

[This Photo](https://en.wikiversity.org/wiki/Python_Concepts) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

Python For Financial Analysis

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **CONTENTS** | **PAGE NUMBER** |
| 1 | Introduction | 1 |
| 2 | Literature Review | 2 - 3 |
| 3 | Data Collection | 4 |
| 4 | Data Cleansing | 5 |
| 5 | Program Code | 6 - 35 |

**INTRODUCTION**

Stock analysis and portfolio optimization is a python program with developed the intent of aiding investors in the analysis of securities and also helping them in optimizing their portfolios.

The entire project is divided into two parts: -

The first part is the stock or securities analysis, whereby the user must input the ticker symbol of the security he wants to analyze. Then using the scraped data the program displays a variety of information and plots about the stock and in the end we run an ARIMA model to predict the future trend of the stock price.

The second part is about portfolio optimization, wherein a user creates a portfolio and by using the concept of Sharpe ratio we plot the efficient frontier and find the most optimal portfolio allocation for our stocks.

Finally, all the plots are presented in the form of a report using HTML, CSS and JAVASCRIPT by creating a website.

**Literature Review**

There are two concepts incorporated in this project –

* **ARIMA Model** – ARIMA stands for Auto Regressive Integrated Moving Average. There are two types of ARIMA models – Seasonal ARIMA and Non – Seasonal ARIMA. Since, stock data is usually Non – Seasonal we will be using the Non – Seasonal ARIMA. It takes 3 parameters (p,d,q) where p,d & q are non – negative positive integers.

Parts of ARIMA Model

* **AR (p) – Autoregression –** A regression model that utilizes the dependant relationship between a current observation and observations over a previous period.
* **I (d) –** **Integrated –** Differencing of observations (Subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
* **MA (q) -** **Moving Average –** A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

When fitting an ARIMA model, we aim to find the values of our parameters p,d and q which optimise or minimise a certain metric of interest. There are many methods to achieve this goal and yet the correct parametrization of ARIMA models can be a tedious process that requires statistical expertise and time. In this tutorial, we hope to overcome this issue by writing a grid search algorithm in python to select the optimal parameter values for our ARIMA(p,d,q) time series model.

The use of a “grid search” is to iteratively explore different combinations of parameters. For each combination of parameters, we fit an ARIMA model with the SARIMAX() function and assess its overall performance. Once we have explored the entire domain of parameters, our optimal set of parameters will be the one that yields the best performance for our criteria of interest. In this scenario, our criteria of interest is the Akaike information criterion (AIC). The AIC measures how well a model fits the data while taking into account the overall complexity of the model. We are therefore interested in finding a model that returns the lowest AIC value.

* **Portfolio Optimization -** The Sharpe ratio was developed by Nobel laureate William F. Sharpe and is used to help investors understand the return of an investment compared to its risk.﻿ The ratio is the average return earned in excess of the risk-free rate per unit of volatility or total risk. Volatility is a measure of the price fluctuations of an asset or portfolio.

The Sharpe Ratio is a measure for calculating risk-adjusted return, and this ratio has become the industry standard for such calculations.

Sharpe ratio = (Mean portfolio return − Risk-free rate)/Standard deviation of portfolio return

MONTE CARLO SIMULATIONS

Since, we have Sharpe ratio to evaluate portfolio allocation against each other, we just guess and chack a bunch of random allocations and see which one has the best sharpe ratio. This is known as Monte Carlo Simulation.

We assign random weights to securities in our portfolio, then calculate its mean daily return and standard deviation of daily return.

This allows us to calculate the sharpe ratio for thousands of randomly selected allocations. We then plot the allocations on a chart showing return vs volatility coloured by the sharpe ratio. From this plot we can find which portfolio has the best sharpe ratio and extract its allocation to create our portfolio.

**LANGUAGE AND BACKEND**

The entire project was developed using Python Programming Language in Jupyter Notebook. Python libraries such as Numpy, Pandas, Matplotlib, Pymongo, Mpld3, etc. were used in this project.

MongoDB is used as the backend. The plots generated using matplotlib are stored in mongodb database and then a report is generated using a website created using HTML, CSS & Javascript. The plots stored in mongodb are displayed in the website.

**DATA COLLECTION AND DATA CLEANSING**

In the initial part of the project, basic stock data and annual reports of the business were obtained using web scraping of some stock screener websites. BeautifulSoup was used to scrape and clean the data.

Python yfinance module was used to obtain historical stock data which consists of Open, High, Low, Close, Adj.Close, Volume, etc,. Of these only Open, High, Low, Adj.Close & Volume is used for analysis.

**PROGRAM CODE**

%load\_ext autoreload  
%autoreload 2  
  
import risk\_kit as rk  
from bs4 import BeautifulSoup  
import pandas as pd  
import numpy as np  
%matplotlib notebook  
import matplotlib.pyplot as plt  
import seaborn as sns  
import mpld3

stock\_name = input("Enter the stock name ").upper()

Enter the stock name INFY

soup = rk.stock\_data(stock\_name)  
cmp = soup.find("span", class\_="Number")  
print("Current Market Price - ₹",cmp.text,sep='.')

Current Market Price - ₹.1127.6

# *Strengths*

strength = soup.find("ul", class\_="strength").text  
print(strength)

The Company is Virtually Debt Free.  
The company has effective cash conversion ratio of 100.186579167471.

# *Limitations*

limitations = soup.find("ul", class\_="limitations").text  
print(limitations)

The company has delivered poor profit growth of 3.9991730983969% over past 3 years.

# Annual Report

reports = rk.balance\_sheet(stock\_name)

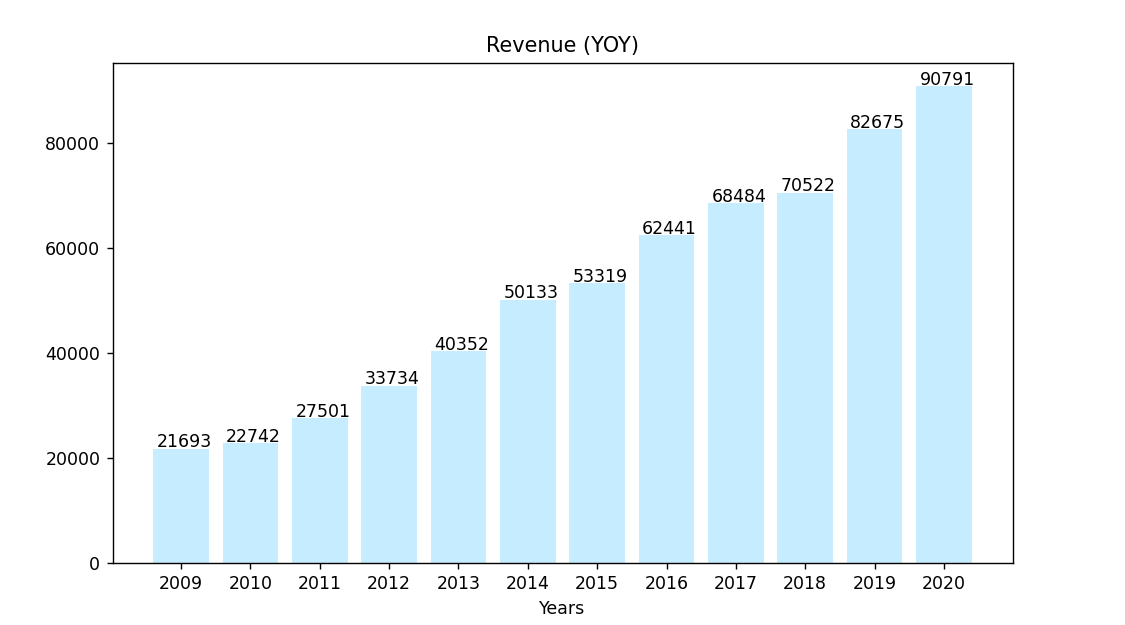
annual\_report = reports[1].rename(columns={'Unnamed: 0':' '})

if 'TTM' in annual\_report.columns:  
 annual\_report.drop('TTM',axis=1,inplace=True)

years=np.array(annual\_report.columns[1:])  
years= [int(item.split()[1]) for item in years]  
years = np.array(years)  
  
revenue=annual\_report.iloc[0][1:]  
revenue = [int(item) for item in revenue]  
revenue = np.array(revenue)

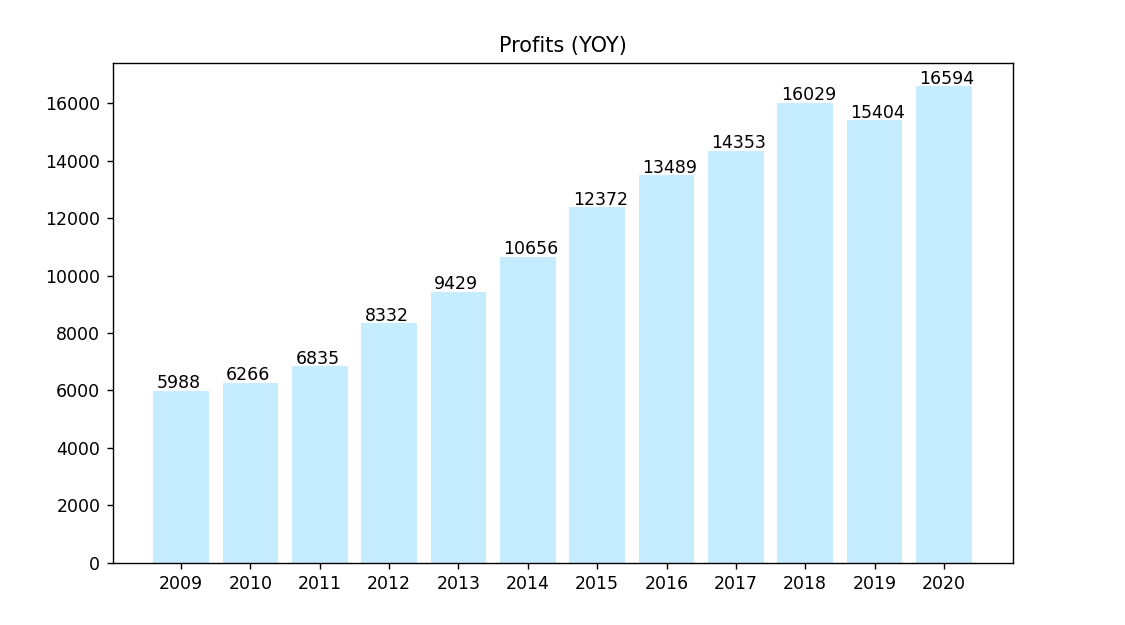
# Revenue (YOY)

fig = plt.figure(figsize=(9,5))  
ax = fig.add\_axes([0.1,0.1,0.8,0.8])  
plt.xticks(years, years)  
ax.bar(years,revenue,color="#C6ECFF")  
ax.set\_title("Revenue (YOY)")  
ax.set\_xlabel("Years")  
  
xlocs, xlabs = plt.xticks()  
for i, v in enumerate(revenue):  
 plt.text(xlocs[i] - 0.35, v + 250, str(v))

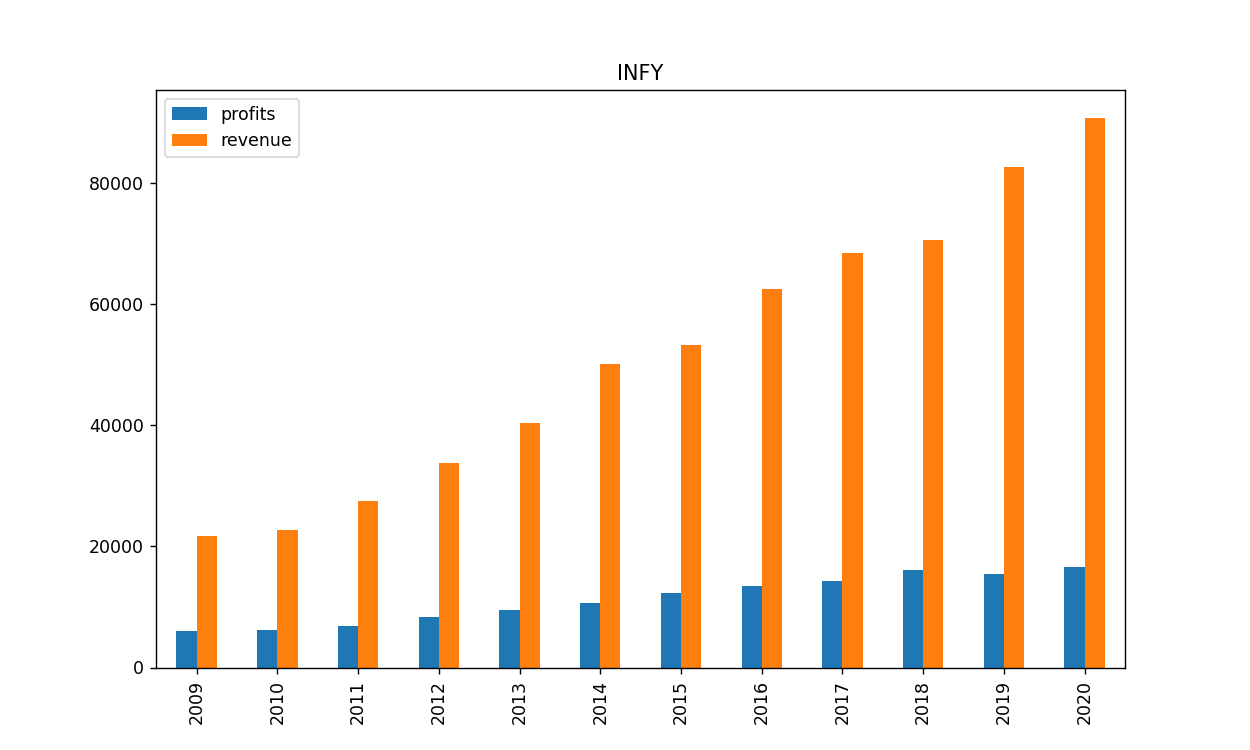


# Profit (YOY)

profit=annual\_report.iloc[9][1:]  
profit = [int(item) for item in profit]  
profit = np.array(profit)  
  
fig = plt.figure(figsize=(9,5))  
ax = fig.add\_axes([0.1,0.1,0.8,0.8])  
plt.xticks(years, years)  
plt.title("Profits (YOY)")  
ax.bar(years,profit,color="#C6ECFF")  
  
xlocs, xlabs = plt.xticks()  
for i, v in enumerate(profit):  
 plt.text(xlocs[i] - 0.35, v + 95, str(v))



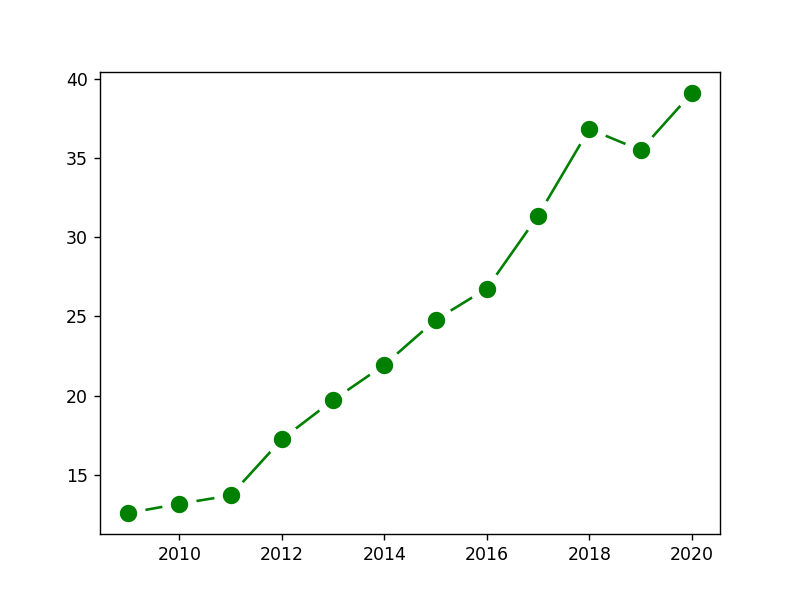
rvpr=pd.DataFrame(zip(profit,revenue),columns=['profits','revenue'],index=years)  
a=rvpr.plot.bar(figsize=(10,6),title=stock\_name);



a.set\_xticklabels([])  
fig1=a.figure

# EPS

eps=annual\_report.iloc[10][1:]  
eps = [float(item) for item in eps]  
plt.plot(years, eps, 'go-', mec='w', mew=5, ms=15);

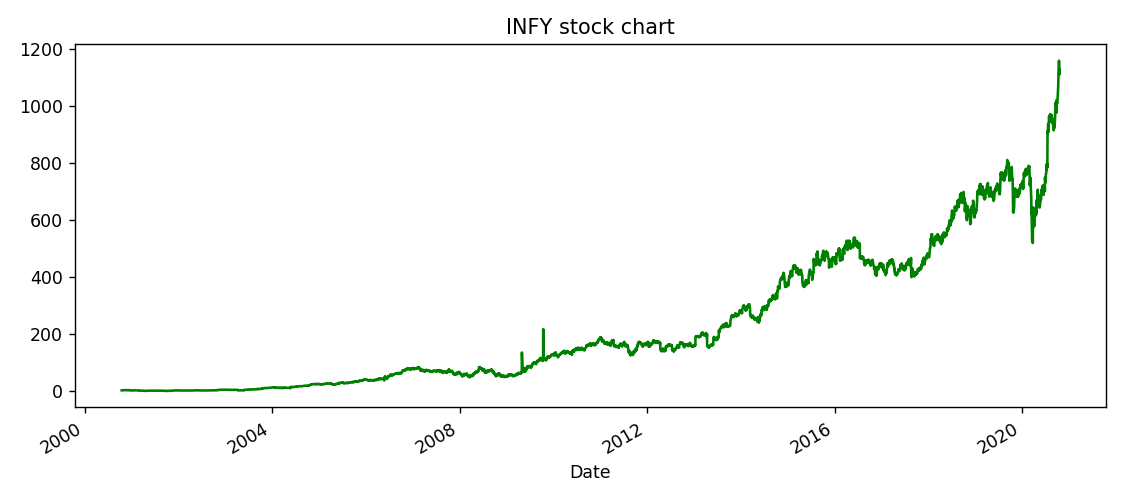


from datetime import date  
import yfinance as yf

stock = yf.Ticker(stock\_name+".NS")  
stocks=[]

stocks.append(stock.history(period="20y"))

first=stocks[0].loc[stocks[0].head(1).index[0],'Close']  
last=stocks[0].loc[stocks[0].tail(1).index[0],'Close']  
if first<last:  
 clr='green'  
else:  
 clr='red'  
fig\_obj=stocks[0]['Close'].plot(figsize=(9,4),color=clr,title=stock\_name+" stock chart")  
fig2=fig\_obj.figure  
plt.tight\_layout()



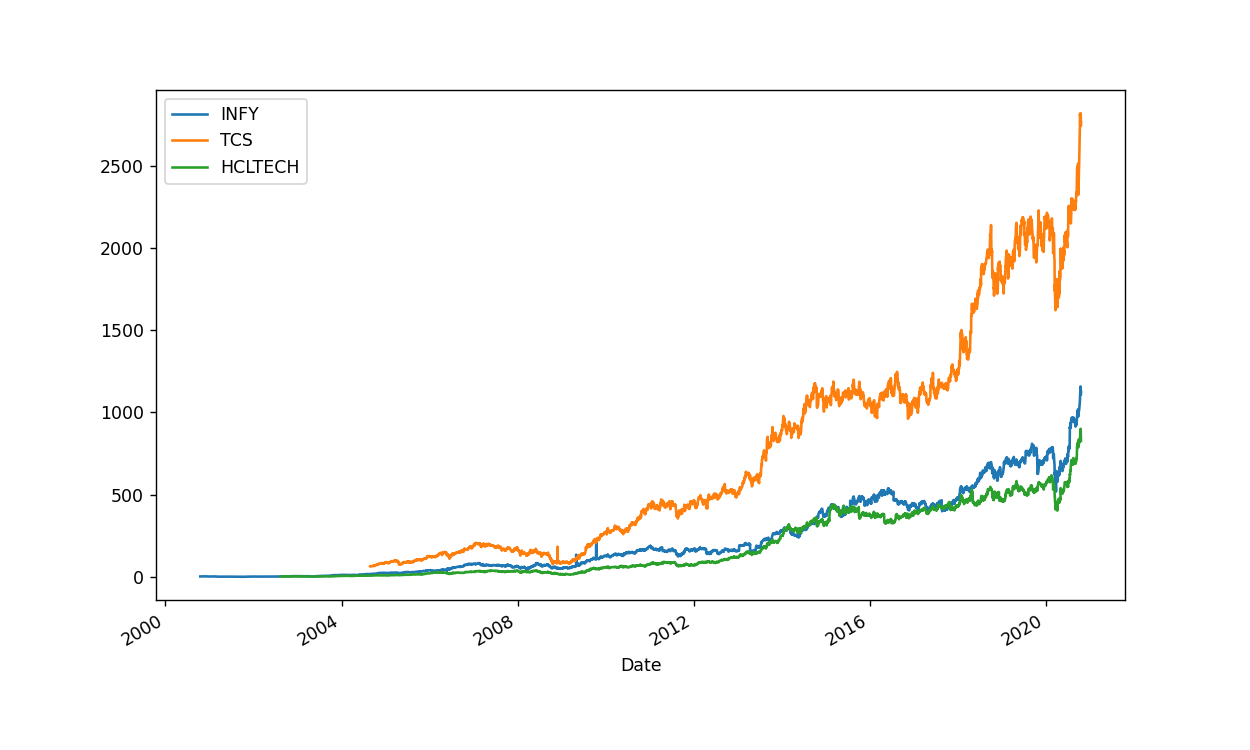
stock1\_name,stock2\_name=input("Enter two more stock names you want to compare with ").split()  
stock1 = yf.Ticker(stock1\_name+".NS")  
stock2 = yf.Ticker(stock2\_name+".NS")  
  
stocks.append(stock1.history(period="20y"))  
stocks.append(stock2.history(period="20y"))

Enter two more stock names you want to compare with TCS HCLTECH

cols=['Dividends', 'Stock Splits']

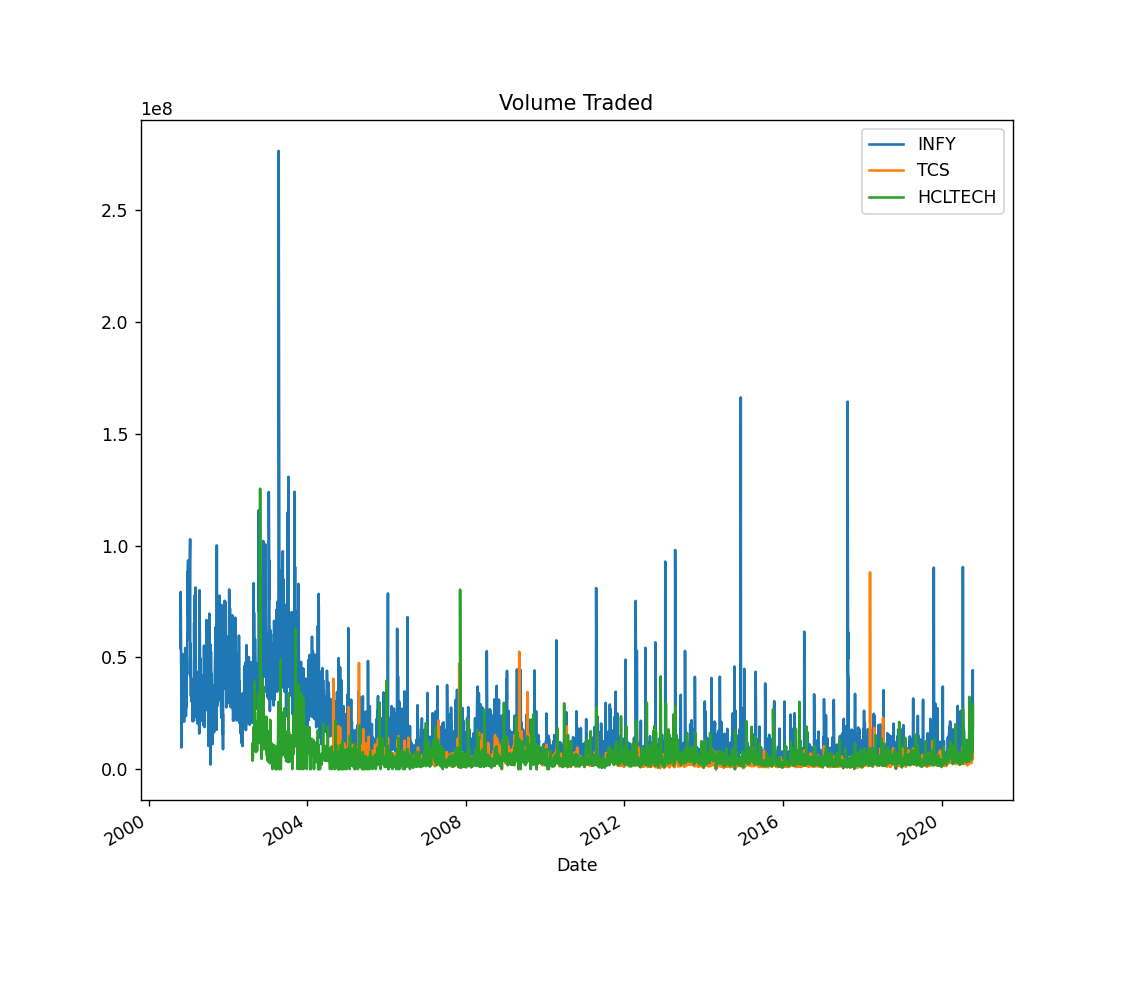
for i in range(3):  
 try:  
 stocks[i].drop(cols,axis=1,inplace=True)  
 except:  
 continue

labl=[stock\_name,stock1\_name,stock2\_name]  
for i in range(3):  
 stocks[i]['Close'].plot(figsize=(10,6))  
plt.legend(labl);  
fig3 = plt.gcf()



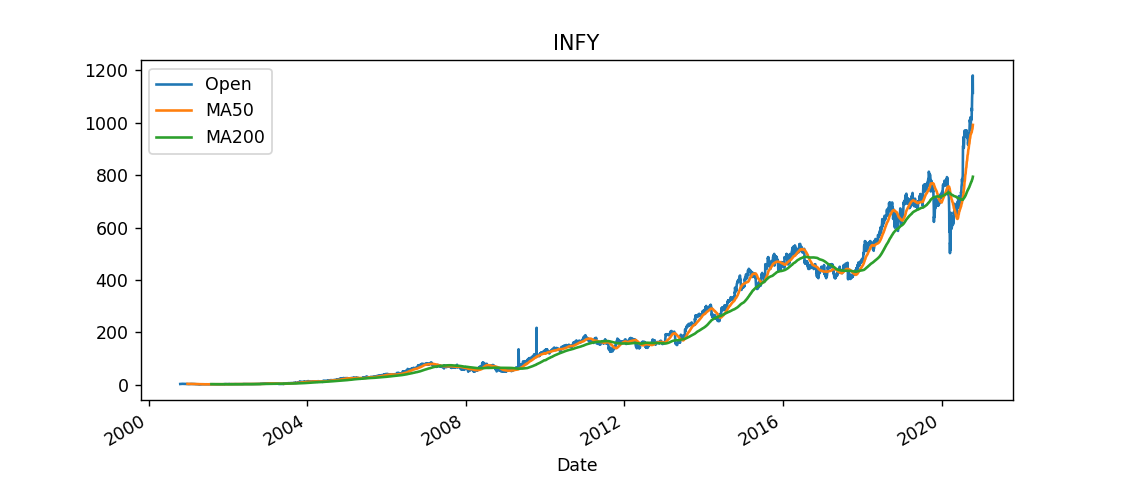
# Plotting volumes

stocks[0]['Volume'].plot(label=stock\_name,figsize=(9,8),title='Volume Traded')  
stocks[1]['Volume'].plot(label=stock1\_name)  
stocks[2]['Volume'].plot(label=stock2\_name)  
plt.legend();



## Moving average 50 & 200 days

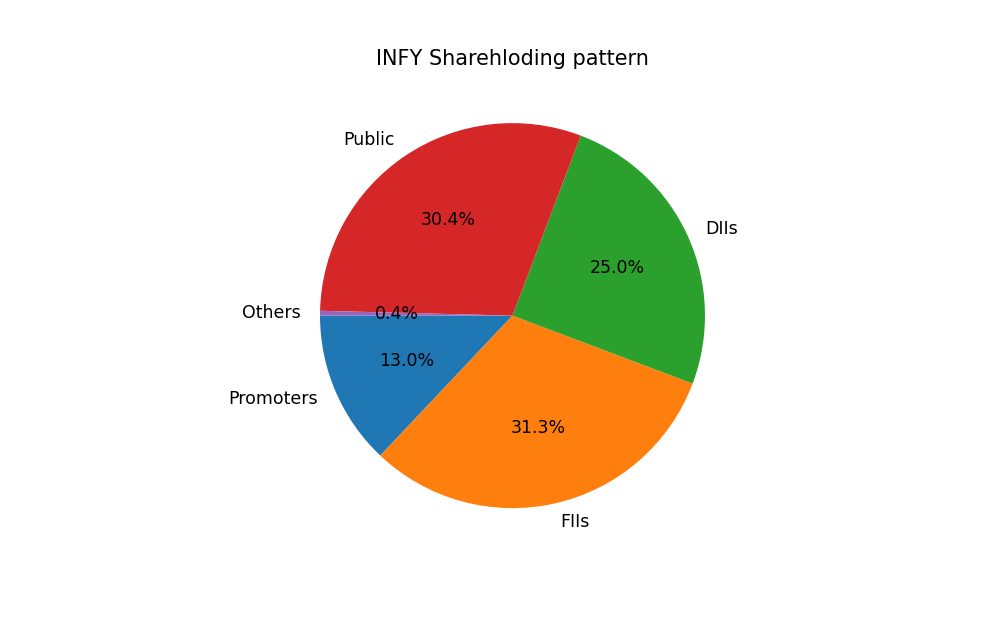
stocks[0]['MA50'] = stocks[0]['Open'].rolling(50).mean()  
stocks[0]['MA200'] = stocks[0]['Open'].rolling(200).mean()  
stocks[0][['Open','MA50','MA200']].plot(title=stock\_name,figsize=(9,4));



# SHAREHOLDING PATTERN

shareholders = list(reports[9]['Unnamed: 0'])  
shareholders = [i.rstrip("\xa0+") for i in shareholders]  
shareholding = list(reports[9][reports[9].columns[-1]])

fig4 = plt.figure(figsize =(8, 5))   
plt.pie(shareholding, labels = shareholders, autopct='%1.1f%%',startangle=180)   
plt.title(stock\_name+" Sharehloding pattern")  
plt.show()



for i in range(3):  
 stocks[i].dropna(inplace=True)

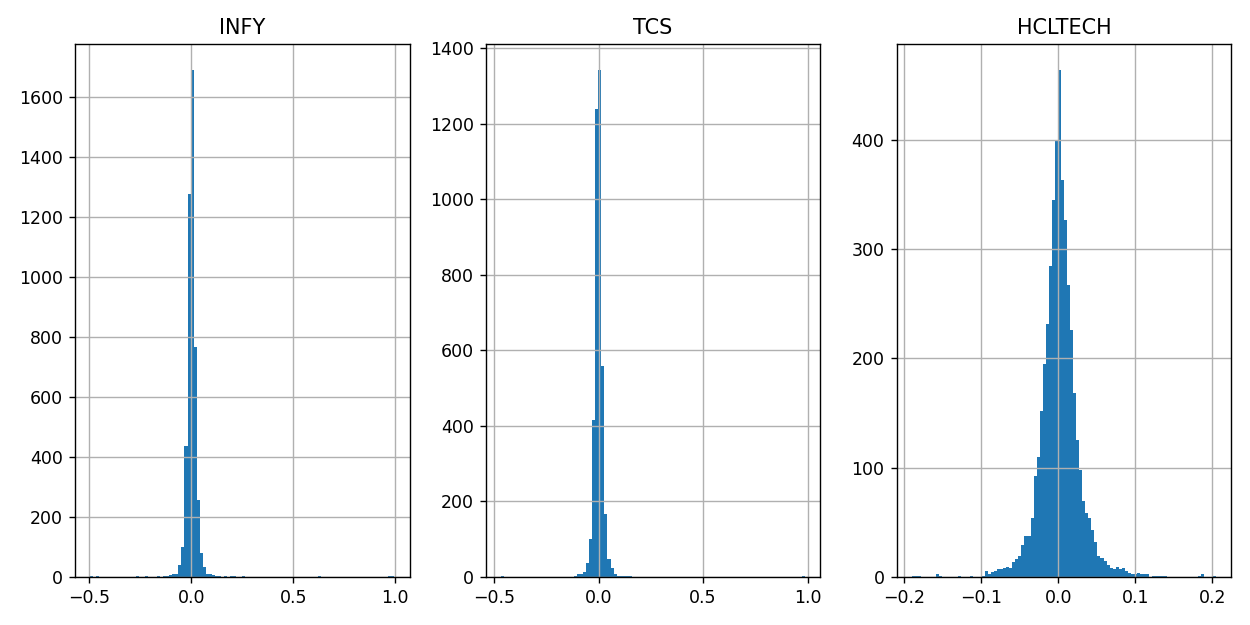
prototype = stocks.copy()

try:  
 indx=shareholders.index('Promoters')  
 promoterHolding=shareholding[indx]  
except:  
 promoterHolding=0.0

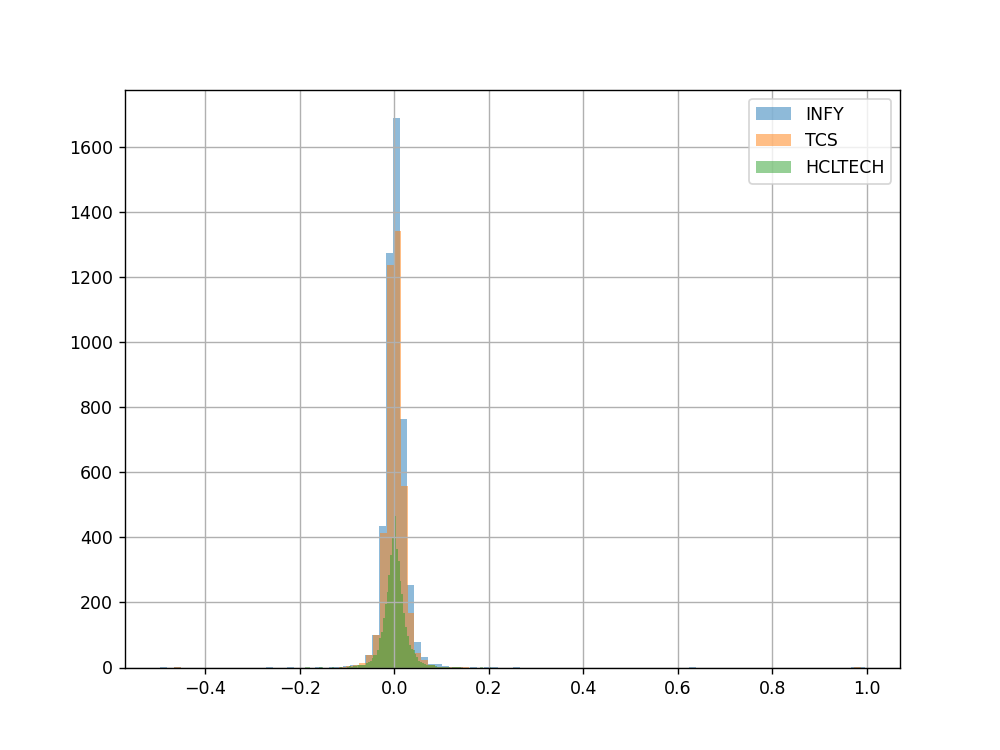
## Daily percentage change

for i in range(3):  
 stocks[i]['returns'] = stocks[i]['Close'].pct\_change(1)

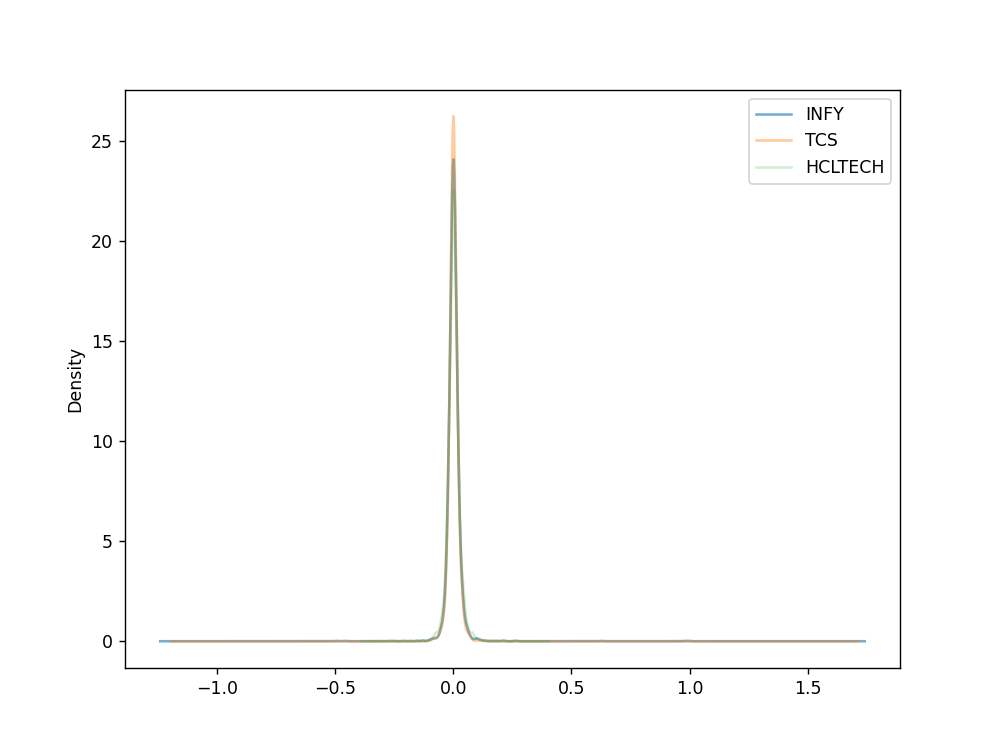
names=[stock\_name,stock1\_name,stock2\_name]  
for i in range(3):  
 plt.subplot(1,3,i+1)  
 stocks[i]['returns'].hist(bins=100,figsize=(10,5));  
 plt.title(names[i])  
plt.tight\_layout()  
fig5 = plt.gcf()



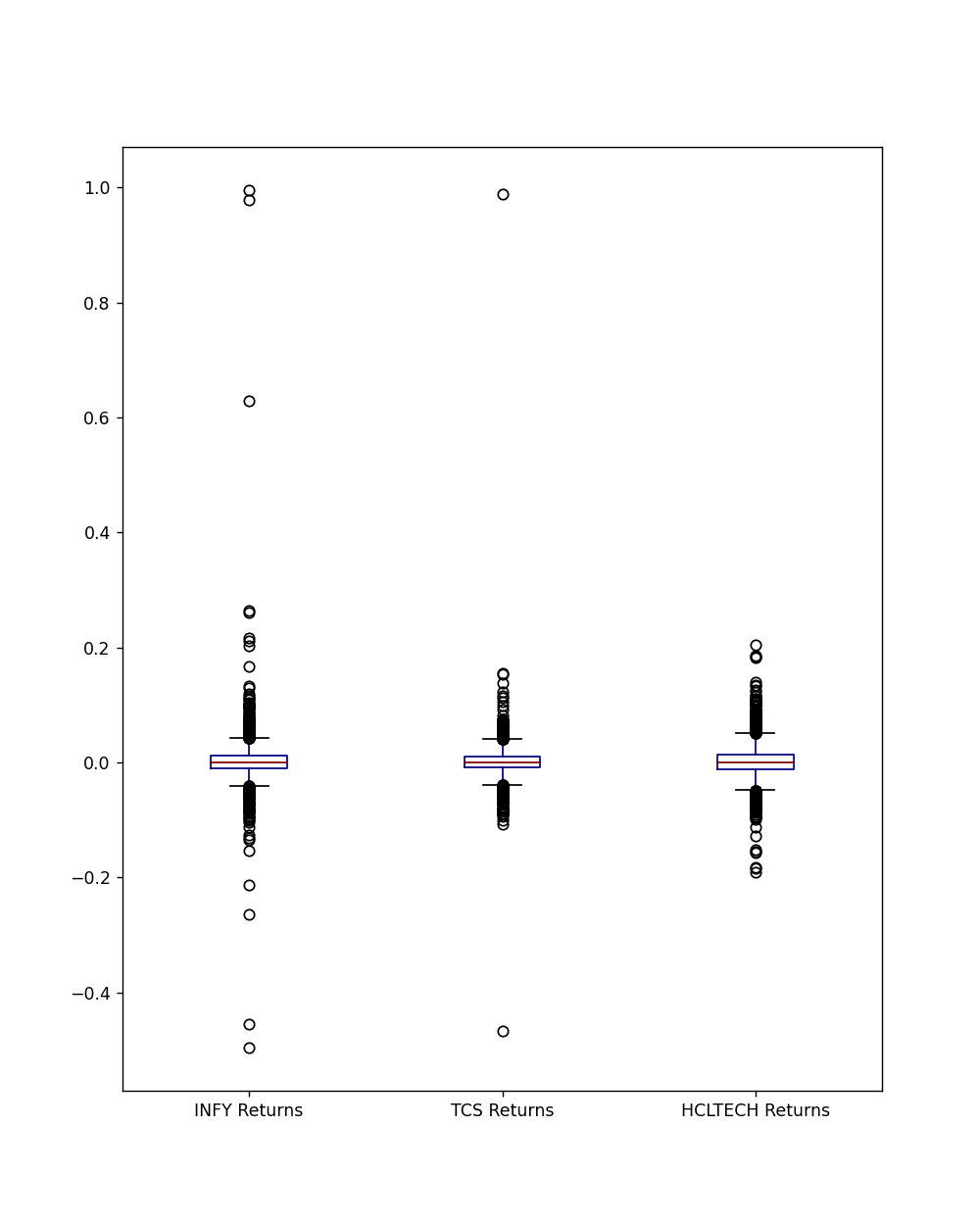
stocks[0]['returns'].hist(bins=100,label=stock\_name,figsize=(8,6),alpha=0.5)  
stocks[1]['returns'].hist(bins=100,label=stock1\_name,alpha=0.5)  
stocks[2]['returns'].hist(bins=100,label=stock2\_name,alpha=0.5)  
plt.legend();



#KDE kernel density estimation instead of histogram  
stocks[0]['returns'].plot(kind='kde',label=stock\_name,figsize=(8,6),alpha=0.6)  
stocks[1]['returns'].plot(kind='kde',label=stock1\_name,alpha=0.4)  
stocks[2]['returns'].plot(kind='kde',label=stock2\_name,alpha=0.2)  
plt.legend();

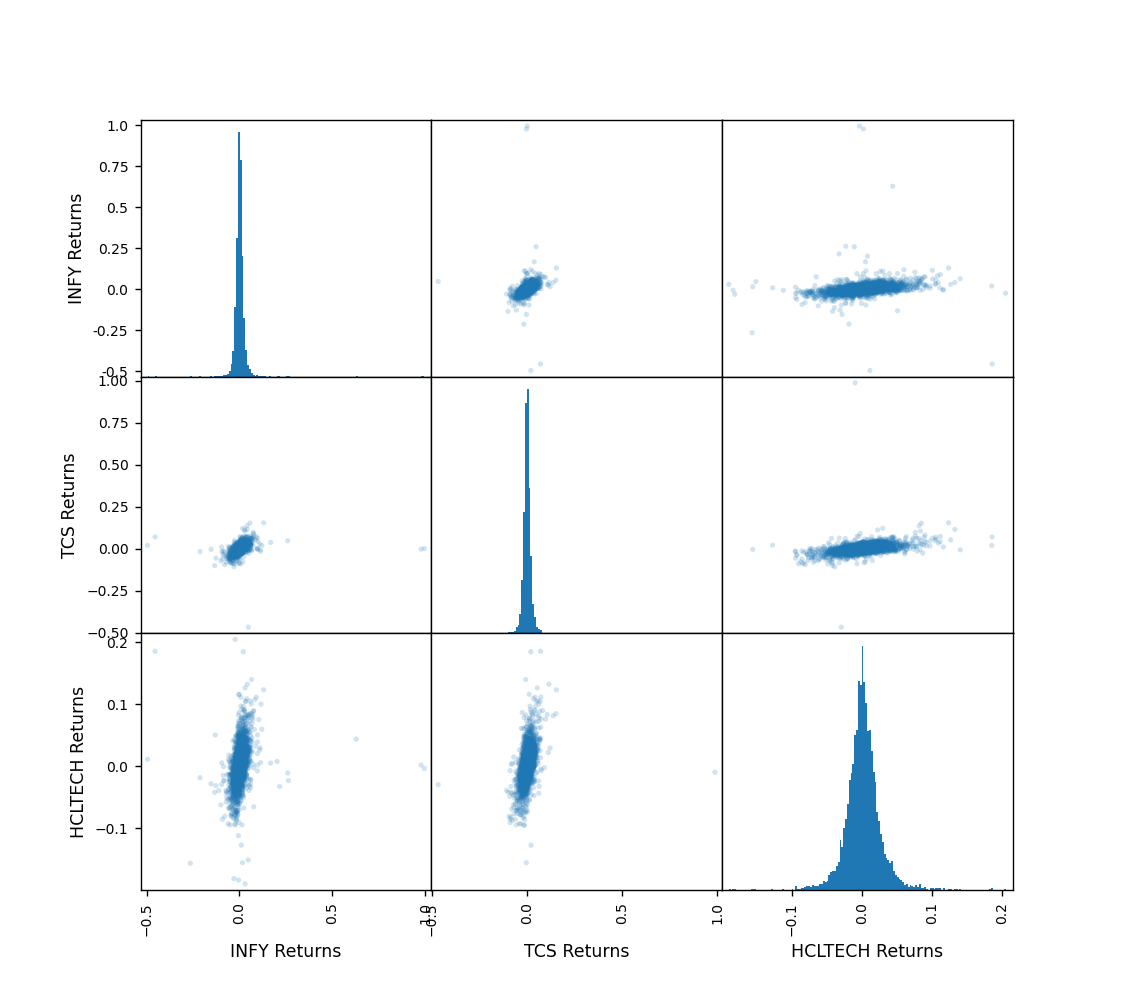


#boxplot to compare returns  
box\_df = pd.concat([stocks[0]['returns'],stocks[1]['returns'],stocks[2]['returns']],axis=1)  
box\_df.columns = [stock\_name+" Returns",stock1\_name+" Returns",stock2\_name+" Returns"]  
box\_df.plot(kind='box',figsize=(8,10),colormap='jet');



## Scatter matrix plot (finding correlation)

from pandas.plotting import scatter\_matrix  
scatter\_matrix(box\_df,figsize=(9,8),alpha=0.2,hist\_kwds={'bins':150});



# PRICE FORECASTING

from statsmodels.tools.eval\_measures import rmse  
import statsmodels.api as sm  
import itertools  
from statsmodels.tsa.arima\_model import ARIMA, ARMA  
import warnings  
warnings.filterwarnings("ignore")

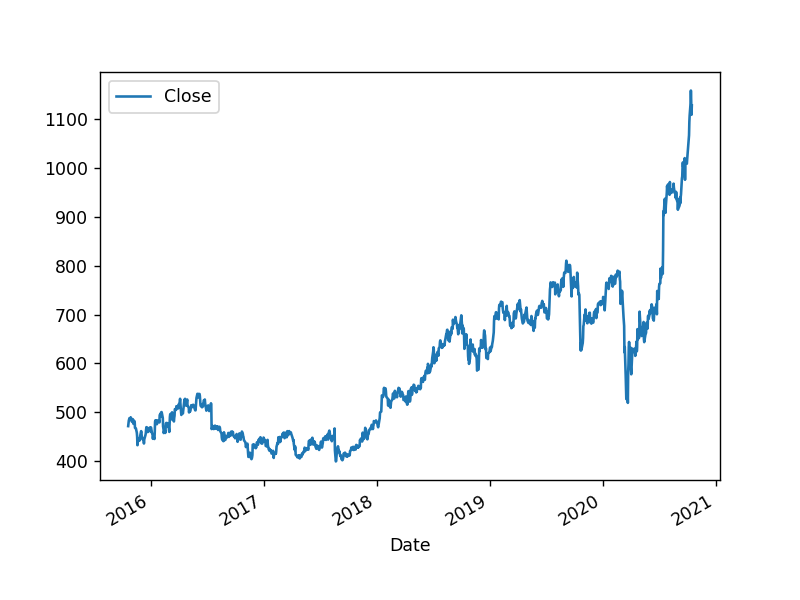
from datetime import date  
import datetime  
  
i=0  
while True:  
 try:  
 d=date.today() - datetime.timedelta(days=1825-i)  
 d=stocks[0].loc[d:]  
 break  
 except:  
 i = i+1

data = stocks[0]

#arima  
df = d['Close']

df = df.to\_frame()

df.plot(style="-");



# Define the p, d and q parameters to take any value between 0 and 3  
p = d = q = range(0, 3)  
# Generate all different combinations of p, q and q  
pdq = list(itertools.product(p, d, q))

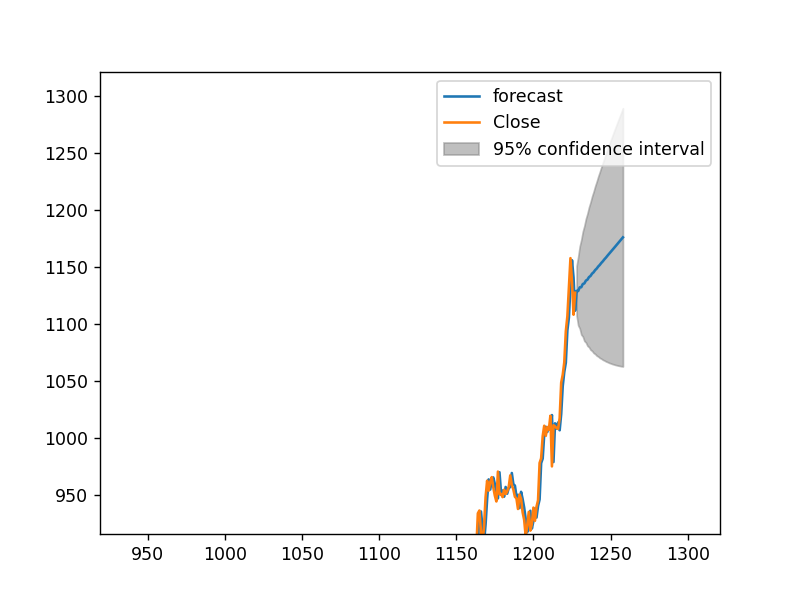
warnings.filterwarnings("ignore")  
aic= []  
parameters = []  
for param in pdq:  
 try:  
 mod = sm.tsa.statespace.SARIMAX(df, order=param,enforce\_stationarity=True, enforce\_invertibility=True)   
 results = mod.fit()  
 # save results in lists  
 aic.append(results.aic)  
 parameters.append(param)  
 #seasonal\_param.append(param\_seasonal)  
 print('ARIMA{} - AIC:{}'.format(param, results.aic))  
 except:  
 continue  
# find lowest aic   
index\_min = min(range(len(aic)), key=aic.\_\_getitem\_\_)   
print('The optimal model is: ARIMA{} -AIC{}'.format(parameters[index\_min], aic[index\_min]))

ARIMA(0, 0, 0) - AIC:19259.381852594655  
ARIMA(0, 0, 1) - AIC:17584.501383689247  
ARIMA(0, 0, 2) - AIC:16140.826685757307  
ARIMA(0, 1, 0) - AIC:9436.333360934033  
ARIMA(0, 1, 1) - AIC:9434.502891760945  
ARIMA(0, 1, 2) - AIC:9435.02702786784  
ARIMA(0, 2, 0) - AIC:10347.86207658548  
ARIMA(0, 2, 1) - AIC:9435.175941356578  
ARIMA(0, 2, 2) - AIC:9432.696798214856  
ARIMA(1, 0, 0) - AIC:9454.61545084832  
ARIMA(1, 0, 1) - AIC:9452.792084677787  
ARIMA(1, 0, 2) - AIC:9453.310498510353  
ARIMA(1, 1, 0) - AIC:9434.270395022337  
ARIMA(1, 1, 1) - AIC:9435.93397754573  
ARIMA(1, 1, 2) - AIC:9436.302807104688  
ARIMA(1, 2, 0) - AIC:9925.123818777107  
ARIMA(1, 2, 1) - AIC:9432.46762115567  
ARIMA(1, 2, 2) - AIC:9433.748134424948  
ARIMA(2, 0, 0) - AIC:9452.697199637803  
ARIMA(2, 0, 1) - AIC:9453.148028307622  
ARIMA(2, 0, 2) - AIC:9448.382517047394  
ARIMA(2, 1, 0) - AIC:9435.390095738923  
ARIMA(2, 1, 1) - AIC:9436.598695985282  
ARIMA(2, 1, 2) - AIC:9436.400085772084  
ARIMA(2, 2, 0) - AIC:9755.277347924995  
ARIMA(2, 2, 1) - AIC:9433.893037282567  
ARIMA(2, 2, 2) - AIC:9426.592958123063  
The optimal model is: ARIMA(2, 2, 2) -AIC9426.592958123063

model = ARIMA(df, order=parameters[index\_min])  
model\_fit = model.fit(disp=0)   
print(model\_fit.summary())

ARIMA Model Results   
==============================================================================  
Dep. Variable: D2.Close No. Observations: 1228  
Model: ARIMA(2, 2, 2) Log Likelihood -4706.994  
Method: css-mle S.D. of innovations 11.147  
Date: Sat, 17 Oct 2020 AIC 9425.989  
Time: 13:49:24 BIC 9456.668  
Sample: 2 HQIC 9437.532  
   
==================================================================================  
 coef std err z P>|z| [0.025 0.975]  
----------------------------------------------------------------------------------  
const 0.0017 0.001 2.020 0.043 4.97e-05 0.003  
ar.L1.D2.Close -0.9606 0.056 -17.293 0.000 -1.069 -0.852  
ar.L2.D2.Close -0.0947 0.029 -3.260 0.001 -0.152 -0.038  
ma.L1.D2.Close -0.0965 0.049 -1.977 0.048 -0.192 -0.001  
ma.L2.D2.Close -0.9034 0.049 -18.526 0.000 -0.999 -0.808  
 Roots   
=============================================================================  
 Real Imaginary Modulus Frequency  
-----------------------------------------------------------------------------  
AR.1 -1.1777 +0.0000j 1.1777 0.5000  
AR.2 -8.9692 +0.0000j 8.9692 0.5000  
MA.1 1.0000 +0.0000j 1.0000 0.0000  
MA.2 -1.1068 +0.0000j 1.1068 0.5000  
-----------------------------------------------------------------------------

fig6=model\_fit.plot\_predict(start=2, end=len(df)+30)



## Run the below cells to save and generate report

import pymongo  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
  
dblist=myclient.list\_database\_names()  
if "test" in dblist:  
 mydb = myclient["test"]  
 coll = mydb['reports']  
 print("Connected to database")  
else:  
 print("Error connecting to database")

Connected to database

username = input("Enter Username ")

Enter Username sagar

reportname = input("Enter Report name ")

Enter Report name final\_infy\_report

info = stock.info

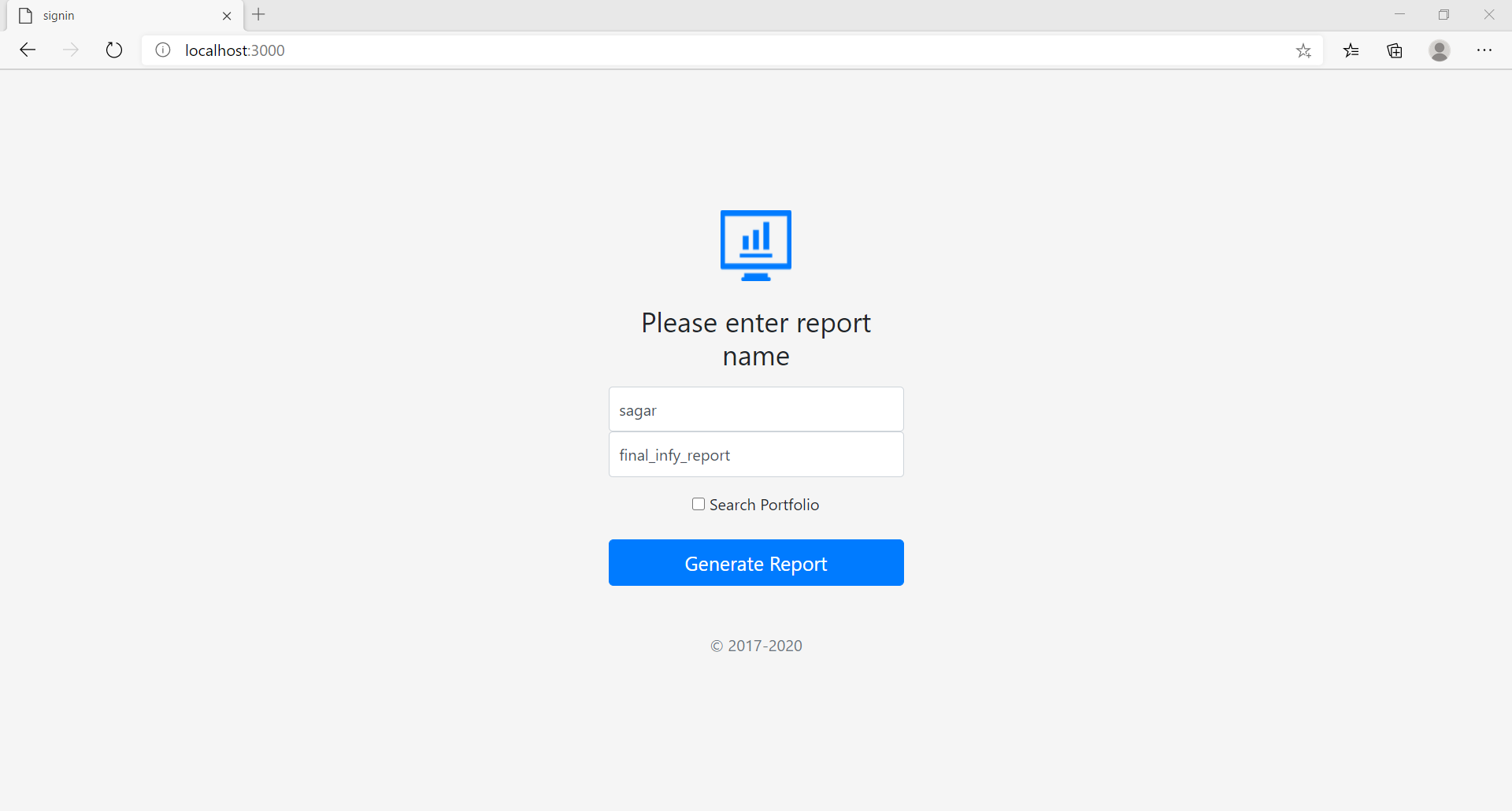
html1 = mpld3.fig\_to\_html(fig1)  
json2 = mpld3.fig\_to\_dict(fig2)  
json3 = mpld3.fig\_to\_dict(fig3)  
json4 = mpld3.fig\_to\_dict(fig4)  
json5 = mpld3.fig\_to\_dict(fig5)  
html6 = mpld3.fig\_to\_html(fig6)

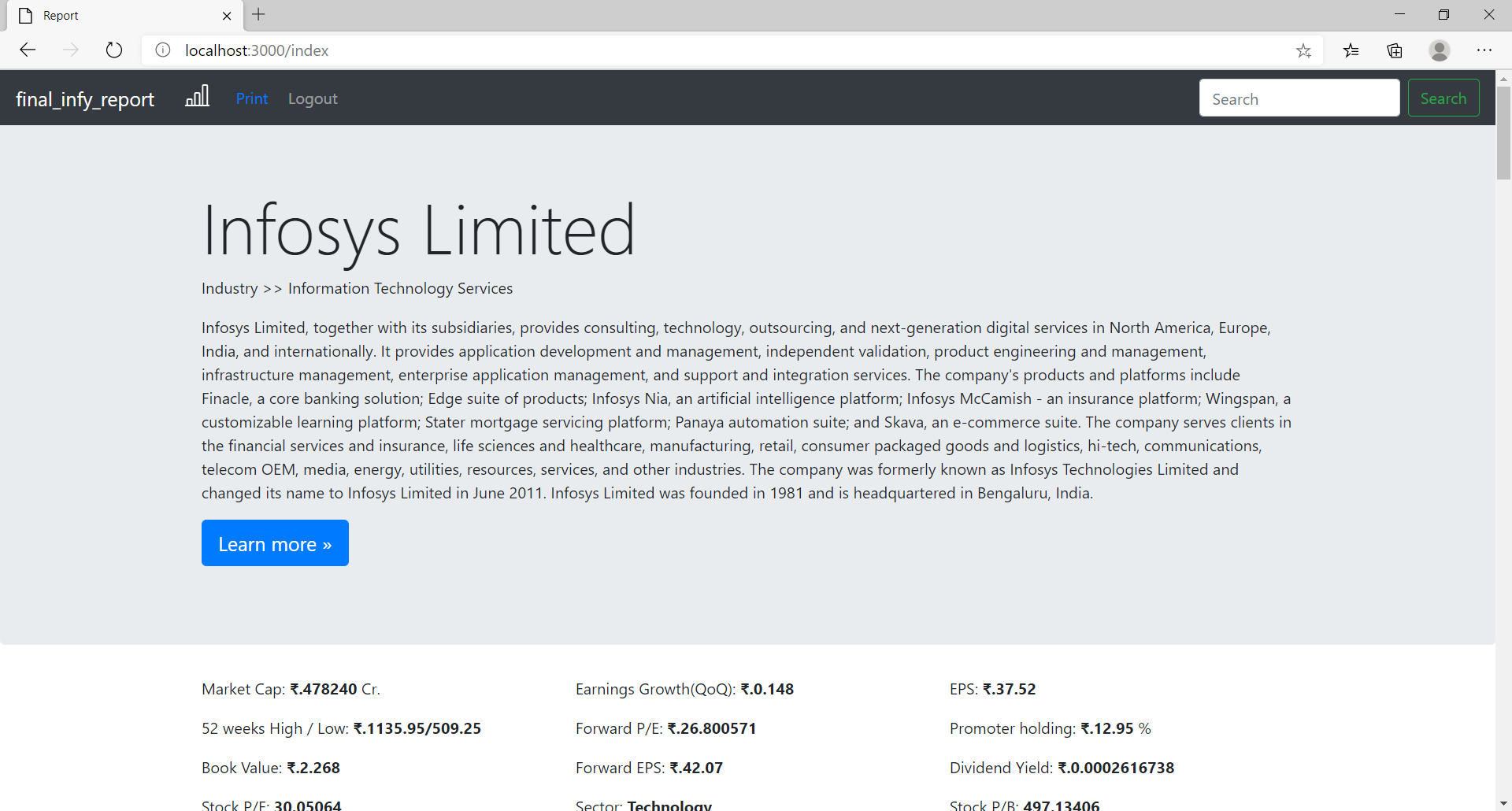
try:  
 coll.insert\_one({"\_id":reportname,"username":username,"data":{"longName":info["longName"],"industry":info["industry"],  
 "marketCap":info["marketCap"],"currentPrice":cmp.text,"fiftyTwoWeekHigh":info["fiftyTwoWeekHigh"],"fiftyTwoWeekLow":info["fiftyTwoWeekLow"],  
 "bookValue":info["bookValue"],"promoterHolding":promoterHolding,"trailingEps":info["trailingEps"],"trailingPE":info["trailingPE"],  
 "website":info["website"],"trailingAnnualDividendYield":info["trailingAnnualDividendYield"],"logo\_url":info["logo\_url"],"longBusinessSummary":info["longBusinessSummary"],"strength":strength,"limitations":limitations,  
 "beta":info["beta"],"twoHundredDayAverage":info["twoHundredDayAverage"],"priceToBook":info["priceToBook"],"earningsQuarterlyGrowth":info["earningsQuarterlyGrowth"],"forwardPE":info["forwardPE"],"forwardEps":info["forwardEps"],"sector":info["sector"],"fiftyDayAverage":info["fiftyDayAverage"]},  
 "hfig1":html1,"fig2":json2,"fig3":json3,"fig4":json4,"fig5":json5,"hfig6":html6})  
 print("Saved to DB")  
except :  
 print("Error")

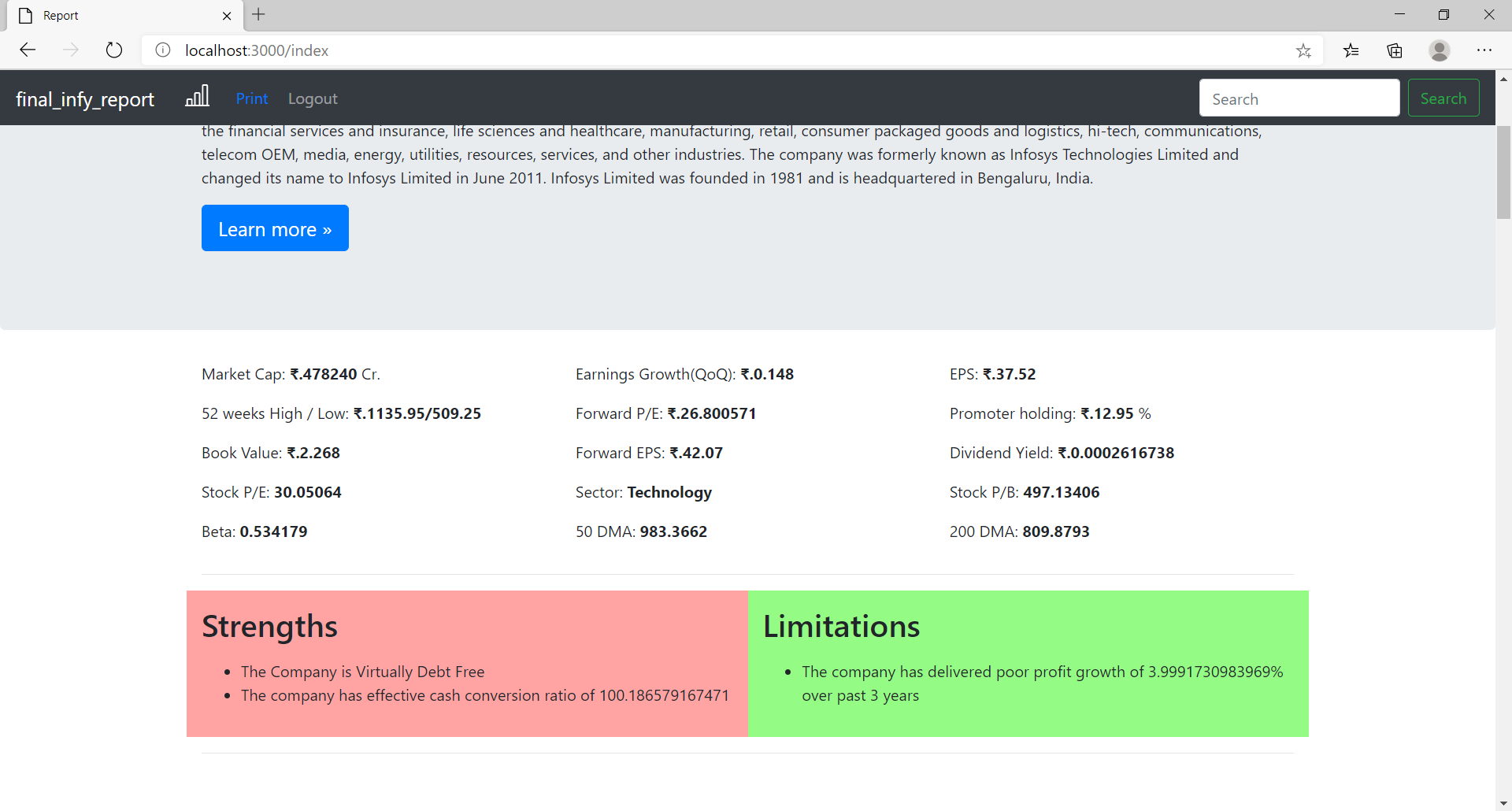
Saved to DB

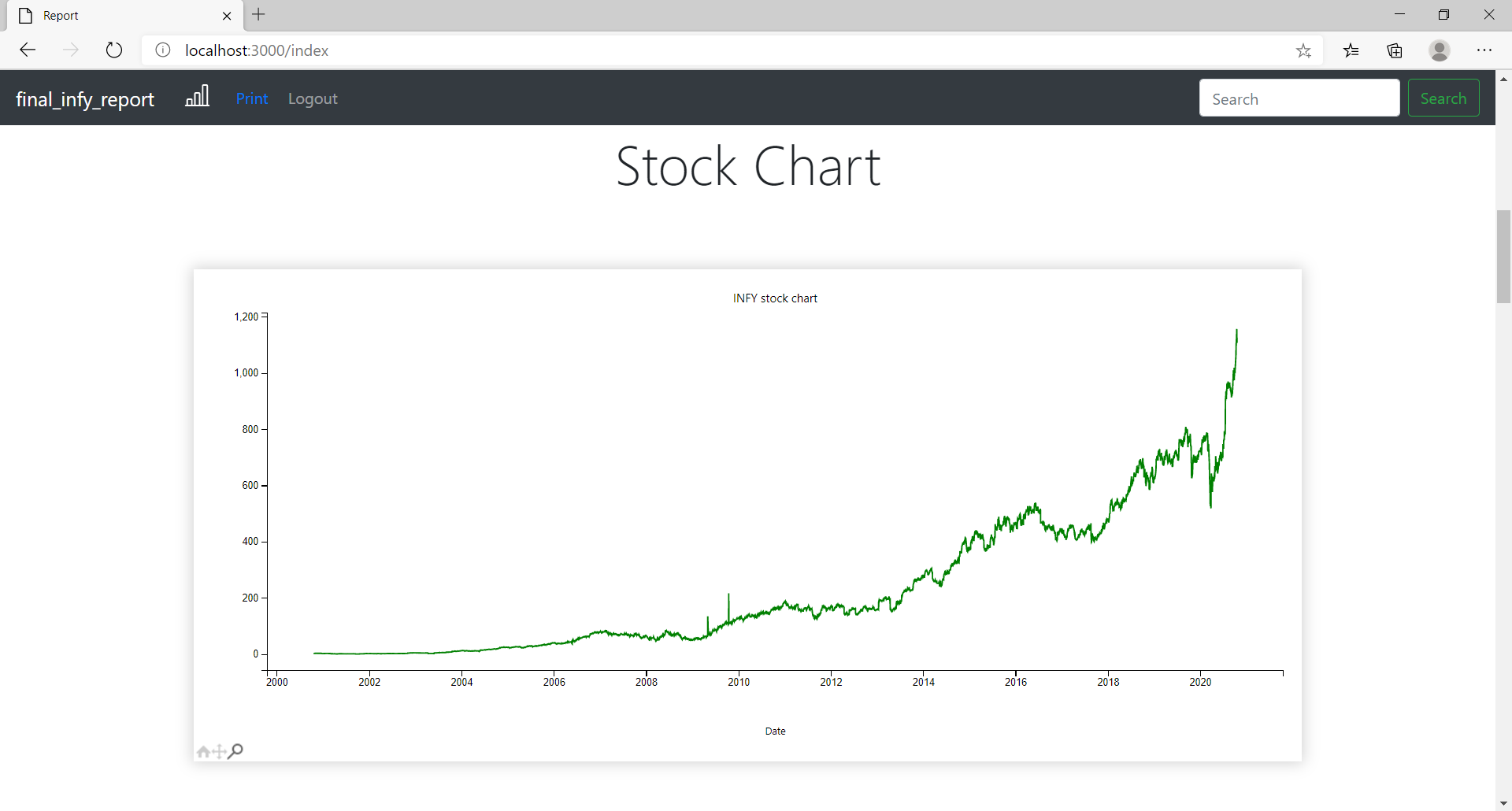
myclient.close()

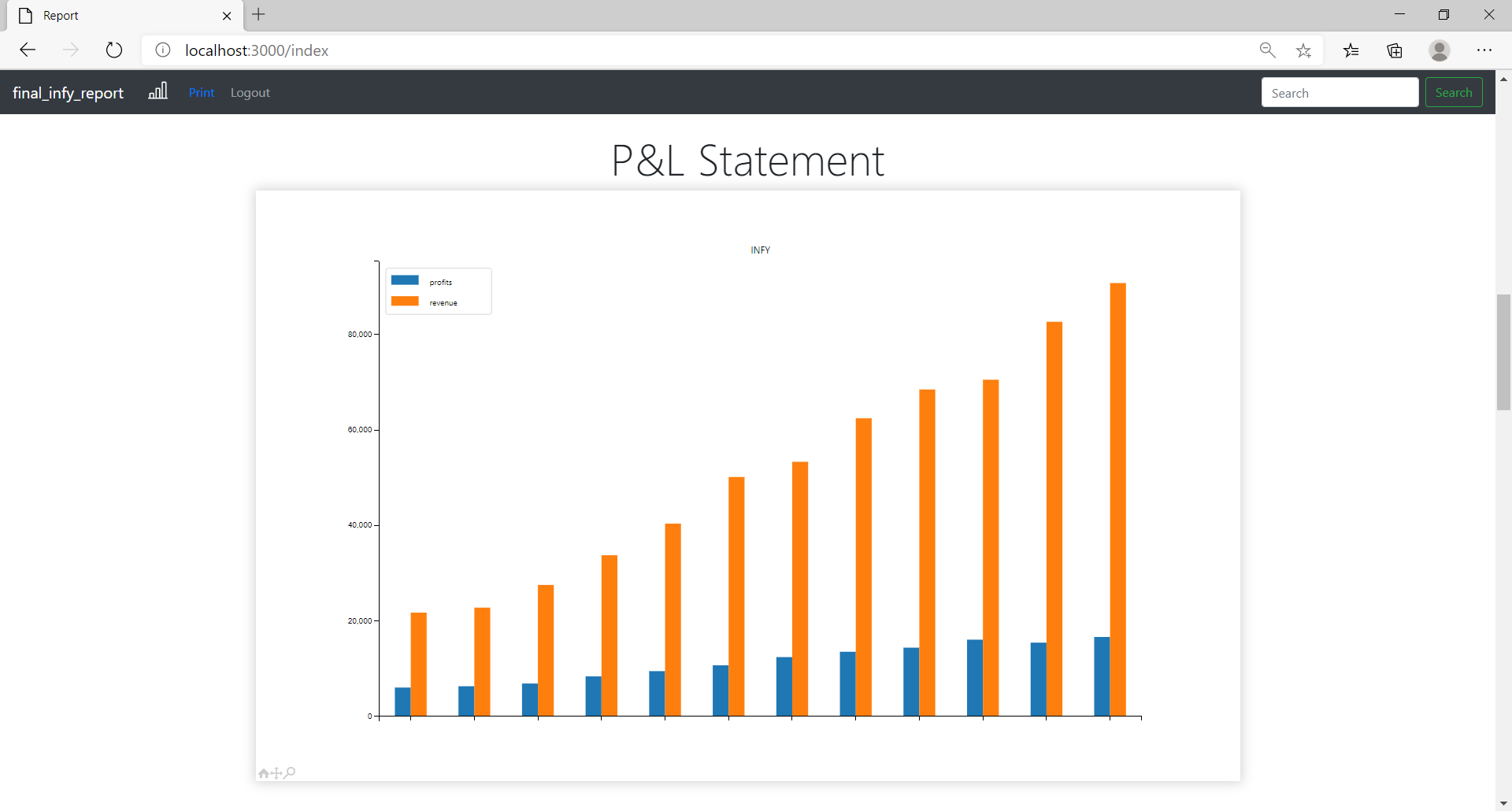
## GO TO http://localhost:3000/

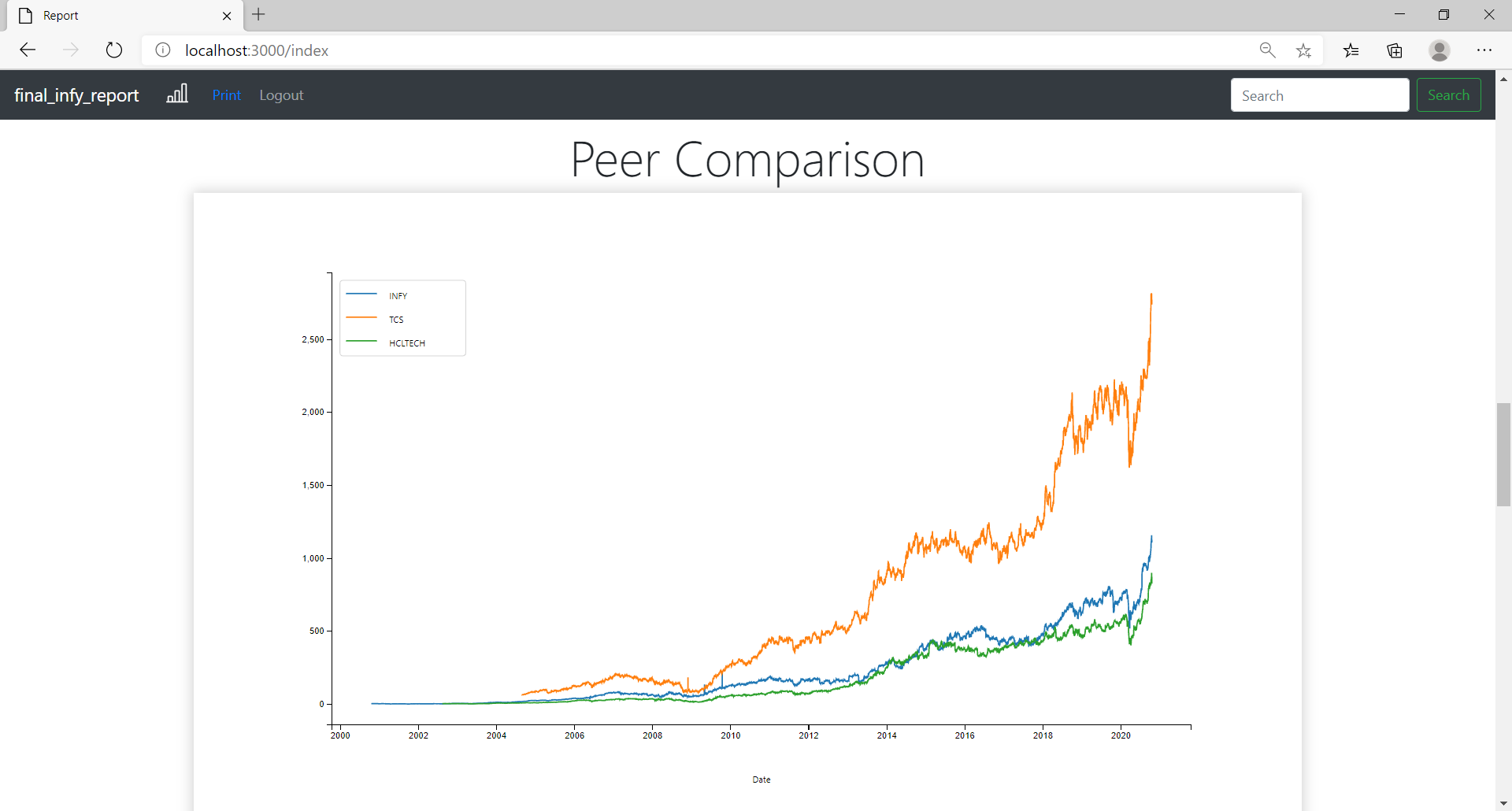


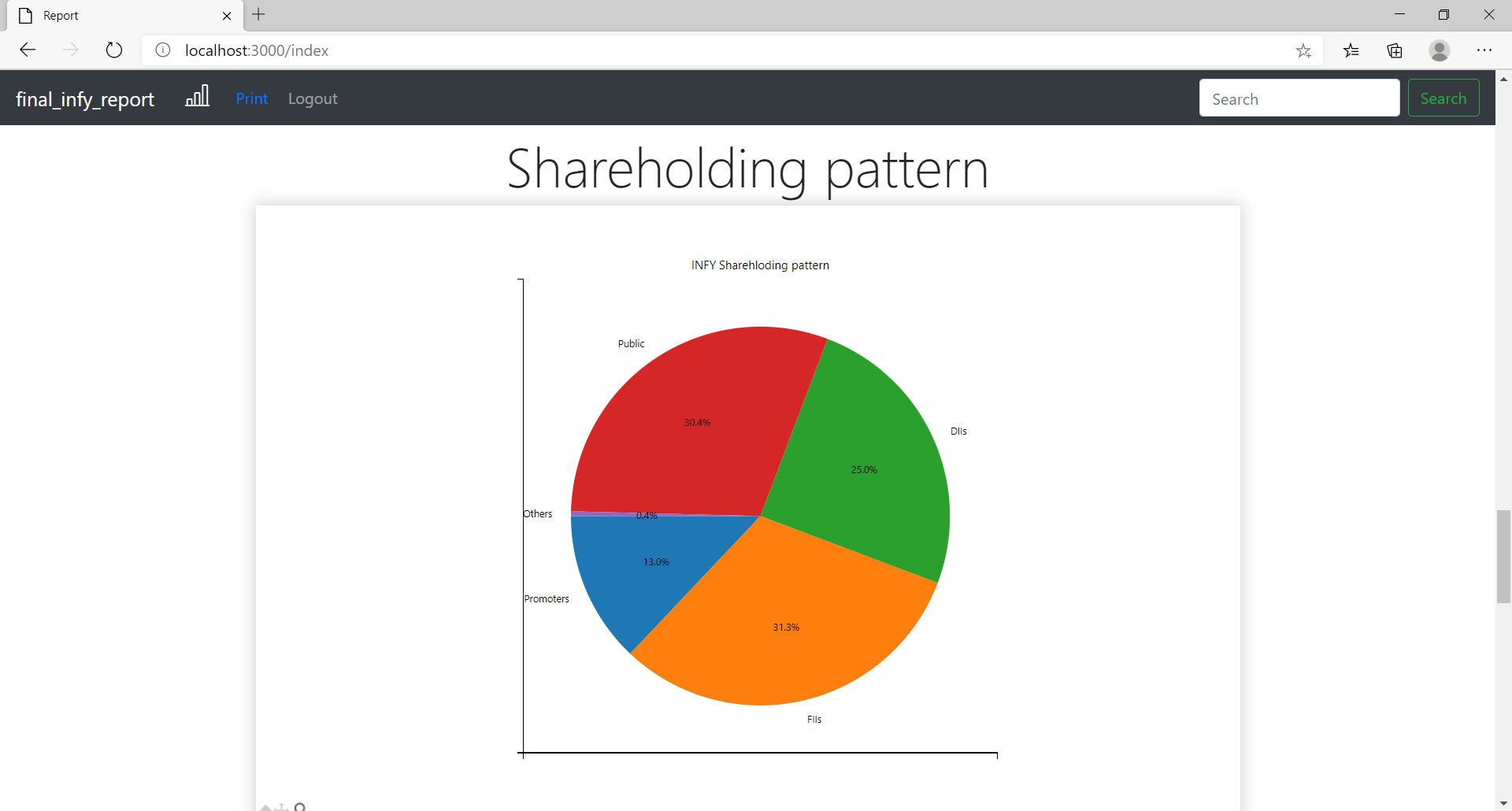
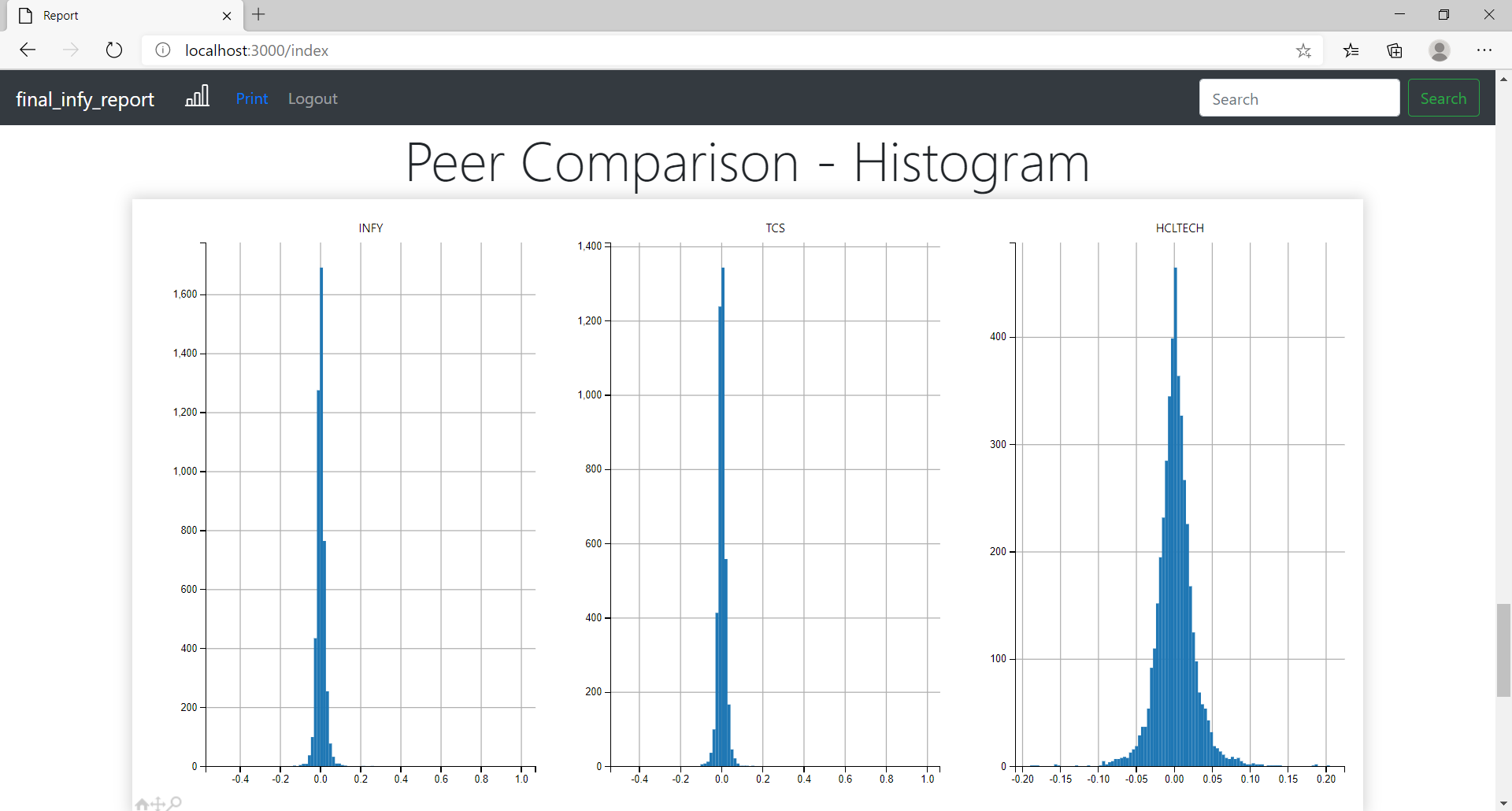


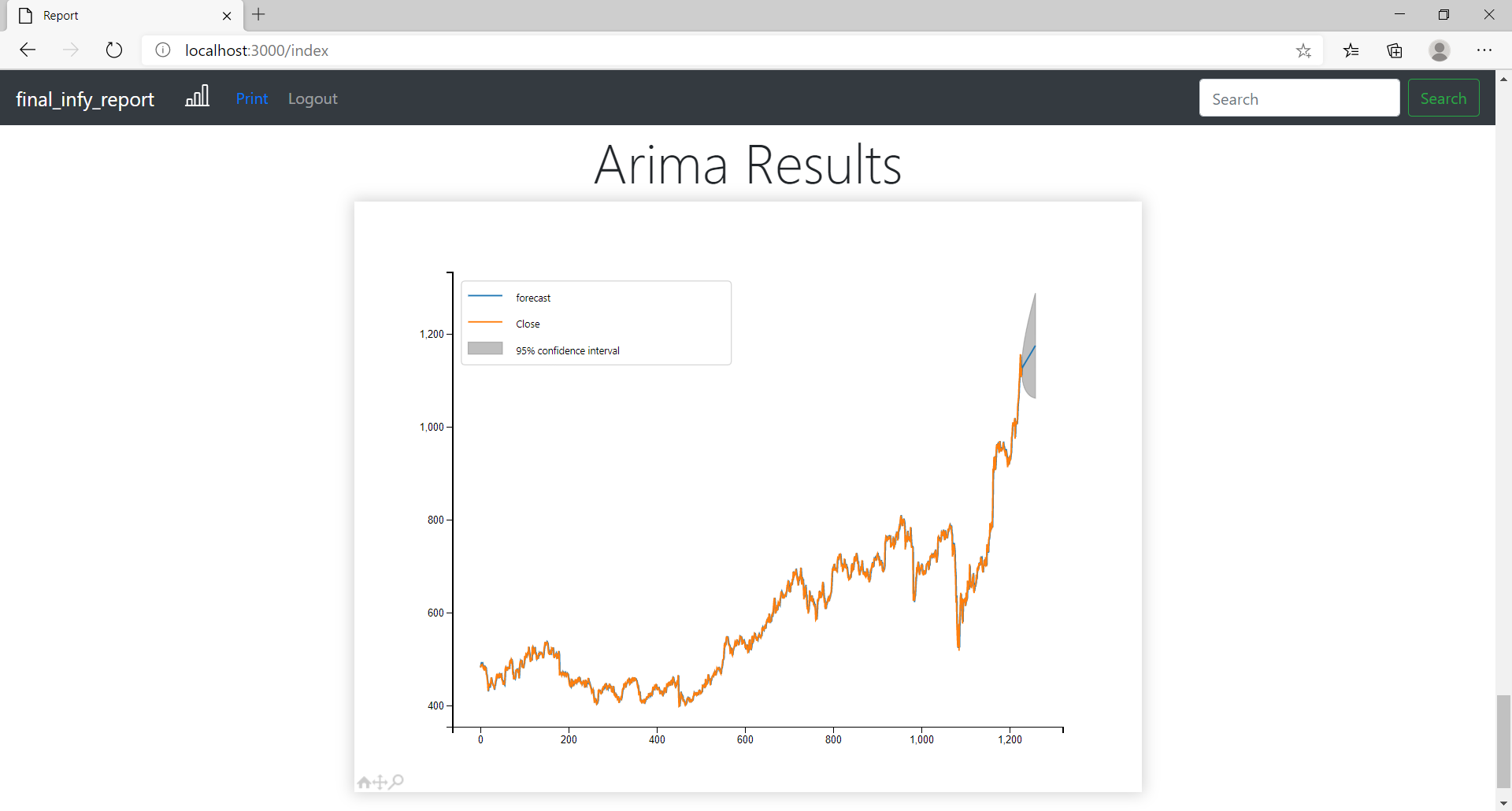










# Portfolio Analysis & Optimization

import pymongo  
myclient = pymongo.MongoClient("mongodb://localhost:27017/")  
  
dblist=myclient.list\_database\_names()  
if "test" in dblist:  
 mydb = myclient["test"]  
 coll = mydb['portfolios']  
   
 print("Connected to database")  
else:  
 print("Error connecting to database")

Connected to database

name=input("Enter the name of the portfolio ")

Enter the name of the portfolio it\_portfolio

choices=[]  
while True:  
 choice=input("Add the ticker symbol of the stock to add to your portfolio - ").upper()  
 choices.append(choice)  
 yn=input("Do you want to add more (y/n) ? ")  
 if yn == 'n' or yn == 'N':  
 break  
amount=int(input("Enter the total investment amount - "))

Add the ticker symbol of the stock to add to your portfolio - INFY  
Do you want to add more (y/n) ? y  
Add the ticker symbol of the stock to add to your portfolio - TCS  
Do you want to add more (y/n) ? y  
Add the ticker symbol of the stock to add to your portfolio - HCLTECH  
Do you want to add more (y/n) ? n  
Enter the total investment amount - 100000

try:  
 coll.insert\_one({"\_id":name,"stocks":choices,"investment\_amount":amount})  
 print("Portfolio created successfully")  
except:  
 print("Portfolio already exists")

Portfolio created successfully

for i in coll.find({"\_id":name}):  
 p\_stocks = i.get("stocks")  
 amount = i.get("investment\_amount")

no\_of\_stocks = len(p\_stocks)

portfolio=[] #contain ticker object  
for i in range(no\_of\_stocks):  
 portfolio.append(yf.Ticker(p\_stocks[i]+".NS"))

for i in range(no\_of\_stocks):  
 portfolio[i]=portfolio[i].history(period="5y") #contains ticker dataframe

#Computing Normed Returns   
for stock\_df in portfolio:  
 stock\_df['Normed Return'] = stock\_df['Close']/stock\_df.iloc[0]['Close']

#Dropping unwanted columns  
portfolio = rk.drop\_divnstksplit(portfolio)

allocations=[]  
for i in range(no\_of\_stocks):  
 print("Enter the portfolio allocation for - ",p\_stocks[i])  
 allocations.append(float(input()))  
if sum(allocations) != 1.0:  
 allocations = []  
 print("(Error) Please Re - Enter, The sum of all allocations must equal to 1")

Enter the portfolio allocation for - INFY  
0.3  
Enter the portfolio allocation for - TCS  
0.4  
Enter the portfolio allocation for - HCLTECH  
0.3

for stock\_df,allo in zip(portfolio,allocations):  
 stock\_df["Allocation"] = stock\_df['Normed Return'] \* allo

print("Your investment amount is ",amount)

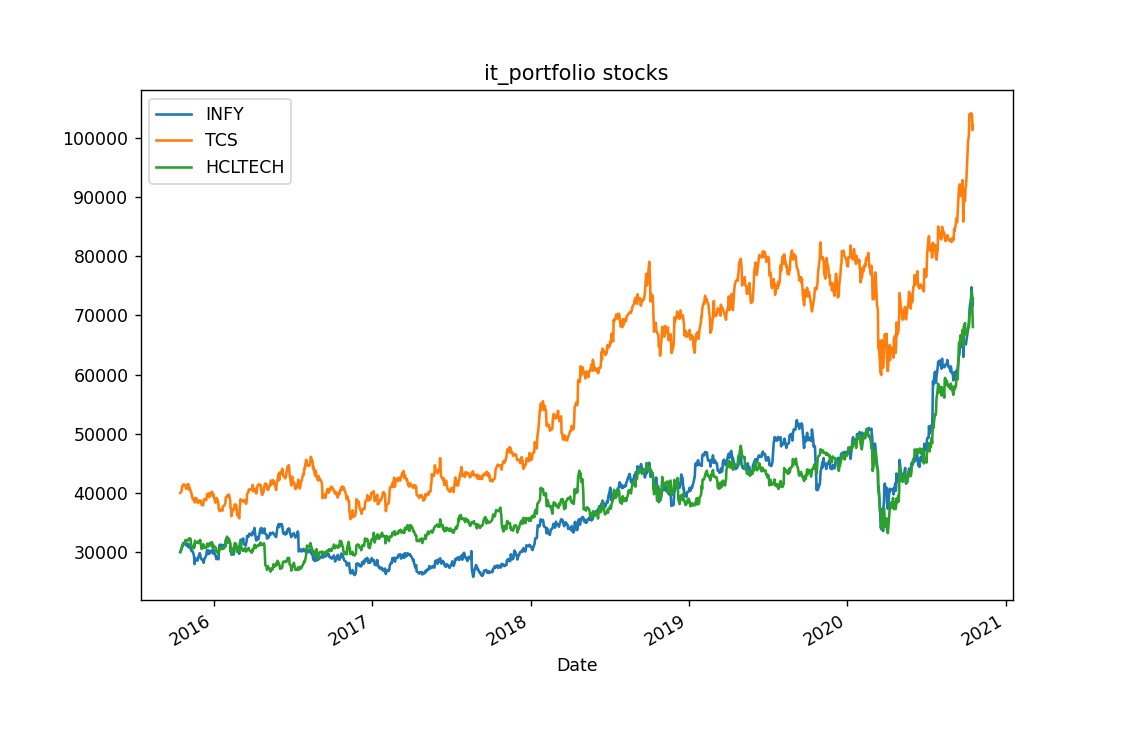
Your investment amount is 100000

for stock\_df in portfolio:  
 stock\_df["Position Values"] = stock\_df["Allocation"] \* amount

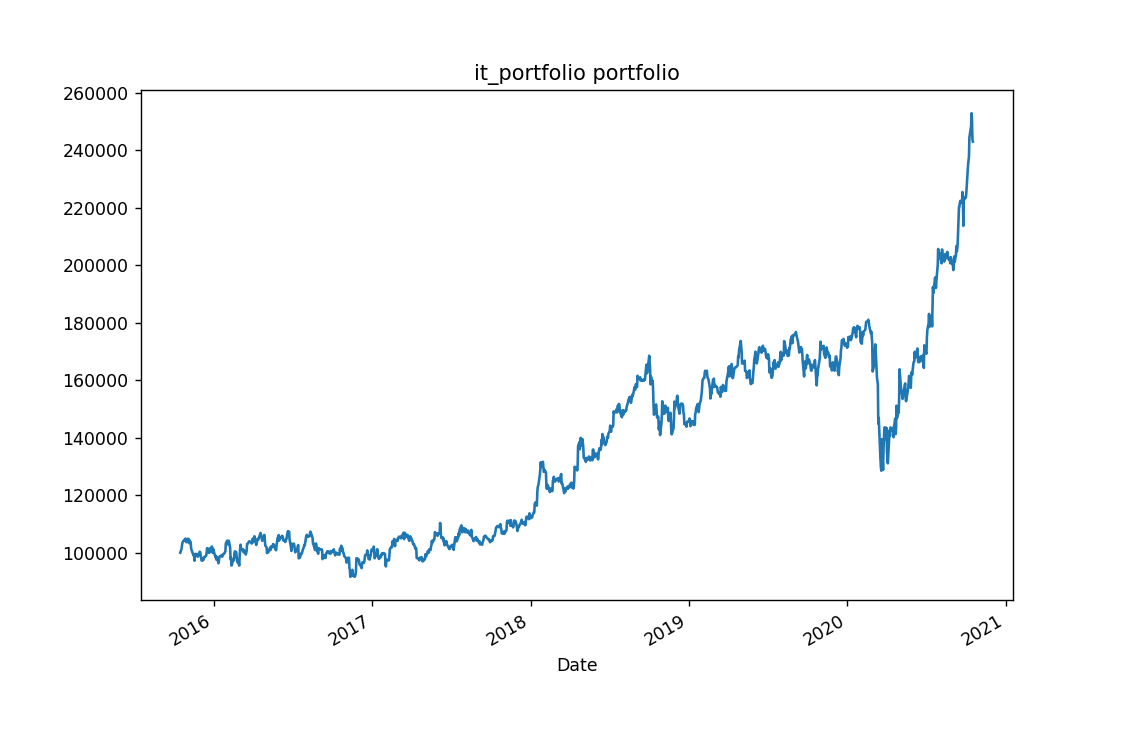
portfolio\_val=pd.DataFrame()  
portfolio\_val = pd.concat(portfolio,axis=1)

portfolio\_val = portfolio\_val['Position Values']  
portfolio\_val.columns = p\_stocks  
portfolio\_val['Total Pos'] = portfolio\_val.sum(axis=1)

fig\_obj=portfolio\_val[p\_stocks].plot(figsize=(9,6),title=name+" stocks");  
pfig1 = fig\_obj.figure



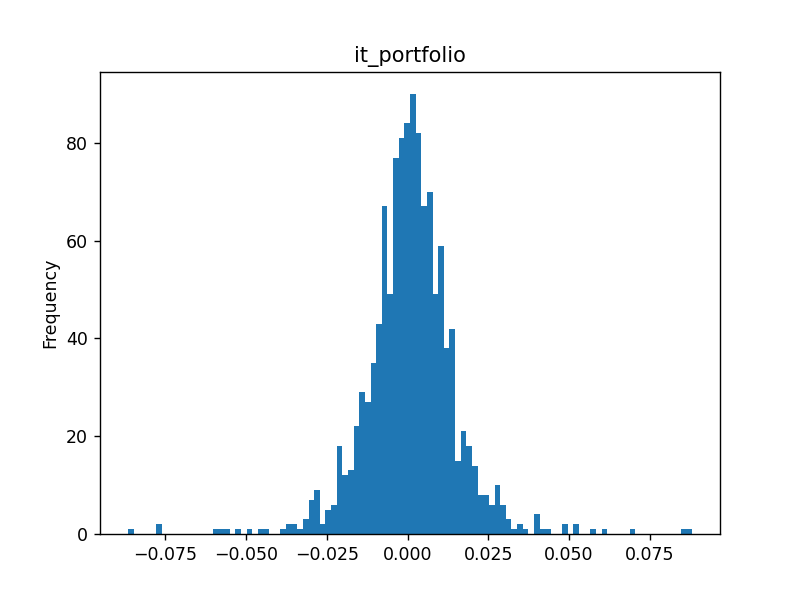
fig\_obj = portfolio\_val['Total Pos'].plot(figsize=(9,6),title=name+" portfolio");  
pfig2 = fig\_obj.figure



#Daily returns  
portfolio\_val['Daily Return'] = portfolio\_val['Total Pos'].pct\_change(1)  
#Cumulative returns i.e total return from the day i invested till now  
cum\_ret = 100 \* (portfolio\_val['Total Pos'][-1]/portfolio\_val['Total Pos'][0] -1 )  
print('Our return was {} percent!'.format(cum\_ret))

Our return was 143.0079838596117 percent!

fig\_obj = portfolio\_val['Daily Return'].plot(kind='hist',bins=100,title=name);  
pfig3 = fig\_obj.figure

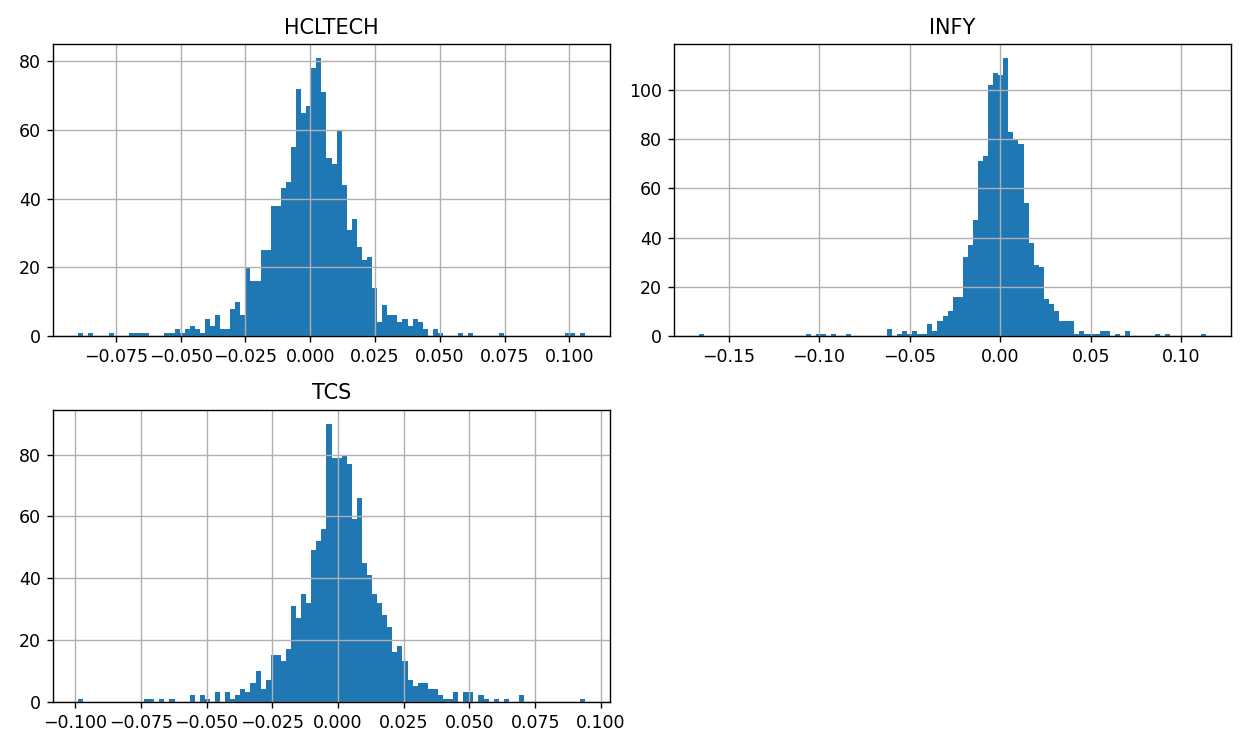


stks=pd.DataFrame()  
stks = pd.concat(portfolio,axis=1)  
stks = stks['Close']  
stks.columns = p\_stocks

stock\_daily\_ret = stks.pct\_change(1)  
stock\_daily\_ret.head()

#log returns  
log\_ret = np.log(stks/stks.shift(1))  
#log\_ret.fillna(0,inplace=True)  
log\_ret.head()

pfig4=[]  
fig\_obj = log\_ret.hist(bins=100,figsize=(10,6));  
plt.tight\_layout()  
pfig4 = plt.gcf()



num\_ports = 25000  
  
all\_weights = np.zeros((num\_ports,len(p\_stocks)))  
ret\_arr = np.zeros(num\_ports)  
vol\_arr = np.zeros(num\_ports)  
sharpe\_arr = np.zeros(num\_ports)  
  
for ind in range(num\_ports):  
  
 # Create Random Weights  
 weights = np.array(np.random.random(no\_of\_stocks))  
  
 # Rebalance Weights  
 weights = weights / np.sum(weights)  
   
 # Save Weights  
 all\_weights[ind,:] = weights  
  
 # Expected Return  
 ret\_arr[ind] = np.sum((log\_ret.mean() \* weights) \*252)  
  
 # Expected Variance  
 vol\_arr[ind] = np.sqrt(np.dot(weights.T, np.dot(log\_ret.cov() \* 252, weights)))  
  
 # Sharpe Ratio  
 sharpe\_arr[ind] = ret\_arr[ind]/vol\_arr[ind]

sharpe\_arr.max()

0.8122732422796562

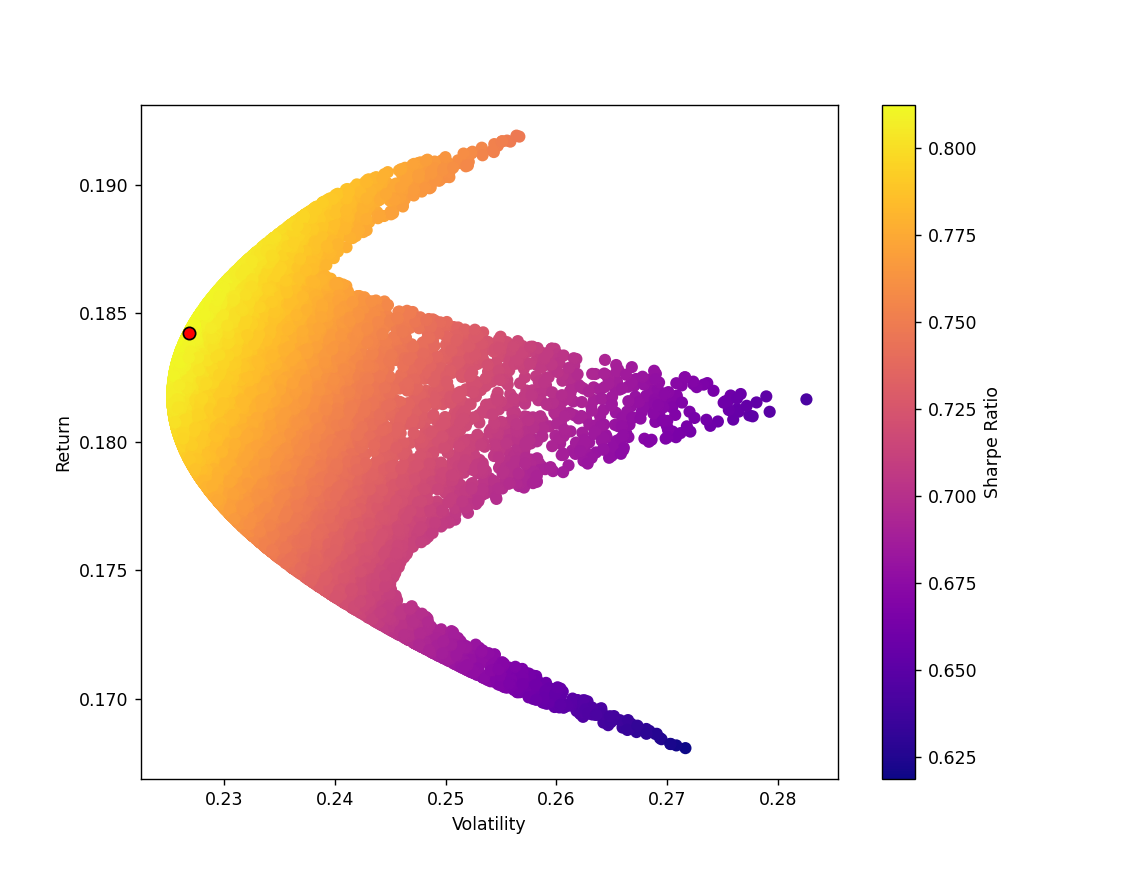
n = sharpe\_arr.argmax()

all\_weights[n,:]

array([0.25642669, 0.53011986, 0.21345346])

max\_sr\_ret = ret\_arr[n]  
max\_sr\_vol = vol\_arr[n]

plt.figure(figsize=(9,7))  
plt.scatter(vol\_arr,ret\_arr,c=sharpe\_arr,cmap='plasma')  
plt.colorbar(label='Sharpe Ratio')  
plt.xlabel('Volatility')  
plt.ylabel('Return')  
  
# Add red dot for max SR  
plt.scatter(max\_sr\_vol,max\_sr\_ret,c='red',s=50,edgecolors='black')  
  
pfig5 = plt.gcf()



# My weights portfolio

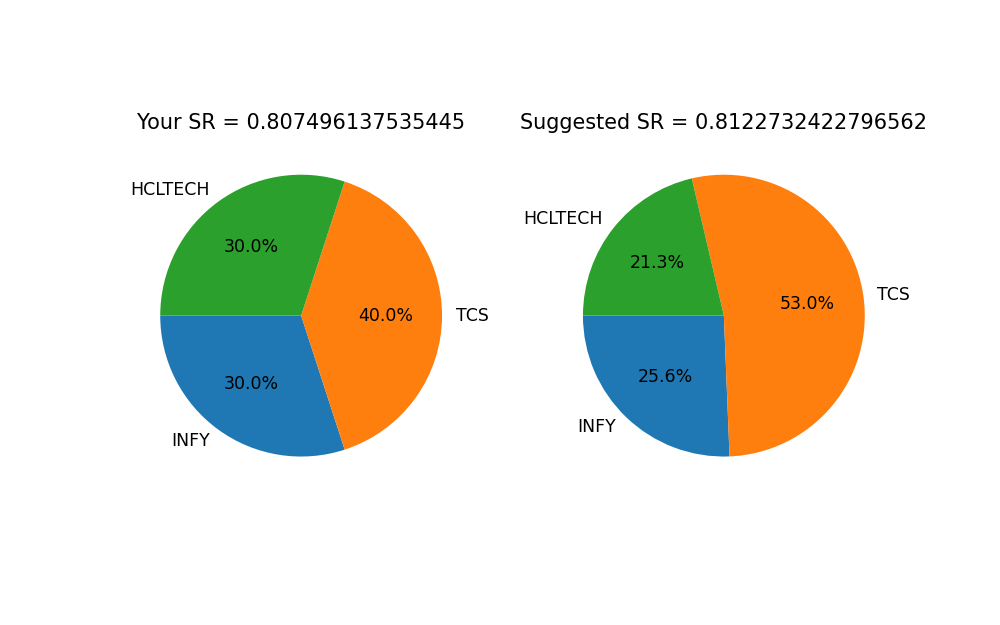
allocations = np.array(allocations)  
  
# Expected Return  
exp\_ret = np.sum((log\_ret.mean() \* allocations) \*252)  
  
# Expected Variance  
exp\_vol = np.sqrt(np.dot(allocations.T, np.dot(log\_ret.cov() \* 252, weights)))  
   
# Sharpe Ratio  
sharpe\_ratio = exp\_ret/exp\_vol

sharpe\_ratio

0.807496137535445

# Sharpe Ratio based on Your allocation V/S Suggested Sharpe Ratio

f = plt.figure(figsize =(8, 5))  
plt.subplot(1,2,1)  
  
plt.pie(allocations,labels=p\_stocks,autopct='%1.1f%%',startangle=180);  
plt.title("Your SR = "+str(sharpe\_ratio));  
  
plt.subplot(1,2,2)  
  
plt.pie(all\_weights[n,:],labels=p\_stocks,autopct='%1.1f%%',startangle=180);  
plt.title("Suggested SR = "+str(sharpe\_arr.max()));



# Save portfolio analysis report

### To save run below cells

json1 = mpld3.fig\_to\_dict(pfig1)  
json2 = mpld3.fig\_to\_dict(pfig2)  
json3 = mpld3.fig\_to\_dict(pfig3)  
json4 = mpld3.fig\_to\_dict(pfig4)  
json5 = mpld3.fig\_to\_html(pfig5)  
json6 = mpld3.fig\_to\_dict(f)

myquery={ "\_id": name }  
newvalues = { "$set": {"pfig1":json1,"pfig2":json2,"pfig3":json3,"pfig4":json4,"hpfig5":json5,"pfig6":json6} }  
coll.update\_one(myquery, newvalues)

<pymongo.results.UpdateResult at 0x1fa629b3748>

myclient.close()

# Go to http://localhost:3000/ to view Portfolio

