move to a higher level. For a down trend prices start high and through a series of advances and pullbacks move to a lower level.

Dow Theory says that the market has three trends. Index/Stock/Commodity is in upper trend when they made higher top and higher lows. It means lows will be higher than previous low and high will be higher than previous high. Index/Stock/Commodity is in lower trend when they made lower top and lower lows as explained in following picture. Third is range bound when stock is neither in up-trend nor in down-trend.

Dow Theory also says that each trend has three phases. Accumulation phase when informed investor buy the stock, public participation phase when prices begin to advance rapidly and distribution phase when informed investor who accumulated stocks on lower prices begin to distribute before anyone else start selling. Dow Theory also says that volume/open interest must confirm the trend. Increase in volume/ open interest with rise in prices will confirm the uptrend and increase in volume / open interest with drop in prices will confirm the downtrend.

So Technical analysis is all about trend – buying stock when uptrend begins, ride the trend, and sell when trend ends a high price. A trend follower buys when the price trend is up and sells when the price trend is down. He believes that, if the trend is up then it will continue to go up. If everything works as expected, the trend follower makes money. The trend is always a matter of time interval. In a long run a stock may be in up-trend but in short run the same may be in down trend.



I have taken Mahindra & Mahindra (M&M) prices from year April 2009 to January 2021 in the above chart to explain technical terminology. As we can see in above chart stock was in uptrend from 2009 to 2011 when prices goes up by 500% from 80 to 400. I have drawn a trend line (trend line is simple a straight line that connects a series of security prices either tops or bottoms). Price of 400 was working as resistance in year 2011 and 2012 and the same

was working as support in 2013 and 2014. (Support is the price level at which there is adequate demand for a security to stop its downward price movement and Resistance is a price level at which there is significant supply of stock to stop its upward price movement). Usually a resistance work as support when prices break resistance as we have seen above. From year 2015 to 2018 this stock was range bound. We have seen first false breakout in prices in year 2017 when stock beaked all time high. Again we have seen a successful breakout in 2018 and prices goes up from 750 to 1000. In the year 2019 and 2020 the stock was in downtrend. Again a new uptrend started in January 2021. Now we have to see that prices will remain range bound or this uptrend will continue.

Hundreds of technical indicators developed for identification of trend. We can divide these indicators in two categories –

1. Lagging indicators – These indicators gives buy/sell signal after price moves.
2. Leading indicators – These indicators give signals before price change.

Chart patterns help us in identification of trend as we have learned in above example. Technicians also have developed many chart patterns for identification of reversal of trend. Some of these discussed below-

Head and Shoulder pattern –

When a price are in uptrend and fails to create a higher high than prices create head and shoulder pattern as shown in following chart. This is an indication of reversal of uptrend and beginning of down trend when neckline is broken as shown in following chart –

The same pattern we can observe in prices chart of many stocks when there is reversal in down trend and breakout in the prices above neckline.

Round Top and Round bottoms –

Round top and round bottoms trend reversal pattern is illustrated in the following charts when prices are gradually coming down or going up against the previous trend.

Double top and Double bottom-

As shown in the following example when price in trend fails to create higher highs in case of uptrend or lower lows in case of down trend they reverse from the previous high/low.

Gaps-

In a uptrend when current period low price is higher than previous period high price (Gap up opening) indicates continuation a uptrend and in case of down trend when a day high price are lower than previous day low price (Gap down opening) is an indication of further down trend. Some technical analyst believes that price comes back once to fill the gap.

Identifying and benefitting from chart patterns constitutes a major part of technical analysis. Different traders use different charts in order to achieve better results in the market. Acquainting yourself with these chart patterns can help you identify better trading opportunities more quickly.

In this book we will work on systematic trading. We will try to develop technical based trading systems that are consistently profitable in various market conditions. When we talk about systematic trading it means all decisions of buy and sell will be taken by system not by intuition of individual. System will generate buy and sell signal. We will also check that these systems are profitable on past data or not. So we are trying to develop

systems for trading. Trading is all about making money. Algorithmic trading is all about developing quantitative rules that give you profit. In this book we are going to take some simple ideas and turn them into successful trading strategies. If you can't turn an idea into profitable returns, then you are wasting your time. Trading is really all about making money. With the help of this book you will be able to develop, modify and backtest your own idea on past data to develop your own profitable strategy. You ideas should be based on common sense and than pyhton can help you to get your ideas tested using historical data to be sure they actually work.

Algo trading is quantified rules to apply on trading. A very simple example is a stock trading above 20 days moving average bought and below 20 days moving average sold. Any strategies that can be quantified can be traded through algos. Benefit of quantified strategy is you can back test on past data. You can check the outcomes on historical data and this give you conviction to trade inspite of draw downs of any strategy. Any strategy that cannot be quantified can't be traded through algos.

This book is divided in 5 parts –

1. First we will learn basics of Python. Python has emerged as one of the most popular languages for programmers in financial trading, due to its ease of availability, user-friendliness, and the presence of sufficient scientific libraries. Python serves as an excellent choice for automated trading. That's why I believe that every trader must have basic knowledge of Python. It's available absolutely free of cost for individuals.
2. Second we will learn how download past data. We need past data for back-testing of our strategies. We will learn to fetch EOD, One minute and real-time data free of cost available through various sources.
3. We will learn to develop python codes for backtesting the performance of Technical based indicators on past data. We will be able to compute return, number of trades generated by any strategy, maximum drawdown, maximum return in a single trade,

maximum loss in a single trade etc on past data.

4. We will learn to install technical analysis library (TA-LIB) for backtesting. Today this library has become one of the most famous libraries for technical analysis of stocks and other financial securities. Ta-lib includes more than 150 indicators such as ADX, MACD, RSI and Bollinger Bands and candlestick pattern recognition etc.
5. At last we will learn to develop a Technical based trading system using combination of various technical indicators.

BASICS OF PYTHON

Whatever a trader does must be based on past data analysis and back tested results. For past data analysis you can download data in excel from exchange websites, but download and analysis of data thru excel could be very time consuming and tedious task. Here Python can help you a lot. What python does it in seconds the same on excel could take days. Most of the machine learning packages are into python, so when you will go to next level of trading / or in automated trading this basic knowledge discussed in this book will help. This book is an attempt to explain the functions of Python which are useful for traders.

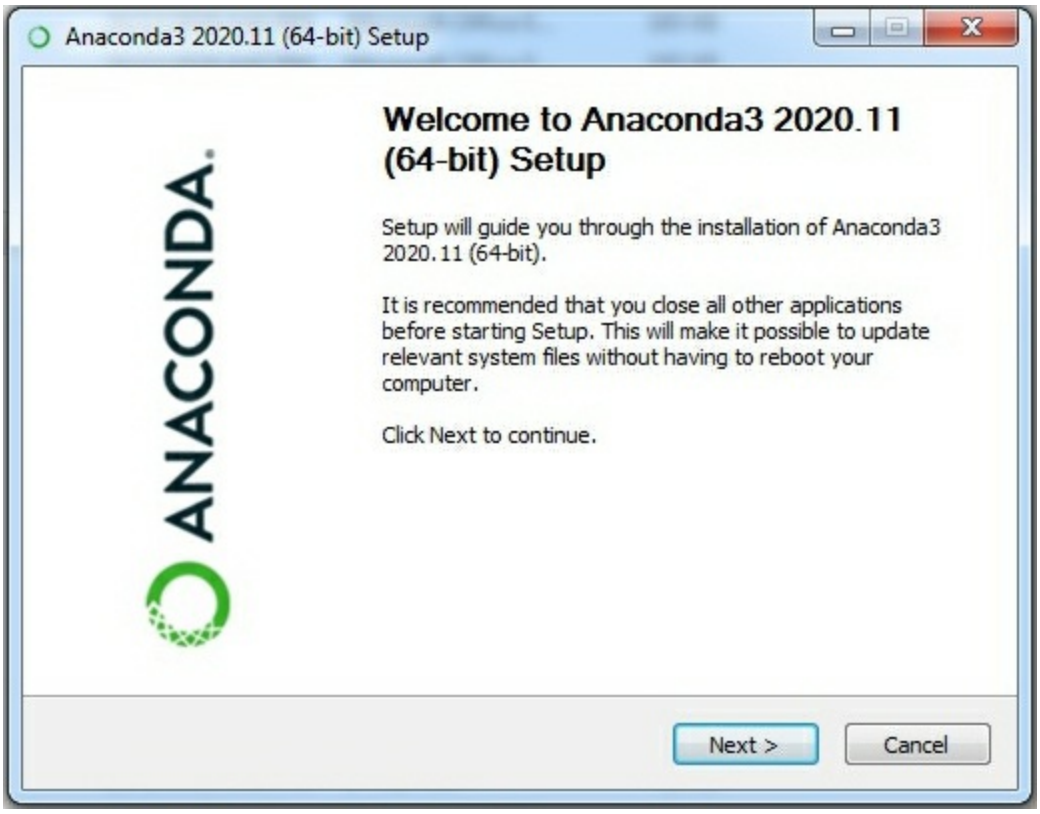
Installation of Python Jupyter Notebook on Laptop

First get python installed on your system/laptop. You can download anaconda installer from following link.

https://www.anaconda.com/distribution/

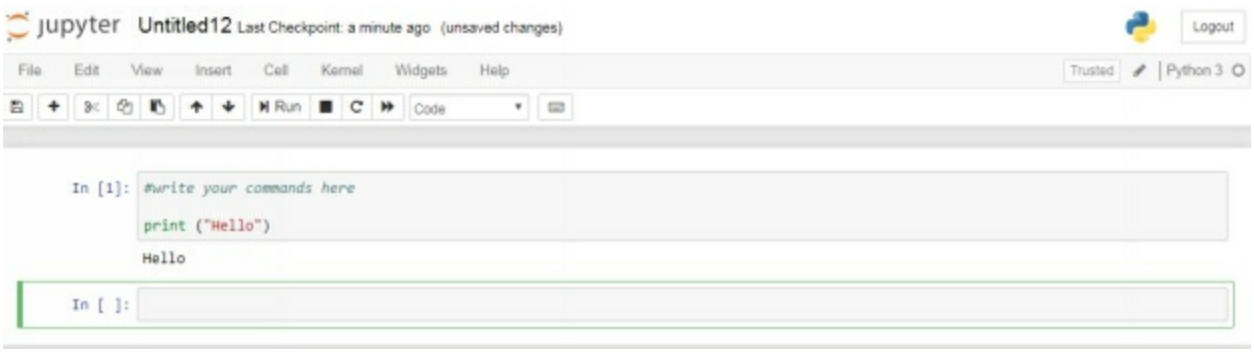
Go to the bottom of the page and dwonload installer as per you operating system -

Run the installer –



Installation of anaconda is like installation of any other program on your system. After installation Go to all program menu of your system/laptop, when you will search all programs you will find one folder with the name of Anaconda. Click on Anaconda, there will be many options, click on Jupyter Notebook. Jupyter notebook will look like this –

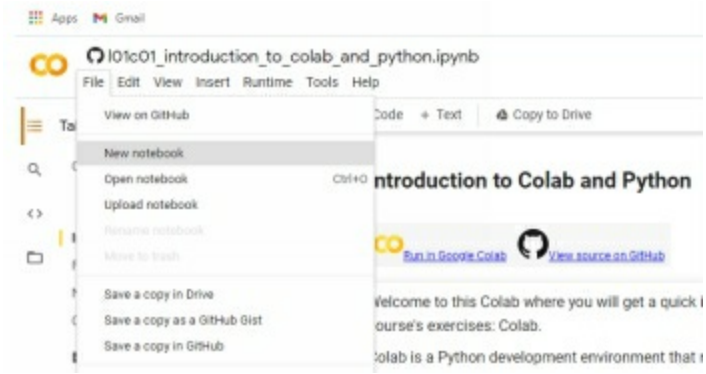
Click on new > Python 3 than a new notebook will open, will look line this –



I have given print command in 1st cell. You can write and execute commands in second highlighted green cell. *Get familiar with the Jupyter Notebook interface.* Now Jupyter Notebook is your playground. Jupyter Notebook is an interactive document. You need to write commands in predefined format and on execution of commands you get results of your commands. It's easy; you can learn basics of python thru any video available on Google. Good thing with python is that past data of exchanges and tools are already available. Most of the professional traders use python or c++ for development and back-testing of strategies. You need not to learn complete programming language. You can analyze data in the way you want for that you need to learn some basics like how to write program and how this software will help in past data analysis for any strategy.

Google Colab (Python Jupyter note book)-

You can run python codes on Google Colab also. Colab is a Python development environment that runs in the browser using Google Cloud. One can use Google colab to run python codes or you may download anaconda on your laptop and you can use Jupyter notebook to run codes. One can open Google colab in any browser. Open Google Colab, click on file and open new notebook.



Let's start with some basic of Python than we will learn how to fetch past data in python and back testing of some strategies on past data.

First you need to understand that you need to give commands in predefined format, it is called coding. Programming languages understand commands that are also in predefined format. Initially you will make lots of mistake in coding and gradually you will learn how to write program code.

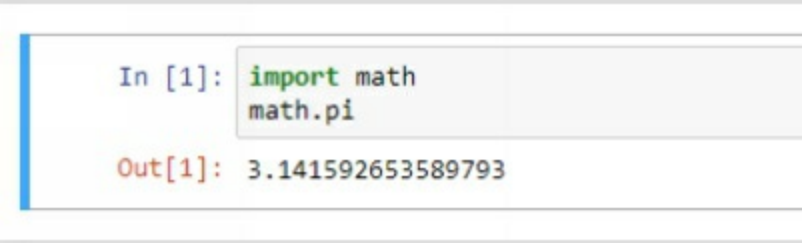
Open a new Jupyter note book. Type following in a cell (code written in blue color is python code. Statement written after '#' is explanation in English, it's not a part of code, it is just to explain you the code) –

```
a=100          # here you assigned value of 100 to a variable 'a'
b=a/2          # here you assigned value of 50 ('a' divide by 2) to variable 'b'
c=a+b          # here you assigned value of 150 (value of a + value of b) to variable 'c'
d=(a*3)-b      # here you assigned value of 250 [(a X 3)-b] to variable 'd'
print (a,b,c,d)    # here you are giving print command to print values of 'a', 'b', 'c' and 'd'
```

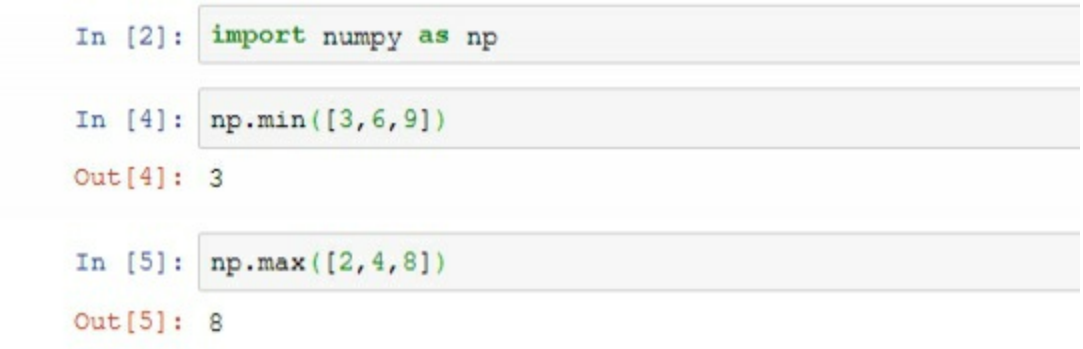
Than you can click on 'run' button or press 'Shift' and 'Enter' buttons on your laptop. You will get the results of your command. Jupyter notebook will print the value of 'a', 'b', 'c' and 'd' as per your command. Python makes use of simple syntax which looks like written English.

Be creative, do various experiment with codes in notebook. This will help you to learn more about programming.

Second you need to understand modules in python. If you want to use mathematical functions in Jupyter sheet than you need to import math module.



For option trader one of the basic module required is NumPy. It is used for Scientific Computing. In cell 2 I have imported module numpy. In cell 4 I have given command to find minimum value out of 3, 6, and 9. We are using Np.min() command from numpy module. You will check out of command is 3. So program itself found he minimum value out of 3 values. In 5th cell I have given command to find maximum value. Result of command 5 is 8. So we are giving commands in predefined format and software is giving output of that command.



Math, NumPy are the Built-in modules, these modules are available by default in Python. But there are many other publicly available module which you may want to use for example backtrader. Backtrader is used for back testing. To install a module following command is used in python –

!pip install <module name>

Third and the most important thing a trader need is past data. Various options are available to get past data. One can download data from yahoo finance.

Exchange wise modules are also available like **nsepy module** is used to get the past data for contracts traded on National Stock Exchange, India. You can install these modules thru following commands –

!pip install yfinance

!pip install nsepy

Forth you need to understand data in tabular format in python. Tabular format comprising of row and columns like Excel spread sheet, in python this is called ‘dataframe’. You can perform anything in a particular row and column thru commands in python. Trader can fetch past data through many sources like yfinance in tabular format, we will learn this in next chapter. You can also import and export csv/excel files in python. You can perform various functions on a table in python like you do on ‘Excel’. Some basic functions required for past data analysis and strategy backtesting are discussed below -

Following command will be used for import of data of a file saved on your laptop –

Table = pd.read_csv('filename.csv')

Following command will be used to save data in a file on your laptop (export data) –

filename.to_csv("giveanyname.csv", index=True, encoding='utf8')

Codes to create table (DataFrame) in Python itself by typing values-

import pandas as pd

data = {'Stocks': ['Reliance', 'Infosys', 'TCS'],

'Price': [1200, 700, 2500]}

Table = pd.DataFrame(data, columns = ['Stocks', 'Price'])

print (Table)

```
In [1]: import pandas as pd

        data = {'Stocks': ['Reliance','Infosys','TCS'],
                'Price': [1200,700,2500]}

        Table = pd.DataFrame(data, columns = ['Stocks', 'Price'])

        print (Table)

          Stocks  Price
        0  Reliance   1200
        1   Infosys    700
        2       TCS   2500

In [ ]:
```

Add new column in Table with following command –
Table['Quantity'] =0

```
In [2]: Table['Quantity'] =0
        print (Table)

          Stocks  Price  Quantity
        0  Reliance   1200         0
        1   Infosys    700         0
        2       TCS   2500         0
```

Check columns with following command –
Table.columns

```
# Inspect the columns
Table.columns

Index(['Stocks', 'Price', 'Quantity'], dtype='object')
```

Select last 2 values of column ‘Price’ in new variable ‘last’ with following command –

last = Table['Price'][-2:]

```
# Select only the last 2 observations of `Price`
last = Table['Price'][-2:]
print (last)
```

```
1     700
2    2500
Name: Price, dtype: int64
```

Select maximum and minimum value in a table from following commands –
Table.column.max()
Table.column.min()

```
Table.Price.min()
```

```
700
```

```
Table.Price.max()
```

```
2500
```

You can insert value in any cell thru following command –
Table.loc[0, 'Quantity'] = 10 #this command will insert value '10' in 1st row of column 'Quantity'

```
Table.loc[0, 'Quantity'] = 10
Table.loc[1, 'Quantity'] = 20
Table.loc[2, 'Quantity'] = 30

print (Table)
```

```
     Stocks  Price  Quantity
0  Reliance   1200        10
1   Infosys    700        20
2       TCS   2500        30
```

You can also perform various mathematical/statistical functions between columns –
Table['Amount']=Table['Price']*Table['Quantity']

```
Table['Amount']=Table['Price']*Table['Quantity']
print (Table)
```

```
     Stocks  Price  Quantity  Amount
0  Reliance   1200        10   12000
1   Infosys    700        20   14000
2       TCS   2500        30   75000
```

Following command can be used for cumulative sum of any column –
Table['Total'] = Table['Amount'].cumsum()

```
Table['Total'] = Table['Amount'].cumsum()
print (Table)
```

```
     Stocks  Price  Quantity  Amount   Total
0  Reliance   1200        10   12000   12000
1   Infosys    700        20   14000   26000
2       TCS   2500        30   75000  101000
```

Following command could be used to delete columns –
Table.drop(["Total", "Amount"], axis = 1, inplace = True)

```
Table.drop(["Total", "Amount"], axis = 1, inplace = True)
print (Table)
```

```
     Stocks  Price  Quantity
0  Reliance   1200        10
1   Infosys    700        20
2       TCS   2500        30
```

Now let's learn some other functions of Python that we are going to use in coming chapters –

Object Oriented Programming (OOP) - The concept of OOP in Python focuses on creating reusable code. You can create your own function in Python. In the following example you have created a new function 'my_function' which multiply 'Price' and 'Quantity' of any table. So you can perform a single command or set of commands in your function and you need not to write these codes again and again. That is why it is called object oriented programming.

```
def my_function(df):
```

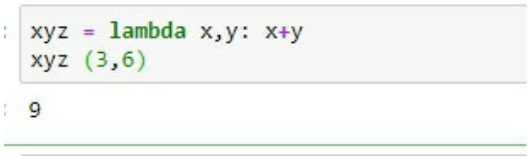
```
    try:
        return (df['Price']*df['Quantity'])
    except:
        return np.NaN

Table['Amount'] = Table.apply(my_function, axis=1)
print (Table)
```

```
def my_function(df):
    try:
        return (df['Price']*df['Quantity'])
    except:
        return np.NaN


Table['Amount'] = Table.apply(my_function, axis=1)
print (Table)
```

```
     Stocks  Price  Quantity  Amount
0  Reliance   1200        10   12000
1   Infosys    700        20   14000
2       TCS   2500        30   75000
```

Lambda - you can create your own function through lambda. Here x and y are the parameters and x+y is the operation performed. So when you will use xyz in your code you will get addition of 2 values as result.

```
xyz = lambda x,y: x+y
xyz (3,6)
```



For –

For loop is used to get numbers/values in a sequence. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation. Following example will explain the use of for loop.

```
val = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
for x in val:
```

print (x)

You will get the following output –

```
val = ['a','b','c','d','e']
for x in val:
    print (x)
```

```
a
b
c
d
e
```

Range - To loop through a set of code a specified number of times we can can use a 'range' function. Let me explain how these loops works. Jupyter note book will print the value of 'x' 4 times if you will write following code -

```
x=1
for x in range (1, 5):
        print (x)
```

```
x=1
for x in range (1, 5):
    print (x)
```

```
1
2
3
4
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter:

```
for x in range (1, 15, 2):
        print (x)
```

```
In [6]: for x in range (1, 15, 2):
            print (x)

1
3
5
7
9
11
13

In [ ]:
```

As you can observe from above command value of x printed with incremental value of 2.

If Statement

The If: elif: else: statement is used in Python for decision making. Program codes evaluate the test expression and will execute statement only if the test expression is true. If test expression is false the statement is not executed. Python uses the colon (:) and indentation/whitespace to group statements.

Use following python codes with different numbers to understand 'If Statement'

```
num = float(input("Enter a number: "))
if num < 0:
    print("Negative number")
elif num == 0:
    print("Zero")
else:
    print("Positive number")
```

```
num = float(input("Enter a number: "))
if num < 0:
    print("Negative number")
elif num == 0:
    print("Zero")
else:
    print("Positive number")
```

```
Enter a number: 0
Zero
```

Break and continue

In Python, break and continue statements can alter the flow of a normal loop.

for x in range (1, 15, 2):
**   if x == 7: break**
**   print (x)**

```
for x in range (1, 15, 2):
    if x == 7: break
    print (x)
```

```
1
3
5
```

for x in range (1, 15, 2):
**   if x < 7: continue**
**   print (x)**

```
for x in range (1, 15, 2):
    if x < 7: continue
    print (x)
```

```
7
9
11
13
```

Indentation

A logical block of statements such as the ones that make up a function should all have the same indentation, if one of the lines in a group has a different indentation; it is flagged as a syntax error.

Code checking on compilation

Python check codes on run time only. Any error will not come to your notice until that line runs.

Help for Python

The help() & dir() function pulls up documentation strings for various modules, functions, and methods.

```
In [1]: help (len)

        Help on built-in function len in module builtins:

        len(obj, /)
            Return the number of items in a container.
```

```
In [2]: dir (len)

Out[2]: ['__call__',
         '__class__',
         '__delattr__',
         '__dir__',
```

Python datetime

Python has a module named **datetime** to work with dates and times. You can fetch today's date with following commands –

Import datetime as date

datetime.date.today()

```
import datetime as date
```

```
print (datetime.date.today())
```

```
2021-01-04
```

```
print (datetime.datetime.now())
```

```
2021-01-04 11:00:00.925773
```

Following command can be used to get date in datetime format.
a_date = datetime.date(2021, 1, 3)

You can fetch month (%m), date (%d) or year (%y) with strftime command –
m_date = a_date.strftime("%m")

```
a_date = datetime.date(2021, 1, 3)
print (a_date)
```

```
2021-01-03
```

```
m_date = a_date.strftime("%m")
print (m_date)
```

```
01
```

Following command can be used to increase or reduce number of days from date.
+-datetime.timedelta(days=1)

To get 4 days prior date from the date store in variable ‘a_date’ following command will be used –
p_date = a_date - datetime.timedelta(days=4)

To get the 4 days after todays date the following command will be used –
n_date = datetime.date.today() + datetime.timedelta(days=4)

```
p_date = a_date - datetime.timedelta(days=4)
print (p_date)
```

```
2020-12-30
```

```
n_date = datetime.date.today() + datetime.timedelta(days=4)
print (n_date)
```

```
2021-01-08
```

The Python Tutorial

I explained the required python basics for a trader. Trader objective is to make money not to be an expert python programmer. Basic Python explained, rest you will learn with the codes written for fetching historical data and backtesting of strategies on past data in this book. If you want to learn more about basic please refer tutorials available on Python website python.org, link given below-

https://docs.python.org/3/tutorial/index.html

Free resources

If you do not understand programming then I will suggest you to learn some basics of python first. You can enroll for free basic course on Python through following link -

https://quantra.quantinsti.com/course/python-trading-basic

FETCHING HISTORICAL DATA

Data is one of the most important resources of the 21st century. Historical market data is essential for strategy development and backtesting. Professional data vendors sometimes are not an economically viable option for retail investors. Fortunately free market data sources available. I will discuss few of them in this chapter.

Daily Historical Data through yfinance

Historical data can be fetched from yfinance. First you need to install required libraries. Following codes could be used to fetch past data.

```
!pip install yfinance
from datetime import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import yfinance as yf
data = yf.download('HDFC.NS', start="2009-01-01", end="2019-01-31")
```

Above commands will fetch 3 years historical data of scrip HDFC trading on NSE in variable data. You can check details of data store in variable ‘data’ with print command.

print (data)

You will get the following output –

```
print (data)
```

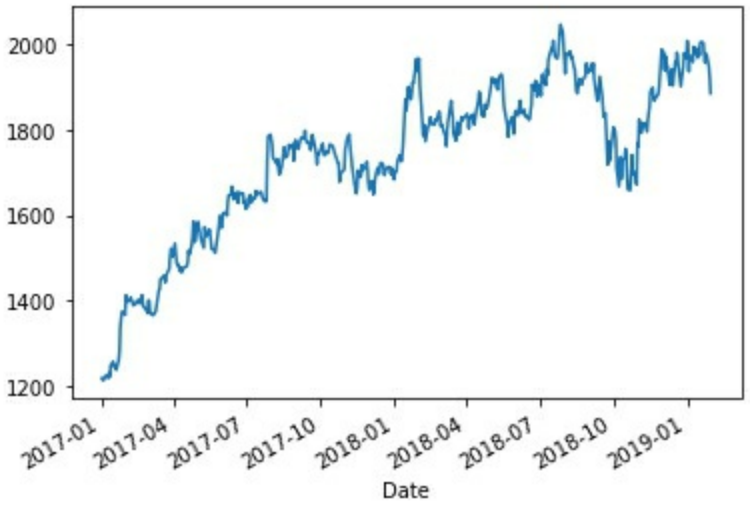
Date	Open	High	Low	Close	Adj Close
2017-01-02	1272.000000	1272.000000	1213.699951	1217.099976	1145.718750
2017-01-03	1218.699951	1222.599976	1200.500000	1214.000000	1142.800537
2017-01-04	1215.800049	1224.500000	1197.250000	1212.099976	1141.011963
2017-01-05	1215.699951	1220.400024	1205.099976	1217.199951	1145.812866
2017-01-06	1219.000000	1232.000000	1218.800049	1222.349976	1150.660767
...	...	...	...	...	...
2019-01-24	1960.000000	1976.000000	1942.150024	1971.099976	1895.751465
2019-01-25	1964.400024	1993.000000	1963.949951	1977.800049	1902.195435
2019-01-28	1978.699951	1982.199951	1938.099976	1946.900024	1872.476685
2019-01-29	1943.699951	1949.900024	1893.300049	1922.300049	1848.817017
2019-01-30	1924.599976	1929.300049	1869.500000	1885.800049	1813.712158

You can check the last 3 year price movement through charts also with the help of following commands-

Data.Close.plot()

```
: data.Close.plot()
```

```
: <AxesSubplot:xlabel='Date'>
```



Following command can be used to download and save data in csv file /Excel on your laptop.

data.to_csv("HDFC.csv", index=True, encoding='utf8')

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	02/01/2017	1272	1272	1213.7	1217.1	1145.719	2369360
3	03/01/2017	1218.7	1222.6	1200.5	1214	1142.801	3203369
4	04/01/2017	1215.8	1224.5	1197.25	1212.1	1141.012	2805972
5	05/01/2017	1215.7	1220.4	1205.1	1217.2	1145.813	3339784
6	06/01/2017	1219	1232	1218.8	1222.35	1150.661	2774461
7	09/01/2017	1225	1229	1212.6	1223.55	1151.79	2016528
8	10/01/2017	1225	1227.95	1213.05	1217.25	1145.86	2669824
9	11/01/2017	1224.7	1233	1218.2	1230.05	1157.909	1353730
10	12/01/2017	1234	1242.95	1217.6	1221.1	1149.484	1477834
11	13/01/2017	1224.95	1250.6	1222.15	1247.5	1174.335	2131709
12	16/01/2017	1245.1	1264	1240.25	1256.9	1183.184	1694061
13	17/01/2017	1258.35	1261	1234.95	1245.8	1172.735	1185487
14	18/01/2017	1246.3	1260.95	1245.65	1249.05	1175.795	1491317
15	19/01/2017	1251.7	1251.7	1232.9	1242.4	1169.535	1599732
16	20/01/2017	1236.15	1245	1230.6	1237.3	1164.734	1408933
17	23/01/2017	1234.7	1262	1232	1259.75	1185.867	2265524

In the same way you can fetch historical data of other exchanges and other scrip.

Per Minutes historical data through yfinance-

You can also download 1 minute historical data of last 10 days from yfinance. Following codes will be used to get 1 minute data –

```
minu_data = yf.download('HDFC.NS', start="2021-01-01", end="2021-01-02", interval="1m")
```

Following is the screen shot –

```
minu_data = yf.download('HDFC.NS', start="2021-01-01", end="2021-01-02", interval="1m")
```

```
[**********100%**********]  1 of 1 completed
```

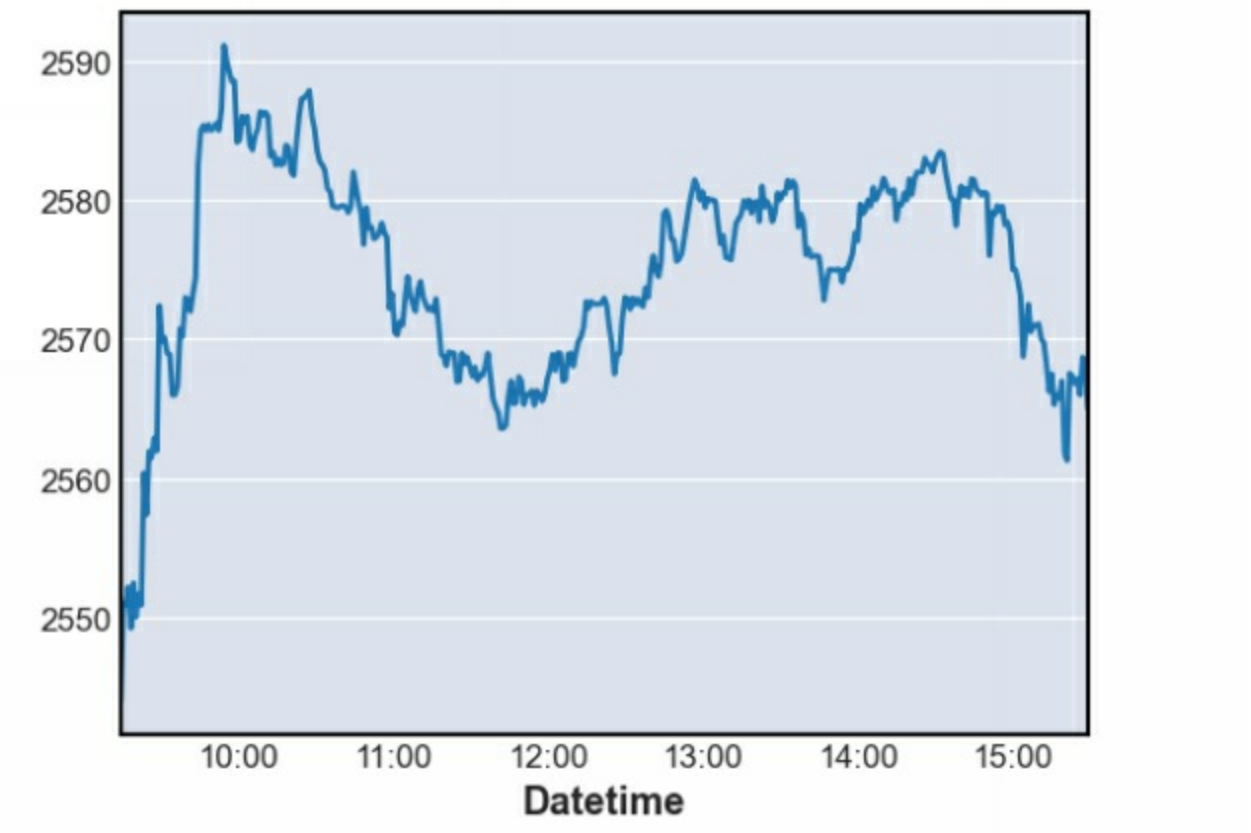
```
print (minu_data)
```

```
                                 Open         High          Low        Close  \
Datetime
2021-01-01 09:15:00+05:30  2553.649902  2554.500000  2542.100098  2544.000000
2021-01-01 09:16:00+05:30  2545.000000  2553.300049  2543.949951  2551.000000
2021-01-01 09:17:00+05:30  2551.100098  2553.000000  2549.449951  2550.899902
2021-01-01 09:18:00+05:30  2550.800049  2553.000000  2549.899902  2552.199951
2021-01-01 09:19:00+05:30  2552.050049  2552.649902  2548.000000  2549.250000
...                                ...          ...          ...          ...
2021-01-01 15:25:00+05:30  2567.250000  2568.600098  2565.550049  2567.149902
2021-01-01 15:26:00+05:30  2567.149902  2567.550049  2565.600098  2566.000000
2021-01-01 15:27:00+05:30  2566.000000  2569.000000  2565.949951  2568.699951
2021-01-01 15:28:00+05:30  2568.500000  2569.000000  2567.050049  2568.500000
2021-01-01 15:29:00+05:30  2568.000000  2568.500000  2561.350098  2565.000000
```

Intraday chart can also be plotted to check price movement –

```
minu_data.Close.plot()
```

```
<AxesSubplot:xlabel='Datetime'>
```



Live Data through yfinance

You need to install package yahoo_fin to get Live stock quotes using web scraping tools. The following code gets the real-time stock price during market hours.

```
!pip install requests_html
!pip install yahoo_fin
from yahoo_fin import stock_info
from datetime import time
now = datetime.now()
price = stock_info.get_live_price("HDFC.NS")
print(now, 'HDFC Price:', price)
```

```
2021-01-04 13:24:12.268340 HDFC.NS: 2560.39990234375
```

Historical Data through QUANDL

Quandl is a platform that provides its users with economic, financial and alternative datasets. According to Quandl, their user amount is over 400,000 people, which ranges from the world's top hedge funds to investment banks and various asset managers. Quandl want to inspire customers to make new discoveries and incorporate them into trading strategies. They believe there are new and better ways to understand the complex information that creates markets and market movement. They believe that data, and alternative data in particular, is going to become the primary driver of active investment performance over the next decade.

Quandl offers both free and premium products. The Quandl API is free to use

and grants access to all free datasets. Quandl users only pay to access Quandl's premium data products. In this book we will use free data available on quandl.

Quandl is really easy to use and is beginner-friendly. All you need to do is to go to their website (https://www.quandl.com/) and register for an account. After that you'll get your API key that allows you to fetch data from quandl.

The best thing with Quandl is that data is free, transparent, easy to find and cleaned.

First you need to create account with Quandl and you will get API Key. After that you need to install Quandl library with following command –

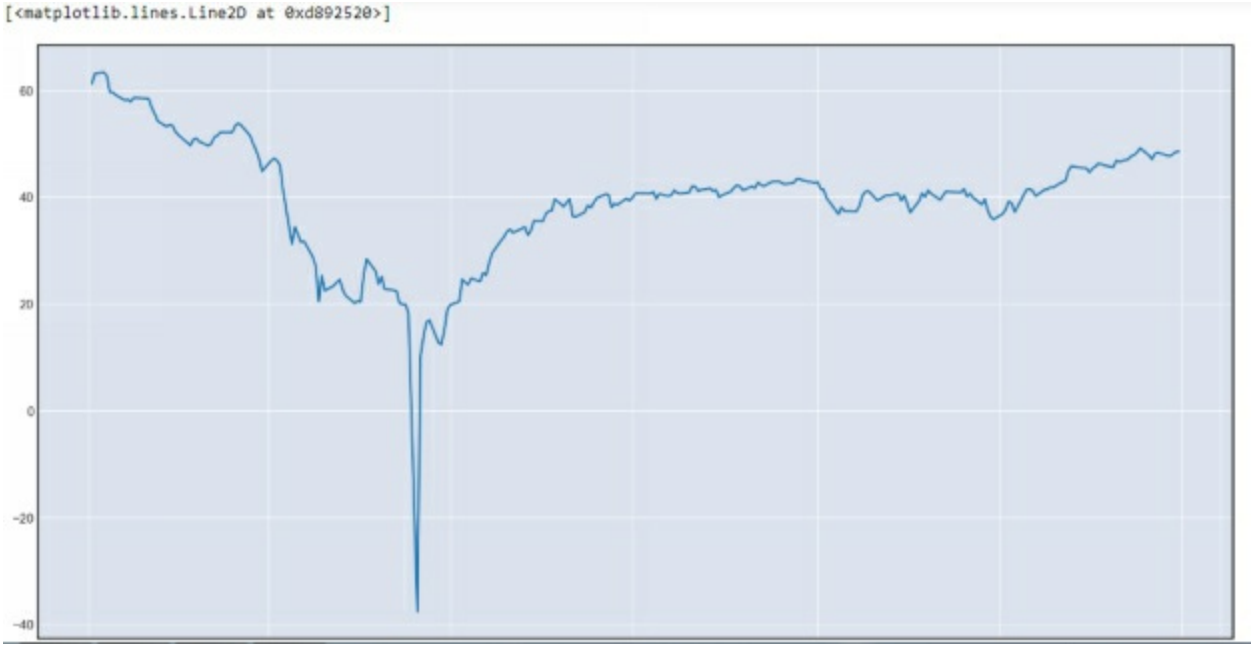
!pip install quandl

Following codes can help you to fetch historical data. In the following example I am fetching WTI Crude oil prices of CME –

import quandl
start = datetime(2020, 1, 1)
end = datetime(2020, 12, 31)
df = quandl.get('CHRIS/CME_CL1', start_date=start, end_date=end, qopts= {'columns': ['Settle']}, authtoken='insert you key that you get on registration with quandl')
plt.figure(figsize=(20,10))
plt.plot(df.Settle)

```
import quandl
start = datetime(2020, 1, 1)
end = datetime(2020, 12, 31)
df = quandl.get('CHRIS/CME_CL1', start_date=start, end_date=end, qopts={'columns': ['Settle']}, authtoken='
plt.figure(figsize=(12,6))
plt.plot(df.Settle)
```

You will get the following output –



Further you can learn from Quandl website itself about how to fetch data in python –

https://www.quandl.com/tools/python

PYHTON CODES FOR TECHNICAL INDICATORS

-

We understood the trends. In this chapter we will understand technical indicators used to predict the trends. There is no perfect method to predict the trend but we will use numbers to uncover the correct direction on prices. As we have seen false breakout in previous chapter the same we may face while trading with numbers. A false prediction will result into loss. At end of the day we have to check sum of total profit earned in successful trade and loss in false trade.

We also need to consider drawdown's because if you will lose 20% of capital in false move then you need to earn 25% return to reach on same level. For example you started with Rs 100/-. You lost 20% in false move and now you have Rs 80/-. You need to earn 25% of Rs 80 you have to reach on same level of Rs 100/-. **So drawdown's in any strategy should as much lower as possible.** That's why we can use options spread to minimize our losses. To learn options and why options are better then future please read my book "Systematic Options Trading: Option Greeks, Strategies & Backtesting in Python".

We can use following indicators to predict the trend.

1. Moving Average

Moving average is one of the most popular indicators to predict the trend. The 200-day moving average seems to have become a benchmark for deciding that a particular stock/index is going up or down. If the price of the

stock is above the 200-day moving average, then the long term trend is up. If the price of the stock is below the 200-day moving average, then the long term trend is down. Let's understand how to compute moving average. First we need to understand mean value or average price.

Mean is average of all numbers. Mean value is sum of all numbers divided by count of all numbers.

For example a stock has given a return of 5%, 8%, 15%, 2% and 10% in last 5 years. What is average return?

Average return = (5+8+15+2+10) / 5 = 8

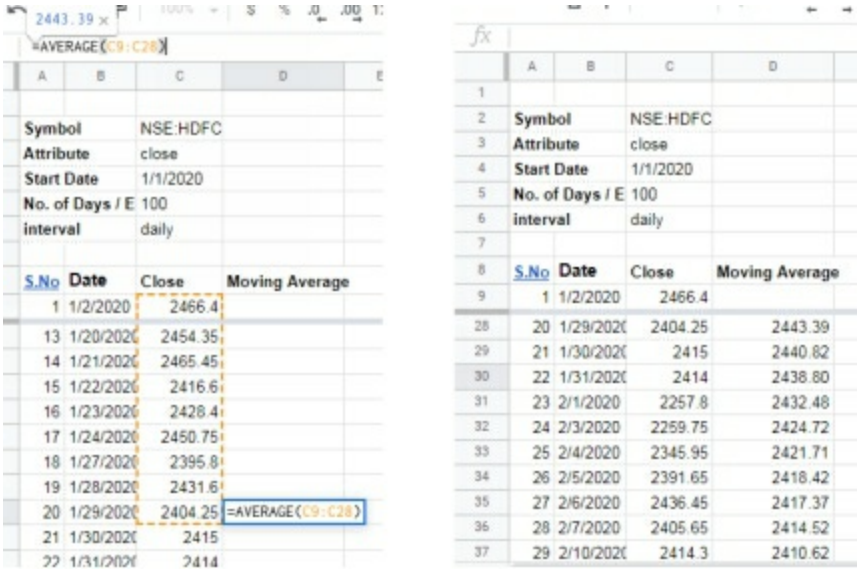
So mean value is 8, you can say stock has given a average return of 8% in last 5 years.

Let's compute mean value in Google Sheet. In the following example we are computing moving average of data of HDFC we fetched in Google sheet. We are computing 20 days moving average of close price of S.No. 1 to 20 in cell D28 as shown in following screenshot. Again 20 days moving average of close price of S.No. 2 to 21 in cell D29 and so on.

D28 = average(C9: C28)

D29 = average(C10: C29)

So 20-days moving average is simply the average price over the past 20 days. Each moving average value is computed using most recent days used in the calculation. Moving average is series of average of different subsets of the full data set. The average is taken over for a specific period of time for example 30minuts, 30 days, 50 days etc.



	A	B	C	D
1				
2	Symbol		NSE:HDFC	
3	Attribute		close	
4	Start Date		1/1/2020	
5	No. of Days / E		100	
6	interval		daily	
7				
8	S.No	Date	Close	Moving Average
9	1	1/2/2020	2466.4	
28	20	1/29/202(	2404.25	2443.39
29	21	1/30/202(	2415	2440.82
30	22	1/31/202(	2414	2438.80
31	23	2/1/2020	2257.8	2432.48
32	24	2/3/2020	2259.75	2424.72
33	25	2/4/2020	2345.95	2421.71
34	26	2/5/2020	2391.65	2418.42
35	27	2/6/2020	2436.45	2417.37
36	28	2/7/2020	2405.65	2414.52
37	29	2/10/202(	2414.3	2410.62

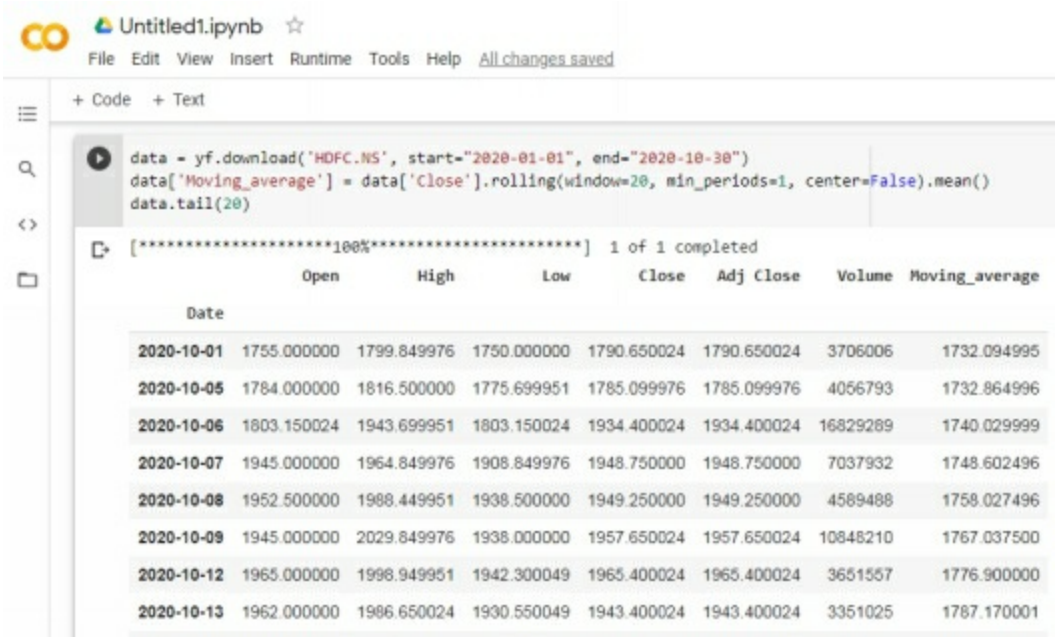
Following commands can be used to compute moving average (rolling average) in a Jupyter note book. We have taken a window of 20 days.

```
data['Moving_average'] = data['Close'].rolling(window=20, min_periods=1,center=False).mean()
```

You can check the output of above using following command.

```
data.tail(20)
```

You will get the following output of above commands. We have moving average of last 20 trading days in last column.

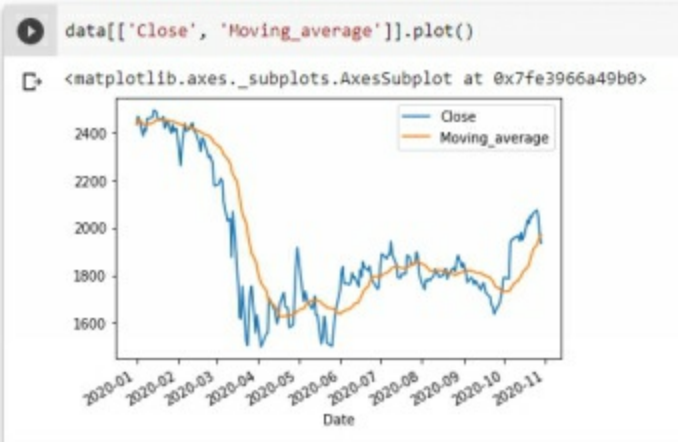


Date	Open	High	Low	Close	Adj Close	Volume	Moving_average
2020-10-01	1755.000000	1799.849976	1750.000000	1790.650024	1790.650024	3706006	1732.094995
2020-10-05	1784.000000	1816.500000	1775.699951	1785.099976	1785.099976	4056793	1732.864996
2020-10-06	1803.150024	1943.699951	1803.150024	1934.400024	1934.400024	16829289	1740.029999
2020-10-07	1945.000000	1964.849976	1908.849976	1948.750000	1948.750000	7037932	1748.602496
2020-10-08	1952.500000	1988.449951	1938.500000	1949.250000	1949.250000	4589488	1758.027496
2020-10-09	1945.000000	2029.849976	1938.000000	1957.650024	1957.650024	10848210	1767.037500
2020-10-12	1965.000000	1998.949951	1942.300049	1965.400024	1965.400024	3651557	1776.900000
2020-10-13	1962.000000	1986.650024	1930.550049	1943.400024	1943.400024	3351025	1787.170001

You can plot the chart of Close price and moving average we have computed with the following command -

```
data[['Close', 'Moving_average']].plot()
```

You will get following output.



In the above chart blue line is close price and orange line is last 20 days moving average. From January 2020 to April 2020 the stock was trading below 20 days moving average, it means stock was in down trend. Moving average crossover is used to generate buy or sell signals. If short period moving average line is above the long period moving average then it's a buy signal (Uptrend) and if short period line is below the long period line it is sell signal (Down-trend). This works in trending market but in rage bound market this strategy may give losses. In range bound market mean reversal strategy will give you profit. One more point to note that the long period moving average will be smoother but longer is later in telling you that the trend has changed direction. You also need to keep in mind that if period will be shorter then number of trades will be higher that will result into more cost especially when cost of trading is high.

Trading on Moving average crossover -

Which moving average use for trading is a very subjective decision and it

may depend on contract you are trading. Let's backtest the returns given by moving average crossovers on past data. I have written following codes in which one can define stock, years, short moving averages and long moving averages to get returns. With the help of following code I am computing return given by Reliance Industries during the year 2020 on moving average crossovers of 10 days and 30 days.

```
# Download necessary libraries
!pip install yfinance
from datetime import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import yfinance as yf


# Fetch Historical Data
Stock = "RELIANCE.NS"
data = yf.download(Stock, start="2020-01-01", end="2020-12-31")


# A separate table created with close price values
T = pd.DataFrame({"Close": data["Close"]})
# Short and Long moving averages defined
SMA=10
LMA=30
# Compute Moving averages of last 10 days and 30 days closing prices
T['Short_average'] = T['Close'].rolling(window=SMA, min_periods=1, center=False).mean()
T['Long_average'] = T['Close'].rolling(window=LMA, min_periods=1, center=False).mean()
T['positions_long'] = np.nan
for x in range (len(T)):
    if T.Short_average[x] > T.Long_average[x]:
        T['positions_long'][x] = 1
    if T.Short_average[x] <= T.Long_average[x]:
        T['positions_long'][x] = 0
T.positions_long = T.positions_long.fillna(method='ffill')
```

```
T['price_difference']= T.Close - T.Close.shift(1)
T['pnllong'] = T.positions_long.shift(1) * T.price_difference
T['cumpnl_long'] = T.pnllong.cumsum()
print(T)
# Download file 'T' in csv format to check computations
T.to_csv("MA_T.csv", index=True, encoding='utf8')
```

Screenshot of file 'T' downloaded is given below -

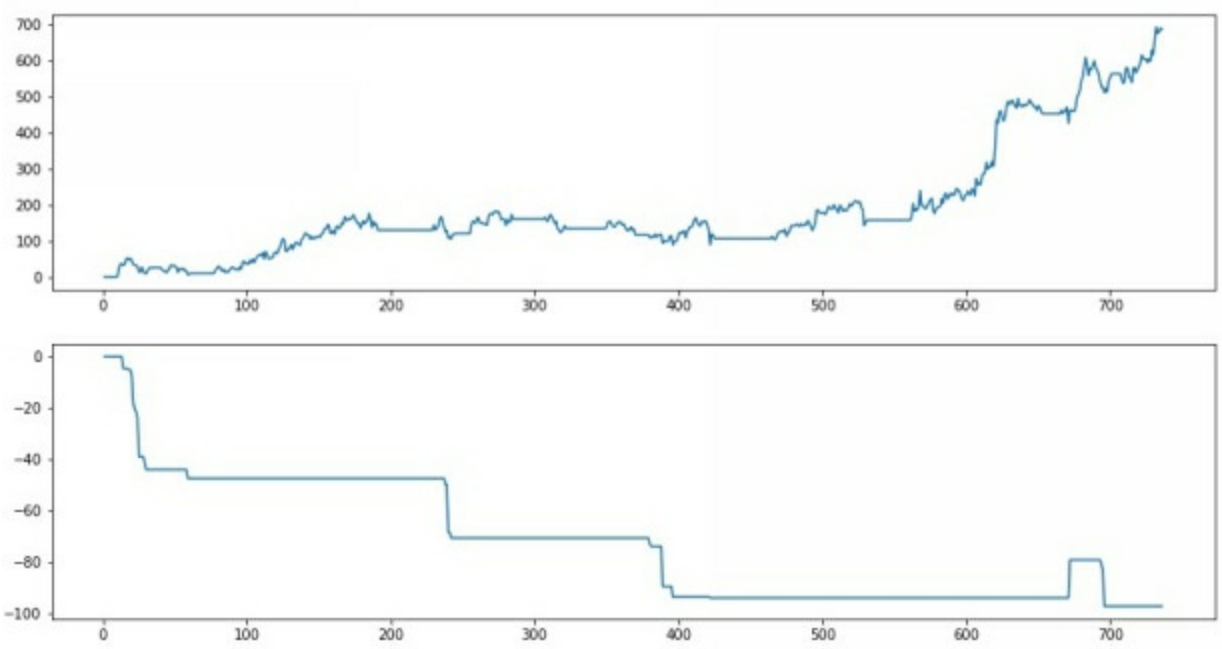
	A	B	C	D	E	F	G	H
1	Date	Close	Short_average	Long_average	positions_long	price_difference	pnllong	cumpnl_long
60	25/03/2020	1072.087646	991.1895203	1242.979458	0	137.5461426	0	-51.61071777
61	26/03/2020	1056.188354	991.5065125	1230.215448	0	-15.89929199	0	-51.61071777
62	27/03/2020	1055.593994	987.5737854	1216.852118	0	-0.594360352	0	-51.61071777
63	30/03/2020	1020.774109	989.0349365	1202.200997	0	-34.81988525	0	-51.61071777
64	31/03/2020	1103.29187	999.5106323	1189.856344	0	82.51776123	0	-51.61071777
65	01/04/2020	1070.304565	1010.600513	1176.720854	0	-32.98730469	0	-51.61071777
66	03/04/2020	1067.332764	1026.425507	1163.844576	0	-2.971801758	0	-51.61071777
67	07/04/2020	1194.77478	1045.063837	1154.014425	0	127.4420166	0	-51.61071777
68	08/04/2020	1180.955688	1075.584528	1144.313049	0	-13.8190918	0	-51.61071777
69	09/04/2020	1208.494629	1102.97984	1136.883472	0	27.53894043	0	-51.61071777
70	13/04/2020	1177.983887	1113.569464	1129.379602	0	-30.51074219	0	-51.61071777
71	15/04/2020	1139.052856	1121.855914	1121.383728	1	-38.93103027	0	-51.61071777
72	16/04/2020	1157.082031	1132.004718	1114.178691	1	18.0291748	18.0291748	-33.58154297
73	17/04/2020	1212.506592	1151.177966	1110.723112	1	55.42456055	55.42456055	21.84301758
74	20/04/2020	1232.120728	1164.060852	1108.33409	1	19.61413574	19.61413574	41.45715332
75	21/04/2020	1225.731323	1179.603528	1104.850448	1	-6.389404297	-6.389404297	35.06774902
76	22/04/2020	1350.795776	1207.949829	1105.639632	1	125.0644531	125.0644531	160.1322021
77	23/04/2020	1358.027222	1224.275073	1107.612598	1	7.231445313	7.231445313	167.3636475
78	24/04/2020	1403.694336	1246.548938	1112.433565	1	45.66711426	45.66711426	213.0307617
79	27/04/2020	1416.324707	1267.331946	1122.854785	1	12.63037109	12.63037109	225.6611328
80	28/04/2020	1414.739624	1291.00752	1131.922168	1	-1.585083008	-1.585083008	224.0760498
81	29/04/2020	1413.550903	1318.457324	1143.939917	1	-1.188720703	-1.188720703	222.8873291
82	30/04/2020	1452.234253	1347.972546	1155.85035	1	38.68334961	38.68334961	261.5706787
83	04/05/2020	1421.723511	1368.894238	1169.70238	1	-30.51074219	-30.51074219	231.0599365

Kindly observe the above output. In short average column we have computed last 10 days moving average and in a Long average column we have computed last 30 days moving average. In column 'positions_long' we have generated buy signal when short moving average is trading above long moving average. This field of 'cumpnl_long' is giving us the per share return given by this strategy.

We also need to check drawdown in any strategy. Following python codes can be used to compute the drawdown.

```
# Calculate the max drawdown in the past window days for each day
T['rolling_max'] = T['cumpnl_long'].rolling(250, min_periods=1).max()
T['daily_drawdown'] = T['cumpnl_long']-T['rolling_max']
# Calculate the maximum daily drawdown
T['max_daily_drawdown'] = T['daily_drawdown'].rolling(250, min_periods=1).min()


# plot the return and drawdown of strategy
fig, axs = plt.subplots(2,figsize=(15,8))
axs[0].plot(T['cumpnl_long'])
axs[1].plot(T['max_daily_drawdown'])
```

Output of the above codes is given below. Total profit of the strategy is approximately 700 per share in the year 2020. Maximum drawdown of the strategy was approximately 100.



We can do further analysis of strategy performance like total number of trade count, maximum profit in a trade, maximum loss in a trade etc. Following code can be used for computation –

```
T['Trade'] = T['positions_long'].diff()
T1 = T.where((T.Trade != 0))
T1 = T1.dropna()
T1.drop(["Short_average", "Long_average", "positions_long",
```

```
"price_difference", "pnllong", "rolling_max", "daily_drawdown",
"max_daily_drawdown"], axis = 1, inplace = True)
T1['Trade_Return'] = (T1['cumpnl_long'].diff()/T1['Close'])*100
print ("Number of Trade count", round(len(T1)/2))
print (T1)
```

Output of the above code is given below –

```
Number of Trade count 12
           Date        Close  cumpnl_long  Trade  Trade_Return
10   2018-01-15   540.575012     0.000000    1.0           NaN
33   2018-02-19   565.875000    25.299988   -1.0      4.470950
40   2018-02-28   586.299988    25.299988    1.0      0.000000
61   2018-04-03   570.224976     9.224976   -1.0     -2.819065
77   2018-04-25   580.174988     9.224976    1.0      0.000000
191  2018-10-10   700.450012   129.500000   -1.0     17.171107
229  2018-12-06   668.500000   129.500000    1.0      0.000000
245  2018-12-31   658.950012   119.950012   -1.0     -1.449273
255  2019-01-14   701.900024   119.950012    1.0      0.000000
285  2019-02-26   742.500000   160.549988   -1.0      5.468010
306  2019-04-01   755.099976   160.549988    1.0      0.000000
322  2019-04-25   728.549988   134.000000   -1.0     -3.644223
349  2019-06-06   735.599976   134.000000    1.0      0.000000
370  2019-07-05   718.000000   116.400024   -1.0     -2.451250
379  2019-07-18   792.700012   116.400024    1.0      0.000000
426  2019-09-27   782.200012   105.900024   -1.0     -1.342368
464  2019-11-27   695.750000   105.900024    1.0      0.000000
531  2020-03-03   747.000000   157.150024   -1.0      6.860776
561  2020-04-21   633.200012   157.150024    1.0      0.000000
653  2020-08-31   928.599976   452.549988   -1.0     31.811326
665  2020-09-16  1001.750000   452.549988    1.0      0.000000
701  2020-11-06  1112.750000   563.549988   -1.0      9.975286
707  2020-11-17  1123.699951   563.549988    1.0      0.000000
```

As you can observe in above output, total numbers of trade in above strategy are 12. If you will observe the ‘Trade_Return’ you will find highest return in a single trade is 31.81% and maximum loss in a trade is -3.64%. One more thing you can observe that moving average strategy is giving small losses and big profits.

Above python code can be used to compute returns given by any stock of any exchange for a given moving average crossover. We can create loops also to compute return of multiple stocks on multiple moving averages that we will learn later on.

Back-testing multiple Moving average crossover -

Which moving averages to take for trading is a very subjective decision. However with the help of Python we can backtest return given by stock on various moving averages in various periods. I have written following codes in which one can define stock, years, short moving averages and long moving averages to get returns. With the help of following code I am computing yearly return given by Reliance Industries during the year 2016 to 2020 on different combinations of moving average crossovers from 1 to 35. Trader is buying when short moving average is crossing long moving average from downside and selling when short moving average is crossing long moving average from upside.

If you will observe the following code you will notice that I have used 3 ‘for’ loops. 1st ‘for’ loop for years, 2nd ‘for’ loop for short moving average and 3rd for loop for long moving average. Pivot Table created with values of ‘cumpnl_long’. This field of ‘cumpnl_long’ is giving us the per share return given by stock when we are having long position if short moving average is above long moving average

Python Code -

```
!pip install yfinance
from datetime import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import yfinance as yf
T3 = pd.DataFrame({"Close": data["Close"]})
T3['Year'] = T3.index.year
T2 = pd.DataFrame({"cumpnl_long":['0'], "SMA":['0'],  "LMA":['0'], "Year":
['0']})
for z in range (2016, 2021, 1):
    T = T3.where(T3.Year == z)
    T = T.dropna()
    for a in range(1,35,2):
        for b in range(a,35,2):
        SMA=a
        LMA=b
```

```
        # Compute Long and short Moving averages
        T['Short_average'] = T['Close'].rolling(window=SMA, min_periods=1,
center=False).mean()
        T['Long_average'] = T['Close'].rolling(window=LMA, min_periods=1,
center=False).mean()
        T['positions_long'] = np.nan
        for x in range (len(T)):
            if T.Short_average[x] > T.Long_average[x]:
                T['positions_long'][x] = 1
            if T.Short_average[x] <= T.Long_average[x]:
                T['positions_long'][x] = 0
        T.positions_long = T.positions_long.fillna(method='ffill')
        T['price_difference']= T.Close - T.Close.shift(1)
        T['pnllong'] = T.positions_long.shift(1) * T.price_difference
        T['cumpnl_long'] = T.pnllong.cumsum()
        T1 = T[['cumpnl_long']].tail(1)
        T1['SMA'] = SMA
        T1['LMA'] = LMA
        T1['Year'] = z
        T2 = T2.append(T1)
#Create pivot table
Pivot_Table1 = pd.pivot_table(T2, values ='cumpnl_long', index =['SMA',
'LMA'], columns =['Year'], aggfunc = np.sum)
#Export pivot table
Pivot_Table1.to_csv("PV_T.csv", index=True, encoding='utf8')
from google.colab import files
files.download('PV_T.csv')
```

You will get the following output in Excel downloaded-

	A	B	C	D	E	F	G
1	SMA	LMA	2016	2017	2018	2019	2020
2	1	1	0	0	0	0	0
3	1	3	-55.6749	432.5246	382.4998	240.9003	323.2694
4	1	5	-9.02502	353.75	367.45	361.8004	326.2994
5	1	7	4.825043	330.9499	77.70001	405.7504	457.6096
6	1	9	-33.35	306.6999	101.8	283.0004	576.5597
7	1	11	-66.425	282.8	150.35	250.05	655.0596
8	1	13	-30.65	268.5501	180.8999	249.35	666.8998
9	1	15	-39.2751	242.325	292.8499	262.9	621.6495
10	1	17	-14.8251	253.3749	250.3498	237.9	717.7498
11	1	19	-2.85007	267.6249	182.6497	227.45	665.7499
12	1	21	9.499969	270.2749	87.14978	189.05	525.4999
13	1	23	-0.19998	220.7999	105.1998	162.8999	553.9
14	1	25	-5.69998	243.65	100.4498	146.7999	545.3002
15	1	27	-19.2249	251.1249	104.6498	205	672.7498
16	1	29	-15.35	255.075	134.8498	263.2999	817.0498
17	1	31	-10.125	278.725	102.5997	279.9501	768.9497
18	1	33	-10.825	259.375	97.59973	266.7002	750.7496
19	3	3	0	0	0	0	0
20	3	5	6.925018	225.6251	123.6001	327.4501	677.3296
21	3	7	8.69989	195.8499	59.04987	108.4	692.5497
22	3	9	7.649841	229.225	74	207.1998	732.9995
23	3	11	1.875	185.325	-5.85004	265.95	701.4995
24	3	13	23.09995	173.6751	9	193.45	915.7999
25	3	15	75.875	185.375	98.14978	245.05	738.6499

PV_T

In the above output of excel you will observe moving average crossover of 1 day and 5 days is giving consistent return year on year (SMA denotes short moving average and LMA denotes long moving average). So technically you can say Reliance is buy if trading above 5 days moving average. But this will result into many trades and cost is associated with every trade. If you will observe moving average cross over of 1 day and 29 days is also giving consistent return. It means if Reliance is trading 30 days moving average its a buy.

One more thing you will notice that in year 2020 all the moving averages has given very good return because prices were trending in year 2020. We have seen rollercoaster ride in year 2020 from Nifty 12000 to 8000 and again back to 12000. In trending market moving average crossover gives good return however in range bound market mean reversal strategies gives good return.

As we have computed return for Reliance, in the same way you can compute moving average return given by any stock on past data for any combination of moving averages.

2. Relative Strength Index(RSI)

The relative strength index is a momentum oscillator commonly used to predict when a company is oversold or overbought. The RSI will then be a value between 0 and 100. It is widely accepted that when the RSI is 30 or below, the stock is oversold and when it is 70 or above, the stock is overbought.

The calculation process is given below:

1. Observe the last 14 closing prices of a stock.
2. Determine whether the current day's closing price is higher or lower than the previous day.
3. If the price has increased from previous day price, we note down the difference in the "Gain" column and if it's a loss, then we note it down in the "Loss" column.
4. Calculate the average of "Gain" column and "Loss" column over the last 14 days.
5. Compute the relative strength (RS): (AvgGain/AvgLoss)
6. Compute the relative strength index (RSI): (100–100 / (1 + RS))

Python code for computation of RSI on simple moving average is given below-

```
# Download necessary libraries
from datetime import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
import yfinance as yf


# Fetch Historical Data
Stock = "RELIANCE.NS"
data = yf.download(Stock, start="2020-01-01", end="2020-12-31")
# Get just the close in file 'T1'
T1 = pd.DataFrame({"close": data["Close"]})
# Window length for moving average
window_length = 14
# Get the difference in price from previous step
T1['delta'] = T1['close'].diff()
# Make the positive gains (up) and negative gains (down) Series
T1['up'] = T1['delta']
T1['down'] = T1['delta']


for x in range (len(T1)):
    if T1.up[x] < 0:
        T1.up[x] = 0
    if T1.down[x] > 0:
        T1.down[x] = 0


# Calculate the SMA
T1['roll_up'] = T1['up'].rolling(window_length).mean()
T1['roll_down'] = T1['down'].abs().rolling(window_length).mean()
# Calculate the RSI based on SMA
T1['RS'] = T1['roll_up'] / T1['roll_down']
T1['RSI'] = 100.0 - (100.0 / (1.0 + T1['RS']))
print (T1)
# Download file ‘T1’ in csv format to check computations
T1.to_csv("RSI_T.csv", index=True, encoding='utf8')
```

Screenshot of file ‘T1’ downloaded is given below -

	A	B	C	D	E	F	G	H	I
1	Date	close	delta	up	down	roll_up	roll_down	RS	RSI
2	01/01/2020	1495.425							
3	02/01/2020	1520.884	25.45862	25.45862	0				
4	03/01/2020	1522.716	1.832642	1.832642	0				
5	06/01/2020	1487.401	-35.3153	0	-35.3153				
6	07/01/2020	1510.284	22.88318	22.88318	0				
7	08/01/2020	1498.942	-11.3425	0	-11.3425				
8	09/01/2020	1533.464	34.52271	34.52271	0				
9	10/01/2020	1533.118	-0.34668	0	-0.34668				
10	13/01/2020	1529.205	-3.91284	0	-3.91284				
11	14/01/2020	1515.039	-14.1658	0	-14.1658				
12	15/01/2020	1509.541	-5.49792	0	-5.49792				
13	16/01/2020	1523.459	13.91809	13.91809	0				
14	17/01/2020	1566.154	42.69531	42.69531	0				
15	20/01/2020	1517.961	-48.1931	0	-48.1931				
16	21/01/2020	1519.497	1.5354	1.5354	0	10.20328	8.483869	1.202668	54.60052
17	22/01/2020	1518.952	-0.5448	0	-0.5448	8.384809	8.522784	0.983811	49.59198
18	23/01/2020	1512.513	-6.43896	0	-6.43896	8.253906	8.98271	0.918866	47.88589
19	24/01/2020	1507.263	-5.25024	0	-5.25024	8.253906	6.835205	1.207558	54.70108
20	27/01/2020	1492.404	-14.8591	0	-14.8591	6.619393	7.896572	0.838262	45.60078
21	28/01/2020	1457.93	-34.4733	0	-34.4733	6.619393	9.548767	0.69322	40.94092
22	29/01/2020	1465.954	8.023926	8.023926	0	4.726624	9.548767	0.494998	33.11029
23	30/01/2020	1430.193	-35.761	0	-35.761	4.726624	12.07836	0.39133	28.12632
24	31/01/2020	1398.395	-31.7986	0	-31.7986	4.726624	14.0702	0.335932	25.14586
25	03/02/2020	1372.49	-25.9045	0	-25.9045	4.726624	14.90868	0.317038	24.07206

As discussed earlier that when the RSI is 30 or below, the stock is oversold so trader will buy the stock and when it is 70 or above, the stock is overbought so trader will sell the stock. The RSI can give false signals too.

The fundamental property of RSI which states that a level above 70 is overbought can be proved wrong in a strong bull market where the company is progressing rapidly and posting good returns to its shareholders in this scenario the RSI can stay above 70 for a long time, which can be disastrous for short sellers. It is a similar case for a bear market where the RSI can stay below 30 and not rise above that level for an extended period of time. Now you have data and tools so you can make your own observations/strategies.

In the following example when 21 days RSI is trading above 70 is an indication that there is up trend in the stock and when 21 days RSI is trading

below 30 there is down trend in the stock. So in following program code I am buying Reliance when RSI is above 70 and squaring off position when it's going below 50. I am selling when RSI is going below 30 and position is covered when coming back to 50.

Python Codes for computation of profit and loss with RSI:

```
T1['positions_long'] = np.nan
for x in range (len(T1)):
    if T1.RSI[x] > 70:
        T1['positions_long'][x] = 1
    if T1.RSI[x] <= 50:
        T1['positions_long'][x] = 0
T1.positions_long = T1.positions_long.fillna(method='ffill')
T1['positions_short'] = np.nan
for x in range (len(T1)):
    if T1.RSI[x] < 30:
        T1['positions_short'][x] = -1
    if T1.RSI[x] >= 50:
        T1['positions_short'][x] = 0
T1.positions_short = T1.positions_short.fillna(method='ffill')
T1['positions'] = T1.positions_long + T1.positions_short
T1['price_difference']= T1.close - T1.close.shift(1)
T1['pnl'] = T1.positions.shift(1) * T1.price_difference
T1['cumpnl'] = T1.pnl.cumsum()
# Calculate the max drawdown in the past window days for each day
T1['rolling_max'] = T1['cumpnl'].rolling(250, min_periods=1).max()
T1['daily_drawdown'] = T1['cumpnl']-T1['rolling_max']
# Calculate the minimum (negative) daily drawdown
T1['max_daily_drawdown'] = T1['daily_drawdown'].rolling(250,
min_periods=1).min()
fig, axs = plt.subplots(2,figsize=(15,8))
axs[0].plot(T1['cumpnl'])
axs[1].plot(T1['max_daily_drawdown'])
```

Output of above Python Codes:

As we can observe in above output return given by strategy is 1000/- per share whereas the maximum drawdown was 200/-.

Python Codes for further analysis of strategy:

```
T1['Trade'] = T1['positions'].diff()
T2 = T1.where((T1.Trade != 0))
T2 = T2.dropna()
T2.drop(["delta", "up", "down", "roll_up", "roll_down", "RS", "positions_long", "positions_short", "positions", "price_difference", "pnl", "rolling_max", "daily_drawdown", "max_daily_drawdown"], axis = 1, inplace = True)
T2['Trade_Return'] = (T2['cumpnl'].diff()/T2['close'])*100
print ("Number of Trade count", round(len(T2)/2))
print (T2)
```

Output of above Python Codes:

```
T1['Trade'] = T1['positions'].diff()
T2 = T1.where((T1.Trade != 0))
T2 = T2.dropna()
T2.drop(["delta", "up", "down", "roll_up", "roll_down", "RS", "positions_
T2['Trade_Return'] = (T2['cumpnl'].diff()/T2['close'])*100
print ("Number of Trade count", round(len(T2)/2))
print (T2)
```

```
Number of Trade count 7
                 close        RSI      cumpnl  Trade  Trade_Return
Date
2020-01-30  1430.193237  28.126320    0.000000   -1.0           NaN
2020-02-17  1464.369263  51.239510  -34.176025    1.0     -2.333839
2020-03-02  1303.791382  29.839665  -34.176025   -1.0      0.000000
2020-04-01  1070.304565  51.050382  199.310791    1.0     21.814988
2020-04-16  1157.082031  72.600669  199.310791    1.0      0.000000
2020-05-19  1408.900024  49.382481  451.128784   -1.0     17.873376
2020-06-04  1579.800049  74.385510  451.128784    1.0      0.000000
2020-08-13  2122.050049  47.395739  993.378784   -1.0     25.553120
2020-09-10  2314.000000  73.900976  993.378784    1.0      0.000000
2020-09-30  2234.350098  37.399173  913.728882   -1.0     -3.564791
2020-10-26  2029.099976  27.152387  913.728882   -1.0      0.000000
2020-11-23  1950.699951  56.907119  992.128906    1.0      4.019071
2020-12-11  2005.800049  70.340629  992.128906    1.0      0.000000
2020-12-21  1939.699951  46.671033  926.028809   -1.0     -3.407749
```

As we can observe in above output maximum profit in a trade was 25.55% and maximum loss as -3.56%. Total 7 trades generated by this strategy in year 2020.

There are hundreds of indicators used in Technical Analysis for prediction of price movement. We should use combination of different indicators which will give us a holistic view of the market and help us extract maximum information from the price action of a particular asset. So a trading setup includes many indicators. We will discuss one trading setup at end of this book to get directional view of the stock.

Writing python codes for each indicator is going to be a Herculean task. So we will use Technical Analysis libraries of Python for computation of these indicators. This will save lots of energy and time. One of the most popular libraries in python for technical analysis is TA-LIB. We will use this to make our job easy. Our objective is to make money not to understand how to write codes. So we will remain focused on development of profitable trading setups.

TECHNICAL ANALYSIS LIBRARY (TA-LIB) IN PYTHON FOR BACKTESTING

If we work on developing a technical indicators code from scratch it will require huge amount of efforts and difficulty. Best thing in python is that we can use the work done by others. Mario Fortier started TA-LIB library in Python as a hobby. Today this library has become one of the most famous libraries for technical analysis of stocks and other financial securities. Ta-lib includes 150+ indicators such as ADX, MACD, RSI and Bollinger Bands and candlestick pattern recognition.

INSTALLING TA-LIB PYTHON LIBRARY

Ta-lib installation is different from other python libraries as it is not available to install directly using pip install.

First, we need to visit the following link and download the whl file of Ta-Lib according to our windows version.

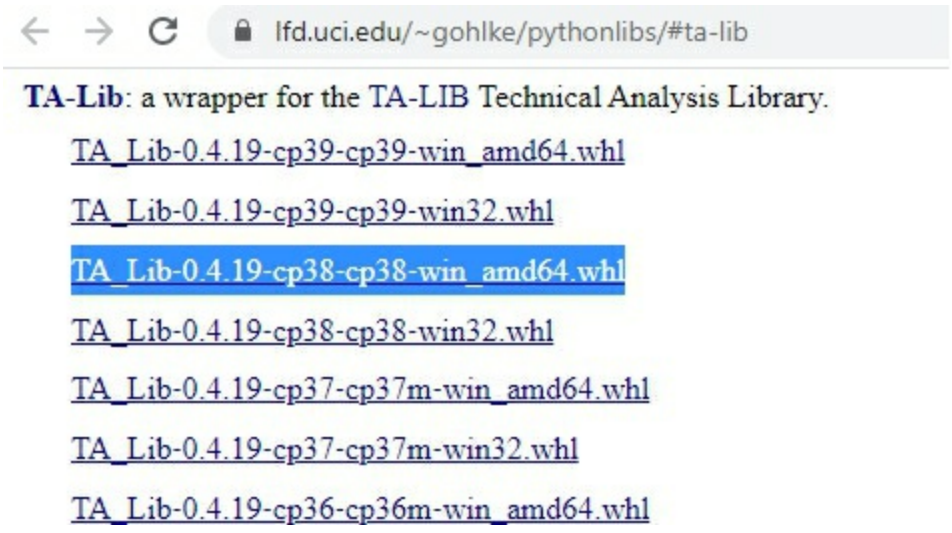
https://www.lfd.uci.edu/~gohlke/pythonlibs/#ta-lib

Following commands can be used to check your windows and version.

```
import platform
import sys

!python -V
print(platform.platform())
print (sys.version)
```

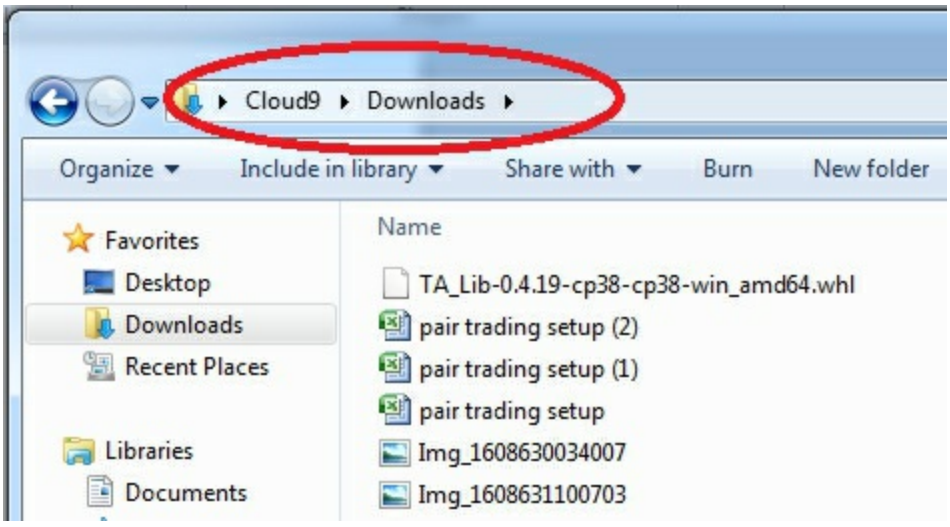
```
Python 3.8.5Windows-7-6.1.7600-SP0
3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
```

One should note that you should download the file keeping your Python version and Windows architecture (32 bit or 64 bit) in mind. Since I have the python version 3.8 installed and 64 bit Windows 7 system, I will download the file, “TA_Lib-0.4.19-cp38-cp38-win_amd64.whl”. As you might have guessed “cp38” implies Python version 3.8 and “win_amd64” implies Windows 64 bit operating system.



By default, the "whl" file gets downloaded in the "Downloads" folder of your system user. You need to copy this file from downloads and paste it to parent folder “<user name of parent folder>”. As shown in following screen shot, my laptop User name is ‘Cloud9’ so “whi” file got downloaded in folder Cloud9 > Downloads.

I just copied the “whi” file from downloads and pasted it to parent folder ‘Cloud9’.



After that, we can install it using pip install as given below.

```
!pip install TA_Lib-0.4.19-cp38-cp38-win_amd64.whl
```

```
!pip install TA_Lib-0.4.19-cp38-cp38-win_amd64.whl
```

```
Processing c:\users\cloud9\ta_lib-0.4.19-cp38-cp38-win_amd64.whl
Installing collected packages: TA-Lib
Successfully installed TA-Lib-0.4.19
```

DOWNLOAD PAST DATA FOR REUSE

Now without writing long complicated codes we can start using Python library Ta_Lib for technical based trading and backtesting.

One more thing you should note that you should download data on your laptop inspite of fetching again and again same data from yfinance. You can use downloaded files for various experiments. In the coming chapters we will also use past data file saved on system. We can use following codes to fetch historical data of stock from yfinance and to save it on laptop/system for reuse –

Python codes-

```
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
import yfinance as yf
# Fetch historical data
data = yf.download('INFY.NS', start="2018-01-01", end="2020-12-31")
# Download file ‘data’ in csv format on Laptop/system to reuse Infosys data
data.to_csv("Infy-His-data2018-20.csv", index=True, encoding='utf8')
#Print data
data.head()
```

```
# Fetch historical data
data = yf.download('INFY.NS', start="2018-01-01", end="2020-12-31")

# Download file 'data' in csv format to reuse infosys data
data.to_csv("Infy-His-data2018-20.csv", index=True, encoding='utf8')

data.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-01	518.849976	522.250000	515.000000	516.775024	474.472595	5431340
2018-01-02	518.625000	521.000000	511.500000	514.849976	472.705139	6112248
2018-01-03	514.250000	515.799988	509.299988	510.649994	468.848938	6846552
2018-01-04	510.500000	510.500000	504.799988	507.700012	466.140472	8947614
2018-01-05	507.649994	513.200012	503.000000	506.000000	464.579559	11025976

In the following example I have selected randomly 10 stocks to download past data of last 3 years from yfinance and saved it on my laptop so that I can use them again and again without increasing load on yfinance.

```
import yfinance as yf
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt


Stock = ['RELIANCE.NS', 'HDFCBANK.NS', 'SBIN.NS', 'ONGC.NS', 'BHEL.NS', 'NESTLEIND.NS', 'INFY.NS']
data = yf.download(Stock, start="2018-01-01", end="2020-12-31")


close = data['Close']
close.to_csv("Daily_Close", index=True, encoding='utf8')
Daily_Close = pd.read_csv('Daily_Close')
Daily_Close.head()
#you will get the following output
```

```
Daily_Close = pd.read_csv('Daily_Close')
Daily_Close.head()
```

	Date	BHEL.NS	HDFCBANK.NS	INFY.NS	NESTLEIND.NS	ONGC.NS	RELIANCE.NS	SBIN.NS
0	2018-01-01	95.150002	927.250000	516.775024	7863.200195	192.350006	909.750000	307.100006
1	2018-01-02	98.199997	936.174988	514.849976	7863.549805	196.850006	911.150024	303.250000
2	2018-01-03	98.550003	926.325012	510.649994	7860.100098	193.449997	914.799988	302.850006
3	2018-01-04	100.750000	929.950012	507.700012	7869.250000	199.500000	920.299988	308.500000
4	2018-01-05	102.349998	931.799988	506.000000	7856.750000	198.449997	923.250000	306.350006

close.describe()
#you will get the following output

```
close.describe()
```

	BHEL.NS	HDFCBANK.NS	INFY.NS	NESTLEIND.NS	ONGC.NS	RELIANCE.NS	SBIN.NS
count	737.000000	737.000000	737.000000	737.000000	737.000000	737.000000	737.000000
mean	57.491113	1096.338256	747.980190	12699.377896	133.155292	1370.113082	268.875441
std	21.413306	128.339688	151.021617	3169.692485	40.419717	372.413590	51.380974
min	19.850000	767.700012	506.000000	6969.649902	60.000000	882.700012	150.850006
25%	37.099998	1001.000000	658.950012	10289.299805	89.849998	1104.750000	242.600006
50%	62.450001	1066.650024	719.700012	11623.000000	142.800003	1278.000000	275.100006
75%	73.099998	1201.849976	781.700012	16048.500000	167.100006	1533.900024	306.350006
max	104.550003	1441.800049	1253.050049	18732.699219	210.850006	2324.550049	372.399994

Same way you can also save volume data for future use.

Volume = data['Volume']
Volume.to_csv("Daily_Volume", index=True, encoding='utf8')

DATA VISUALISATION IN PYTHON

Through Data visualization we try to understand data by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed.

We will use Plotly Python library in python program code. Plotly is a python library which helps in data visualisation in an interactive manner. You can zoom in our zoom out at any data point to get clearer picture. For that first you need to install Plotly through command -

!pip install plotly.

```
!pip install plotly
```

```
Collecting plotly
  Downloading plotly-4.14.3-py2.py3-none-any.whl (13.2 MB)
Collecting retrying>=1.3.3
  Downloading retrying-1.3.3.tar.gz (10 kB)
Requirement already satisfied: six in c:\users\cloud9\anaconda3\lib\site-
Building wheels for collected packages: retrying
  Building wheel for retrying (setup.py): started
  Building wheel for retrying (setup.py): finished with status 'done'
  Created wheel for retrying: filename=retrying-1.3.3-py3-none-any.whl s:
a249f7fcb153addf8d9cefaf0e
  Stored in directory: c:\users\cloud9\appdata\local\pip\cache\wheels\c4'
56
Successfully built retrying
Installing collected packages: retrying, plotly
Successfully installed plotly-4.14.3 retrying-1.3.3
```

TECHNICAL ANALYSIS WITH PYTHON LIBRARY TA_LIB

1. OVERLAP STUDIES

1.1 SIMPLE MOVING AVERAGE (SMA)

A simple moving average (SMA) calculates the average of a selected range of closing prices, by the number of periods in that range. Short term moving averages are more sensitive to price. When price declines short term average declines much faster than long term average. As decline continues short term average will decline further below long term average.

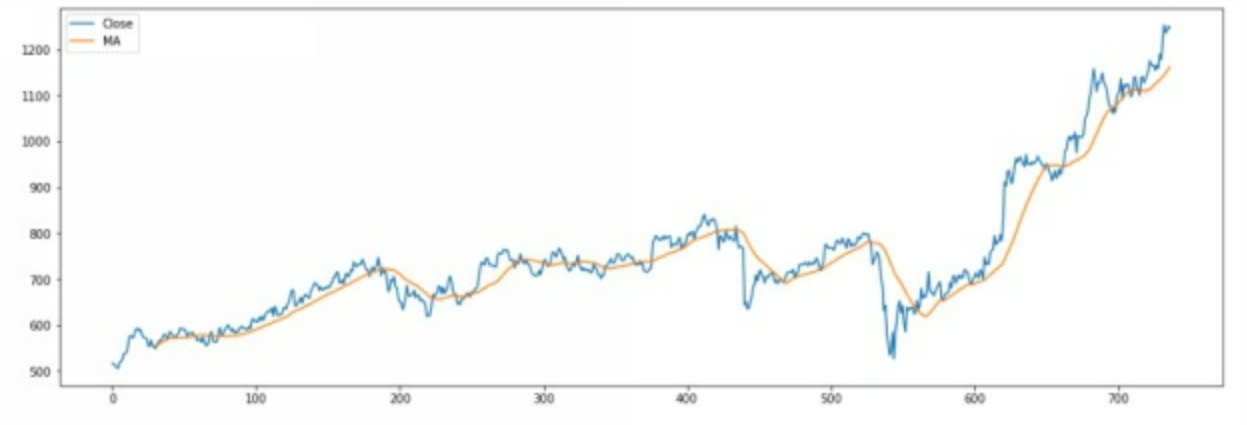
1. If short period moving average line is above the long period moving average then it's a buy signal and if short period line is below the long period line it is sell signal.
2. Short period Moving averages give early signal of entry and exit as compare to long period moving average.
3. Moving Average works in trending market but in rage bound market this strategy may give losses.

Python codes for computation of Moving averages are given below. We are not downloading data from yfinance. We are using files saved on out laptop in the following python codes.

```
# past data uploaded from file saved on system
Daily_Close = pd.read_csv('Daily_Close')
Daily_Close.head()
# Seperate table 'Infy' created with Infosys data
Infy = pd.DataFrame({"Date": Daily_Close["Date"], "Close": Daily_Close["INFY.NS"]})
Infy.head()
# Computation of simple moving average using python library TA-LIB
Infy['MA'] = ta.SMA(Infy['Close'],30)
Infy[['Close','MA']].plot(figsize=(18,6))
```

```
plt.show()
```

```
Infy['MA'] = ta.SMA(Infy['Close'],30)
Infy[['Close','MA']].plot(figsize=(18,6))
plt.show()
```

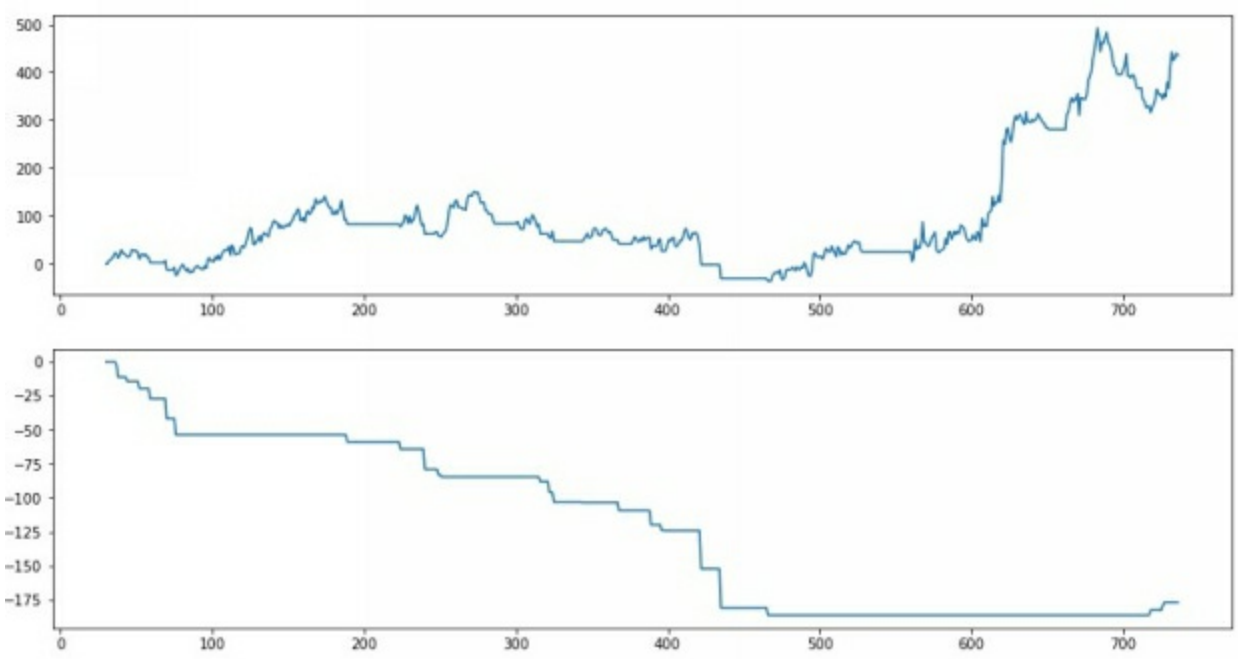


We have computed moving average. With the help of following codes we can compute profit and loss generated by strategy on past data if trader bought Infosys share when stock was trading above 30 days moving average (bought when close price crossed average prices from downside to upside) and sold when price went below 30 days moving average.

```
# Calculate trades and profits generated by strategy
Infy['positions_long'] = np.nan
for x in range (len(Infy)):
    if Infy.Close[x] > Infy.MA[x]:
        Infy['positions_long'][x] = 1
    if Infy.Close[x] <= Infy.MA[x]:
        Infy['positions_long'][x] = 0
Infy.positions_long = Infy.positions_long.fillna(method='ffill')
Infy['price_difference']= Infy.Close - Infy.Close.shift(1)
Infy['pnllong'] = Infy.positions_long.shift(1) * Infy.price_difference
Infy['cumpnl_long'] = Infy.pnllong.cumsum()
# Calculate the max drawdown in the past window days for each day
Infy['rolling_max'] = Infy['cumpnl_long'].rolling(250, min_periods=1).max()
Infy['daily_drawdown'] = Infy['cumpnl_long']-Infy['rolling_max']
# Calculate the minimum (negative) daily drawdown
Infy['max_daily_drawdown'] = Infy['daily_drawdown'].rolling(250,
min_periods=1).min()
```

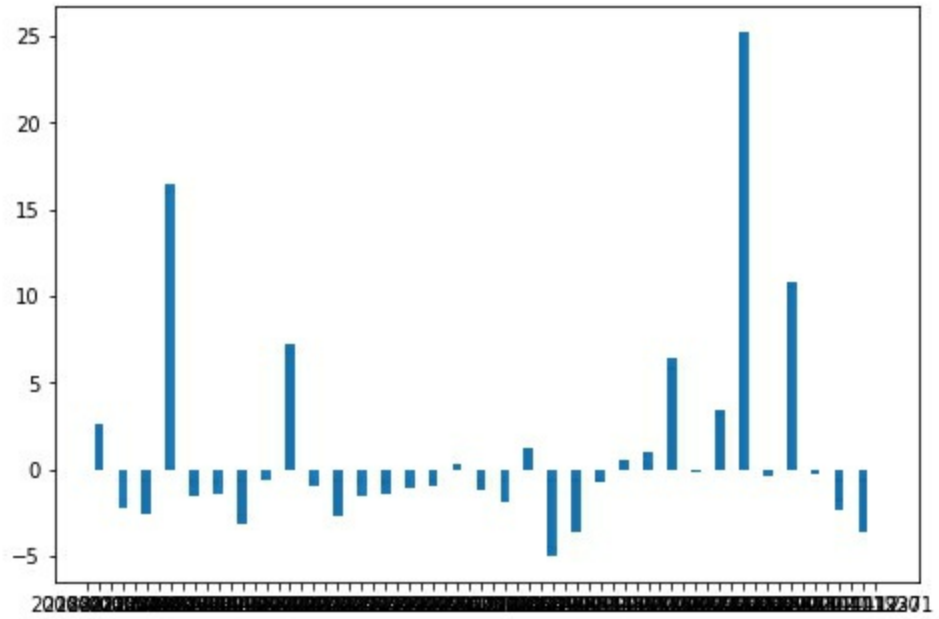
```
# Plot Cumulative retruns and maximum drawdown of strategy
fig, axs = plt.subplots(2,figsize=(15,8))
axs[0].plot(Infy['cumpnl_long'])
axs[1].plot(Infy['max_daily_drawdown'])
```

You will get the following output. As we can observe in following chart, first chart of cumulative profit and loss showing profit of approximately Rs 450/- per share generated by strategy and second chart of maximum daily drawdown is Rs 175/- per share.



With the help of following codes we can further analyse the strategy to check total numbers of trade and return on each trade generated by strategy in last 3 years from 2018 to 2021 (as we are taking 3 years data for backtesitng) -

Number of Trade count 34



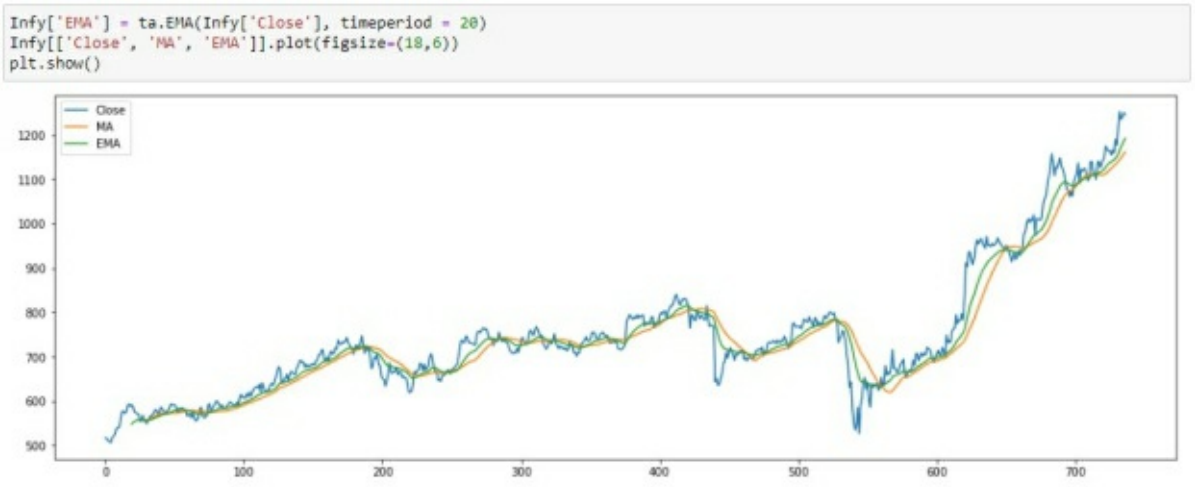
As we can observe from the above chart total 34 trades generated by strategy, 22 trades resulted into loss and 12 trades given profit. One can download files ‘Infy’ or ‘T2’ for further analysis of trade.

```
# Download file ‘Infy’ in csv format to check computations
Infy.to_csv("Infy.csv", index=True, encoding='utf8')
```

1.2. EXPONENTIAL MOVING AVERAGE (EMA)

An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points.

```
# Computation of exponential moving average using ‘TA-LIB’
Infy['EMA'] = ta.EMA(Infy['Close'],20)
Infy[['Close','MA', 'EMA']].plot(figsize=(18,6))
plt.show()
```

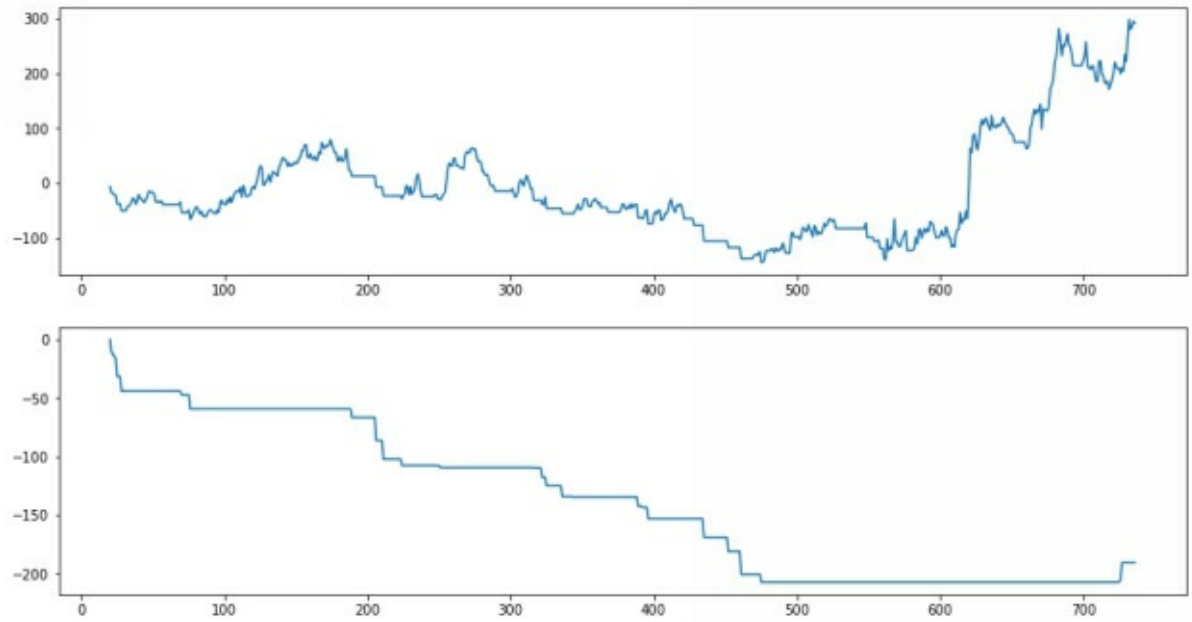


We can use the same code for computation of return generated by Exponential Moving Average as we used for moving averages-

```
# Calculate trades and profits generated by strategy
Infy['positions_long'] = np.nan
for x in range (len(Infy)):
    if Infy.Close[x] > Infy.EMA[x]:
        Infy['positions_long'][x] = 1
```

```
    if Infy.Close[x] <= Infy.EMA[x]:
        Infy['positions_long'][x] = 0
Infy.positions_long = Infy.positions_long.fillna(method='ffill')
Infy['price_difference']= Infy.Close - Infy.Close.shift(1)
Infy['pnllong'] = Infy.positions_long.shift(1) * Infy.price_difference
Infy['cumpnl_long'] = Infy.pnllong.cumsum()
# Calculate the max drawdown in the past window days for each day
Infy['rolling_max'] = Infy['cumpnl_long'].rolling(250, min_periods=1).max()
Infy['daily_drawdown'] = Infy['cumpnl_long']-Infy['rolling_max']
# Calculate the minimum (negative) daily drawdown
Infy['max_daily_drawdown'] = Infy['daily_drawdown'].rolling(250, min_periods=1).min()
# Plot Cumulative retruns and maximum drawdown of strategy
fig, axs = plt.subplots(2,figsize=(15,8))
axs[0].plot(Infy['cumpnl_long'])
axs[1].plot(Infy['max_daily_drawdown'])
```

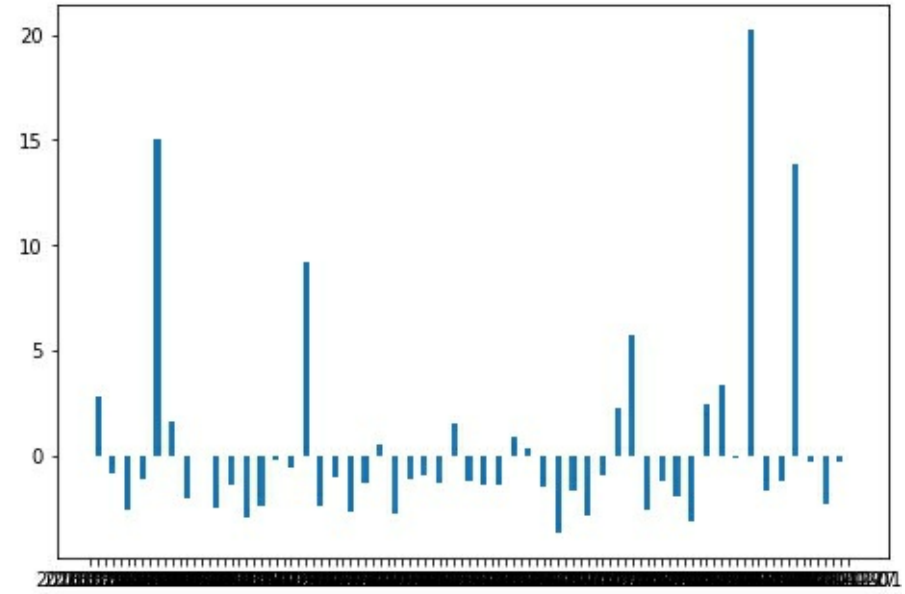
```
[<matplotlib.lines.Line2D at 0xb8e2430>]
```




```
# Calculate number of trades and return of each trade
Infy['Trade'] = Infy['positions_long'].diff()
T2 = Infy.where((Infy.Trade != 0))
T2 = T2.dropna()
T2.drop(["positions_long", "price_difference", "pnllong", "rolling_max",
```

```
"daily_drawdown", "max_daily_drawdown"], axis = 1, inplace = True)
T2['Trade_Return'] = (T2['cumpnl_long'].diff()/T2['Close'])*100
print ("Number of Trade count", round(len(T2)/2))
# Plot trade retruns
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(T2['Date'], T2['Trade_Return'])
plt.show()
```

```
Number of Trade count 52
```



1.3. BOLLINGER BANDS

Bollinger Bands are a type of statistical chart characterizing the prices and volatility over time of a financial instrument or commodity. Bollinger Bands are envelopes plotted at a standard deviation level above and below a simple moving average of the price. Because the distance of the bands is based on standard deviation, they adjust to volatility swings in the underlying price. Bollinger Bands use 2 parameters, Period and Standard Deviations. The default values are 20 for period, and 2 for standard deviations, although you may customize the combinations.

Statistically 68% values should remain within the range of +- one standard deviation from the mean value, so if price is touching upper band or lower band than they should come back to mean value but mean value is also moving up or down with the prices that's why in range bound market price reverse to mean value will be true but in case of trending market this will not be true. In a trending market price touching upper band or a lower band may be a breakout upside or downside respectively.

You can observe in following chart when price is touching the upper band the stock is in uptrend and when price is touching the lower band stock is in down trend. How to use the Bollinger band for trading will depend on the instrument and price behavior of that instrument.

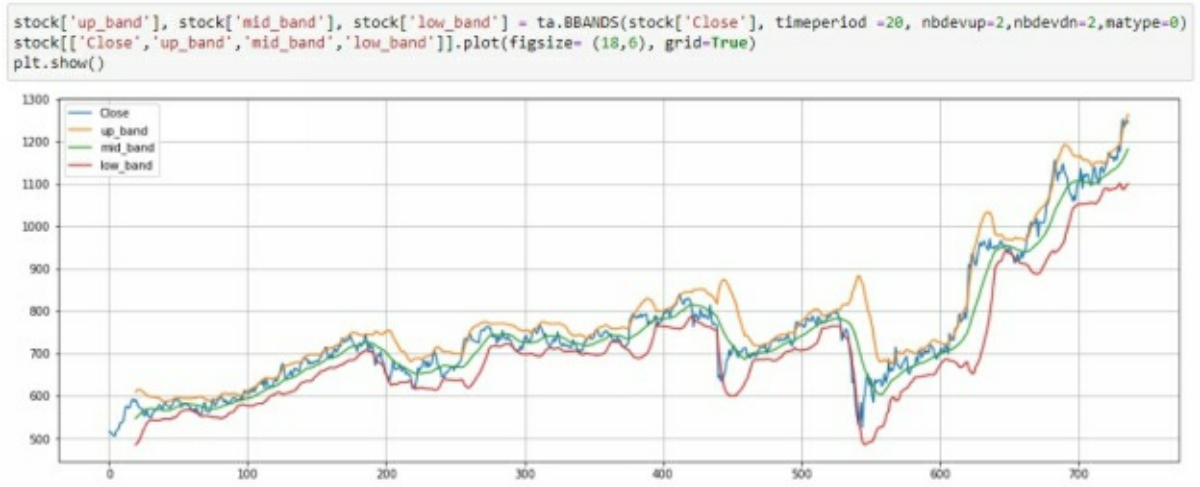
Following codes will be used for computation of Bollinger band with TA-LIB –

upper, middle, lower = BBANDS(Close Price, Time Period, Number of Standard Deviation from Mean, Moving Average Type)

In the following example we are taking time period of 20 days for computation of simple moving average and 2 standard deviation upside and 2 standard deviation downside for computation of bands.

```
# Computation of bollinder bands using 'TA-LIB'
stock['up_band'], stock['mid_band'], stock['low_band'] =
```

```
ta.BBANDS(stock['Close'], timeperiod =20,
nbdevup=2,nbdevdn=2,matype=0)
stock[['Close','up_band','mid_band','low_band']].plot(figsize= (18,6),
grid=True)
plt.show()
```

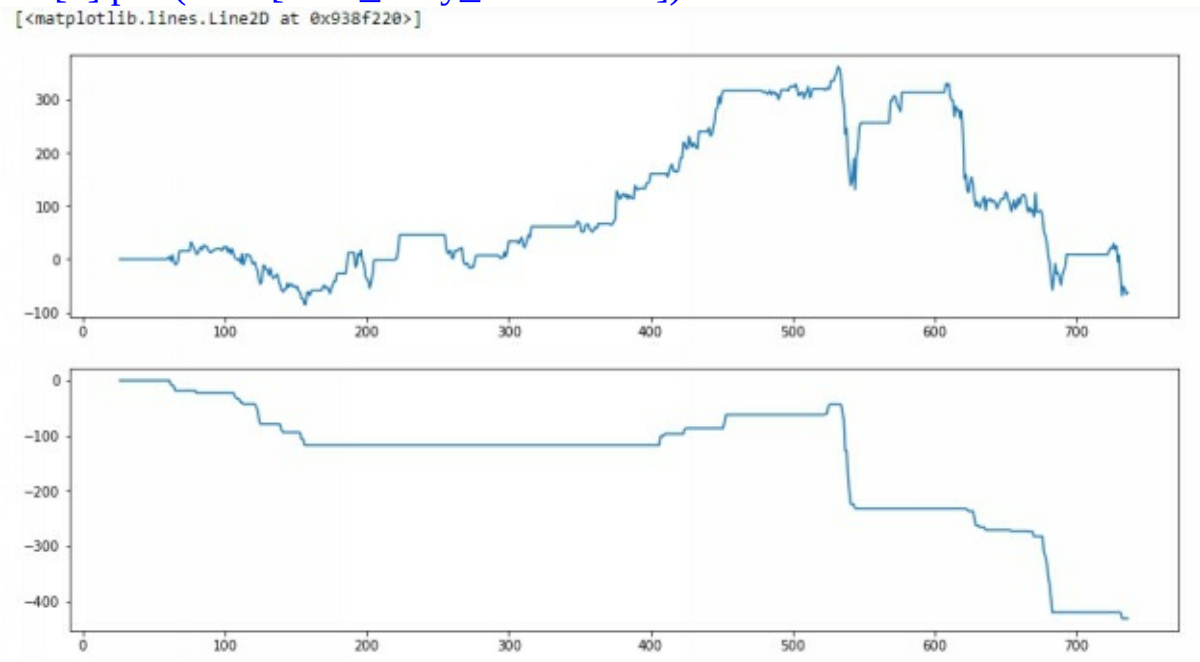


We can use the following codes for computation of return generated by Bollinger Bands. There is slight difference in python codes we used for moving averages because in case of moving averages we were taking only buy side trades, however in case of Bollinger band we buying when stock price is going below lower band and we are selling when stock price is are going above upper band in the hope that price will come back to average prices. So Bollinger band is a mean reversal strategy. Python codes are given below-

```
# Generate trade signals and compute profit loss on trades
stock['positions_long'] = np.nan
for x in range (len(stock)):
    if stock.Close[x] < stock.low_band[x]:
        stock['positions_long'][x] = 1
    if stock.Close[x] >= stock.mid_band[x]:
        stock['positions_long'][x] = 0
stock.positions_long = stock.positions_long.fillna(method='ffill')
stock['positions_short'] = np.nan
for x in range (len(stock)):
    if stock.Close[x] > stock.up_band[x]:
```

```
            stock['positions_short'][x] = -1
        if stock.Close[x] <= stock.mid_band[x]:
            stock['positions_short'][x] = 0
stock.positions_short = stock.positions_short.fillna(method='ffill')
stock['positions'] = stock.positions_long + stock.positions_short
stock['price_difference']= stock.Close - stock.Close.shift(1)
stock['pnl'] = stock.positions.shift(1) * stock.price_difference
stock['cumpnl'] = stock.pnl.cumsum()
# Calculate the max drawdown in the past window days for each day
stock['rolling_max'] = stock['cumpnl'].rolling(250, min_periods=1).max()
stock['daily_drawdown'] = stock['cumpnl']-stock['rolling_max']
# Calculate the minimum (negative) daily drawdown
stock['max_daily_drawdown']    =    stock['daily_drawdown'].rolling(250,
min_periods=1).min()
# Plot cumulative profit loss and maximum drawdown
fig, axs = plt.subplots(2,figsize=(15,8))
axs[0].plot(stock['cumpnl'])
axs[1].plot(stock['max_daily_drawdown'])
```

```
[<matplotlib.lines.Line2D at 0x938f220>]
```



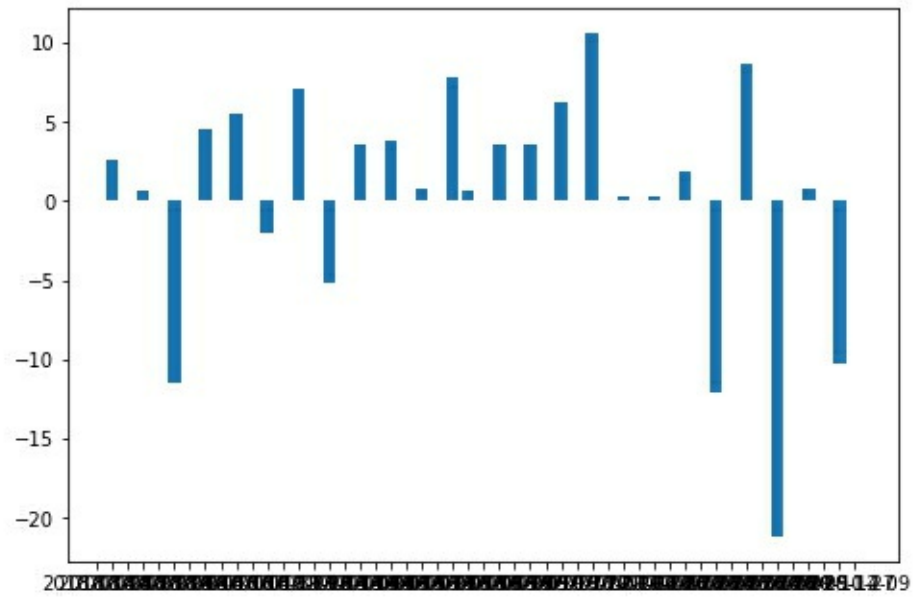
As you can observe in above output Trading Infosys with 20 days moving average and 2 standard deviation generated a negative returns and maximum drawdown was more than total returns generated by strategy.

We can do further analysis for computation of return generated by each trade

with the help of following codes-

```
# Calculate number of trades and return of each trade
stock['Trade'] = stock['positions'].diff()
T2 = stock.where((stock.Trade != 0))
T2 = T2.dropna()
T2.drop(["positions_long", "positions_short", "positions",
"price_difference", "pnl", "rolling_max", "daily_drawdown",
"max_daily_drawdown"], axis = 1, inplace = True)
T2['Trade_Return'] = (T2['cumpnl'].diff()/T2['Close'])*100
print ("Number of Trade count", round(len(T2)/2))


# Plot trade retruns
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(T2['Date'], T2['Trade_Return'])
plt.show()
```

```
Number of Trade count 25
```



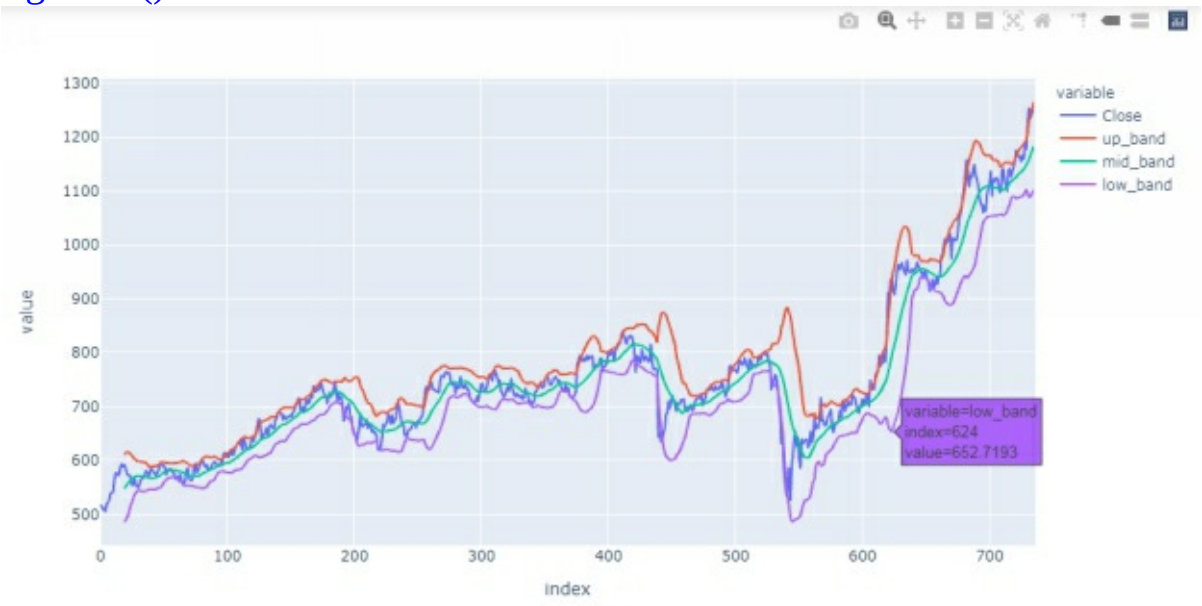
Total 25 trades generated by strategy from year 2018 to 2020. As you can observe in above chart in initial days most of the trades resulted into profit

of approx 3% to 10% in each trade when stock was range bound but when stock started trending strategy started generating losses.

We are using same data for computation of signals and returns generated by various technical indicators. If you will compare the charts of Moving Averages and Bollinger Band you will find that Bollinger bands are giving return when stocks are range bound and Moving Averages are giving return when stocks are trending.

As discussed earlier we can use ploty python library for better visualization. In the following graph I am using ploty library to view interactive chart of Bollinger band –

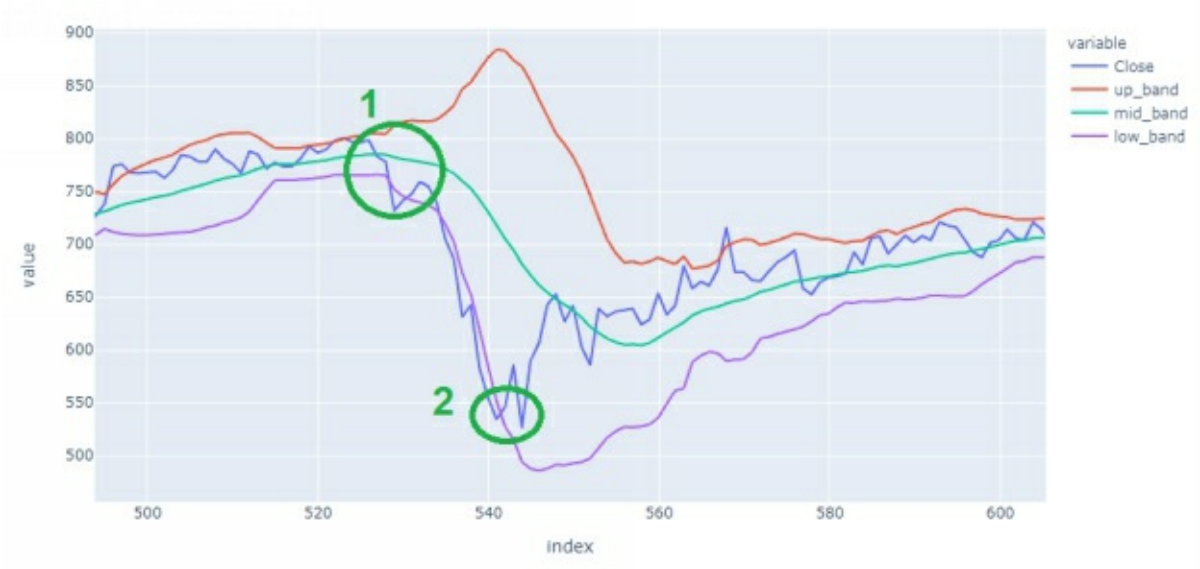
```
import plotly.express as px
fig = px.line(stock[['Close','up_band','mid_band','low_band']])
fig.show()
```



Benefit with plotly is that you can zoom in or zoom out chart at any data point. For example I can zoom in to check the clear graph from 500 to 600

data points (Index Values/trading days) of the above chart of Bollinger band. Screen shot of this zoomed in chart is given below. I also have marked 2 circles in the foll0wing chart. In the present strategy we are used in above example for trading is buying on point, as marked by circle 1, when stock price going below lower band. So strategy bought stock approx Rs 750 when price went below lower band and sold when prices came back to mean value of 650. This single trade resulted into a loss of more than 10% because prices price were trending and fall down sharply before coming back to the mean value. So we can fine-tune our strategy by buying a stock when is coming back from lower band, as marked by circle 2, inspite of buying when it's going below lower band. Program code of this strategy is given in the last chapter of this book when we are using combination of two are more indicators to generate buy sell signals.

So one can say the key benefit of learning Python is that you can adjust your strategy in way you want. You need not to depend on standard software's where you do not know how they are computing things.



2. MOMENTUM INDICATORS

Traders may use momentum indicators to:

- Identify the direction of a trend.
- Find divergences between the price and the momentum indicator in order to identify a potential trend reversal or trend continuation setup.
- Take advantage of overbought and oversold conditions.

2.1. RATE OF CHANGE (ROC)

The Price Rate of Change (ROC) is a momentum-based technical indicator that measures the change in price between the current price and the price a certain number of periods ago.

ROC above zero is sign of uptrend and below zero is the sign of down trend. A sharp increasing in value indicates increasing momentum. This signal is generally not used for trading purposes, but rather to simply alert traders that a trend change may be underway if ROC is going negative to positive territory (indication of reversal of down trend to uptrend) or positive to negative territory (Indication of reversal of uptrend to down trend).

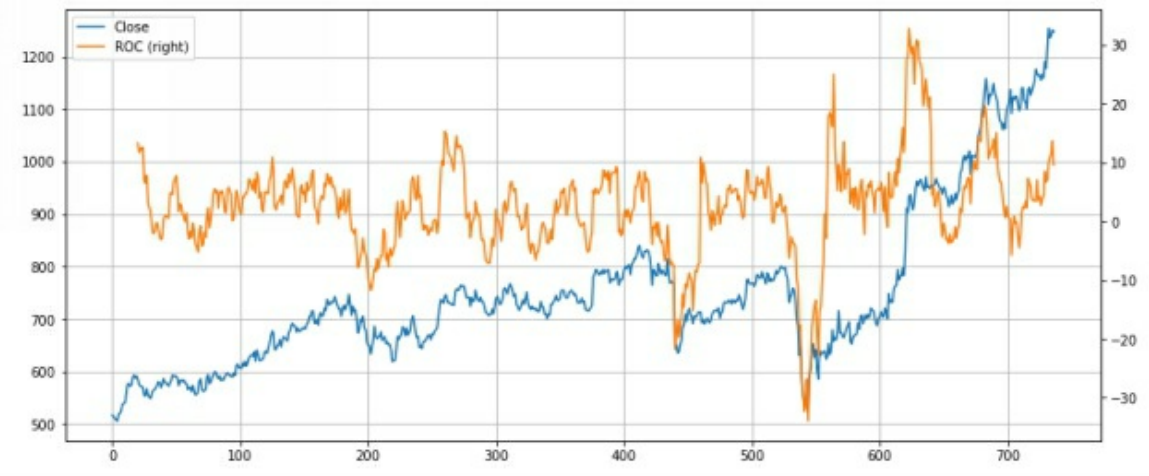
Rate of change = ((Current price/Previous Price)-1)*100

Python codes:-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
Daily_Close = pd.read_csv('Daily_Close')
Infy = pd.DataFrame({"Date": Daily_Close["Date"], "Close": Daily_Close["INFY.NS"]})
Infy['ROC'] = ta.ROC(Infy['Close'],20)
Infy[['Close', 'ROC']].plot(grid=True, secondary_y='ROC', figsize=(14,6))
```

```
Infy['ROC'] = ta.ROC(Infy['Close'],20)
Infy[['Close', 'ROC']].plot(grid=True, secondary_y='ROC', figsize=(14,6))
```

```
<AxesSubplot:>
```



As you can observe in above chart closing price plotted against primary axes of right side and ROC of 20 days is plotted against secondary axes on left side.

2.2. COMMODITY CHANNEL INDEX (CCI)

The CCI is designed to detect beginning and ending market trends. The range of 100 to -100 is the normal trading range. When the CCI is above +100, this means the price is well above the average price as measured by the indicator. When the indicator is below -100, the price is well below the average price. CCI values outside of this range indicate overbought or oversold conditions.

When a stock is in overbought zone you can sell it and when stock is in oversold zone you can buy it in expectation of reversal in prices. However a Momentum indicator can remain in a overbought zone for a long time in bull run and in oversold zone for a long time in a bear run.

You can also look for price divergence in momentum indicators. If prices are making new high but momentum indicator is not making new high then it's a sign of correction in prices. The same is true in case of CCI if the price is making new highs, and the CCI is not, then a price correction is likely.

The formula for CCI is

CCI = Typical Price – MA / 0.015 X Mean Deviation

Typical Price = High + Low + Close / 3

Python codes-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
```

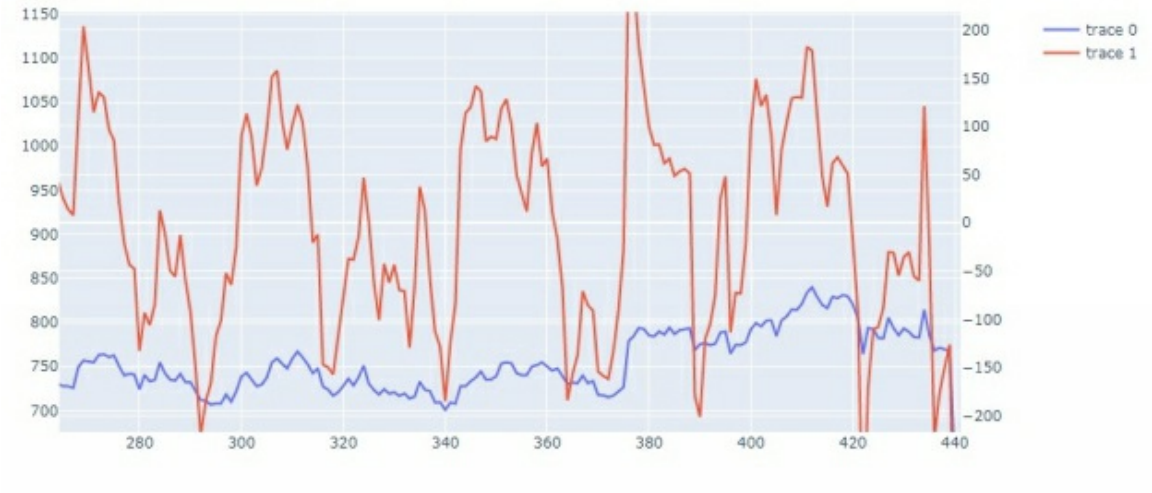
```
import yfinance as yf
data = yf.download('INFY.NS', start="2018-01-01", end="2020-12-31")
# Download file ‘data’ in csv format to reuse infosys data in other indicators
data.to_csv("Infy-His-data2018-20.csv", index=True, encoding='utf8')
# Computation of CCI
data['CCI'] = ta.CCI(data['High'], data['Low'], data['Close'], timeperiod=14)
# Plot CCI and Close price in a graph
 data[['Close', 'CCI']].plot(grid=True, secondary_y='CCI', figsize=(14,6))

 plt.axhline(100, color='red', linestyle='--')

 plt.axhline(-100, color='red', linestyle='--')

 plt.xlabel('Time')

 plt.legend(['CCI', '+-100'])

 plt.show()
```

As above chart is not much clear, we can use Python Plotly library for interactive charts with the help of following codes.

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
# Create figure with secondary y-axis
```

```
fig = make_subplots(specs=[[{"secondary_y": True}]])
# Add traces
fig.add_trace(go.Scatter(y=data['Close']),secondary_y=False,)
fig.add_trace(go.Scatter(y=data['CCI']),secondary_y=True,)
fig.show()
```

Year 2019 zoomed in the chart created with the help of above code for better clarity in following screenshot. Redline in the following chart is CCI and blue line are closing price of stock. One can observe that prices were range bound from 700 to 800. When CCI was above 100 stock came down and when CCI was below -100 stock recovered.



2.3. RELATIVE STRENGTH INDEX (RSI)

The relative strength index is a technical indicator used in the analysis of financial markets. It is intended to chart the current and historical strength or weakness of a stock or market based on the closing prices of a recent trading period.

RS = Average of x days up close / Average of x days down close

RSI = 100 – (100 / 1+RS)

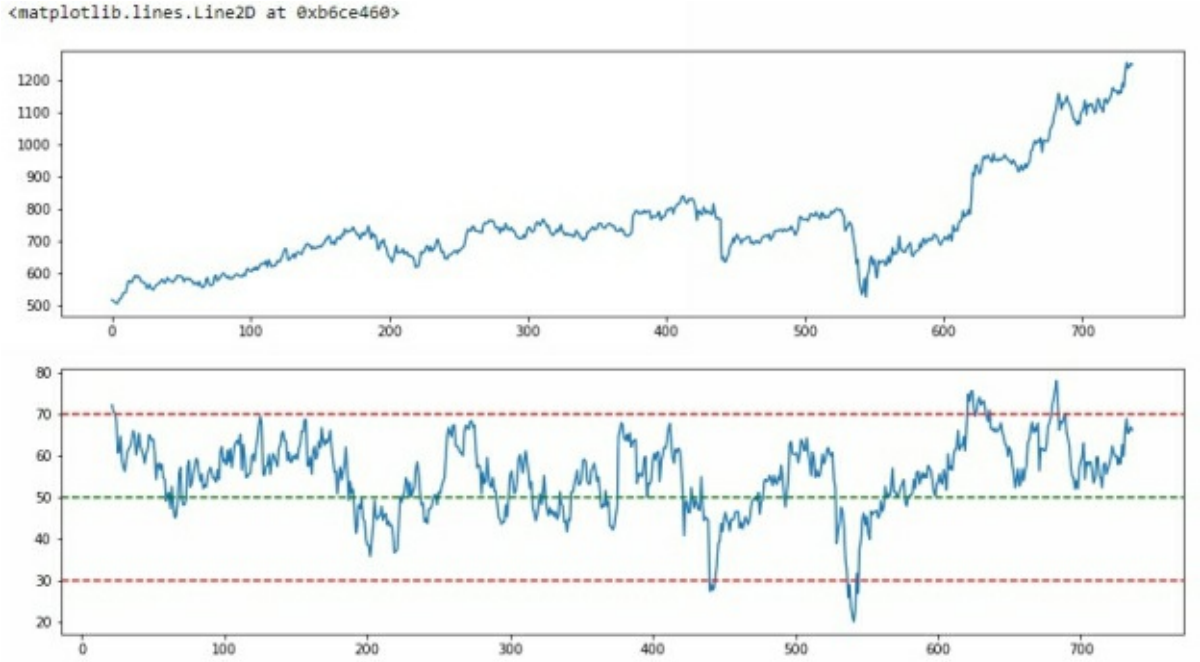
RSI is plotted on scale of 0 to 100. RSI value above 70 is considered as overbought and RSI value below 30 is considered as oversold. Some traders take crossing back above 30 line as a confirmation that trend has turned up and crossing back under 70 line as a confirmation of down trend to sell.

Python codes for RSI-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
#Import Data from file saved on system earlier
Infy = pd.read_csv('Infy-His-data2018-20.csv')
# Compute 21 days RSI in colum 'RSI' in Table 'Infy'
Infy['RSI'] = ta.RSI(Infy['Close'],21)
#print Chart
fig, axs = plt.subplots(2,figsize=(15,8))
axs[0].plot(Infy['Close'])
axs[1].plot(Infy['RSI'])
plt.axhline(70, color='red', linestyle='--')
```

```
plt.axhline(50, color='green', linestyle='--')
plt.axhline(30, color='red', linestyle='--')
```

```
<matplotlib.lines.Line2D at 0xb6ce460>
```



Limitation with RSI is that in a bull market RSI main remain above 70 or in a bear market RSI main remain below 30 for a long period of time so buying when oversold or selling when overbought can give you losses. You can also observe in above chart when 21 days RSI was above 50 stock was in uptrend and when RSI was below 50 stock was in downtrend.

As we discussed earlier also you can also look for price divergence in momentum indicators. RSI indicator can also be used to predict a divergence in the trend before the price trend actually reverses. Divergence can usually be spotted if the price line is moving higher but the RSI indicator slumps due to the fact that the relative strength of the asset weakens when compared to the previous periods' growth. This is an indication that prices will go down. The same is true when the closing price has been bearish for a while but the RSI starts posting higher values, it means that the prices will pick up.

2.4. MOVING AVERAGE CONVERGENCE/DIVERGENCE (MACD)

MACD is based on the point spread difference between two exponential moving averages of the closing price. The MACD indicator is created by subtracting a longer-term exponential moving average from a shorter-term exponential moving average. Usually we take 26 days slower moving average and 12 days faster moving average. This difference is further smoothed by and even faster exponential moving average (usually 9 periods) which is called signal line. MACDs rely on three exponential moving averages instead or one or two.

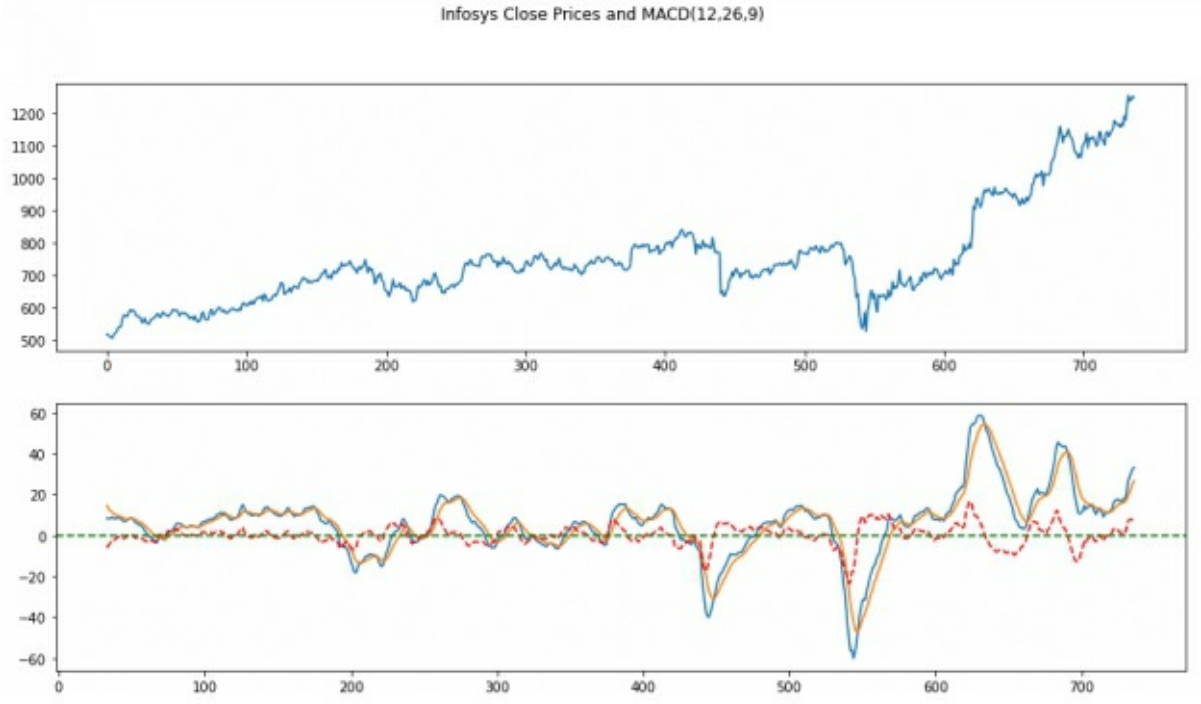
MACD generally rises if shorter-term trends are gaining strength and generally declines if shorter-term trends are losing strength. Buy shares when MACD line crosses the signal line upwards (it is considered bullish signal) on the other hand sell shares when MACD line crosses the signal line downwards (it is considered bearish signal). When MACD is extremely high it indicates top has been made and when MACD is extremely low it indicates bottom has been made.

Following command can be used for computation of MACD in TA-Lib-
macd, macdsignal, macdhist = MACD(close, fastperiod=12, slowperiod=26, signalperiod=9)

Ptyhon Codes-

```
#Import Necessary Libraries
import talib as ta
import pandas as pd
```

```
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')
# Compute MACD
Infy['macd'], Infy['macdsignal'], Infy['macdhist'] = ta.MACD(Infy['Close'],
fastperiod=12, slowperiod=26, signalperiod=9)
#Plot Chart
fig1, ax = plt.subplots(2, figsize=(15,8))
ax[0].plot(Infy['Close'])
ax[1].plot(Infy['macd'])
ax[1].plot(Infy['macdsignal'])
ax[1].plot(Infy['macdhist'], color='red', linestyle='--')
plt.axhline(0, color='green', linestyle='--')
plt.suptitle('Infosys Close Prices and MACD(12,26,9)')
plt.show()
```

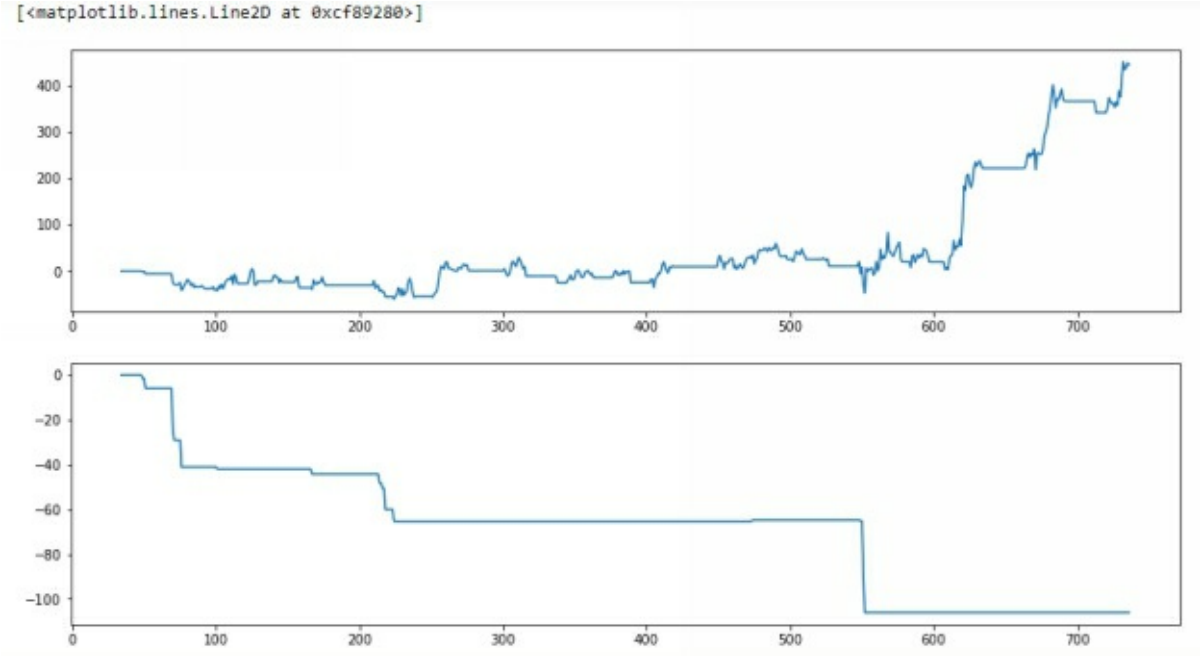


In the following code we are computing return generated by buy and hold strategy on past 3 years data of Infosys. We are buying Infosys when MACD line crossing MACD Signal line upwards and holding position

until MACD line crossing MACD Signal line downwards –

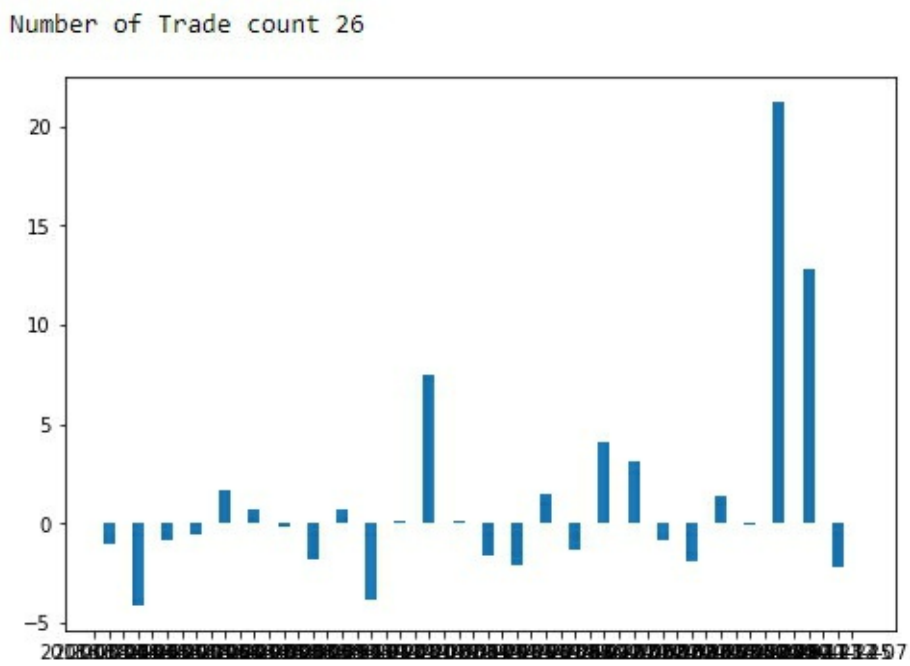
Calculate trades and profits generated by strategy Buying Infosys when MACD is crosses MACDSignal line upwards and hold position until it crosses downwards

```
Infy['positions_long'] = np.nan
for x in range (len(Infy)):
    if Infy.macd[x] > Infy.macdsignal[x]:
        Infy['positions_long'][x] = 1
    if Infy.macd[x] <= Infy.macdsignal[x]:
        Infy['positions_long'][x] = 0


Infy.positions_long = Infy.positions_long.fillna(method='ffill')
Infy['price_difference']= Infy.Close - Infy.Close.shift(1)
Infy['pnllong'] = Infy.positions_long.shift(1) * Infy.price_difference
Infy['cumpnl_long'] = Infy.pnllong.cumsum()


# Calculate the max drawdown in the past window days for each day
Infy['rolling_max'] = Infy['cumpnl_long'].rolling(250, min_periods=1).max()
Infy['daily_drawdown'] = Infy['cumpnl_long']-Infy['rolling_max']
# Calculate the minimum (negative) daily drawdown
Infy['max_daily_drawdown'] = Infy['daily_drawdown'].rolling(250, min_periods=1).min()
# Plot Cumulative retruns and maximum drawdown of strategy
fig, axs = plt.subplots(2,figsize=(15,8))
axs[0].plot(Infy['cumpnl_long'])
axs[1].plot(Infy['max_daily_drawdown'])
```



As we can observe in above charts, strategy generated a profit of approximately Rs 400/- per share and maximum drawdown was Rs 100/- per share. With the help of following codes we are computing total trades and return from each trade-

```
# Calculate number of trades and return of each trade
Infy['Trade'] = Infy['positions_long'].diff()
T2 = Infy.where((Infy.Trade != 0))
T2 = T2.dropna()
T2.drop(["positions_long", "price_difference", "pnllong", "rolling_max", "daily_drawdown", "max_daily_drawdown"], axis = 1, inplace = True)
T2['Trade_Return'] = (T2['cumpnl_long'].diff()/T2['Close'])*100
print ("Number of Trade count", round(len(T2)/2))
# Plot trade retruns
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(T2['Date'], T2['Trade_Return'])
plt.show()
```



As we can observe in above charts, total 26 trades generated by strategy and return from each trade was approximately -5% to 20%.

2.5. BALANCE OF POWER (BOP)

Balance of Power (BOP) is an oscillator that measures the strength of buying and selling pressure. The Balance of Power indicator measures the market strength of buyers against sellers by assessing the ability of each side to drive prices to an extreme level.

Balance of Power = (Close price – Open price) / (High price – Low price)

Balance of Power could be used to generate trading signals on the crossovers with its center line. A simple trading system based on the Balance of Power indicator would suggest:

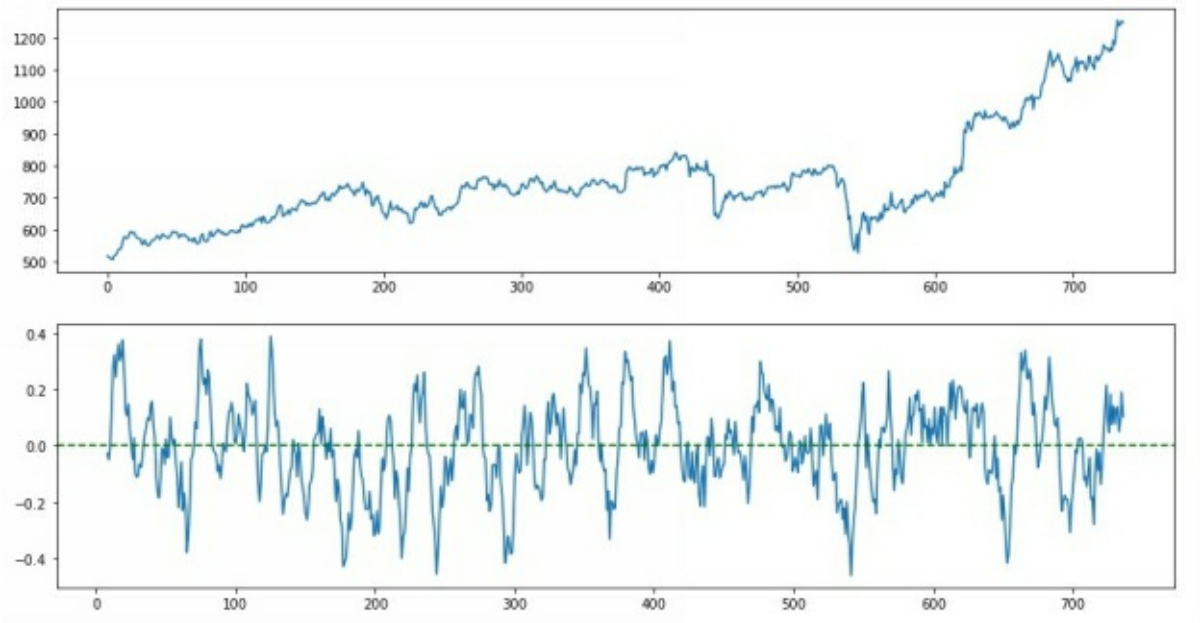
- buy when BOP crosses above zero line - becomes positive;
- sell when BOP crosses below zero line - becomes negative.

The resulting value can be smoothed by a moving average. The level or center line in BOP represents a stock's accumulation above or below its zero line. When the indicator is in positive territory, the bulls are in charge; and sellers dominate when the indicator is negative. A reading near the zero line indicates a balance between the two and can mean a trend reversal.

Python Codes-

```
#Download necessary libraries
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
#Import data from file saved on system
```

```
Infy = pd.read_csv('Infy-His-data2018-20.csv')
#Calculate Balance of Power
Infy['BoP'] = ta.BOP(Infy['Open'], Infy['High'], Infy['Low'], Infy['Close'])
# BOP value smoothed by a 9 days moving average
Infy['BoP_MA'] = ta.SMA(Infy['BoP'],9)
#Plot BoP
fig1, ax = plt.subplots(2, figsize=(15,8))
ax[0].plot(Infy['Close'])
ax[1].plot(Infy['BoP_MA'])
plt.axhline(0, color='green', linestyle='--')
plt.suptitle('Infosys Close Prices and Balance of Power Technical Indicator')
plt.show()
```



2.6. STOCHASTIC

It is based on observation that as prices increases, closing prices tend to be closer to the upper end of price range. Same in case of downtrend the closing prices tend to be closer to the lower end of price range.

%K = (C-L) / (H-L) X 100

%K = Stochastic

C = Latest Close Price

L = Low price during last N Periods

H = High price during last N prriods

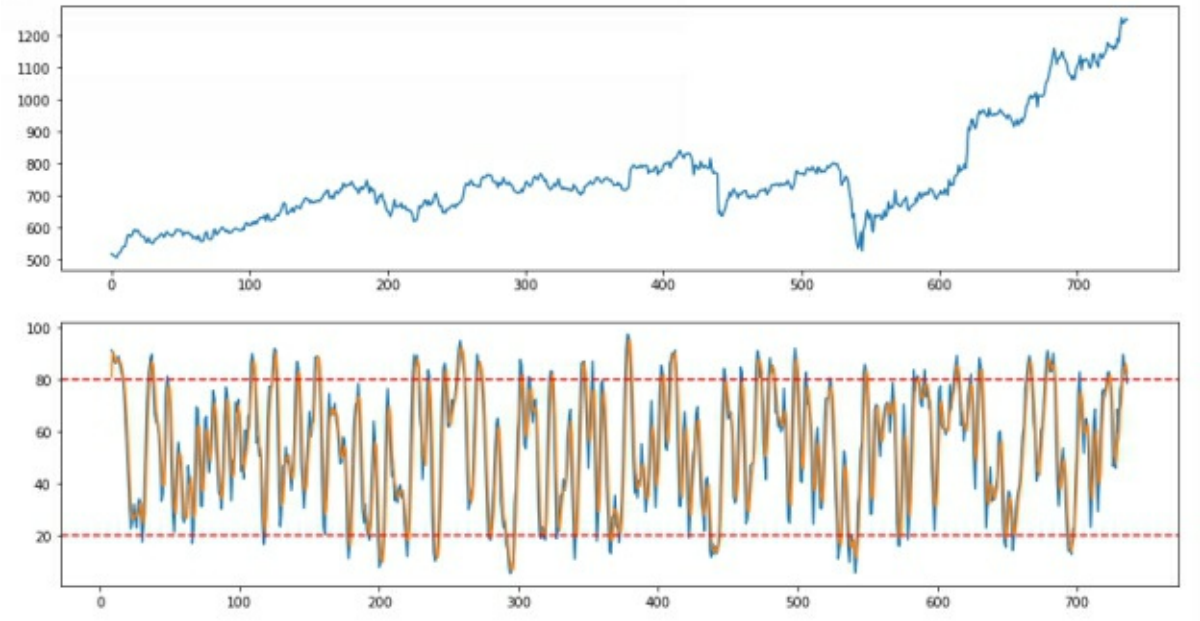
%D = 3 days simple moving average of %K

The TA-Lib Stochastic function returns two lines slowk and slowd which can then be used to generate the buy/sell indicators. A crossover signal occurs when the two lines cross in the overbought region (commonly above 80) or oversold region (commonly below 20). When a slowk line crosses below the slowd line in the overbought region it is considered a sell indicator. Conversely, when an increasing slowk line crosses above the slowd line in the oversold region it is considered a buy indicator.

Python codes-

```
#Download necessary libraries
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
#Import data from file saved on system
```

```
Infy = pd.read_csv('Infy-His-data2018-20.csv')
#Compute ‘slowk’ and ‘slowd’ values
Infy['slowk'], Infy['slowd'] = ta.STOCH(Infy['High'], Infy['Low'], Infy['Close'], fastk_period=5, slowk_period=3, slowk_matype=0, slowd_period=3, slowd_matype=0)
#Plot Values
fig1, ax = plt.subplots(2, figsize=(15,8))
ax[0].plot(Infy['Close'])
ax[1].plot(Infy['slowk'])
ax[1].plot(Infy['slowd'])
plt.axhline(20, color='red', linestyle='--')
plt.axhline(80, color='red', linestyle='--')
plt.suptitle('Stochastic')
plt.show()
```




```
#Generate Buy and sell signals
Infy['Signal_Sell'] = ((Infy['slowk'] < Infy['slowd']) & (Infy['slowk'].shift(1) > Infy['slowd'].shift(1))) & (Infy['slowd'] > 80)
Infy['Signal_Buy'] = ((Infy['slowk'] > Infy['slowd']) & (Infy['slowk'].shift(1) < Infy['slowd'].shift(1))) & (Infy['slowd'] < 20)
Infy.tail()
```

```
Infy['Signal_Sell'] = ((Infy['slowk'] < Infy['slowd']) & (Infy['slowk'].shift(1) > Infy['slowd'].shift(1))) & (Infy['slowd'] > 80
Infy['Signal_Buy'] = ((Infy['slowk'] > Infy['slowd']) & (Infy['slowk'].shift(1) < Infy['slowd'].shift(1))) & (Infy['slowd'] < 20)
Infy.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume	slowk	slowd	Signal_Sell	Signal_Buy
732	2020-12-23	1238.000000	1258.849976	1230.550049	1253.050049	1253.050049	15878346	77.904418	71.058521	False	False
733	2020-12-24	1249.900024	1249.900024	1226.000000	1236.050049	1236.050049	7313885	89.518461	81.144616	False	False
734	2020-12-28	1238.449951	1248.000000	1236.000000	1240.300049	1240.300049	4607051	85.284078	84.235652	False	False
735	2020-12-29	1235.000000	1254.449951	1235.000000	1250.300049	1250.300049	6878105	83.650215	86.150918	True	False
736	2020-12-30	1253.000000	1253.300049	1238.150024	1246.800049	1246.800049	5194690	78.539071	82.491121	False	False

2.7. STOCHASTIC RELATIVE STRENGTH INDEX -

The stochastic RSI is a technical indicator used to measure the strength and weakness of the relative strength indicator (RSI) over a set period of time.

Stochastic RSI = RSI – min[RSI] / max[RSI] – min[RSI]

RSI = Current Relative Strength Index

min[RIS] = Lowest RSI reading over last N period

max[RSI] = Highest RSI reading over last N period

Stochastic RSI ranges between zero and 100. Reading above 80 is considered as overbought and reading below 20 is considered as oversold. A reading of zero means the RSI is at its lowest level in 14 periods (or whatever lookback period is chosen). A reading of 1 (or 100) means the RSI is at the highest level in the last 14 periods.

One downside to using the Stochastic RSI is that it tends to be quite volatile, rapidly moving from high to low. Smoothing with 10 days simple moving average may help in this regard.

RSI is more useful when stock is trending whereas Stochastic RSI is more useful in sideways.

Python codes-

```
#Download necessary libraries
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
```

```
from datetime import datetime
import numpy as np
#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')
#Compute ‘fastk’ and ‘fastd’ values
Infy['fastk'], Infy['fastd'] = ta.STOCHRSI(Infy['Close'],
timeperiod=14, fastk_period=5, fastd_period=3, fastd_matype=0)
#Plot Values
fig1, ax = plt.subplots(2, figsize=(15,8))
ax[0].plot(Infy['Close'])
ax[1].plot(Infy['fastk'])
ax[1].plot(Infy['fastd'])
plt.axhline(20, color='red', linestyle='--')
plt.axhline(80, color='red', linestyle='--')
plt.suptitle('Stochastic RSI')
plt.show()
```

3. VOLUME INDICATORS

Volume is one piece of information that is often neglected by many market players, especially the beginners. However, learning to interpret volume brings many advantages and could be of tremendous help when it comes to analyzing the markets. Volume plays a very integral role in technical analysis as it helps us to confirm trends and patterns. When institutional investors buy or sell, they obviously do not transact in small chunks. Rise in volume with rise in price confirm the uptrend and rise in volume with fall in price confirm the downtrend.

3.1. CHAIKIN A/D OSCILLATOR

Chaikin Accumulation Distribution Line is a volume-based indicator designed to measure the cumulative flow of money into and out of a security. The Chaikin Oscillator is the difference between the 3-day and 10-day EMAs of the Accumulation Distribution Line. A momentum change is the first step to a trend change. Anticipating trend changes in the Accumulation Distribution Line can help chartists anticipate trend changes in the underlying security.

Money Flow Multiplier = [(Close - Low) - (High - Close)] /(High - Low)

Money Flow Volume = Money Flow Multiplier x Volume for the Period

ADL = Previous ADL + Current Period's Money Flow Volume

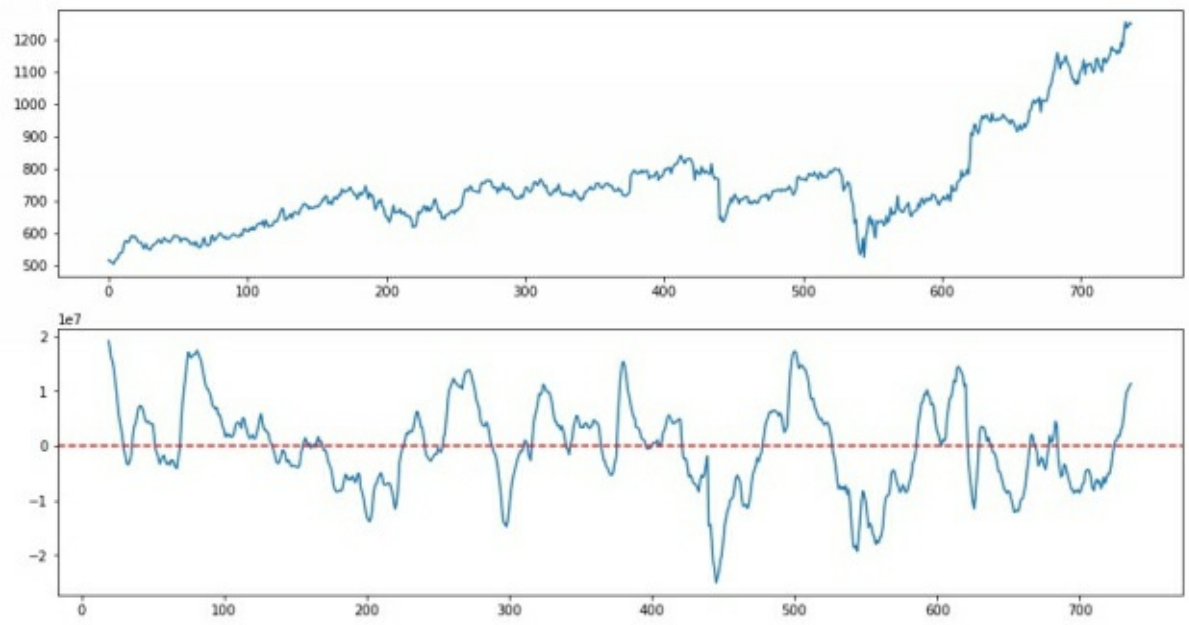
Chaikin Oscillator = (3-day EMA of ADL) - (10-day EMA of ADL)

A move into positive territory indicates that the Accumulation Distribution Line is rising and buying pressure prevails. A move into negative territory indicates that the Accumulation Distribution Line is falling and selling pressure prevails.

Python code-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
```

```
#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')
#Compute Chaikin A/D Oscillator
Infy['C_A/D_Osc'] = ta.ADOSC(Infy['High'], Infy['Low'],
Infy['Close'], Infy['Volume'], fastperiod=6, slowperiod=20)
#Plot Graph
fig1, ax = plt.subplots(2, figsize=(15,8))
ax[0].plot(Infy['Close'])
ax[1].plot(Infy['C_A/D_Osc'])
plt.axhline(0, color='red', linestyle='--')
plt.show()
```




```
#Plot Graph with plotly
import plotly.graph_objects as go
from plotly.subplots import make_subplots
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])
# Add traces
fig.add_trace(
    go.Scatter(y=Infy['Close']),
    secondary_y=False,
)
fig.add_trace(
```

```
        go.Scatter(y=Infy['C_A/D_Osc']),
        secondary_y=True,
    )
    fig.show()
```

3.2. ON BALANCE VOLUME (OBV)

On Balance Volume (OBV) measures buying and selling pressure as a cumulative indicator that adds volume on up days and subtracts volume on down days. When the security closes higher than the previous close, all of the day's volume is considered up-volume. When the security closes lower than the previous close, all of the day's volume is considered down-volume.

If the OBV is rising, accumulation may be taking place, a warning of an upward breakout. If the OBV is falling, distribution may be taking place, a warning of a downward breakout. When both price and OBV are making higher peaks the upward trend is likely to continue. When both price and OBV are making lower peaks the downward trend is likely to continue.

When price continues to make higher peaks and OBV fails to make higher peaks, the upward trend is likely to stall or fail. This is called a negative divergence. When price continues to make lower troughs and OBV fails to make lower troughs, the downward trend is likely to stall or fail. This is called a positive divergence.

Let's take an example to understand-

Day	Price	Volume	OBV
1	100	100	100
2	101	150	250
3	100	120	130

Day 1 – OBV 100

If today's close is equal to yesterday's close then:

OBV = Yesterday’s OBV

Day 2 – OBV 250

If today's close is greater than yesterday's close then:

OBV (250) = Yesterday’s OBV (100) + Today’s Volume (150)

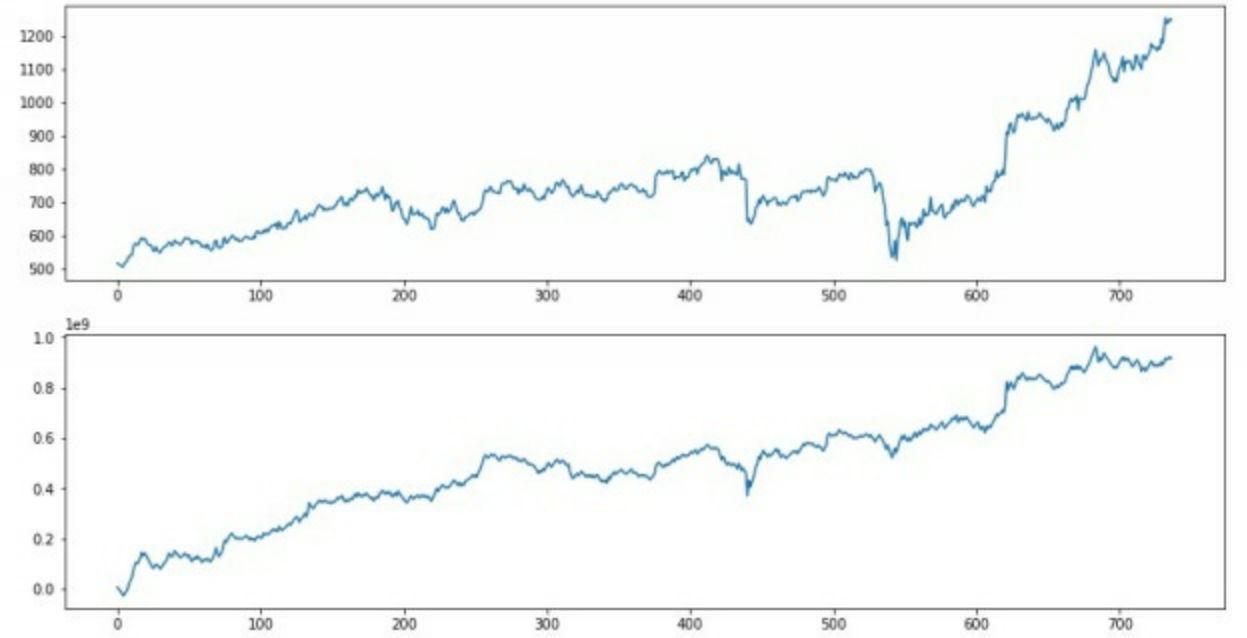
Day 3 – OBV 130

If today’s close is less than yesterday’s close then:

OBV (130) = Yesterday’s OBV (250) – Today’s Volume (120)

Python Code-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')
#Compute OBV
Infy['OBV'] = ta.OBV(Infy['Close'], Infy['Volume'])
#Plot close price and OBV
fig1, ax = plt.subplots(2, figsize=(15,8))
ax[0].plot(Infy['Close'])
ax[1].plot(Infy['OBV'])
plt.show()
```


1200
1100
1000
900
800
700
600
500
0
100
200
300
400
500
600
700
1e9
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600
700

4. VOLATILITY INDICATORS

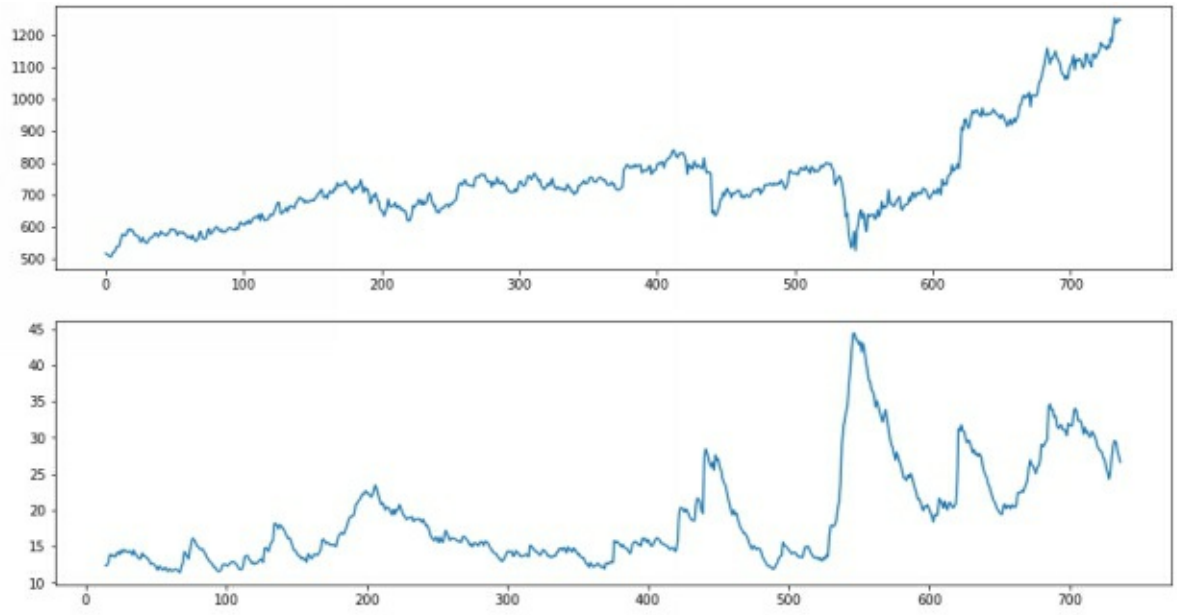
4.1 AVERAGE TRUE RANGE (ATR)

Average True Range or ATR is a measurement of volatility. It measures the average of true price ranges over time. If system is giving me a ATR number of $1.75 for XYZ stock based on 15 days parameter, then it means that the stock moved an average of $1.75 per day over the past 15 days. The indicator does not indicate the price direction; rather it is used primarily to measure volatility.

High ATR values often occur at market bottoms or at market tops following a panic sell-off or aggressive buying. Low ATR values are often found during extended sideways movements or consolidation periods.

Python Code-

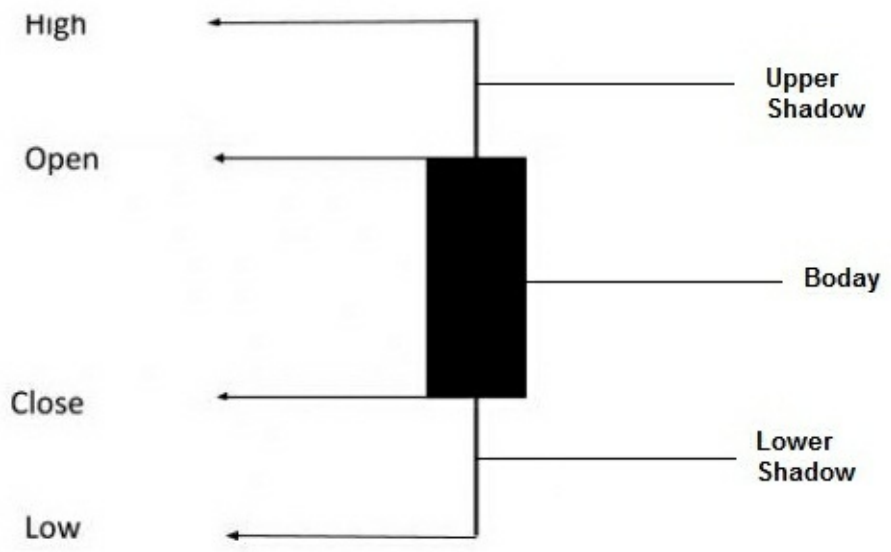
```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')
#Compute ATR
Infy['ATR'] = ta.ATR(Infy['High'], Infy['Low'], Infy['Close'], timeperiod=14)
#Plot Close price and ATR
fig1, ax = plt.subplots(2, figsize=(15,8))
ax[0].plot(Infy['Close'])
ax[1].plot(Infy['ATR'])
plt.show()
```


1200
1100
1000
900
800
700
600
500
0
100
200
300
400
500
600
700

45
40
35
30
25
20
15
10
0
100
200
300
400
500
600
700

5. PATTERN RECOGNITION

In a candle chart there are several horizontal bars or candles that form the chart. Each candle has three parts Upper Shadow, Body and Lower Shadow. In a bar the high, low, open and close prices are plotted. The body is colored either Red or Green. Bar will be green if close price is higher than open price and Bar will be red if close price is lower than open price.
Vertical line connects high and low prices.



More than 50 candlestick patterns can be identified with the help of TA-Lib. TA-Lib creates individual column for each pattern. While 0 corresponds to no pattern, positive values represent bullish patterns and negative values represent bearish patterns. Some of the patterns explained in this chapter, for more pattern you can refer TA-Lib.

5.1. **Marubozu** – Marubozu as a candlestick with no upper and lower

shadow. The red candle represents, selling on every price point, the bearish marubozu. The blue represents, buying on every price point, the bullish marubozu.

One can identify Marubozu pattern with the help of following codes.

Python Codes-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np


#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')
#Compute Marubozu
Infy['Marubozu'] = ta.CDLMARUBOZU(Infy['Open'], Infy['High'], Infy['Low'], Infy['Close'])
#Plot Marubozu Signals with candle chart
from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = make_subplots(rows=2, cols=1)
fig.append_trace(go.Candlestick(x=Infy['Date'], open=Infy['Open'], high=Infy['High'], low=Infy['Low'], close=Infy['Close'],
), row=1, col=1)
fig.append_trace(go.Scatter(
    x=Infy['Date'],
    y=Infy['Marubozu'],
), row=2, col=1)
fig.update_layout(height=800, width=1000, title_text="Marubozu")
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```

If you will observe the above output you will find that chart is plotted with 2 sub-plots. In first subplot we have can Infosys price with candle chart and in second subplot we have Marubozu signals. As you can observe second Marubozu signal was generated on 17th December 2018. Value is -100 that is why this is a sell signal. In the first plot we can zoom in to get a clear view of candle pattern and price direction after signal. View of 1st subplot after zoom in is given below-

5.2. **Morning Star** -The morning star candlestick pattern is considered a sign of hope in a bleak market downtrend. It is a three-stick pattern: one short-bodied candle between a long red and a long green. Traditionally, the ‘star’ will have no overlap with the longer bodies, as the market gaps both on open and close. It signals that the selling pressure of the first day is subsiding, and a bull market is on the horizon.

Python Codes-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np


#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')


#Generate Signals
Infy['Morning Star'] = ta.CDLMORNINGSTAR(Infy['Open'],
Infy['High'], Infy['Low'], Infy['Close'], penetration=0)
#Plot Signal and price with candle chart
from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = make_subplots(rows=2, cols=1)
fig.append_trace(go.Candlestick(x=Infy['Date'],      open=Infy['Open'],
high=Infy['High'], low=Infy['Low'], close=Infy['Close'],
), row=1, col=1)
fig.append_trace(go.Scatter(
    x=Infy['Date'],
    y=Infy['Morning Star'],
), row=2, col=1)
fig.update_layout(height=800, width=1000, title_text="Morning Star")
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```

5.3. **Evening Star** - The evening star is a three-candlestick pattern that is the equivalent of the bullish morning star. It is formed of a short candle sandwiched between a long green candle and a large red candlestick.

It indicates the reversal of an uptrend, and is particularly strong when the third candlestick erases the gains of the first candle.

Python Codes-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np


#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')


#Generate Signals
Infy['Evening Star'] = ta.CDLEVENINGSTAR(Infy['Open'],
```

```
Infy['High'], Infy['Low'], Infy['Close'], penetration=0)
#Plot Signal and price with candle chart
from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = make_subplots(rows=2, cols=1)
fig.append_trace(go.Candlestick(x=Infy['Date'],      open=Infy['Open'],
high=Infy['High'],low=Infy['Low'], close=Infy['Close'],
), row=1, col=1)
fig.append_trace(go.Scatter(
    x=Infy['Date'],
    y=Infy['Evening Star'],
), row=2, col=1)
fig.update_layout(height=800, width=1000, title_text="Evening Star")
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```

5.4. **Hammer** - This is a candle with a short body and a long lower wick. It is usually located at the bottom of a downward trend. It indicates that despite selling pressures, a strong buying surge pushed the prices up.

Python Codes-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np


#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')


#Generate Signals
Infy['Hammer'] = ta.CDLHAMMER(Infy['Open'], Infy['High'], Infy['Low'], Infy['Close'])
#Plot Signal and price with candle chart
from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = make_subplots(rows=2, cols=1)
fig.append_trace(go.Candlestick(x=Infy['Date'], open=Infy['Open'], high=Infy['High'], low=Infy['Low'], close=Infy['Close'],
), row=1, col=1)
fig.append_trace(go.Scatter(
    x=Infy['Date'],
    y=Infy['Hammer'],
), row=2, col=1)
fig.update_layout(height=800, width=1000, title_text="Hammer")
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```

5.5. **Inverted hammer** - A similarly bullish pattern is the inverted hammer. The only difference being that the upper wick is long, while the lower wick is short.

Python Codes-

```
import talib as ta
import pandas as pd
```

```
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np

#Import data from file saved on system
Infy = pd.read_csv('Infy-His-data2018-20.csv')

#Generate Signals
Infy['Inv_Hammer']    =    ta.CDLINVERTEDHAMMER(Infy['Open'],
Infy['High'], Infy['Low'], Infy['Close'])
#Plot Signal and price with candle chart
from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = make_subplots(rows=2, cols=1)
fig.append_trace(go.Candlestick(x=Infy['Date'],
                                open=Infy['Open'], high=Infy['High'],
                                low=Infy['Low'], close=Infy['Close'],
), row=1, col=1)
fig.append_trace(go.Scatter(
    x=Infy['Date'],
    y=Infy['Hammer'],
), row=2, col=1)
fig.update_layout(height=800, width=1000, title_text="Hammer")
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```


trace 0
trace 1
1200
1000
800
600
Jan 2018
Jul 2018
Jan 2019
Jul 2019
Jan 2020
Jul 2020
100
80
60
40
20
0
Jan 2018
Jul 2018
Jan 2019
Jul 2019
Jan 2020
Jul 2020

TRADING SETUPS

You can develop your own trading system based on your own idea. Technical Analysis is a science once can tech you technical indicators, its merits, demerits. However use of these indicators for prediction of trend/prices is an art. There could be thousands of permutations and combinations of these technical indicators for prediction of buy or sell signals. Our objective is develop a mechanical system of trading that is giving good return on past data and expected to perform in future based on assumptions that past price behavior and pattern will continue in future. Let's try to develop of profitable system. First you need to have a basic idea, what exactly you want to do. Then you need to identify technical tools which you want to use for execution of your idea. Back-test these technical tools on past data. Apply in trading if you are satisfied with results.

Basic Idea – I want to ride the trend, so my objective is to find stocks which are in uptrend and buying when momentum starts.

Strategy and Analysis – It is assumed that if a stock is trading above 200 moving average is in long term uptrend. So I am selecting a stock only when they are in long term uptrend. One can use the Oscillator to discover short-term overbought or oversold conditions. So I am using RSI to predict short term trend. I will use 21 days RSI for generating buy and sell signal. So my strategy is to buy stock that is trading above 200 days moving average when 21 days RSI is above 50% and sell when its below 50%.

Strategy – Buy a stock in a long term uptrend based on short term buying signal.

1. Stock must be trading above 200 days moving average.

2. Buy when 21 days RSI is above 70% and cover your long position if RSI goes below 50%.

Coding and Analysis for back testing – In the following code we are computing returns generated by 8 different stocks from different sectors to check the performance of strategy.

Python Codes-

```
import talib as ta
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
import yfinance as yf
Stock = ['RELIANCE.NS', 'HDFC.NS', 'SBIN.NS', 'LT.NS', 'TATAMOTORS.NS', 'ICICIBANK.NS', 'NESTLEIND.NS', 'INFY.NS']
data = yf.download(Stock, start="2010-01-01", end="2021-02-28")
close = data['Close']
close.to_csv("Daily_Close", index=True, encoding='utf8')
Daily_Close = pd.read_csv('Daily_Close')
Stocks = ['RELIANCE.NS', 'HDFC.NS', 'SBIN.NS', 'LT.NS', 'TATAMOTORS.NS', 'ICICIBANK.NS', 'NESTLEIND.NS', 'INFY.NS']
import warnings
warnings.filterwarnings("ignore")
Table = pd.DataFrame({"Stock":['0'], "Close":0, "Total Profit":0, "Percent_Return":0, "Max_Drawdown":0, "Number_Trades":0, "Max_Return":0, "Min_Return":0})
Table1 = pd.DataFrame({"Stock":['0'], "Close":0, "Total Profit":0, "Percent_Return":0, "Max_Drawdown":0, "Number_Trades":0, "Max_Return":0, "Min_Return":0})
for y in range (len(Stocks)):
    #Get Historical data
    Infy = pd.DataFrame({"Date": Daily_Close["Date"], "Close":
```

```
Daily_Close[Stocks[y]]})
    #Compute Moving Average & RSI
    Infy['MA'] = ta.SMA(Infy['Close'],200)
    Infy['RSI'] = ta.RSI(Infy['Close'],21)
    #Generate Buy & Sell Signals
    Infy['MA_long'] = np.nan
    for x in range (len(Infy)):
        if Infy.Close[x] > Infy.MA[x]:
            Infy['MA_long'][x] = 1
        if Infy.Close[x] <= Infy.MA[x]:
            Infy['MA_long'][x] = 0
    Infy.MA_long = Infy.MA_long.fillna(method='ffill')
    Infy['RSI_long'] = np.nan
    for x in range (len(Infy)):
        if Infy.RSI[x] > 50:
            Infy['RSI_long'][x] = 1
        if Infy.RSI[x] <= 50:
            Infy['RSI_long'][x] = 0
    Infy.RSI_long = Infy.RSI_long.fillna(method='ffill')
    Infy['positions_long'] = Infy.MA_long * Infy.RSI_long
    Infy['price_difference']= Infy.Close - Infy.Close.shift(1)
    Infy['pnllong'] = Infy.positions_long.shift(1) *
Infy.price_difference
    Infy['cumpnl_long'] = Infy.pnllong.cumsum()
    # Calculate the max drawdown in the past window days for each
day
    Infy['rolling_max'] = Infy['cumpnl_long'].rolling(250,
min_periods=1).max()
    Infy['daily_drawdown'] = Infy['cumpnl_long']-Infy['rolling_max']
    # Calculate the minimum (negative) daily drawdown
    Infy['max_daily_drawdown'] = Infy['daily_drawdown'].rolling(250,
min_periods=1).min()
    # Calculate number of trades and return of each trade
    Infy['Trade'] = Infy['positions_long'].diff()
    T2 = Infy.where((Infy.Trade != 0))
    T2 = T2.dropna()
```

```
        T2['Trade_Return'] = (T2['cumpnl_long'].diff()/T2['Close'])*100
        Table['Stock'] = Stocks[y]
        Table['Close'] = Infy.Close.iloc[-1]
        Table['Total Profit'] = Infy.cumpnl_long.iloc[-1]
        Table['Percent_Return'] = (Table['Total Profit']/Table['Close'])*100
        Table['Max_Drawdown'] = Infy.max_daily_drawdown.min()
        Table['Number_Trades'] = round(len(T2)/2)
        Table['Max_Return'] = T2.Trade_Return.max()
        Table['Min_Return'] = T2.Trade_Return.min()
        Table1= Table1.append(Table)
print (Table1)
```

We will get the following output –

	A	B	C	D	E	F	G	H	I
1		Stock	Close	Total Profit	Percent_Return	Max_Drawdown	Number_Trades	Max_Return	Min_Return
2	0	0	0	0	0	0	0	0	0
3	0	RELIANCE.NS	2085.80	1295.73	62.12	-224.90	107.00	37.92	-4.74
4	0	HDFC.NS	2539.40	309.60	12.19	-443.65	100.00	21.59	-15.09
5	0	SBIN.NS	390.15	182.03	46.66	-90.00	70.00	32.22	-5.73
6	0	LT.NS	1442.50	495.69	34.36	-337.55	79.00	35.38	-7.19
7	0	TATAMOTORS.NS	322.95	317.36	98.27	-100.28	76.00	30.43	-8.13
8	0	ICICIBANK.NS	597.75	135.46	22.66	-98.25	95.00	25.95	-4.94
9	0	NESTLEIND.NS	16101.60	6621.00	41.12	-3629.80	82.00	26.99	-6.15
10	0	INFY.NS	1253.30	294.27	23.48	-188.80	84.00	39.62	-27.00

As we can observe in the output this strategy is giving 12% to 98% return in various stocks with an average 75 to 100 trades per stock. Maximum return in a single trade is given in column ‘H’ and maximum loss in a single trade is given in column ‘I’.

As I have taken 2 indicators Moving average and RSI for development of this strategy in the above example, in the same way you can take 2 or more indicators for back-testing.

GITHUB LINK

You can download pythoncode used in this book from follwing link -

https://github.com/OptionsnPython/Python-for-Trading-on-Technical

COST OF TRADE

One more important factor to consider is the cost of Trade also. Apart from the cost of fund we have many other expenses also. For example in India we have Security Transaction Tax (STT) that is Rs 1000/- per crore on sell side (We can take it Rs 500/- per crore both side – buy and sell), we have stamp duty that is Rs 100 per crore of turnover on both side, Exchange Transaction charges of Rs 200 per Crore, SEBI Fee of Rs 15/- per crore, GST of 18% on Exchange Transaction Charges. So on a round trip transaction cost is approx .0002%. So I am taking 0.0001% on each side. (I have taken trading cost of Future, However buying selling shares in cash equity segment will be 10 times more costly because we have Security Transaction Tax of Rs 10,000 per crore on both side, so its Rs 20,000 per crore on round trip. This cost of trade can make any good strategy unprofitable. So if you don't trade derivatives in India and only buy to take delivery of shares in you demat than better you select a strategy with minimum number of trades to bring down cost of trade)

In the following example I am taking various moving averages for computation of profit generated by strategy. I am creating 2 loops, first loop for different moving average period (20 to 200) and second loop for different RSI period (14 and 21). I am also computing cost of Trade in last column.

Python Codes –

```
Daily_Close = pd.read_csv('Daily_Close')
Stocks = ['RELIANCE.NS', 'HDFC.NS', 'SBIN.NS', 'LT.NS', 'TATAMOTORS.NS', 'ICICIBANK.NS', 'NESTLEIND.NS', 'INFY.NS']
import warnings
warnings.filterwarnings("ignore")
```

```
Table = pd.DataFrame({"MA":0, "RSI":0, "Stock":['0'], "Close":0,
"Total Profit":0, "Percent_Return":0, "Max_Drawdown":0,
"Number_Trades":0, "Max_Return":0, "Min_Return":0,
"Trade_Cost":0})
Table1 = pd.DataFrame({"MA":0, "RSI":0, "Stock":['0'], "Close":0,
"Total Profit":0, "Percent_Return":0, "Max_Drawdown":0,
"Number_Trades":0, "Max_Return":0, "Min_Return":0,
"Trade_Cost":0})
for a in range (20, 220, 20):
    for b in range (14, 28, 7):
        for y in range (len(Stocks)):
            #Get Historical data
            Infy = pd.DataFrame({"Date": Daily_Close["Date"],
"Close": Daily_Close[Stocks[y]]})
            #Compute Moving Average & RSI
            Infy['MA'] = ta.SMA(Infy['Close'],a)
            Infy['RSI'] = ta.RSI(Infy['Close'],b)
            #Generate Buy & Sell Signals
            Infy['MA_long'] = np.nan
            for x in range (len(Infy)):
                if Infy.Close[x] > Infy.MA[x]:
                    Infy['MA_long'][x] = 1
                if Infy.Close[x] <= Infy.MA[x]:
                    Infy['MA_long'][x] = 0
            Infy.MA_long = Infy.MA_long.fillna(method='ffill')
            Infy['RSI_long'] = np.nan
            for x in range (len(Infy)):
                if Infy.RSI[x] > 50:
                    Infy['RSI_long'][x] = 1
                if Infy.RSI[x] <= 50:
                    Infy['RSI_long'][x] = 0
            Infy.RSI_long = Infy.RSI_long.fillna(method='ffill')
            Infy['positions_long'] = Infy.MA_long * Infy.RSI_long
            Infy['price_difference']= Infy.Close - Infy.Close.shift(1)
            Infy['pnllong'] = Infy.positions_long.shift(1) *
Infy.price_difference
```

```
            Infy['cumpnl_long'] = Infy.pnllong.cumsum()
            # Calculate the max drawdown in the past window days for
each day
            Infy['rolling_max'] = Infy['cumpnl_long'].rolling(250,
min_periods=1).max()
            Infy['daily_drawdown'] = Infy['cumpnl_long']-
Infy['rolling_max']
            # Calculate the minimum (negative) daily drawdown
            Infy['max_daily_drawdown'] =
Infy['daily_drawdown'].rolling(250, min_periods=1).min()
            # Calculate number of trades and return of each trade
            Infy['Trade'] = Infy['positions_long'].diff()
            T2 = Infy.where((Infy.Trade != 0))
            T2 = T2.dropna()
            T2['Trade_Return'] =
(T2['cumpnl_long'].diff()/T2['Close'])*100
            T2['Cost_of_Trade'] = T2['Close']*0.0001
            Table['MA'] = a
            Table['RSI'] = b
            Table['Stock'] = Stocks[y]
            Table['Close'] = Infy.Close.iloc[-1]
            Table['Total Profit'] = Infy.cumpnl_long.iloc[-1]
            Table['Percent_Return'] = (Table['Total
Profit']/Table['Close'])*100
            Table['Max_Drawdown'] = Infy.max_daily_drawdown.min()
            Table['Number_Trades'] = round(len(T2)/2)
            Table['Max_Return'] = T2.Trade_Return.max()
            Table['Min_Return'] = T2.Trade_Return.min()
            Table['Trade_Cost'] = T2.Cost_of_Trade.sum()
            Table1= Table1.append(Table)
Table1.to_csv("Table1.csv", index=True, encoding='utf8')
```

Output of the above program –

	A	B	C	D	E	F	G	H	I	J	K	L
1		MA	RSI	Stock	Close	Total Profit	Percent_Return	Max_Drawdown	Number_Trades	Max_Return	Min_Return	Trade_Cost
139	0	180	21	RELIANCE.NS	2085.80	1128.42	54.10	-272.50	115.00	37.92	-4.74	16.83
140	0	180	21	HDFC.NS	2539.40	300.40	11.83	-443.65	102.00	21.59	-15.09	25.73
141	0	180	21	SBIN.NS	390.15	193.08	49.49	-89.20	74.00	33.14	-5.73	3.93
142	0	180	21	LT.NS	1442.50	534.36	37.04	-338.75	80.00	35.38	-7.19	16.84
143	0	180	21	TATAMOTORS.NS	322.95	295.71	91.56	-134.41	78.00	32.94	-10.82	5.01
144	0	180	21	ICICIBANK.NS	597.75	166.83	27.91	-98.25	92.00	25.95	-6.30	4.95
145	0	180	21	NESTLEIND.NS	16101.60	7199.90	44.72	-3137.60	80.00	26.99	-6.15	114.41
146	0	180	21	INFY.NS	1253.30	227.46	18.15	-220.30	88.00	39.62	-27.00	9.59
147	0	200	14	RELIANCE.NS	2085.80	1287.67	61.73	-301.55	119.00	34.97	-4.37	19.22
148	0	200	14	HDFC.NS	2539.40	227.30	8.95	-473.00	126.00	20.01	-9.95	32.99
149	0	200	14	SBIN.NS	390.15	152.31	39.04	-71.70	86.00	32.99	-6.29	4.70
150	0	200	14	LT.NS	1442.50	780.16	54.08	-267.90	84.00	28.82	-3.87	17.99
151	0	200	14	TATAMOTORS.NS	322.95	185.40	57.41	-120.43	98.00	25.04	-6.78	6.65
152	0	200	14	ICICIBANK.NS	597.75	134.82	22.56	-103.00	108.00	27.49	-6.24	6.14
153	0	200	14	NESTLEIND.NS	16101.60	6571.50	40.81	-3202.30	112.00	30.05	-5.43	171.01
154	0	200	14	INFY.NS	1253.30	168.12	13.41	-165.75	115.00	19.43	-27.00	12.93
155	0	200	21	RELIANCE.NS	2085.80	1295.73	62.12	-224.90	107.00	37.92	-4.74	15.76
156	0	200	21	HDFC.NS	2539.40	309.60	12.19	-443.65	100.00	21.59	-15.09	25.46
157	0	200	21	SBIN.NS	390.15	182.03	46.66	-90.00	70.00	32.22	-5.73	3.77
158	0	200	21	LT.NS	1442.50	495.69	34.36	-337.55	79.00	35.38	-7.19	16.60
159	0	200	21	TATAMOTORS.NS	322.95	317.36	98.27	-100.28	76.00	30.43	-8.13	4.90
160	0	200	21	ICICIBANK.NS	597.75	135.46	22.66	-98.25	95.00	25.95	-4.94	5.16
161	0	200	21	NESTLEIND.NS	16101.60	6621.00	41.12	-3629.80	82.00	26.99	-6.15	119.92
162	0	200	21	INFY.NS	1253.30	294.27	23.48	-188.80	84.00	39.62	-27.00	9.23

In the above output B column is Moving Average column and C column is RSI period taken for computation of Retruns. In the last column we have total cost of trade.

You can develop you're your own strategy in python with different combinations of technical indicators computed with different time periods. I always feel Options spreads are always better than trading future. For example buying in-the-money option along with selling out-of-money option with equal time value will have a benefit of limited risk and lower margins as compare to buying necked future. You can learn option strategies and back testing in python in my next book 'Option Greeks, Strategies & backtesting in Python'. Happy Trading.

ABOUT AUTHORS / ACKNOWLEDGMENTS

Anjana Gupta, I am author of this book. I am having master degree in science and management. I am having more than 10 years of experience. Special thanks to Puneet Kanwar, who was instrumental in the completion and editing of this book. Puneet Kanwar is having an experience of 15 years in Indian capital market. He has worked with BSE Limited, formally known as Bombay Stock Exchange, for 6 years. He has also worked with prestigious broking house, Edelweiss prior to BSE. In 2017 Puneet resigned from BSE for his own venture. Currently he is a successful option trader and arbitrager.

For feedback / suggestions / query / doubt you can write to me at optionsnpython@gmail.com.

Happy learning. Anjana Gupta

BOOKS BY THIS AUTHOR

Option Greeks, Strategies & Backtesting In Python

Books By This Author Option Greeks, Strategies & Backtesting in Python is divided into three parts -

1. First part cover option Greeks - Delta, Gamma, Theta, Vega, Delta hedging & Gamma Scalping, implied volatility with the example of past closing prices (Basics of Future and options explaind).

2. Second part covers option trading strategies with examples and computation of returns of a strategy on past data. (You will get an idea how professional traders think)

3. Third part covers Python for traders. After reading this book a novice trader will also be able to use python from installation of Anaconda on his laptop & extracting past data to back-testing and development of his own Option strategies.

Many program codes and their results also explained for back-testing of strategies likes ratios, butterfly etc.

BOOKS BY THIS AUTHOR

Trading Pairs With Python : Advance Statistical Tools For Trading & Backtesting A Strategy In Python

This is a comprehensive guide on pair trading. Concepts are explained from very basic so that any trader who does not understand statistics can understand and learn. Trading with the help of statistical tools explained in detail.

Models are developed in Google spread sheet and Python for back testing and finding opportunities.

This book will cover following –
1.Basics of Python so that a non-programmer can understand Python for backtesting on past data.

2.Fetching Historical data in Google Spreadsheet (Excel) and Python through various free data sources –
a.Daily data in Google Spreadsheet and Python
b.Per minute historical data in Python
c.Live data in Google spreadsheet and Python

3.Basics of statistics and use in trading. Trading with advance statistical tools.

4.Pair trading concepts, development of pair trading models and backtesting of models for getting results on past data.

5.Machine learning tools for pair trading.

This book is written for individuals and traders. With help of this book individual trader, investor can understand statistical tools of pair trading and machine learning for pair trading.