A Monte Carlo Project on
Portfolio Optimization for different Risk Measures

Submitted by

Deepak Kumar
Deepak Punjabi
Tanmay Sah

Under the guidance of

Dr. Yunan Liu
Ms. Yining Huang

INDEX

Introduction

Portfolio is a collection of investments held by an investment company, hedge fund, financial institution or individual. Investments. Investments can be anything like stocks, bonds, bitcoin, fixed income and so on. Portfolio optimization is the process of selecting the best portfolio (asset distribution), out of the set of all portfolios being considered, according to some objective. The objective typically maximizes factors such as expected return and minimizes costs like financial risk. Factors being considered may range from tangible (such as assets, liabilities, earnings or other fundamentals) to intangible (such as selective divestment).

There are different ways to optimize the portfolio. Harry Markowitz was the first one to popularize the portfolio theory. It assumes that an investor wants to maximize a portfolio's expected return contingent on any given amount of risk. For portfolios that meet this criterion, known as efficient portfolios, achieving a higher expected return requires taking on more risk, so investors are faced with a trade-off between risk and expected return.

There are different ways to optimize the portfolio

1.Quadratic programming

2.Nonlinear programming

3.Mixed integer programming

4.Meta-heuristic methods

5.Stochastic programming for multistage portfolio optimization

6.Copula based methods

7.Principal component-based methods

8.Deterministic global optimization

9.Hamilton Jacobi Bellman Equations

10.Monte Carlo Simulation

So, in this paper our aim is to use Monte Carlo simulation techniques to optimize the portfolio for best return and at the same time aim is that risk is minimized.

Risks are either known, unknown or unknowable. When we either do not know the outcome or we cannot estimate the outcome then in such situation risk arises.

So, in our case we are looking for a specific risk Market Risk which is the risk of losses in positions arising from movements in market prices. So, to quantify the risks we have different measures and we have chosen three measure to quantify the risk which are volatility, Value at Risks, Expected Shortfall.

Use of Monte Carlo in Portfolio Optimization

Monte Carlo simulation is computer simulation of a stochastic model repeated numerous times to estimate the probability distribution of the outcome of the stochastic model. This is useful when the probability distribution is not possible to derive analytically, either because it is too complex or because the stochastic variables of the model are not from simple, well-behaved probability distributions. Monte Carlo

simulation allows for arbitrary probability distributions so that very rare events can also be modelled.

Objectives of the project

1.To analyze the stocks history from different industries to check independencies

2.To verify the conditions to apply Geometric Brownian motions

3.Application of Monte Carlo Simulations to get possible random weight combinations for the portfolio

4.To analyze and calculate different Risk Measures used in industries: Volatility, VaR and Expected Shortfall

5.To Find the optimized portfolio having minimum risks and maximum returns

6.Generate each Stock's paths for next 1 year using Geometric Brownian motion

7.Using MC simulations to calculate different risk measures for different generated weights

8.To find the optimized weight of each stocks in newly generated portfolio having maximum return and minimum risks

9.To compare the return, risks and optimized weight of historical portfolio and generated future portfolio

Portfolio Creation

Stocks used: Google(G), Intel(I), Johnson & Johnson (JJ) and Toyota Motors (TOY)

Industries covered: Internet services and products, Computer manufacturing, Medical and pharmacy and Automobile

Stocks data Used: 5 years data from 04-17-2014 to 04-17-2019

Google, Intel and JJ highly correlated

Diversification of stocks

Diversification is a technique that reduces risk by allocating investments among various financial instruments, industries, and other categories. It aims to maximize returns by investing in different areas that would each react differently to the same event.

So, in this paper we have selected stocks from different industries and based on their market capitalization so that we have a good diversification in our portfolio overall

Even though any two stock are correlated but our aim is that overall portfolio is well diversified

Portfolio Risks

1. Volatility-Volatility is a statistical measure of the dispersion of returns for a given security or market index. In most cases, the higher the volatility, the riskier the security.

2. Value at Risks-Value at risk (VaR) is a statistic that measures and quantifies the level of financial risk within a firm, portfolio or position over a specific time frame. This metric is most commonly used by investment and commercial banks to determine the extent and occurrence ratio of potential losses in their institutional portfolios.

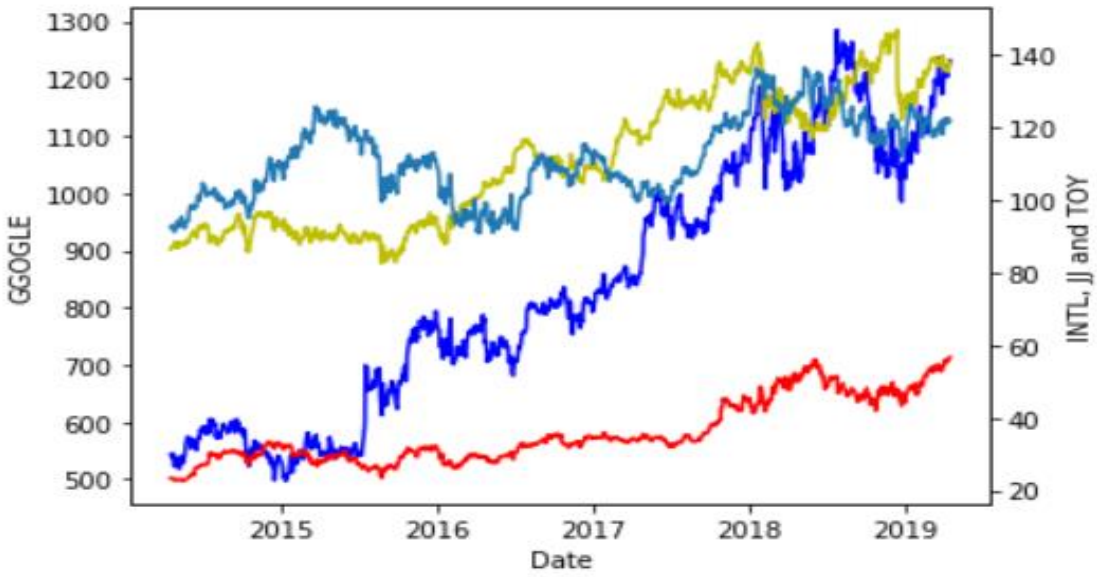
3. Expected Shortfall-Expected shortfall (ES) is a risk measure—a concept used in the field of financial risk measurement to evaluate the market risk or credit risk of a portfolio. The "expected shortfall at q% level" is the expected return on the portfolio in the worst q% of cases. ES is an alternative to value at risk that is more sensitive to the shape of the tail of the loss distribution.

4. Expected shortfall is also called conditional value at risk (CVaR), average value at risk, (AVaR) and expected tail loss (ETL)

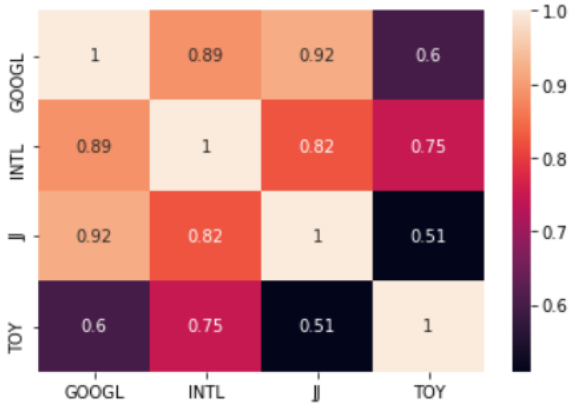
Historical Portfolio Statistical Analysis

Our Stock has four stocks namely Google, Intel, Johnson & Johnson and Toyota Motors. In an order to see how our portfolio have behaving in the past and find the parameters of Geometrical Brownian Motion and generate each stock path, we need to analyze the historical performance or each stock and statistical relations between them.

We have accumulated data of 5 years stock performance of each stock from 04-17-2014 to 04-17-2019. We can see the historical path of each stocks



Here Google is blue in color and appreciated consistently and most among all four giving highest return to investor. Intel is red in color, Johnson and Johnson is yellow and Toyota is purple. All three stocks besides google didn't appreciated most. We can see the correlation between stocks to quantify the relation between their movements with this heatmap created through Seaborn library.



There is strong correlation between Google, Intel and Johnson. Only explanation that help us quantify this correlation is that, they were in similar phase of their growth. After crisis, almost all major American companies have performed well having huge demand and revenue in respective industries.

Our focus is not stock price, rather our interest lies in stock returns. We are going to calculate logarithmic return rather than simple return for the sake of ease and applicability of Stochastic calculus. Also, return and logarithmic return doesn't have much difference in it

Below are the statistics of our stock return like correlations among return, their mean, standard deviations, mean and quantiles. How we are going to use GBM to generate stocks would be explained in later part of the report. But we are going to show the mean and standard deviation of each stocks which would be used in algorithm to generate them.

Statistics of each stock: We can see that Intel stock return has highest standard deviation but highest mean return. This makes it worth for risk loving investors.

	GOOGL	INTL	JJ	TOY
count	1257.000000	1257.000000	1257.000000	1257.000000
mean	0.000651	0.000703	0.000373	0.000219
std	0.014455	0.015815	0.010024	0.011721
min	-0.055662	-0.095432	-0.105781	-0.062797
25%	-0.005941	-0.007499	-0.004194	-0.005800
50%	0.000784	0.000943	0.000388	0.000341
75%	0.008269	0.008671	0.005905	0.006877
max	0.150645	0.100315	0.048395	0.050474

Below is the correlation between return of each stocks in portfolio. Although stock return has strong correlation between them, return on stocks are not that highly correlated. We are going to account this correlation in our generation of stocks values for next year using Multivariate Geometric Brownian Motion.

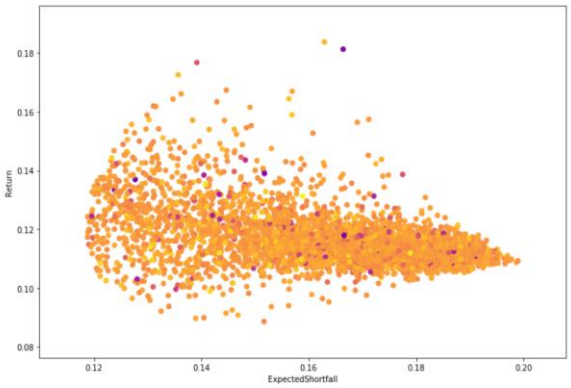
	GOOGL	INTL	JJ	TOY
GOOGL	1.000000	0.439011	0.375379	0.424550
INTL	0.439011	1.000000	0.355773	0.403426
JJ	0.375379	0.355773	1.000000	0.365957
TOY	0.424550	0.403426	0.365957	1.000000

Optimization of Stock Portfolio

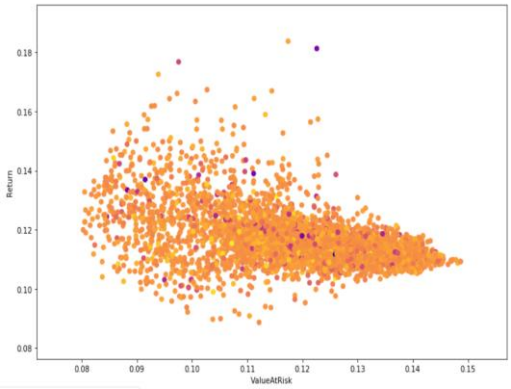
For finding the optimization of portfolio we are going to use Monte Carlo simulations of large number of random weights and calculating all three risk measures, Volatility or standard deviation, Value at Risk, and Expected Shortfall for different combination of weights and weights having minimum risk and highest return would be selected.

Following algorithms have been used to optimize the historical weight:

1. Finding log return of four stocks in portfolio
2. Run a loop of 10000 to do the following:
2.1. Generate 4 random weights between 0 and 1 using uniform distribution
2.2. Rebalancing all four weight by dividing them by sum of generated weight to aggregate sum equal to 1
3. Multiply weight array with log return array and sum them up to get single return of portfolio
4. Store return of portfolio in an empty array each time for 10000 combination of weights
5. Calculate Volatility for each weight combination by finding dot product of covariance matrix and weight and then again doing dot product of the result with transpose of weigh array. Then square root of overall result is taken
6. We then calculate ratio between Return and volatility for each 10000 weight combinations and store them in empty array
7. In step 4 we have stored all portfolio return, now we are going to find the value at risk with 90% confidence interval by finding 10^{th} percentile lowest return from below of the array
8. Calculate the ratio between Value at Risk and return and store them in separate array
9. Expected Shortfall is calculated by finding the mean of lowest 10 percent return
10. Calculating ratio between Expected Shortfall and return in separate empty array for 10000 weight combinations.
11. We then find the weight on which Volatility, Value at risk and Expected Shortfall was minimum respectively.



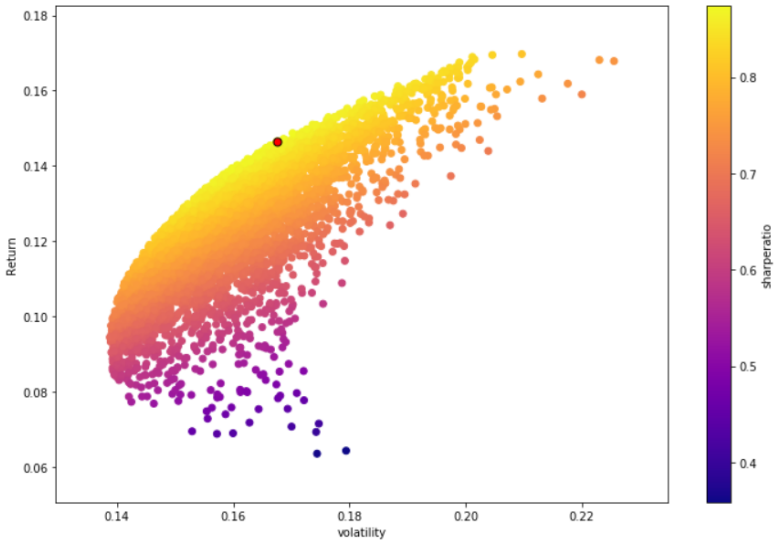
Expected Shortfall vs Return

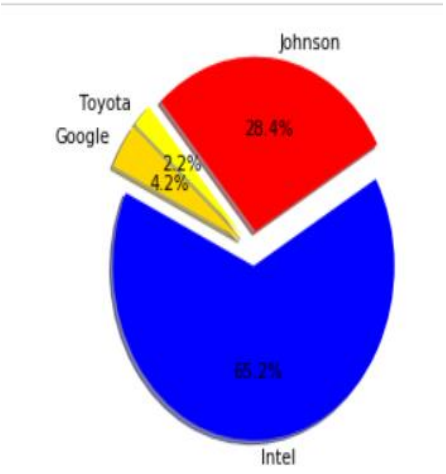


Plot of Value at Risk vs Return

Optimized Weight for 10000 weight combinations comes out to be red dot in plot below which is 65.2% Intel, 28.4% Johnson & Johnson, 4.2% Google and 2.2% Toyota Motors.

Expected Volatility and Return





We will compare this optimized weight with optimized future generated portfolio in later part of this report.

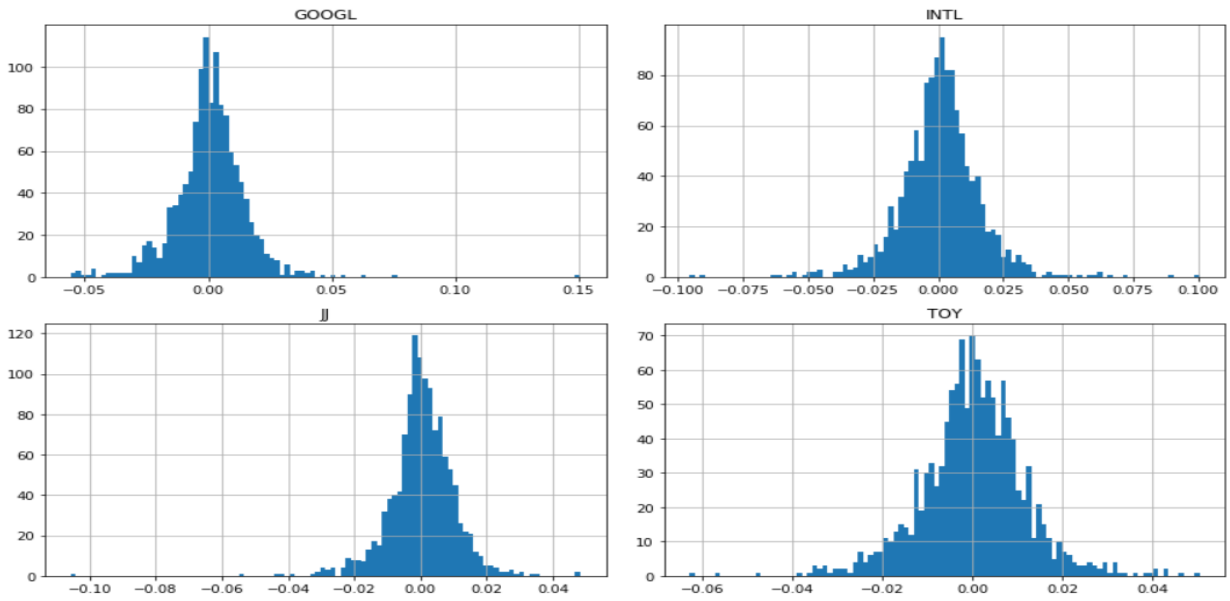
Geometric Brownian Motions

One of the common assumptions in stochastic calculus is that Stock Returns follows a random path of Geometric Brownian Motion. Before applying GBM to we need to see if our stock log return satisfies each condition of GBM or not.

Following are the conditions of GBM.

- The company is a going concern, and its stock prices are continuous in time and value.
- Stocks follow a Markov process, meaning only the current stock price is relevant for predicting future prices.
- The proportional return of a stock is log-normally distributed.
- The continuously compounded return for a stock is normally distributed.

First two conditions are easily satisfied. We tried to see the distribution of our log return and it can easily be noted that it follows normal distribution.



Generating Stock paths using GBM

After we have historical means and Variances of the individual stocks and correlational between them. We are assuming that that will hold true in future. And using those assumptions We are generating future stock prices for all the stocks in the portfolio using Multivariate Geometric Brownian motion and Cholesky decomposition.

To Generate future stock price, we are using Geometric Brownian motion and while generating normal (0,1) random variables. We are not generating independent random Variables but correlated Normal (0,1) variables to use them in Multivariate GBM using python.

```
Google,Intel,Jhonson,Toyota=np.random.multivariate_normal(mean,coreltion,1000).T
```

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right).$$

Historical Mean daily return and Variance of the stocks in portfolio:

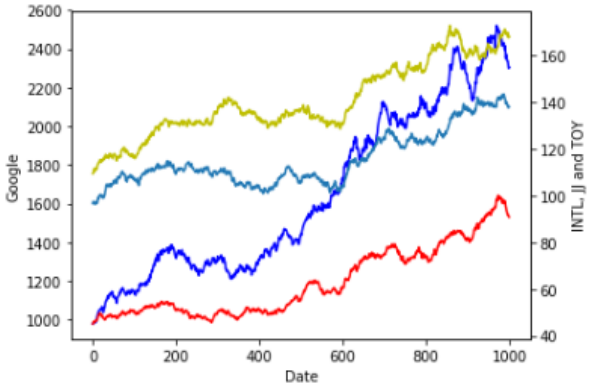
	GOOGL	INTL	JJ	TOY
count	1257.000000	1257.000000	1257.000000	1257.000000
mean	0.000651	0.000703	0.000373	0.000219
std	0.014455	0.015815	0.010024	0.011721
min	-0.055662	-0.095432	-0.105781	-0.062797
25%	-0.005941	-0.007499	-0.004194	-0.005800
50%	0.000784	0.000943	0.000388	0.000341
75%	0.008269	0.008671	0.005905	0.006877
max	0.150645	0.100315	0.048395	0.050474

Historical Correlation between the stocks in portfolio:

	GOOGL	INTL	JJ	TOY
GOOGL	1.000000	0.439011	0.375379	0.424550
INTL	0.439011	1.000000	0.355773	0.403426
JJ	0.375379	0.355773	1.000000	0.365957
TOY	0.424550	0.403426	0.365957	1.000000

Future stock prices of the stocks in portfolio using Monte Carlo Simulation:

	Google	Intel	Johnson	Toyota
0	978.296229	45.072198	109.436569	96.662106
1	977.126442	45.534998	109.629412	97.504740
2	979.578859	45.941589	110.644627	97.375224
3	982.230626	45.739621	110.305949	97.238455
4	983.299284	46.202324	111.658210	97.124608

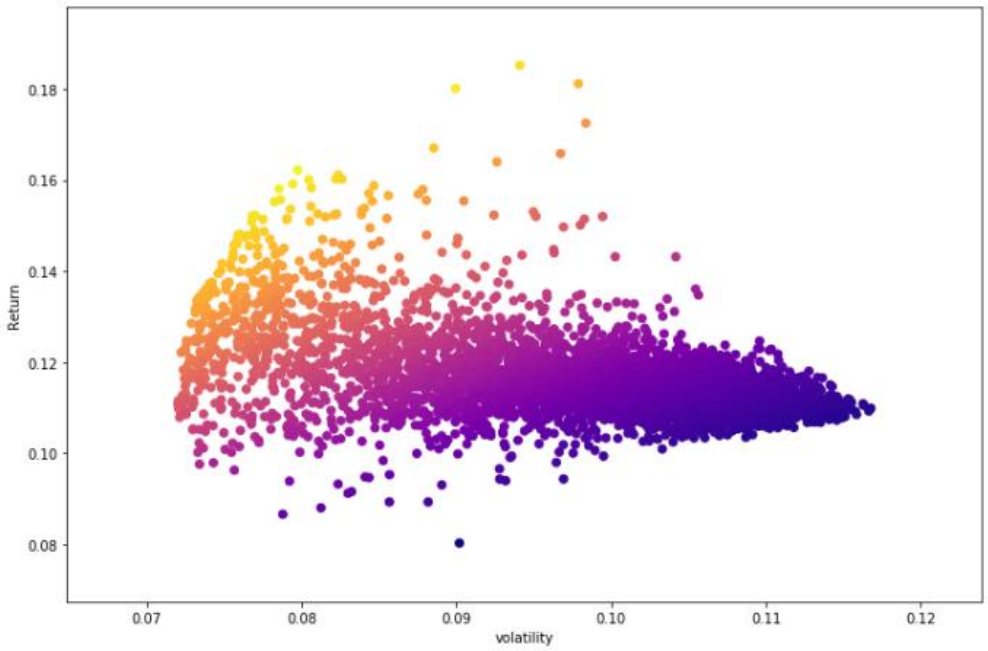


Optimization of Generated portfolio using Mean-Variance optimization

After we have generated the future path of the individual stock prices using Monte Carlo and GBM. Our aim is to make the portfolio of stocks.

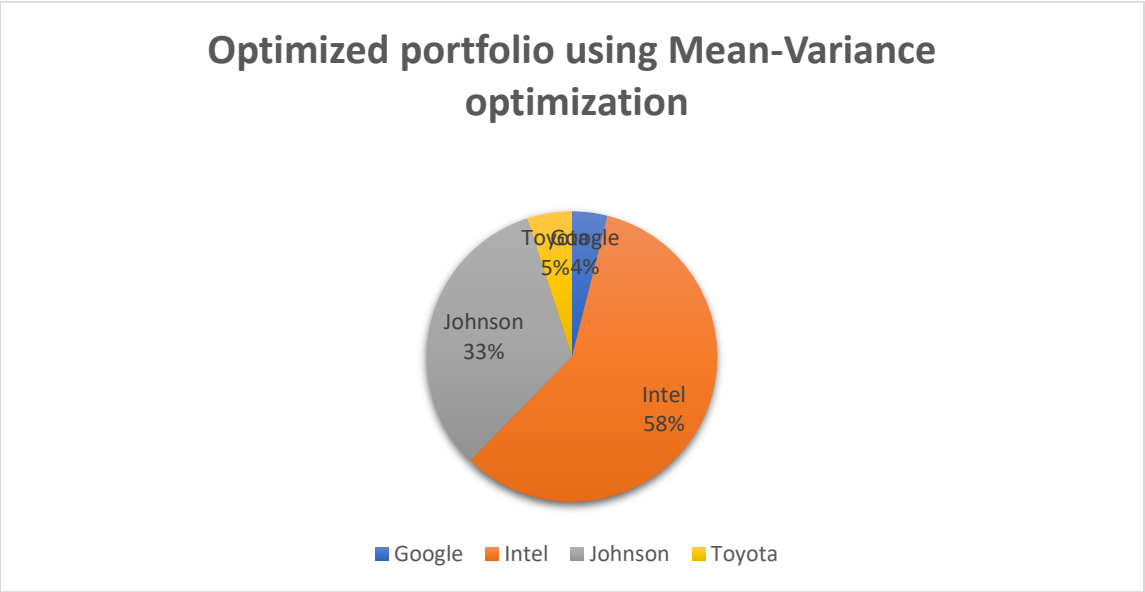
We are Generating 5000 random weights using Monte-Carlo simulation to build a portfolio of stocks. And our aim is to optimize the portfolio. We will calculate Standard deviation and Mean returns of portfolio for all the 5000 weights and check which combination and weights maximizes Risk Adjusted returns I.e. maximizes the ratio of Returns to Risk so that we can invest in that portfolio.

Plot of Mean and Variance of different portfolios weights (Efficient frontier):



Optimized portfolio using Mean-Variance optimization

Mean variance optimization techniques says that we should invest 33% in Johnson and Johnson,5% in Toyota, 4% in Google, and 58% in Intel to maximize the Risk Adjusted return.

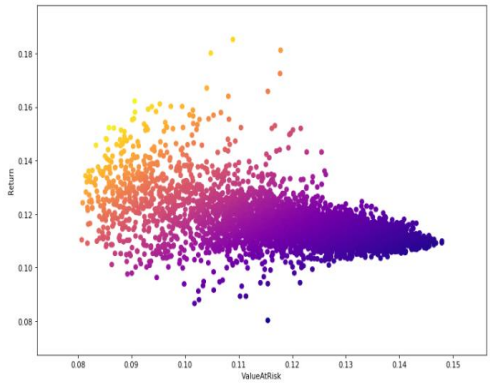


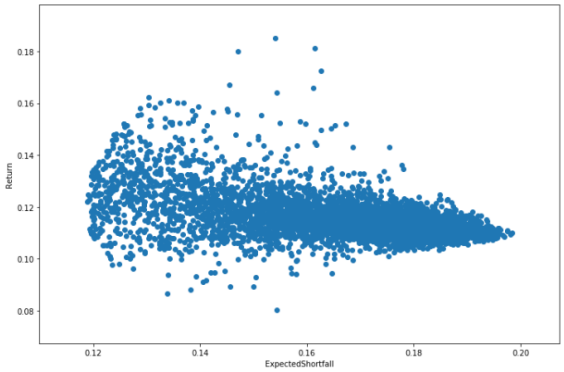
Optimization of Generated portfolio using Mean-Value at Risk and Mean-Expected Shortfall

Mean-Variance optimization considers Volatility as a Risk measure which encompasses upside and downside surprises in stock prices. But often in Risk Management we just want to see what the worst-case scenarios will be. In Risk Management Value at Risk and Expected shortfall are used as alternate Risk Measures as a more coherent measures as compared to Volatility as they as are the better proxy of risk for worst case scenarios.

Hence, we are optimizing our portfolio using Mean-Value at Risk and Mean-Expected Shortfall:

Plot of Mean-VAR and Mean-ES of different portfolios weights (Efficient frontier):

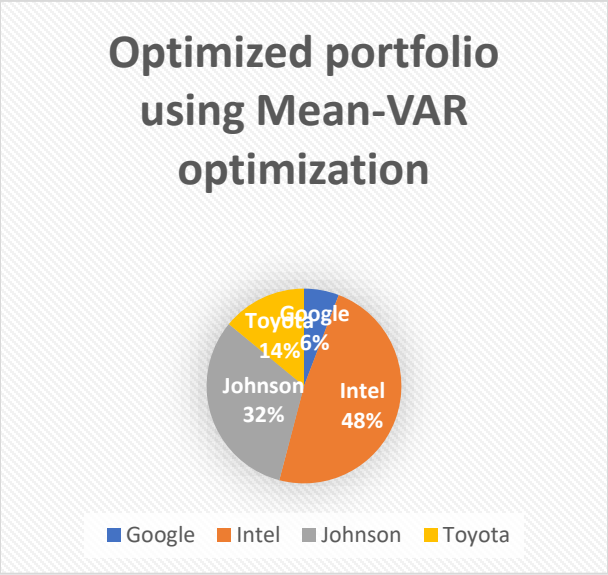


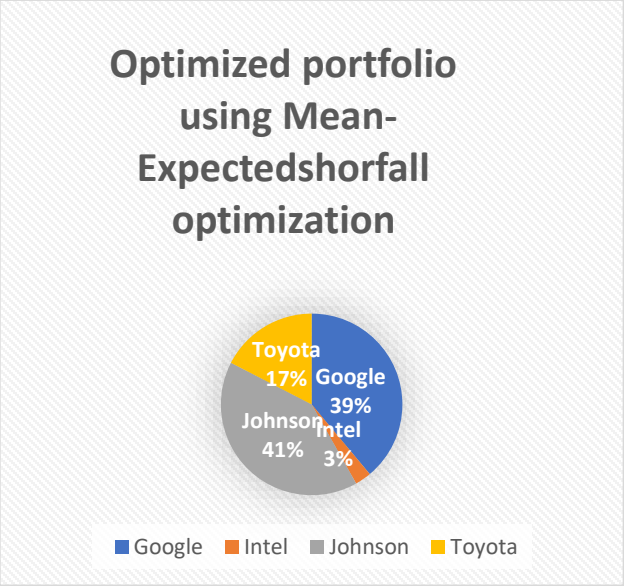


Optimized portfolio using Mean-VAR and Mean-ES optimization:

Mean Value at Risk optimization technique says that we should invest 32% in Johnson and Johnson,14% in Toyota, 6% in Google, and 48% in Intel to maximize the Risk Adjusted return.

Mean Expected-shortfall optimization technique says that we should invest 41% in Johnson and Johnson,17% in Toyota, 39% in Google, and 3% in Intel to maximize the Risk Adjusted return.





Summary

Different portfolio optimization suggests different weights to invest. It depends on the investor presence on what to choose. If you care about the worst-case scenarios, then Mean-Expected shortfall optimization makes much more sense as Expected shortfall captures the tail risk of portfolio.

Citations

- Historical Stocks Data from Yahoo Finance
- Paper: Simulating Stock Prices Using Geometric Brownian Motion: Evidence from Australian Companies by Krishna Reddy The University of Waikato, New Zealand, krishna@waikato.ac.nz Vaughan Clinton University of Waikato, New Zealand
- Investopedia
- Wikipedia
- Article on “Sharpe Ratio” from Wallstreetmojo.com
- Blog on “Exposing Downside Variance” from r3analytics.com

Appendix

This part just includes codes of our project for reference.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
%matplotlib inline
```

In [5]:
```
google=pd.read_csv("C:/Class/Monte carlo/project/GOOGL.csv",index_col='Date',parse_dates=True)
google.drop(['Open','High','Low','Close','Volume'],axis=1,inplace=True)
google.rename(columns={'Adj Close':'GOOGL'},inplace=True)
```

In [6]:
```
type(google)
```

Out[6]:
```
pandas.core.frame.DataFrame
```

In [7]:
```
intel=pd.read_csv("C:/Class/Monte carlo/project/INTC.csv",index_col='Date',parse_dates=True)
intel.drop(['Open','High','Low','Close','Volume'],axis=1,inplace=True)
intel.rename(columns={'Adj Close':'INTL'},inplace=True)
```

In [8]:
```
johnson=pd.read_csv("C:/Class/Monte carlo/project/JNJ.csv",index_col='Date',parse_dates=True)
johnson.drop(['Open','High','Low','Close','Volume'],axis=1,inplace=True)
johnson.rename(columns={'Adj Close':'JJ'},inplace=True)
```

In [9]:
```
toyota=pd.read_csv("C:/Class/Monte carlo/project/TM.csv",index_col='Date',parse_dates=True)
toyota.drop(['Open','High','Low','Close','Volume'],axis=1,inplace=True)
toyota.rename(columns={'Adj Close':'TOY'},inplace=True)
```

In [10]:
```
portfolio=pd.concat([google,intel,johnson,toyota],axis=1)
```

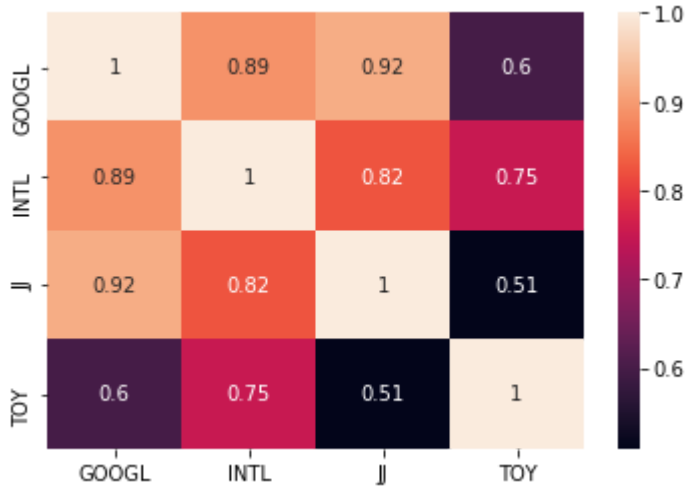
In [12]:
```
portfolio.head()
```

Out[12]:

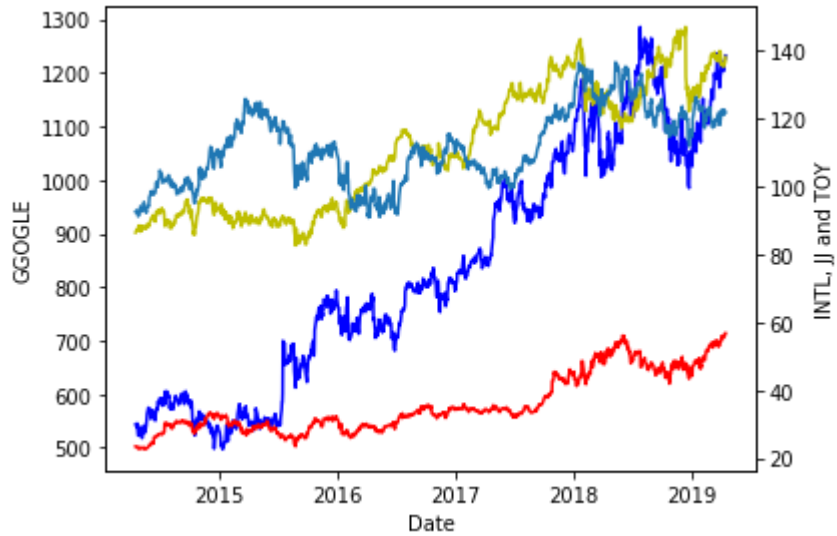
	GOOGL	INTL	JJ	TOY
Date				
2014-04-17	543.340027	23.439812	86.352402	92.458313
2014-04-21	539.369995	23.361790	87.259903	92.988976
2014-04-22	545.500000	23.266438	87.416962	92.800674
2014-04-23	537.510010	23.188421	87.451874	92.569572
2014-04-24	534.440002	23.188421	87.225006	91.619484

In [79]:
```
sns.heatmap(portfolio.corr(),annot=True)
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x23934ca4ba8>

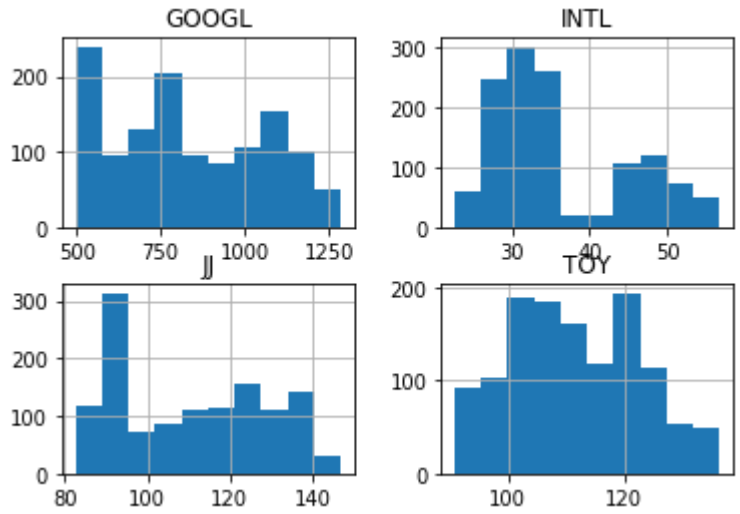


In [18]:
```
fig, ax1 = plt.subplots()

ax1.set_xlabel('Date')
ax1.set_ylabel('GGOGLE')
ax1.plot(portfolio['GOOGL'],'b')
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

ax2.set_ylabel('INTL, JJ and TOY')  # we already handled the x-label with ax1
ax2.plot(portfolio['INTL'],'r')
ax2.plot(portfolio['JJ'],'y')
ax2.plot(portfolio['TOY'])
ax2.tick_params(axis='y')

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



In [83]:
```
portfolio.hist()
```

Out[83]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000023936406978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000023936465160
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002393641E5F8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000239364A2908
>]],
      dtype=object)
```



In [9]:
```
log_ret=np.log(portfolio/portfolio.shift(1))
```

In [10]:
```
log_ret
log_ret.describe()
```

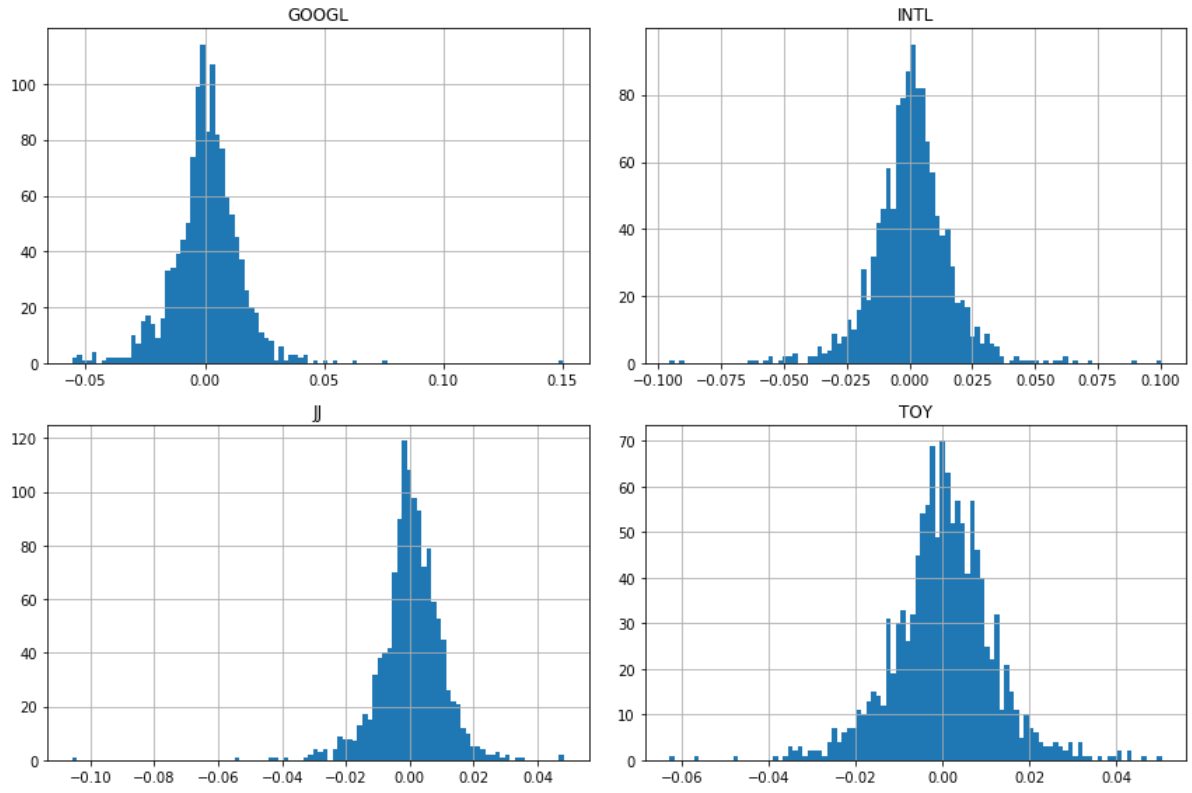
Out[10]:

	GOOGL	INTL	JJ	TOY
count	1257.000000	1257.000000	1257.000000	1257.000000
mean	0.000651	0.000703	0.000373	0.000219
std	0.014455	0.015815	0.010024	0.011721
min	-0.055662	-0.095432	-0.105781	-0.062797
25%	-0.005941	-0.007499	-0.004194	-0.005800
50%	0.000784	0.000943	0.000388	0.000341
75%	0.008269	0.008671	0.005905	0.006877
max	0.150645	0.100315	0.048395	0.050474

In [11]:
```
port_cov=log_ret.corr()
print(port_cov)
```

```
          GOOGL      INTL        JJ       TOY
GOOGL  1.000000  0.439011  0.375379  0.424550
INTL   0.439011  1.000000  0.355773  0.403426
JJ     0.375379  0.355773  1.000000  0.365957
TOY    0.424550  0.403426  0.365957  1.000000
```

In [12]:
```
log_ret.hist(figsize=(12,8),bins=100)
plt.tight_layout()
```



In [ ]:
```
log
```

In [13]:
```
########################
number_port=5000
all_weights=np.zeros((number_port,len(portfolio.columns)))
return_arr=np.zeros(number_port)
volatility_arr=np.zeros(number_port)
sharperatio_arr=np.zeros(number_port)
for i in range(number_port):
    #weights
    weights=np.array(np.random.random(4))
    #rebalance weights
    weights=weights/np.sum(weights)
    #save weights
    all_weights[i,:]=weights
    #expected returns
    return_arr[i]=np.sum((log_ret.mean()*weights)*252)
    #expected volatility
    volatility_arr[i]=np.sqrt(np.dot(weights.T,np.dot(port_cov,weights)))
    #sharpe ratio
    sharperatio_arr[i]=return_arr[i]/volatility_arr[i]
    ######################################
```

In [30]:
```
ValueAtRisk=[]
ExpectedShortfall=[]
MeanReturn=[]
StandardDeviation=[]

number_port=5000
all_weights=np.zeros((number_port,len(portfolio.columns)))
for i in range(number_port):
    #weights
    weights=np.array(np.random.random(4))
    #rebalance weights
    weights=weights/np.sum(weights)
    all_weights[i,:]=weights
    x=weights*portfolio
    x['portfoliovalue']=x['GOOGL']+x['INTL']+x['JJ']+x['TOY']
    x['dailyreturn']=np.log(x['portfoliovalue']/x['portfoliovalue'].shift(1))
    y=x['dailyreturn'].sort_values()
    ValueAtRisk.append(y.quantile(0.1))
    ES=(y<y.quantile(0.1))
    ExpectedShortfall.append(y[ES].mean())
    MeanReturn.append(y.mean())
    StandardDeviation.append(y.std())
```

In [31]:
```
ValueAtRisk_arr=np.array(ValueAtRisk)*(-1)*np.sqrt(252)
ExpectedShortfall_arr=np.array(ExpectedShortfall)*(-1)*np.sqrt(252)
MeanReturn_arr=np.array(MeanReturn)*252
StandardDeviation_arr=np.array(StandardDeviation)*np.sqrt(252)
sharperatio_arr=np.array(MeanReturn_arr/StandardDeviation_arr)
var_mean_ratio=np.array(MeanReturn_arr/ValueAtRisk_arr)
shortfall_mean_ratio=np.array(MeanReturn_arr/ExpectedShortfall_arr)
```

In [32]:
```
print(sharperatio_arr.max(),sharperatio_arr.argmax())
```

```
0.8667536161837194 634
```

In [33]:
```
all_weights[sharperatio_arr.argmax(),:]
```

Out[33]:
```
array([0.04234457, 0.65171018, 0.28371364, 0.02223161])
```

In [34]:
```
print(var_mean_ratio.max(),var_mean_ratio.argmax())
```

```
0.8095046234886399 634
```

In [35]:
```
all_weights[var_mean_ratio.argmax(),:]
```

Out[35]:
```
array([0.04234457, 0.65171018, 0.28371364, 0.02223161])
```

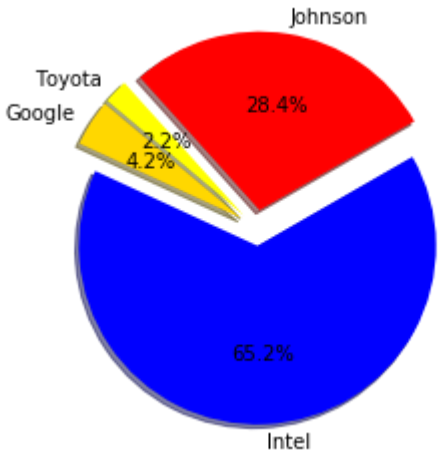
In [36]:
```
print(shortfall_mean_ratio.max(),shortfall_mean_ratio.argmax())
```

```
0.46825891096841576 634
```

In [37]:
```
all_weights[shortfall_mean_ratio.argmax(),:]
```

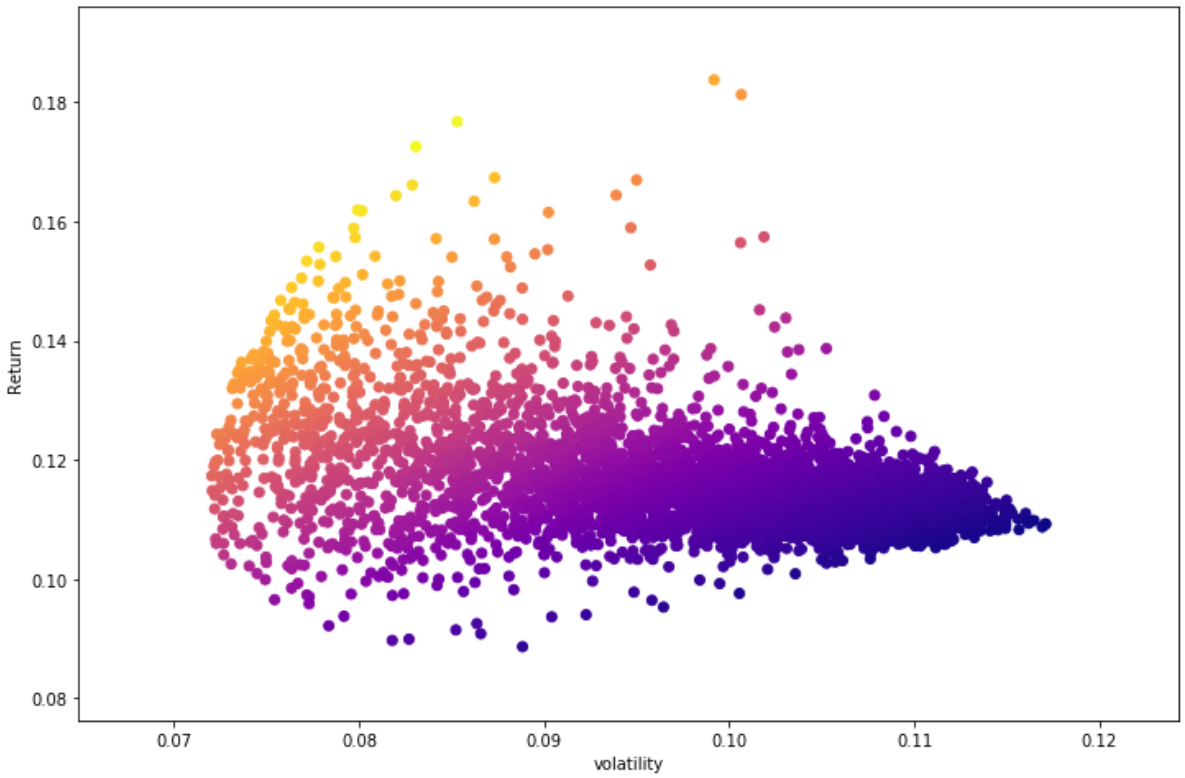
Out[37]:
```
array([0.04234457, 0.65171018, 0.28371364, 0.02223161])
```

In [126]:
```
labels = 'Google','Intel','Johnson','Toyota'
sizes = [0.04234457, 0.65171018, 0.28371364, 0.02223161]
colors = ['gold','blue','red','yellow','orange','purple']
explode = (0.1,0.1,0.1,0.1)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```



In [80]:
```
plt.figure(figsize=(12,8))
plt.scatter(StandardDeviation_arr,MeanReturn_arr,c=sharperatio_arr,cmap='plasma')
#plt.colorbar(label='sharperatio')
plt.xlabel('volatility')
plt.ylabel('Return')
plt.figure(figsize=(0.1,0.1))



#plt.scatter(max_sr_vol,max_sr_ret,c='red',edgecolors='black',s=50)
```

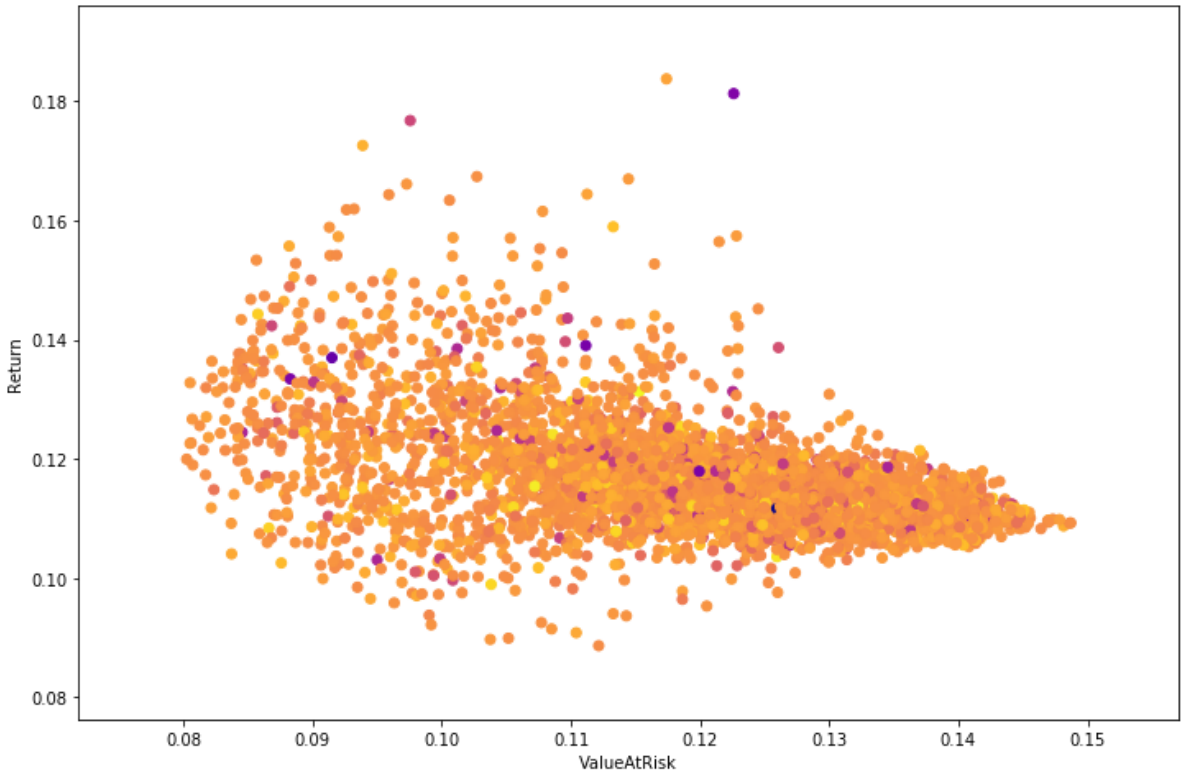
Out[80]: <Figure size 7.2x7.2 with 0 Axes>



<Figure size 7.2x7.2 with 0 Axes>

In [81]:
```
plt.figure(figsize=(12,8))
plt.scatter(ValueAtRisk_arr,MeanReturn_arr,c=var_mean_ratio,cmap='plasma')
#plt.colorbar(label='sharperatio')
plt.xlabel('ValueAtRisk')
plt.ylabel('Return')
plt.figure(figsize=(0.1,0.1))

#plt.scatter(max_sr_vol,max_sr_ret,c='red',edgecolors='black',s=50)
```

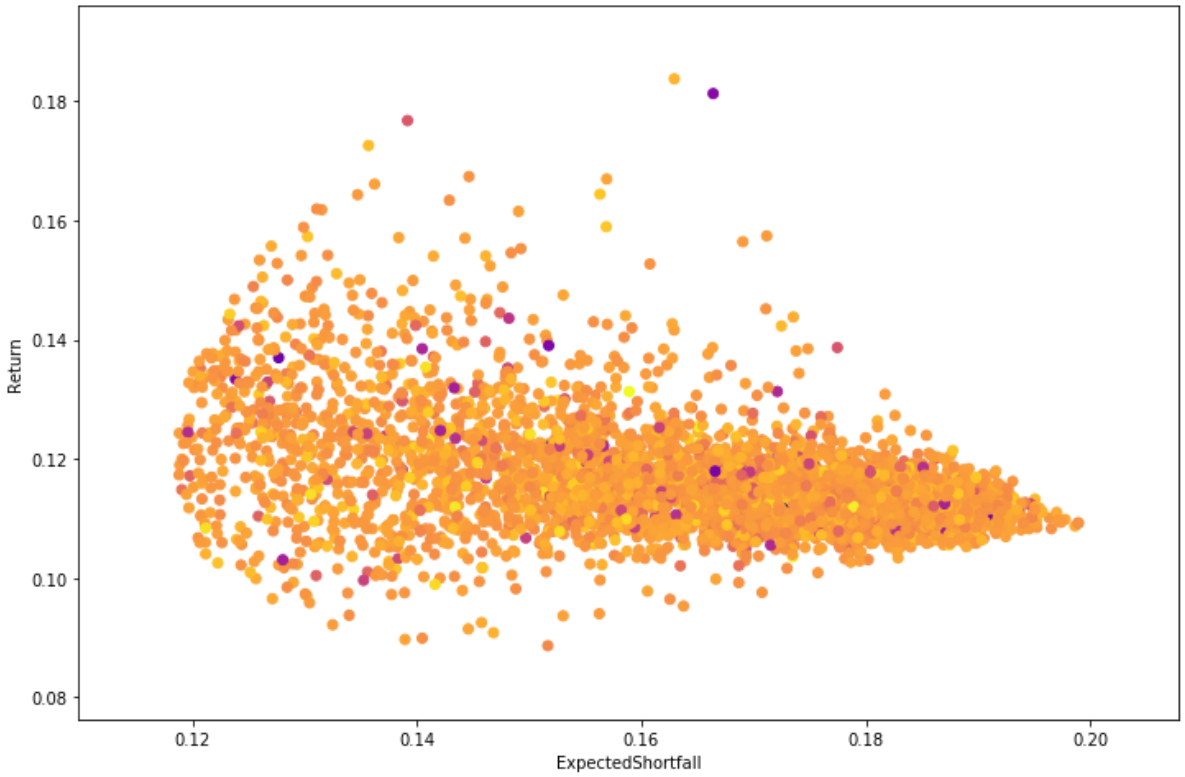
Out[81]: <Figure size 7.2x7.2 with 0 Axes>



<Figure size 7.2x7.2 with 0 Axes>

In [82]:
```
plt.figure(figsize=(12,8))
plt.scatter(ExpectedShortfall_arr,MeanReturn_arr,c=shortfall_mean_ratio,cmap=
'plasma')
#plt.colorbar(label='sharperatio')
plt.xlabel('ExpectedShortfall')
plt.ylabel('Return')
plt.figure(figsize=(0.1,0.1))

#plt.scatter(max_sr_vol,max_sr_ret,c='red',edgecolors='black',s=50)
```

Out[82]: <Figure size 7.2x7.2 with 0 Axes>



<Figure size 7.2x7.2 with 0 Axes>

In [ ]:
```
#Generating stocks values for next year
```

In [47]:
```
T=1
mea=[0.000651,0.000703,0.000373,0.000219]
Goog_Int=0.439011
Goog_JJ=0.375379
Goog_Toy=0.424550
Int_JJ=0.355773
INT_Toy=0.403426
JJ_Toy=0.365957
cov=[[1,Goog_Int,Goog_JJ,Goog_Toy],[Goog_Int,1,Int_JJ,INT_Toy],[Goog_JJ,Int_JJ
,1,JJ_Toy],[Goog_Toy,INT_Toy,JJ_Toy,1]]


#Google

mean_goog=0.000651
std_goog=0.014455
var_goog=std_goog**2
drift_goog=mean_goog-0.5*var_goog
s0_goog=1231.91
futureprice_goog=pd.DataFrame()

#Intel
mean_int=0.000703
std_int=0.015815
var_int=std_int**2
drift_int=mean_int-0.5*var_int
s0_int=56.7
futureprice_int=pd.DataFrame()

#jhonson and Jhonson
mean_jj=0.000373
std_jj=0.010024
var_jj=std_jj**2
drift_jj=mean_jj-0.5*var_jj
s0_jj=138.02
futureprice_jj=pd.DataFrame()

#Toyota

mean_toy=0.000219
std_toy=0.011721
var_toy=std_toy**2
drift_toy=mean_toy-0.5*var_toy
s0_toy=121.75
futureprice_toy=pd.DataFrame()


for i in range(5):
    G,I,J,T=np.random.multivariate_normal(mea,cov,1000).T
    dailyreturn_goog=np.exp(drift_goog+std_goog*G)
    dailyreturn_int=np.exp(drift_int+std_int*I)
    dailyreturn_jj=np.exp(drift_jj+std_jj*J)
    dailyreturn_toy=np.exp(drift_toy+std_toy*T)
    if i==1:
        temp_goog=+pd.DataFrame(s0_goog*dailyreturn_goog.cumprod())
        futureprice_goog=temp_goog
```

```
        temp_int=+pd.DataFrame(s0_int*dailyreturn_int.cumprod())
        futureprice_int=temp_int
        temp_jj=+pd.DataFrame(s0_jj*dailyreturn_jj.cumprod())
        futureprice_jj=temp_jj
        temp_toy=+pd.DataFrame(s0_toy*dailyreturn_toy.cumprod())
        futureprice_toy=temp_toy
    else:
        futureprice_goog=futureprice_goog+pd.DataFrame(s0_goog*dailyreturn_goo
g.cumprod())
        futureprice_int=futureprice_int+pd.DataFrame(s0_int*dailyreturn_int.cu
mprod())
        futureprice_jj=futureprice_jj+pd.DataFrame(s0_jj*dailyreturn_jj.cumpro
d())
        futureprice_toy=futureprice_toy+pd.DataFrame(s0_toy*dailyreturn_toy.cu
mprod())

futureprice_goog=futureprice_goog/10000
futureprice_int=futureprice_int/10000
futureprice_jj=futureprice_jj/10000
futureprice_toy=futureprice_toy/10000
```

In [48]:
```
futureprice_goog
```

Out[48]:

	0
0	0.490656
1	0.490567
2	0.495346
3	0.499302
4	0.500074
5	0.504867
6	0.507087
7	0.508561
8	0.507554
9	0.507576
10	0.507098
11	0.512230
12	0.509447
13	0.514001
14	0.516936
15	0.515267
16	0.512567
17	0.516047
18	0.515483
19	0.513821
20	0.517179
21	0.516848
22	0.520614
23	0.526095
24	0.532795
25	0.529807
26	0.528861
27	0.527159
28	0.523201
29	0.524790
...	...
970	0.727888

	0
971	0.724651
972	0.729511
973	0.738365
974	0.732843
975	0.731779
976	0.731353
977	0.742077
978	0.737641
979	0.737051
980	0.736071
981	0.729778
982	0.729736
983	0.730342
984	0.731067
985	0.737513
986	0.744918
987	0.748210
988	0.748590
989	0.752876
990	0.745171
991	0.744820
992	0.747178
993	0.742187
994	0.749442
995	0.762060
996	0.759436
997	0.762780
998	0.766351
999	0.755759

1000 rows × 1 columns

In [49]:
```
futureprice_int
```

Out[49]:

	0
0	0.022683
1	0.022577
2	0.022516
3	0.022482
4	0.022572
5	0.022669
6	0.022624
7	0.022722
8	0.023060
9	0.022984
10	0.023015
11	0.022881
12	0.022797
13	0.023012
14	0.023020
15	0.022805
16	0.022455
17	0.022686
18	0.022550
19	0.022299
20	0.022317
21	0.022270
22	0.022432
23	0.022579
24	0.022713
25	0.022246
26	0.021969
27	0.021834
28	0.021919
29	0.022094
...	...
970	0.055194

	0
971	0.055017
972	0.054786
973	0.055073
974	0.054783
975	0.055339
976	0.055548
977	0.056384
978	0.055536
979	0.055080
980	0.055351
981	0.055614
982	0.055425
983	0.055536
984	0.055602
985	0.056328
986	0.056620
987	0.056592
988	0.057023
989	0.057339
990	0.057580
991	0.056998
992	0.057489
993	0.057099
994	0.057630
995	0.057868
996	0.057813
997	0.058180
998	0.059362
999	0.059149

1000 rows × 1 columns

In [50]:
```
futureprice_jj
```

Out[50]:

	0
0	0.055033
1	0.054750
2	0.054476
3	0.054466
4	0.054753
5	0.054614
6	0.054411
7	0.054697
8	0.054769
9	0.054449
10	0.054264
11	0.054019
12	0.053828
13	0.053927
14	0.053800
15	0.053931
16	0.053708
17	0.054095
18	0.054142
19	0.054215
20	0.054007
21	0.053871
22	0.054037
23	0.054172
24	0.054158
25	0.053534
26	0.053303
27	0.053504
28	0.053896
29	0.054015
...	...
970	0.091141

	0
971	0.091387
972	0.091417
973	0.091895
974	0.091069
975	0.090883
976	0.090974
977	0.090857
978	0.089789
979	0.088934
980	0.088506
981	0.087613
982	0.088416
983	0.088356
984	0.089187
985	0.089336
986	0.090134
987	0.089659
988	0.089789
989	0.090218
990	0.090971
991	0.090540
992	0.091284
993	0.091620
994	0.092214
995	0.092874
996	0.093149
997	0.093393
998	0.093492
999	0.093799

1000 rows × 1 columns

In [51]:
```
futureprice_toy
```

Out[51]:

	0
0	0.048518
1	0.048134
2	0.047960
3	0.048017
4	0.048070
5	0.047901
6	0.048051
7	0.047773
8	0.047796
9	0.047882
10	0.048074
11	0.048134
12	0.047951
13	0.047970
14	0.047828
15	0.047498
16	0.047128
17	0.047423
18	0.047135
19	0.046587
20	0.046917
21	0.046604
22	0.046733
23	0.046693
24	0.047099
25	0.047136
26	0.047157
27	0.046767
28	0.047016
29	0.047243
...	...
970	0.060971

	0
971	0.060804
972	0.061224
973	0.061645
974	0.060983
975	0.061235
976	0.061079
977	0.062184
978	0.061565
979	0.061794
980	0.061752
981	0.061812
982	0.062111
983	0.061822
984	0.062072
985	0.061660
986	0.061842
987	0.061552
988	0.061771
989	0.062254
990	0.061969
991	0.062028
992	0.062001
993	0.061148
994	0.061084
995	0.061307
996	0.060806
997	0.061025
998	0.061302
999	0.061336

1000 rows × 1 columns

In [92]:
```
future_port=pd.concat([futureprice_goog,futureprice_int,futureprice_jj,futureprice_toy],axis=1)
```

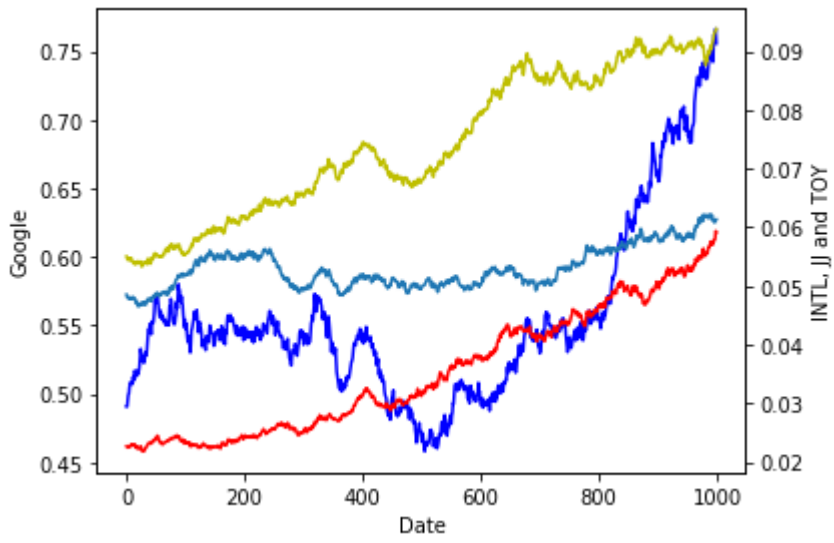
In [93]:
```
future_port.columns=['Google','Intel','Johnson','Toyota']
```

In [124]:
```
future_port.corr()
```

Out[124]:

	Google	Intel	Johnson	Toyota
Google	1.000000	0.628057	0.517240	0.816003
Intel	0.628057	1.000000	0.961371	0.644465
Johnson	0.517240	0.961371	1.000000	0.570861
Toyota	0.816003	0.644465	0.570861	1.000000

In [95]:
```
fig, ax1 = plt.subplots()

ax1.set_xlabel('Date')
ax1.set_ylabel('Google')
ax1.plot(future_port['Google'],'b')
ax1.tick_params(axis='y')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

ax2.set_ylabel('INTL, JJ and TOY')  # we already handled the x-label with ax1
ax2.plot(future_port['Intel'],'r')
ax2.plot(future_port['Johnson'],'y')
ax2.plot(future_port['Toyota'])
ax2.tick_params(axis='y')

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



In [96]:
```
ValueAtRisk=[]
ExpectedShortfall=[]
MeanReturn=[]
StandardDeviation=[]

number_port=5000
all_weights=np.zeros((number_port,len(portfolio.columns)))
for i in range(number_port):
    #weights
    weights=np.array(np.random.random(4))
    #rebalance weights
    weights=weights/np.sum(weights)
    all_weights[i,:]=weights
    x=weights*future_port
    x['portfoliovalue']=x['Google']+x['Intel']+x['Johnson']+x['Toyota']
    x['dailyreturn']=np.log(x['portfoliovalue']/x['portfoliovalue'].shift(1))
    y=x['dailyreturn'].sort_values()
    ValueAtRisk.append(y.quantile(0.1))
    ES=(y<y.quantile(0.1))
    ExpectedShortfall.append(y[ES].mean())
    MeanReturn.append(y.mean())
    StandardDeviation.append(y.std())
```

In [97]:
```
ValueAtRisk_arr=np.array(ValueAtRisk)*(-1)*np.sqrt(252)
ExpectedShortfall_arr=np.array(ExpectedShortfall)*(-1)*np.sqrt(252)
MeanReturn_arr=np.array(MeanReturn)*252
StandardDeviation_arr=np.array(StandardDeviation)*np.sqrt(252)
```

In [98]:
```
 sharperatio_arr1=MeanReturn_arr/StandardDeviation_arr
```

In [99]:
```
sharperatio_arr1.max()
```

Out[99]:
```
2.034453090663059
```

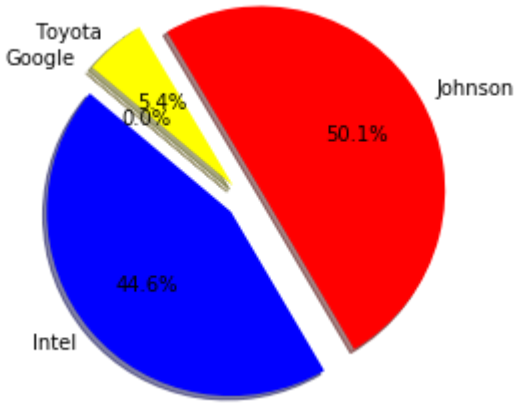
In [100]:
```
sharperatio_arr1.argmax()
```

Out[100]:
```
2145
```

In [101]:
```
all_weights[sharperatio_arr1.argmax(),:]
```

Out[101]:
```
array([6.85471219e-05, 4.45579131e-01, 5.00644966e-01, 5.37073557e-02])
```

In [127]:
```
labels = 'Google','Intel','Johnson','Toyota'
sizes = [6.85471219e-05, 4.45579131e-01, 5.00644966e-01, 5.37073557e-02]
colors = ['gold','blue','red','yellow','orange','purple']
explode = (0.1,0.1,0.1,0.1)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```



In [102]:
```
sharperatioVAR_arr=MeanReturn_arr/ValueAtRisk_arr
```

In [103]:
```
sharperatioVAR_arr.max()
```

Out[103]: 1.7905383801066708

In [104]:
```
sharperatioVAR_arr.argmax()
```

Out[104]: 2145

In [105]:
```
all_weights[sharperatioVAR_arr.argmax(),:]
```

Out[105]: array([6.85471219e-05, 4.45579131e-01, 5.00644966e-01, 5.37073557e-02])

In [ ]:
```
labels = 'Google','Intel','Johnson','Toyota'
sizes = [6.85471219e-05, 4.45579131e-01, 5.00644966e-01, 5.37073557e-02]
colors = ['gold','blue','red','yellow','orange','purple']
explode = (0.1,0.1,0.1,0.1)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```

In [106]:
```
sharperatioES_arr=MeanReturn_arr/ExpectedShortfall_arr
```

In [107]:
```
sharperatioES_arr.max()
```

Out[107]: 1.2446382538035274

In [108]:
```
n=sharperatioES_arr.tolist()
```

In [109]:
```
n.pop(3955)
```

Out[109]: 0.7758190971138389

In [110]:
```
sharperatioES_arr=np.array(n)
```

In [111]:
```
sharperatioES_arr.max()
```

Out[111]: 1.2446382538035274

In [112]:
```
sharperatioES_arr.argmax()
```

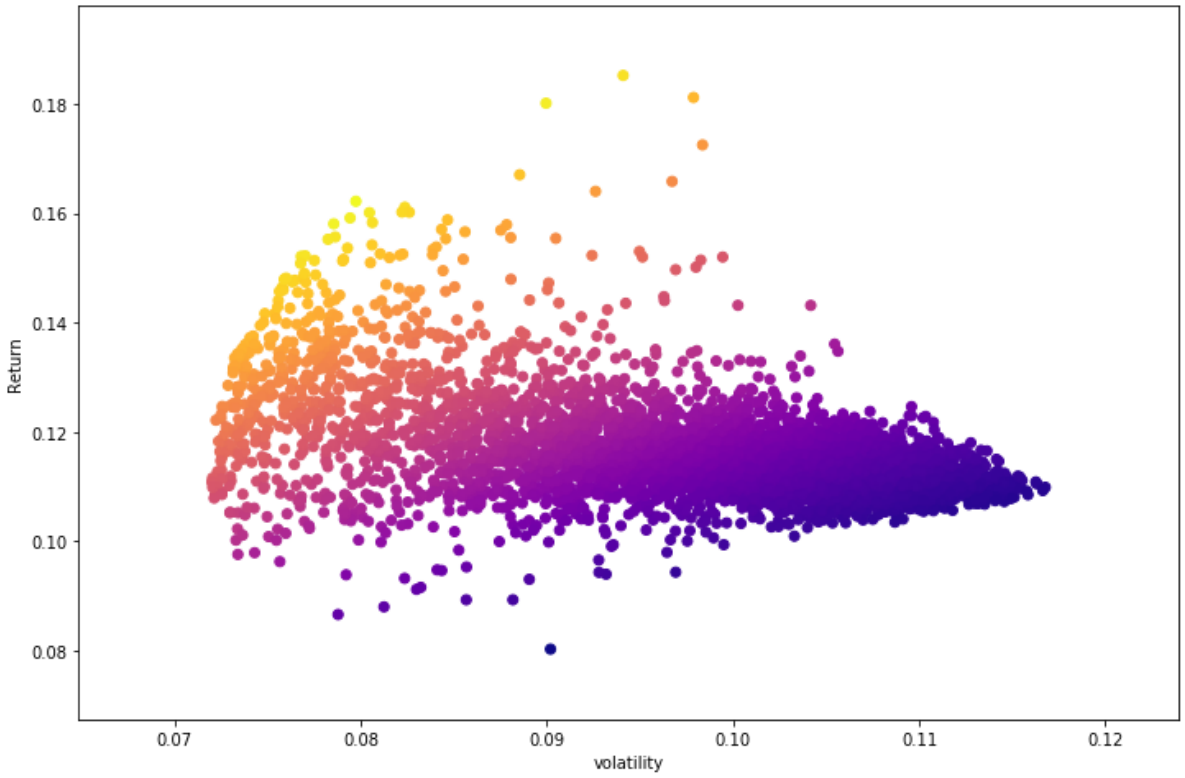
Out[112]: 2145

In [113]:
```
all_weights[3941,:]
```

Out[113]: array([0.22287988, 0.28677431, 0.17317719, 0.31716861])

In [115]:
```
plt.figure(figsize=(12,8))
plt.scatter(StandardDeviation_arr,MeanReturn_arr,c=sharperatio_arr1,cmap='plas
ma')
#plt.colorbar(label='sharperatio')
plt.xlabel('volatility')
plt.ylabel('Return')
plt.figure(figsize=(0.1,0.1))

#plt.scatter(max_sr_vol,max_sr_ret,c='red',edgecolors='black',s=50)
```

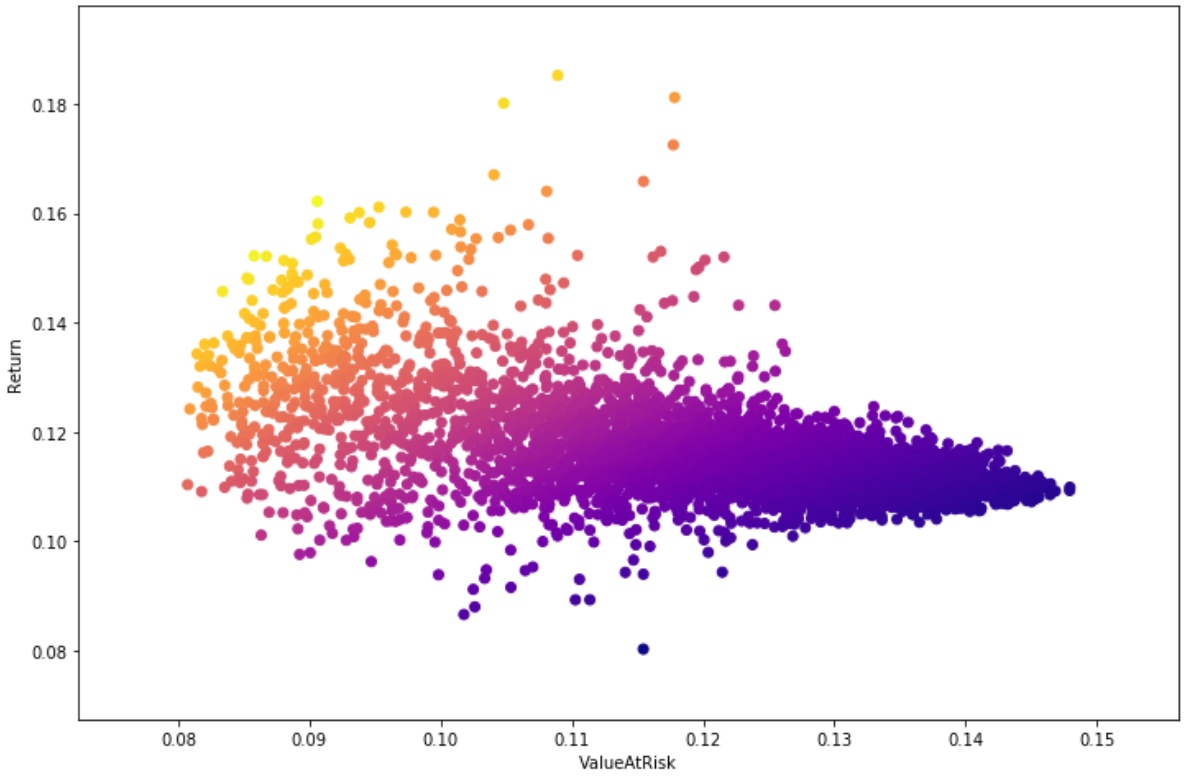
Out[115]: <Figure size 7.2x7.2 with 0 Axes>



<Figure size 7.2x7.2 with 0 Axes>

In [116]:
```
plt.figure(figsize=(12,8))
plt.scatter(ValueAtRisk_arr,MeanReturn_arr,c=sharperatioVAR_arr,cmap='plasma')
#plt.colorbar(label='sharperatio')
plt.xlabel('ValueAtRisk')
plt.ylabel('Return')
plt.figure(figsize=(0.1,0.1))

#plt.scatter(max_sr_vol,max_sr_ret,c='red',edgecolors='black',s=50)
```

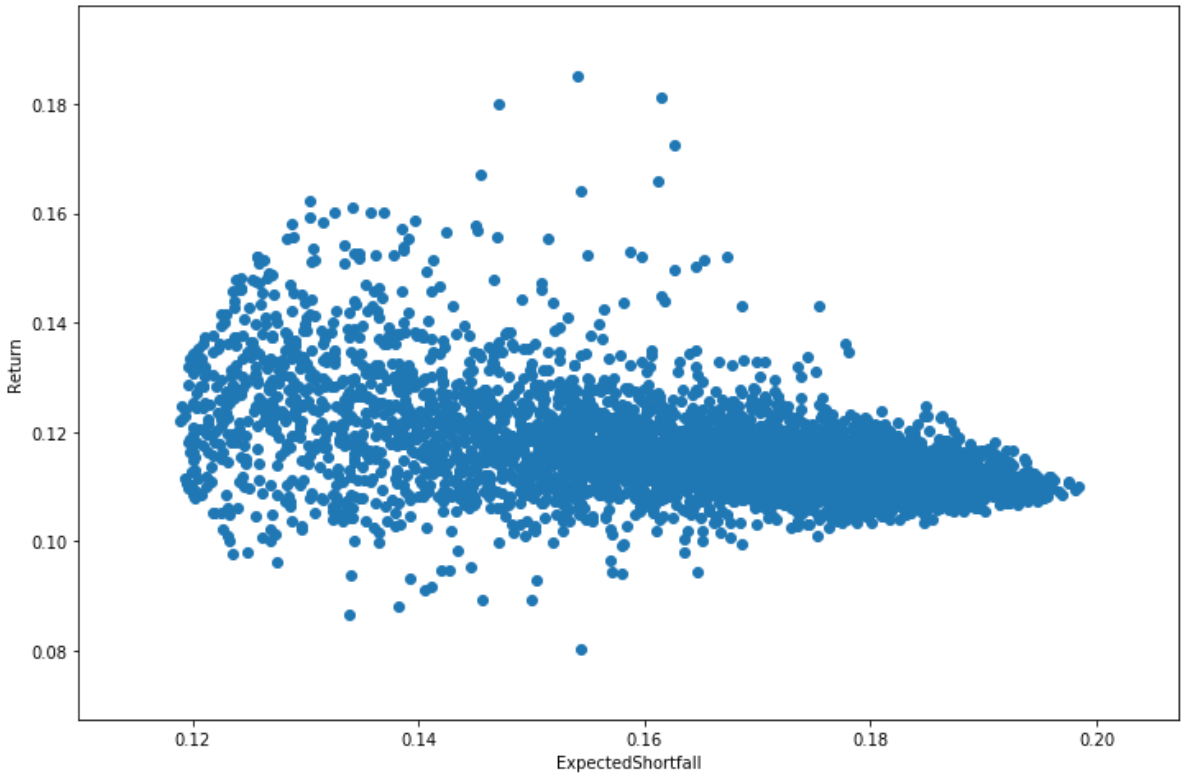
Out[116]: <Figure size 7.2x7.2 with 0 Axes>



<Figure size 7.2x7.2 with 0 Axes>

In [123]:
```
plt.figure(figsize=(12,8))
plt.scatter(ExpectedShortfall_arr,MeanReturn_arr,cmap='plasma')
#plt.colorbar(label='sharperatio')
plt.xlabel('ExpectedShortfall')
plt.ylabel('Return')
plt.figure(figsize=(0.1,0.1))

#plt.scatter(max_sr_vol,max_sr_ret,c='red',edgecolors='black',s=50)
```

Out[123]: <Figure size 7.2x7.2 with 0 Axes>



<Figure size 7.2x7.2 with 0 Axes>