QF634 APPLIED QUANTITATIVE RESEARCH METHODS
LECTURE 1

Lecturer: Prof Emeritus Lim Kian Guan

Correlations and Portfolios

- In the early 1900s till early 1930s, statistical theories including rigorous formulations of probability distributions by Pearson, the design of experiments and the maximum likelihood method by Fisher, concepts of testing by Neyman and others, were born. The extreme value theory was discovered in the 1920s. The practice of statistics had become rigorous within the framework of probability theory. The latter could have been from the time of Pascal and Fermat, though in modern times, it could have been based on the measure-theoretic approach by Kolmogorov.
- The march of statistical applications continues into modern rigorous and analytical forms would not have been possible without the great efforts and tirelessness of generations of brilliant minds in these fields.
- Machine learning is a new wave of approach that picked up momentum since the 1990s, partly statistical in perspectives, and partly engineering in spirit, driven by explosions in data quantities or big data and the commensurate increase in the capacity of computer machines to manage and scrutinize these data or data science.

- Machine learning is a type of artificial intelligence using computer algorithms (in solving specific problems, typically involving optimization) to mainly derive prediction or else classification from a set of data with possibly complex patterns in a high dimension space. Prediction can also mean prediction of a class or category, and thus can be a more generally applied terminology. The solution of the problem leads to better decision-making. The algorithms are largely based on mathematical and statistical models as well as numerical methods of optimization.
- In data analytics for prediction (or we may simply call it predictive analytics) – that part of machine learning for prediction and classification – there is usually not an economic-theoretical model. An explicitly assumed probability distributions underlying the variable data may or may not be utilized. The basic philosophy is that the realized data should speak loudest for themselves.

Machine Learning versus Statistical Modelling

- It is useful to understand the machine learning approach in comparison with the more traditional statistical approach. They have large overlaps but also some differences. Classical statistics and econometrics (including regressions) involve estimation, inference (or testing), prediction, and grouping.
- Estimation of embedded model parameters is a frequent practice in the statistical evaluation of a theoretical model whether it is based on economics or based on assumed probability distributions of the underlying variable data or based on both. It is usually accompanied by the testing or inference on the model, whether the model is acceptable or to be rejected.
- In machine learning, however, embedded model coefficients or feature impact are often part of a 'black box' – oftentimes inconvenient to enunciate, much less test, such as in neural networks.
- The impact of features or explanatory variables may however be measured in other metrics, typically not by underlying statistical distributions of the parameters. Machine learning approach became important due to its ability to perform good forecasts or predictions, often based on complex and unstructured data that more traditional statistical approach finds it inconvenient to model.

The Terms for Prediction/Classification in Machine Learning (ML)

- Dependent Variables (**Targets or Labels**)
- Explanatory Variables (**Features**)
- Regression Coefficients (**Weights**)
- Nonlinear Regression Function (**Activation Function**)
- Optimization Criterion (**Loss Function or Error Function**)
- In-Sample Data (**Training Data Set)**
- Out-of-Sample Data (**Test Data or Hold-Out Set)**
- Out-of-Sample Prediction Error **(Generalization Error)**
- ML calls predictions with target data or labels – **supervised learning**. Typically, **unsupervised learning** is used to form clusters or distinct groupings, reduce dimensions of data, and perform associations.

Some Differences of ML with Regression Models

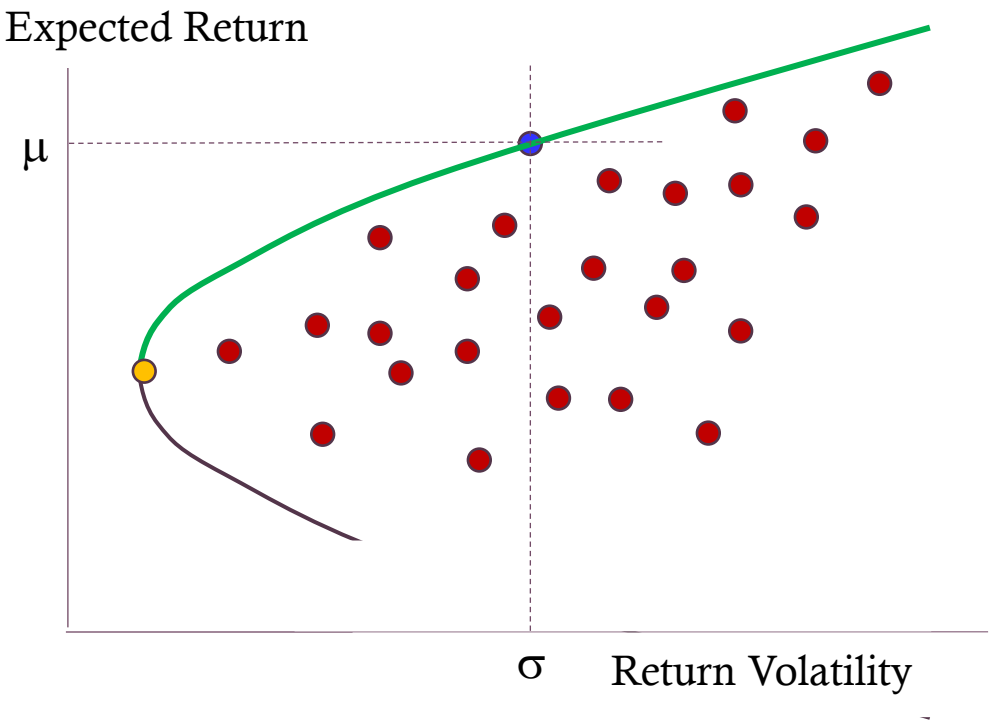
- ML typically does not employ an underlying theoretical model that provides an analytical relationship amongst explanatory and dependent variables.
- ML typically does not explicitly assume and employ underlying probability distributions. But ML relies on large data set(s) and implicitly some forms of data stationarity, e.g., historical patterns in training set repeat for the future test set. While strong assumptions of stationarity are typically not required in machine learning, any data deemed non-stationary or having unit roots in time series could best be pre-processed to ensure that the machine learning prediction results can be improved.
- However, there are exceptions, particularly in smaller data sets. Some ML methods are used in conjunction with more traditional statistical theory.
- Thus, there is typically no statistical model testing or testing hypotheses in ML. In machine learning models, the main objectives are to predict and/or to classify. **A major advantage of statistical model is to provide policy makers (not just forecasters) with what (future) decision variables to control**.
- Sometimes an intermediate step using another cut-out piece of separate validation data set before the final test data set is used to fine-tune hyperparameters before applying the test set. Hyperparameters typically arise in ML algorithms – and is not part of the ML model, e.g., not the weights.

Preparing for the In-Class Exercises:

- Bring your laptops/notebooks
- Download Python from https://www.python.org/downloads/
- Download the latest version of Anaconda from https://www. anaconda.com/distribution/. Ensure you download the version for your device's Operating System.
- We use Jupyter Notebook (a web-based interactive computing platform) for the exercises.
- Several of Python's open-source libraries or packages are commonly used, e.g., Pandas, Numpy, Matplotlib, Scikit-Learn (somewhat similar to Sklearn), SciPy, Keras, TensorFlow, math, etc. These libraries or packages contain more efficiently written reusable modules with codes that need not be rewritten from scratch. For installation and import of the libraries and packages, check for latest update in the documentations on public websites, where necessary. Many of the packages call up other numerical methods to search for optimums – we explain some of these, but details of numerical routines (the numerical methods) are outside the scope of this book.

Portfolio Diversification

- Markowitz efficient portfolio frontier represents the boundary of portfolios of risky stocks that that have the minimum return variance given the expected portfolio return above the minimum variance portfolio return. The hyperbola is obtained when there are no short-sales constraints.



Each red dot represents a risky stock with associated expected return and return volatility (based on historical return mean and standard deviation).

Blue dot on the efficient frontier (green) represents a portfolio (linear combination of the stocks) with expected return μ and portfolio return volatility σ.

μ is a linear combination of the portfolio stocks' expected returns. σ is computed from the weights and the estimated covariance matrix of the stocks' returns.

Orange dot is the minimum variance portfolio (MVP).

MVP Portfolio with Short-sale Constraints

Let there be N stocks and a feasible portfolio is formed with weight w_i on stock i. It is feasible based on the constraints $\sum_{i=1}^{N} w_i = 1$, and $w_i \geq 0$ for every i. Let the weight vector be $w = (w_1, w_2, w_3, \ldots, w_N)^T$. Let the expected return of stock i be $E(r_{it}^*)$ at time t. The expected return vector is $\mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_N)^T$ where $\mu_i = E(r_{it}^*)$. If r_{it}^* is stationary, then its unconditional expectation is constant for every t. The covariance matrix of $(r_{1t}^*, r_{2t}^*, r_{3t}^*, \ldots, r_{Nt}^*)^T$ for each t is given as $\Sigma_{N\times N}$. A minimum variance portfolio (portfolio with the minimum return variance) can be found as follows.

$$\min_{w} \quad w^T \Sigma w \quad \text{subject to } \sum_{i=1}^{N} w_i = 1 \text{ and } w_i \geq 0 \text{ for every i}$$

The objective function under the solution w_{MVP} is the minimum return variance, ${w_{MVP}}^T \Sigma w_{MVP}$, associated with the minimum variance portfolio (MVP).

* or asterisk to r_{it} denotes that the return is adjusted for dividends.

Maximum Sharpe Ratio Portfolio with Short-sale Constraints

Another type of optimal portfolio is that of maximizing the Sharpe ratio defined as

$$w^T \mu \, / \, \sqrt{w^T \Sigma \, w}$$

which is expected portfolio return per unit of risk or per unit of portfolio return standard deviation. Numerator is usually excess return, but we assume risk-free rate is small and negligible.

This is solved as:

$$\min_{w} \sqrt{w^T \Sigma \, w} \, / \, w^T \mu$$

subject to $\sum_{i=1}^{N} w_i = 1$ and $w_i \geq 0$ for every i

Minimum Variance of a Portfolio with Given Expected Return and Short-sale Constraints

The minimum variance of a portfolio for a given expected return k greater or equal to the return variance of the minimum variance portfolio can be found by solving:

$$\min_{w} \quad w^T \Sigma \, \mathrm{w}$$

subject to $\sum_{i=1}^{N} w_i = 1$ and $w_i \geq 0$ for every i, and $w^T \mu = \mathrm{k}$

for expected return vector μ.

Note that the maximum k for $\sum_{i=1}^{N} w_i = 1$ and $w_i \geq 0$ occurs when $\mu_m = \max(\mu_1, \mu_2, \mu_3, \ldots, \mu_N)$ and $w_m = 1$ while $w_{i \neq m} = 0$. Then max $k = \mu_m$.

For any k, ${w_{MVP}}^T \mu \leq k \leq \mu_m$, we can find solution w(k) such that the efficient portfolio frontier occurs at expected return k and portfolio return volatility $\sqrt{w(k)^T \Sigma \, w(k)}$.

Worked Example – Data

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

- Stock price data are obtained from public source Yahoo Finance. In this first part of the data work, 8 of the largest ecommerce listed companies, viz. Alibaba, Amazon, EBay, Rakuten, Suning, Wayfair, Zalando, and JD.Com are examined.
- Use Pandas dataframe to download .csv data. **Follow the codes** via the *.ipynb files.
- Print the first 5 rows of dataframe for ###Alibaba_USD.csv

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	19/9/2014	92.699997	99.699997	89.949997	93.889999	93.889999	271879400
1	22/9/2014	92.699997	92.949997	89.500000	89.889999	89.889999	66657800
2	23/9/2014	88.940002	90.480003	86.620003	87.169998	87.169998	39009800
3	24/9/2014	88.470001	90.570000	87.220001	90.570000	90.570000	32088000
4	25/9/2014	91.089996	91.500000	88.500000	88.919998	88.919998	28598000

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

- Code line [9] selects only rows in the data set df that have years (one of the features or columns) between 2017 and 2021 inclusive.

	Adj Close	day	month	year
576	88.599998	3.0	1.0	2017.0
577	90.510002	4.0	1.0	2017.0
578	94.370003	5.0	1.0	2017.0
579	93.889999	6.0	1.0	2017.0
580	94.720001	9.0	1.0	2017.0
...	...	...	...	...
1830	116.589996	27.0	12.0	2021.0
1831	114.800003	28.0	12.0	2021.0
1832	112.089996	29.0	12.0	2021.0
1833	122.989998	30.0	12.0	2021.0
1834	118.790001	31.0	12.0	2021.0

[1259 rows x 4 columns]

U.S. public firm typically pays dividends on a quarterly basis. Suppose on date Y, a firm announces that some dividends are planned to be paid (payment of cash) on date Y+20 days.

Y+10 is the ex-dividend date and Y+11 is the record date.

Any investor J who buys the stock from investor I on or after the ex-dividend date of Y+10 will not be paid the announced dividends. Investor I who had purchased the share before ex-dividend date and had not sold by Y+10 will receive the dividends even if he/she sold by Y+20.

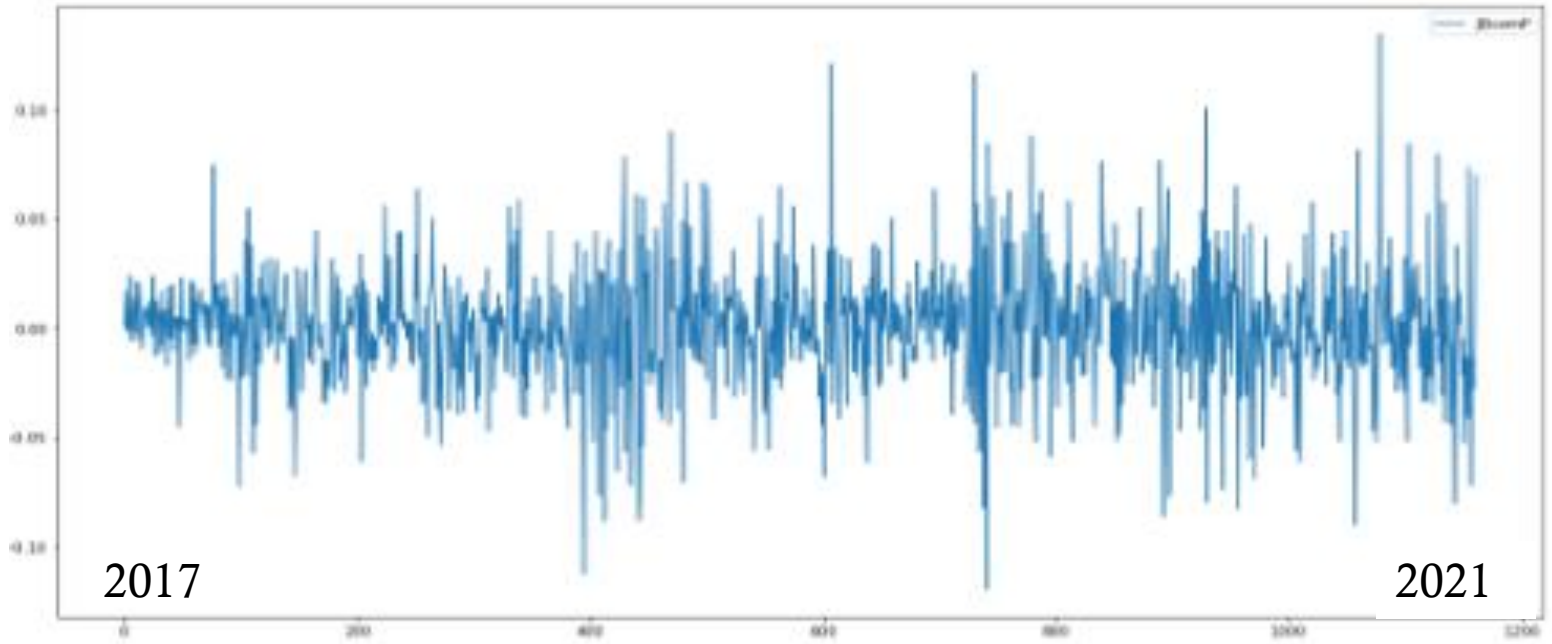
The price of the stock **typically falls on the ex-dividend date** by an amount equal to the dividend (ignoring tax). Thus, it is important to consider adding the dividend so that the computed return rate would not appear to drop due to ex-dividend.

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

- Two ways to compute cum-dividend return rates. The obvious one is to include expected future dividend payout D_{Y+20} on ex-dividend date, i.e. at Y+10, the return rate is $(P_{Y+10} + D_{Y+20})/P_{Y+9} - 1$. Or $r_{Y+10} = \ln (P_{Y+10} + D_{Y+20})/P_{Y+9}$ could be computed. Note that the dividend is not exactly collected at Y+10.
- Another way is to adjust the time series of all daily closing prices as follows. Subtract D_{Y+20} from the price the day prior to ex-dividend date, i.e., Y+9. Call this the adjusted closing price at Y+9, $P_{Y+9}^* = P_{Y+9} - D_{Y+20}$. Then the continuously compounded return rate at Y+10 is computed as $r_{Y+10}^* = \ln P_{Y+10}/P_{Y+9}^*$. (This is to compensate for fall in price on Y+10 due to dividend issue.)
- For daily return, this is approximately the same as r_{Y+10}. Moreover, all closing prices prior to Y+10 are adjusted by the ratio P_{Y+9}^*/ P_{Y+9}, i.e., $P_{Y+8}^* = P_{Y+8} \times (P_{Y+9}^*/ P_{Y+9})$, $P_{Y+7}^* = P_{Y+7} \times (P_{Y+9}^*/ P_{Y+9})$, and so on. So, return rate at Y+9 calculated using the adjusted closing prices is $\ln P_{Y+9}^*/P_{Y+8}^* = \ln P_{Y+9}/P_{Y+8}$.
- If a firm announces a 1:1 stock dividend or else a 2:1 stock split, the effect is the same – the investor gets an additional share for every share he/she owns. At effective date of distribution, the closing price of the share will typically drop to half. All prices prior would be adjusted to $P_t^*=P_t/2$.

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

- In code lines [25] to [27], the adjusted closing price data sets of the 8 stocks from 2017 trough 2021 are merged so that only all stock prices with the same day, month, and year are included. A few missing day gaps are ignored. Daily continuously compounded return rates are computed.
- The time series of the 8 columns of stock returns are shown in code line [29]. Example of JD.com is shown (China's largest online retailer).



Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

```
In [32]: # Finding the simple correlation matrix from a series of returns
         corr_matrix = dfret.corr()
         print(corr_matrix)
```

- Returns Correlation Matrix. Code line [32].

	AlibabaP	AmazonP	EbayP	RakutenP	SuningP	WayfairP	ZalandoP	JDcomP
AlibabaP	1.000000	0.469633	0.304793	0.192598	0.148897	0.280226	0.203696	0.651556
AmazonP	0.469633	1.000000	0.356632	0.183431	0.067568	0.348132	0.242445	0.458592
EbayP	0.304793	0.356632	1.000000	0.181259	0.070084	0.322172	0.236042	0.304356
RakutenP	0.192598	0.183431	0.181259	1.000000	0.091049	0.191544	0.161165	0.184363
SuningP	0.148897	0.067568	0.070084	0.091049	1.000000	0.108223	0.108273	0.113146
WayfairP	0.280226	0.348132	0.322172	0.191544	0.108223	1.000000	0.293582	0.277591
ZalandoP	0.203696	0.242445	0.236042	0.161165	0.108273	0.293582	1.000000	0.247294
JDcomP	0.651556	0.458592	0.304356	0.184363	0.113146	0.277591	0.247294	1.000000

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

Inputs for Portfolio Optimization

- Mean annual return is 252 × E(r_{it}^*) where E(.) is the expectation operator, and E(r_{it}^*) is estimated using $\frac{1}{T}\sum_{t=1}^{T} r_{it}^*$ with daily returns r_{it}^* , t=1,2,…..,T. The mean annualized return is estimated as 252 $\frac{1}{T}\sum_{t=1}^{T} r_{it}^*$ where 252 is approximately the number of trading days in U.S. For all stocks, the vector of mean annualized return is r = μ. A portfolio with weights vector w would have portfolio mean annualized return as $w^T r$.
- For stocks i=1,2,…,N, with return vector r_t^* at day t, the variance of the daily portfolio return is var($w^T r_t^*$). Variance of portfolio annual return is var($\sum_{t=1}^{252} w^T r_t^*$) that is approximately $\sum_{t=1}^{252}$ var($w^T r_t^*$) or 252 × var($w^T r_t^*$) when intertemporal correlations are assumed to be approximately zero. var($w^T r_t^*$) is estimated using w^T var(r_t^*) w for a given w. Each ijth element of var(r_t^*) is estimated as $\frac{1}{T-k}\sum_{t=1}^{T}(r_{it}^* - \mu_i)(r_{jt}^* - \mu_j)$ where k =1 for i=j, and k=2 for i≠j. Annualized portfolio return volatility is $\sqrt{252 \times (w^T \text{ var}(r_t^*)\, w)}$

Portfolio Solutions Using Scipy.optimize

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

In [36]:
```
### Ref: https://www.kaggle.com/code/trangthvu/efficient-frontier-optimization/notebook
import scipy
### All weights, of course, must be between 0 and 1. Thus we set 0 and 1 as the boundaries.
from scipy.optimize import Bounds
bounds = Bounds(0, 1)

### The second boundary is the sum of weights.
from scipy.optimize import LinearConstraint
linear_constraint = LinearConstraint(np.ones((dfret.shape[1],), dtype=int),1,1)
  ### 1,1 in argument of LinearConstraint refers to Lb, Up in Lb <= A.dot(w) <= Ub; if Lb=Ub, it implies equality constraint
  ### df.shape[0] refers to no. rows, .shape[1] refers to no. cols; np.ones fill up with ones
  ### Above -- np.ones((dfret.shape[1],), dtype=int) is A, i.e. 1 x 8 elements of ones since dfret.shape[1] gives dim of cols
  ### Then A'w = 1 is the constraint, i.e. sum of wts must equal to one

covar = dfret.cov()
r = np.mean(dfret,axis=0)*252
### axis=0 means to apply calculation "column-wise", axis=1 means to:apply calculation "row-wise",
### r is annualized vector mean return
### Here. mean is calculated for each XYZ stock return time series (column)

def ret(r,w):
    return r.dot(w)                                ### Note ret(r,w) is defined here. r.dot(w) is matrix multiplication of r and w
def vol(w,covar):                                  ### Risk level or volatility
    return np.sqrt(np.dot(w,np.dot(w,covar))*252)### same as sqrt of w^T \Sigma w *252 -- annualized return volatility
def sharpe (ret,vol):
    return ret/vol
```

Portfolio Solutions Using Scipy.optimize

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

```
### Find a portfolio with the minimum risk.
from scipy.optimize import minimize
### Create x0, the first guess at the values of each stock's weight.
weights = np.ones(dfret.shape[1])
x0 = weights/np.sum(weights)

### Define a function to calculate volatility
portfstderr = lambda w: np.sqrt(np.dot(w,np.dot(w,covar))*252)  ### w is input to function lambda that outputs portfvola
res1 = minimize(portfstderr,x0,method='trust-constr',constraints = linear_constraint,bounds = bounds)
   ### constraint means unit vector .dot(w) = 1; minmize chooses wts w

### Objective function is portfstderr
### 'trust-constr' is to minimize a scalar function subject to constraints -- algorithm updates x0 till obj fn portfvola is min
### minimize(..) function returns optimal weight w_min
### These are the weights of the stocks in the portfolio with the lowest level of risk possible.
w_min = res1.x
### optimization full output.x gives the solution array

np.set_printoptions(suppress = True, precision=3)
### Suppress=True means always printing floating point numbers to 3 decimal places

print(w_min)
print('return: % .4f'% (ret(r,w_min)), 'risk: % .4f'% vol(w_min,covar))  ### this is min var portfolio
### "print" treats the % as a special character you need to add, so it can know, that when you type "f"
###   the number (result) that will be printed will be a floating point type, and the ".4" tells your "print"
###   to print only the first 4 digits after the point.
```

- For the minimum variance portfolio solution in [36], the optimal weights indicate 3.6% of total investment wealth allocated to Alibaba, 21% to Amazon, 20.9% to Ebay, 16.5% to Rakuten, 27.8% to Suning, and 10.2% to Zalando. Close to zero % are put into Wayfair and JD.Com. The weights are all positive and sum to one.

Portfolio Solutions Using Scipy.optimize

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

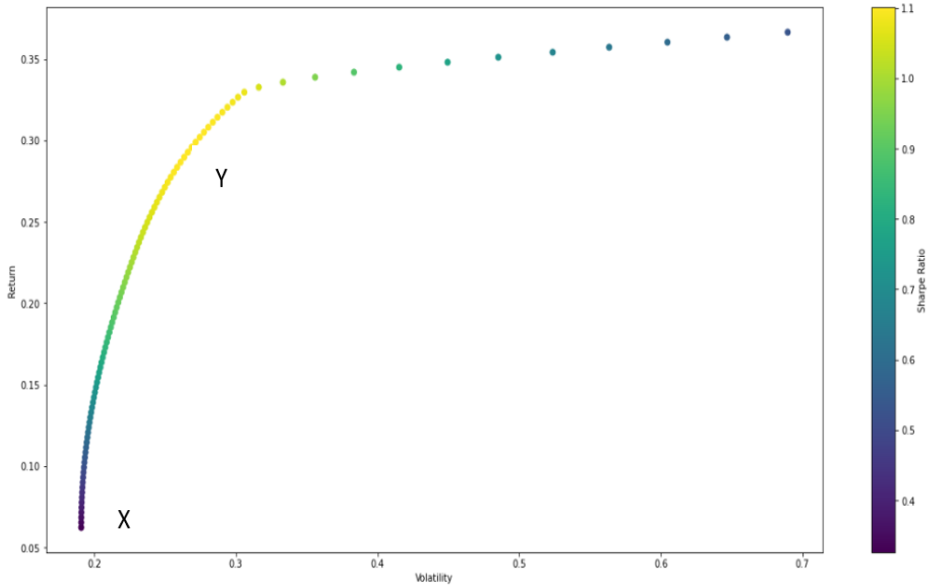
In [37]:
```
### Define 1/Sharpe_ratio as invSharpe
invSharpe  = lambda w: np.sqrt(np.dot(w,np.dot(w,covar))*252)/r.dot(w)
res2 = minimize(invSharpe,x0,method='trust-constr',constraints = linear_constraint,bounds = bounds)
### Objective function is invSharpe -- inverse of Sharpe ratio
#These are the weights of the stocks in the portfolio with the highest Sharpe ratio - call the weight vector w_Sharpe

w_Sharpe = res2.x
   ### constraint means unit vector .dot(w) = 1; minmize chooses wts w
   ### optimization full output.x gives the solution array of optimal weights in min inverse Sharpe ratio or max Sharpe ratio

print(w_Sharpe)
print('return: % .4f'% (ret(r,w_Sharpe)), 'risk: % .4f'% vol(w_Sharpe,covar))  ### this is max Sharpe ratio portfolio
### "print" treats the % as a special character you need to add, so it can know, that when you type "f"
###   the number (result) that will be printed will be a floating point type, and the ".4" tells your "print"
###   to print only the first 4 digits after the point.

print( 1/(  np.sqrt(np.dot(w_Sharpe,np.dot(w_Sharpe,covar))*252)/r.dot(w_Sharpe) ) )
### Above is optimized objective function -- the max Sharpe ratio.
### It can also be found using print(sharpe(ret(r,w_Sharpe),vol(w_Sharpe,covar)))
```

```
[0.    0.768 0.171 0.    0.    0.03  0.031 0.   ]
return:  0.2969 risk:  0.2697
1.1006526861512465
```

- In [37], the objective function to be minimized is the inverse of the Sharpe ratio – the solution is the same as maximizing Sharpe ratio. The solution to [37] shows the maximized Sharpe ratio is 1.1007 whereas the portfolio return, and standard error are respectively 29.69% and 26.97%. This appears to be superior to that of minimum variance portfolio with a portfolio return of a much lower 6.22% and volatility of 19.08%.

Please upload Chapter1-1portecom.ipynb and follow the computing steps in Jupyter Notebook

In [38]:
```
w = w_min ### w is now optimal portfolio weights, sum to 1
num_ports = 100
gap = (np.amax(r) - ret(r,w_min))/num_ports
all_weights = np.zeros((num_ports, len(dfret.columns))) ### all_weights is 2D 100 x 8 zero matrix
ret_arr = np.zeros(num_ports) ### this is a 1-tuple of 100 zeros
vol_arr = np.zeros(num_ports)

for i in range(num_ports): ### this means looping from i=0 to 1,2,3,4,....,99 (100 loops in total)
    port_ret = ret(r,w) + i*gap
    double_constraint = LinearConstraint([np.ones(dfret.shape[1]),r],[1,port_ret],[1,port_ret])
    ### above means np.ones(dfret.shape[1]) or A * w >< 1,1; r * w >< port_ret, i.e. constraints port wts sum to 1 and
    ### portf exp return == port_ret

    ### Create x0: initial guesses for weights.
    x0 = w_min
    ### Define a function for annualized portfolio volatility.
    portfvola = lambda w: np.sqrt(np.dot(w,np.dot(w,covar))*252)
    res = minimize(portfvola,x0,method='trust-constr',constraints = double_constraint,bounds = bounds)
       ### Above double constraints mean unit vector .dot(w) = 1; r .dot(w) = port_ret; minimize chooses wts w
    all_weights[i,:]=res.x ### i row x 8 optimal wts (at row i)
    ret_arr[i]=port_ret
    vol_arr[i]=vol(res.x,covar)

sharpe_arr = ret_arr/vol_arr  ### sharpe_arr is 100 x 1 array since it is ret_arr[100]/vol_arr[100] element by element

plt.figure(figsize=(20,10))
plt.scatter(vol_arr, ret_arr, c=sharpe_arr, cmap='viridis')
### in plt.scatter, c is a scalar or sequence of n numbers to be mapped to colors using cmap
### in plt, for sequential plots, 'viridis' gives colors across the 3D representation of vol_arr, ret_arr, sharpe_arr
### c= in front of third dimension sharpe_arr gives the colors in that dimension, otherwise dots will be all blue
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.show()
```



In code line [38], the Markowitz mean-variance efficient portfolio frontier is drawn under the constraints of positive weights and for different required returns k

Forming Optimal Portfolios

Please upload Chapter1-2.ipynb and follow the computing steps in Jupyter Notebook

- Combine the 8 ecommerce stocks with 8 stocks from technology -- Broadcomm, Intel, Microchip, Micron, Qualcomm, Samsung, SK Hynix, and SMIC, and 8 stocks from the food industry -- Tyson, Pepsico, Nestle, Mondelez, Kweichow, Diageo, Danone, and Anheuser-Busch.

In [2]:
```
### We can import the data set. The dataframe name is now df
dftech = pd.read_csv('ret_porttech.csv',index_col=[0]) ### index_col=[0] removes default unnamed index column
dfecom = pd.read_csv('ret_portecom.csv',index_col=[0])
dffood = pd.read_csv('ret_portfood.csv',index_col=[0])
```

In [3]:
```
dfmerge = pd.merge(dftech,dfecom, on = ['day', 'month','year'])
dfmerge.columns=['BroadR','IntelR','MicroCR','MicronR','QualR','SamR','SKR','SMICR','day','month',
                 'year','AliR','AmazR','EbayR','RakR','SunR','WayR','ZalR','JDR'] ### Rename the columns
dfmerge = pd.merge(dfmerge,dffood, on = ['day', 'month','year'])
```

In [4]:
```
print(dfmerge)
```

```
      BroadR    IntelR   MicroCR   MicronR     QualR      SamR       SKR  \
0  -0.007147 -0.005204 -0.000470 -0.008461  0.001070 -0.008811 -0.016000
1  -0.015882 -0.001649 -0.019461 -0.011244  0.001221 -0.016732  0.009631
2   0.013167  0.003570  0.017265 -0.003171 -0.000305  0.017838  0.022118
3   0.002150  0.003557  0.011554  0.013520  0.001830  0.027787  0.031781
4   0.020138 -0.001914  0.006653  0.006247 -0.000305  0.000538  0.004028
```

Please upload Chapter1-2.ipynb and follow the computing steps in Jupyter Notebook

- Use the training data set from 2017 till 2020, and a test set of 112 data points from beginning of 2021 to about June 2021. See [5]:

```
trainingset = dfmerge[(dfmerge['year']<2021)]
testset = dfmerge[(dfmerge['year']==2021)]
testset = testset.iloc[0:111,:] ### choose only the first half year of test set
```

See [11]:

```
import scipy
from scipy.optimize import Bounds
bounds = Bounds(0, 1)  ### No shortsale
```

```
from scipy.optimize import LinearConstraint
linear_constraint = LinearConstraint(np.ones((trainingset.shape[1],), dtype=int),1,1)
```

```
covar=trainingset.cov()
r = np.mean(trainingset,axis=0)*252
```

```
def ret(r,w):
    return r.dot(w)
def vol(w,covar):
    return np.sqrt(np.dot(w,np.dot(w,covar))*252)
def sharpe (ret,vol):
    return ret/vol
```

After finding the MVP in [11], and the max Sharpe ratio portfolio in [13], the program computes the efficient frontier with positive weights in [16]:

Please upload Chapter1-2.ipynb and follow the computing steps in Jupyter Notebook

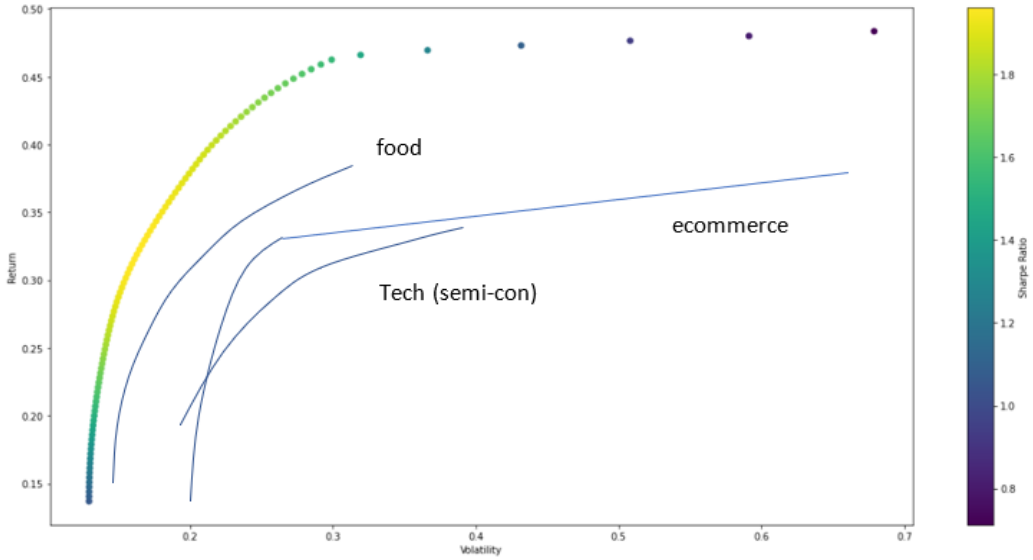
```
w = w_min   ### w is now optimal portfolio weights, sum to 1
num_ports = 100
gap = (np.amax(r) - ret(r,w_min))/num_ports
### np.amax in numpy returns max in the array -- since weights sum to 1 and are bounded in (0,1). max portf ret is amax(r)
### The aboove range given by gap starts at ret given by Min Var Portf to Max of all mean returns -- maximum possibe

all_weights = np.zeros((num_ports, len(trainingset.columns)))   ### all_weights is 2D 100 x 24 zero matrix
### Note: len(trainingset.columns) is 24 -- there are 24 stocks here
### print(np.shape(all_weights)) gives (100,24) -- same as print(all_weights.shape) that gives (100,24)
ret_arr = np.zeros(num_ports) ### this is a 1-tuple of 100 zeros
vol_arr = np.zeros(num_ports)

for i in range(num_ports): ### this means looping from i=0 to 1,2,3,4,....,99 (100 loops in total)
    port_ret = ret(r,w) + i*gap
    double_constraint = LinearConstraint([np.ones(trainingset.shape[1]),r],[1,port_ret],[1,port_ret])
      ### Above, objective minimization is doubly constrained to have wts sum to one and Sum wts x rets sum to port_ret
      ### above means np.ones(dfret.shape[1]) or A * w >< 1,1; r * w >< port_ret, i.e. constraints port wts sum to 1 and
      ### portf exp return == port_ret
    ### Create x0: initial guesses for weights.
    x0 = w_min
    ### Define a function for portfolio volatility.
    portfstderr = lambda w1: np.sqrt(np.dot(w1,np.dot(w1,covar))*252)
    optweight = minimize(portfstderr,x0,method='trust-constr',constraints = double_constraint,bounds = bounds)

    all_weights[i,:]=optweight.x  ### 24 x 1 optimal wts at row i
    ret_arr[i]=port_ret
    vol_arr[i]=vol(optweight.x,covar)
sharpe_arr = ret_arr/vol_arr  ### sharpe_arr is 100 x 1 array since it is ret_arr[100]/vol_arr[100] element by element
```

[16]: Produce the efficient frontier of the portfolio of 24 stocks (2017 – 2020) as shown on RHS.

Please upload Chapter1-2.ipynb and follow the computing steps in Jupyter Notebook

```
plt.figure(figsize=(20,10))
plt.scatter(vol_arr, ret_arr, c=sharpe_arr, cmap='viridis')
### in plt.scatter, c is a scalar or sequence of n numbers to be mapped to colors using cmap
### in plt, for sequential plots, 'viridis' gives colors across the 3D representation of vol_arr, ret_arr, sharpe_arr
### c= in front of third dimension sharpe_arr gives the colors in that dimension, otherwise dots will be all blue
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.show()
```

Clearly the larger combined industry set of stocks produce a better performing efficient frontier – for any given expected return, the portfolio volatility is smaller. This is evidence of the benefit of portfolio diversification when more stock returns are added which have lower variances and low correlations with the others.



Test Set Prediction Results for Portfolio Optimization

Please upload Chapter1-2.ipynb and follow the computing steps in Jupyter Notebook

After the optimal weights for every k are computed using the training set, suppose we apply these weights to fresh data that is out-of-sample in the test data set. Would we obtain the out-of-sample or test set predicted portfolio returns and volatilities that are similar to that mapped in the training set? We use the test set to construct the out-of-sample mean and covariance matrix.

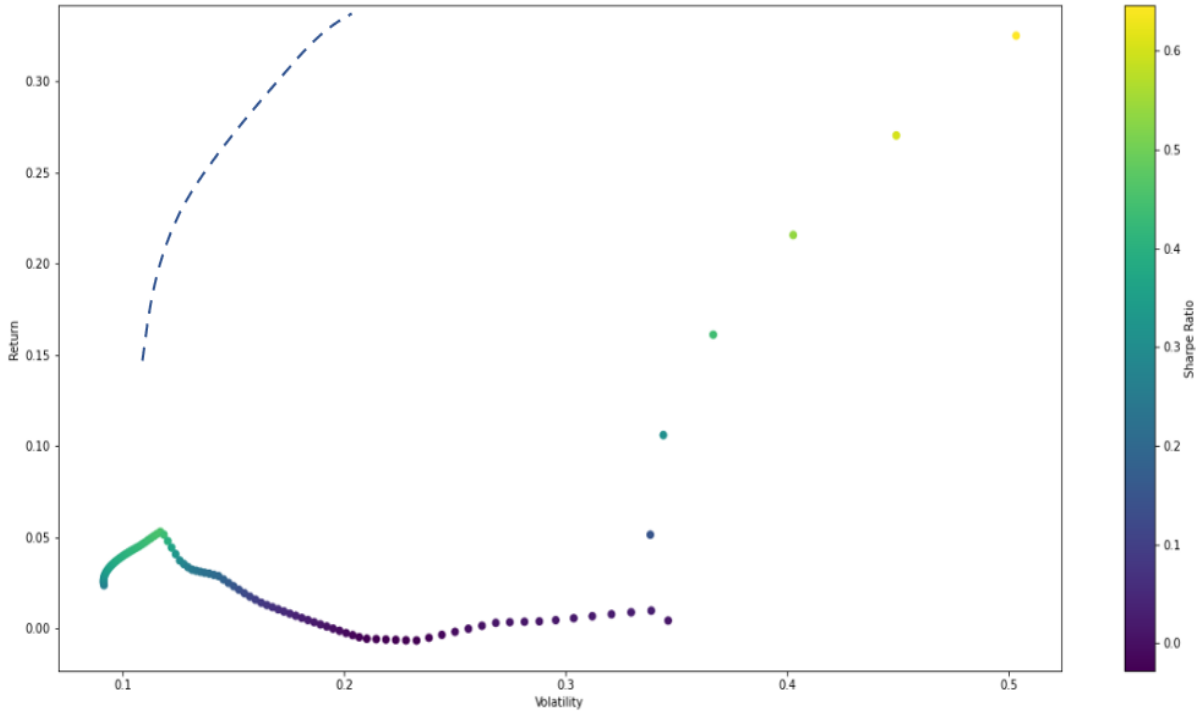
In [21]:
```
### Below the test set data is used with the optimal wts computed with training set to form the eff portf frontier
testcovar=testset.cov()
testr = np.mean(testset,axis=0)*252

### initialize
testport_ret = np.zeros(num_ports)
testport_vol = np.zeros(num_ports)

for i in range(num_ports):
    testport_ret[i] = ret(testr,all_weights[i,:])
    testport_vol[i]= vol(all_weights[i,:],testcovar)

testport_sharpe = testport_ret/testport_vol

plt.figure(figsize=(20,10))
plt.scatter(testport_vol, testport_ret, c=testport_sharpe, cmap='viridis')
### in plt.scatter, c is a scalar or sequence of n numbers to be mapped to colors using cmap
### in plt, for sequential plots, 'viridis' gives colors across the 3D representation of vol_arr, ret_arr, sharpe_arr
### c= in front of third dimension sharpe_arr gives the colors in that dimension, otherwise dots will be all blue
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.show()
```

Test Set Prediction Results for Portfolio Optimization

Please upload Chapter1-2.ipynb and follow the computing steps in Jupyter Notebook

Clearly, the training set mean return μ and portfolio return volatility based on training set Σ would be different for the test set mean return and covariance matrix Σ. The resulting portfolio performances based on the fresh test returns data show substandard performance as compared with the in-sample dotted efficient frontier curve.

Output from [21]:



Reasons Why Out-of-Sample Results are Substandard

- Performance errors are created due to the randomness of returns that could produce large deviations from the expectations in μ. In this case, the stocks with high weights computed from the training data set has generally lower returns in the test data set than in the training data set.
- Another possible reason is that the estimation of the covariance matrix Σ using the training set data could contain random errors and similarly this would affect the out-of-sample portfolio return volatility based on the fresh returns.
- Both mean return and portfolio return volatility errors could be reduced if we have a model to better explain the stock returns such as in a multi-factor linear model. In this case, if the factors are correctly found and anticipated, the errors remain only in the residuals of the linear model, and these errors would be smaller.
- We do not go into multi-factor models here but suggest another method popular in machine learning approach to attempt to reduce the error in the use of in-sample estimate of Σ.
- Later, we show how a more accurate forward prediction of future expected return could be used to improve on training set optimal portfolio weights for future investing results.

Denoising the Correlation Matrix

- One often thinks of the stock return covariance and corresponding correlation matrices as constant matrices. Sample estimates are random due to small sampling errors. With fixed N number of stocks and a time series T that approaches $+\infty$, i.e. $N/T \downarrow 0$, the sample estimate of the correlation matrix $\hat{\Sigma}$ would converge to the population constant of $\Sigma_{N\times N}$ given stationary time series of the stock returns.
- However, suppose N, T are increasingly large, so that N/T does not converge to zero, but instead converge as N, $T \to +\infty$, to some finite positive number, i.e., $0 < N/T < 1$. In small sample, it is also the case that $0 < N/T < 1$. Then every element in the (small) sample correlation matrix is a random variable. We may treat the sample correlation matrix in this context as a random matrix which is in general a matrix-valued random variable.
- A square matrix A is positive definite if for any non-zero conformable vector w, $w^T A w > 0$. If $A_{N\times N}$ $w_{iN\times 1} = \lambda_i\ w_{iN\times 1}$ for scalar λ_i , then λ_i is called an eigenvalue of A while w_i is the corresponding eigenvector. There are N number of eigenvalues and eigenvectors. The eigenvalues are found by solving the determinant, $|A - \lambda_i I|$. Some of these may be repeated. In general, these eigenvectors are non-unique, so additional constraints to make them normalized and orthogonal to each other are imposed. In other words, for any i,j eigenvectors, w_i , w_j: $w_i^T w_i = 1$, $w_j^T w_j = 1$, and $w_i^T w_j = 0$ for $i \neq j$.

Denoising the Correlation Matrix

- A positive definite symmetric matrix has strictly positive eigenvalues. A matrix is positive definite if it is symmetric, and all its eigenvalues are strictly positive. Hence sample or else population covariance and correlation matrices of stock returns, that are both symmetric and positive definite, have positive eigenvalues. The important characterization of the random matrix is that the eigenvalues are random variables.
- Suppose a vector of random variables change over time due to independent and identically distributed noise (with mean 0 and variance of 1) and not signals. Noises, unlike signals, are not systematic factors in the market. Then the **Marcenko-Pastur Theorem** states that as N, T $\rightarrow +\infty$, and N/T converge to a finite positive number in (0,1), then the eigenvalues λ_i of the sample correlation matrix converge to a probability density function (pdf) as follows (and not a degenerate single value distribution of probability one).

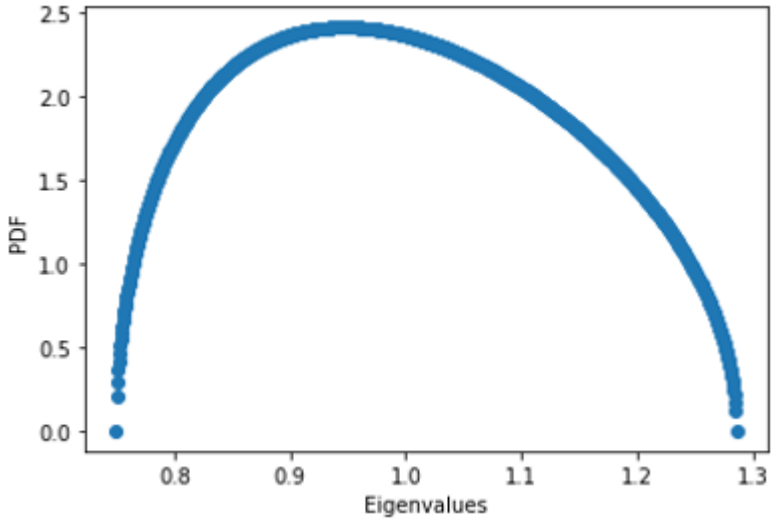
$$\text{pdf}\ (\lambda_i) = \frac{\sqrt{(U-\lambda_i)(\lambda_i-L)}}{2\pi\lambda_i\sigma^2 N/T}$$

where $U = \sigma^2\left(1+\sqrt{N/T}\right)^2$ and $L = \sigma^2\left(1-\sqrt{N/T}\right)^2$, for $\lambda_i \in [L,U]$. Outside of $[L,U]$, pdf $(\lambda_i) = 0$.

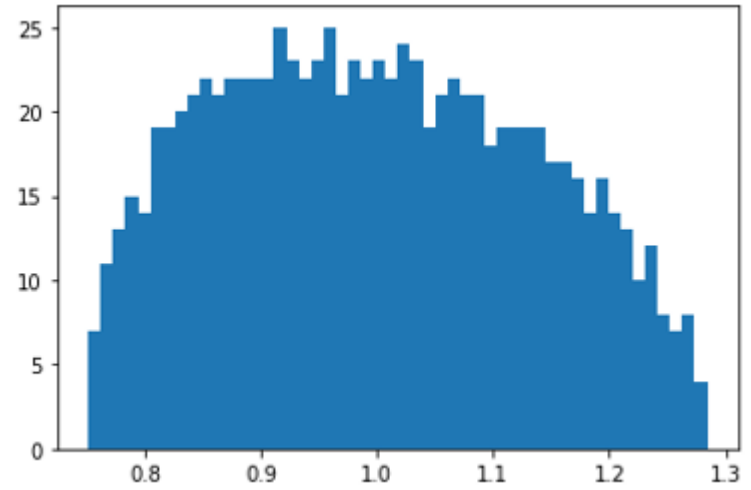
Denoising the Correlation Matrix

- T × N (T=50,000 and N=900) random normal numbers with mean 0 and variance 1 are generated as noises to find estimates of the eigenvalues of the sample correlation matrix of the simulated N number of random numbers.

Theoretical Distribution of eigenvalues

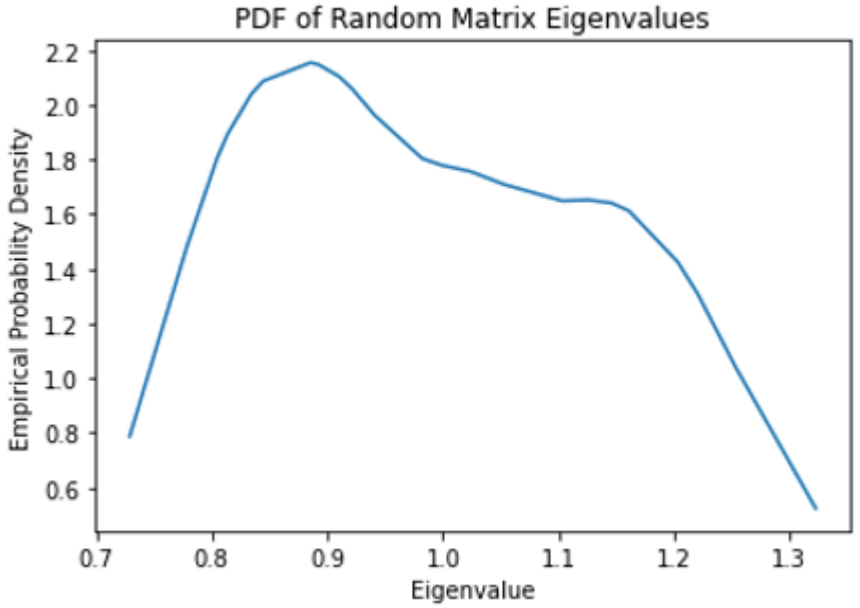


Empirical Distribution of eigenvalues



Denoising the Correlation Matrix

However, when we use only N=24, T=889 (size of the training set data) similar random normal numbers with mean zero and variance one, the empirical pdf is as follows. This empirical pdf does not approximate the theoretical pdf as closely since N, T are small. Nevertheless, it still shows that when the randomness is just independently identically distributed noises, eigenvalues can still range from 0.7 to 1.3.



The idea is to look for significantly larger values of eigenvalues that would signal the presence of signals and not just noises.

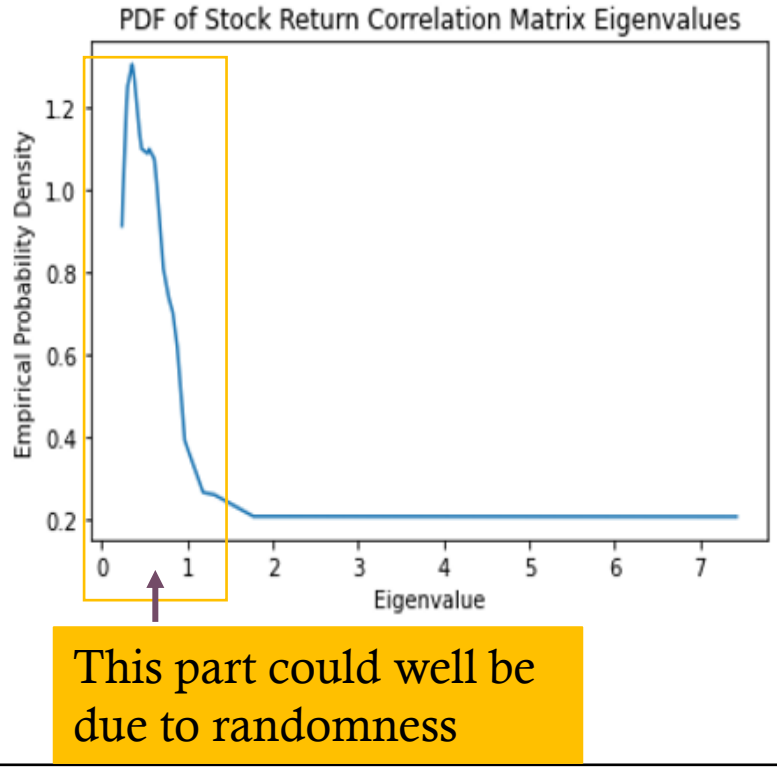
Eigenvalues for true cov matrix I_{24x24} based on 889 time values from (0,1) should be all ones, i.e. $Iv_1=\lambda_1 v_1$, so $\lambda_1=\lambda_2=\lambda_3=\ldots=\lambda_{24}=1$ repeated for 24 times.
Yet random cov matrix produces the eigenvalue distribution 0.7-1.3.Average is about 1.

Denoising the Correlation Matrix

Suppose we use the actual stock return data of the N=24 stocks over time series T=889 in the training data set to form the correlation matrix $\Sigma_{24\times24}$, 'corr_matrix'. In the program, this is computed as corr_matrix = trainingset.corr(). The eigenvalues and eigenvectors of $\Sigma_{24\times24}$ are found using the following. 'eigenvalues' show the list of 24 computed eigenvalues.

Note that the correlation matrix 'corr_matrix' is the sample correlation matrix of the training set returns, which is also the correlation matrix of the standardized training set returns. It may be different from the simulated random normal numbers with mean zero and variance one in that the off-diagonal correlation numbers are theoretically not necessarily zeros in the returns sample.

The computed empirical pdf of the eigenvalues is shown. Most of the eigenvalues falls below 1.3. Only those eigenvalues > 1.3 represent combinations of the standardized stock return random variables (with mean 0 and variance 1) that produce bigger variances that could be due to true market signals and not white noise.

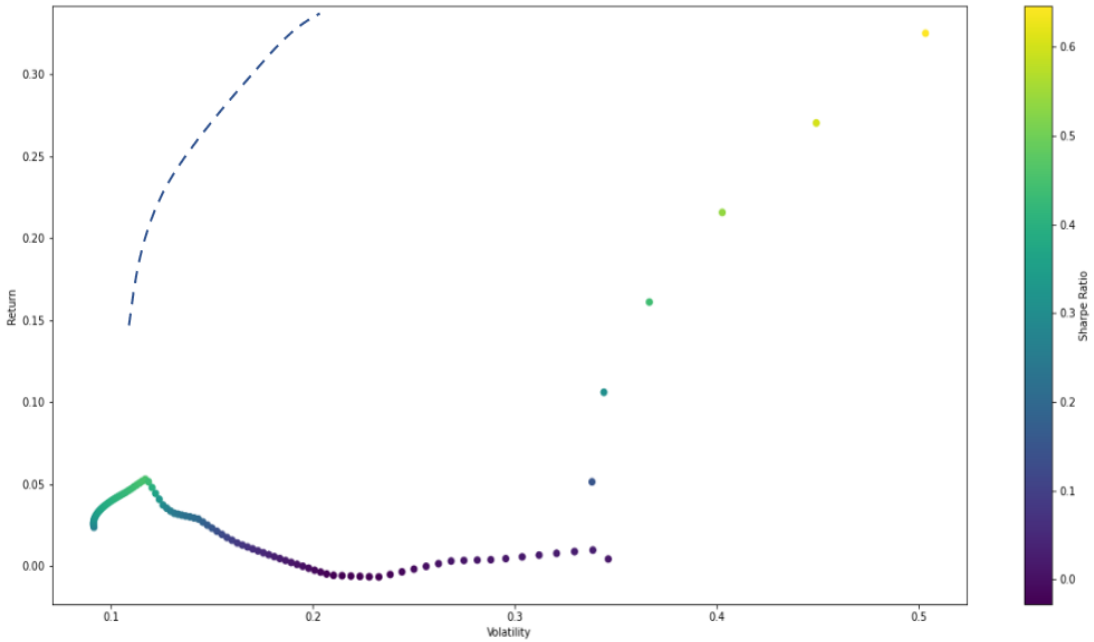


Chapter1-5.ipynb goes on to denoise the correlation matrix using the constant residual eigenvalue method.

Constant Residual Eigenvalue Method

- Denoising the empirical correlation matrix is one way to produce a more accurate correlation matrix, hence also covariance matrix, for portfolio optimization. We show one method of denoising – the constant residual eigenvalue method. This approach is to set a constant eigenvalue for all random eigenvalues that are smaller than or equal to U which we can approximate using 1.3 as seen in the pdf of the random matrix eigenvalues with N=24 and T=889. This constant is the average of all the eigenvalues below 1.3. Thus, the total variance is preserved.
- We use the denoised covariance matrix to re-compute the optimal weights, 'all_weights', of portfolio based on the training data set with the same portfolio return means but using the denoised covariance matrix.
- After that we employ the test data with the optimal weights computed as above to find the new portfolio results. There is no significant change[+] from the case when the returns covariance matrix is not denoised. This could be due to larger errors in the portfolio expected returns than in the portfolio return variances. Such methods may be more effective in other problems where the means do not change as much.

[+]Chapter1-6.ipynb



+Chapter1-6.ipynb. Test set frontier using denoised cov from training set

Using a More Accurate Forward Predictor

As shown earlier, diversification is limited in improving results if the future returns have means that are not similar to the means based on historical returns used to compute the optimal portfolio weights. In fact, most of the times, accurate predictions of future means count more toward better portfolio performance than diversification itself. This is why prediction of future (expected) returns or similarly prediction of future stock prices is such an important business in financial data science. In this section, we show how a more accurate forward prediction of future expected return could be used to improve on training set optimal portfolio weights for future investing results.

[+]Chapter1-7.ipynb

Using a More Accurate Forward Predictor

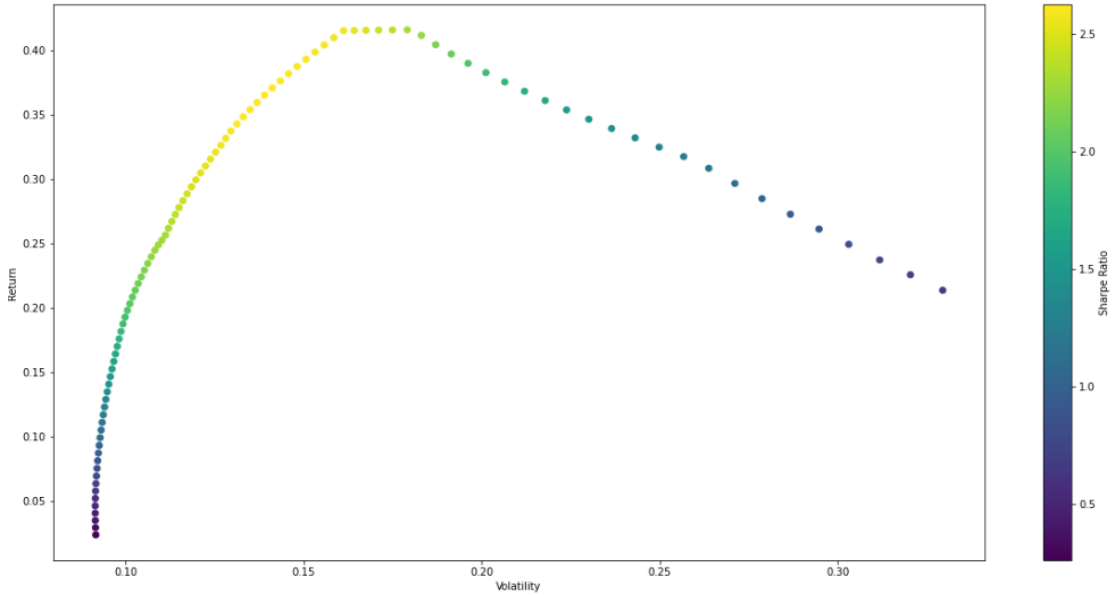
We employ the same training set return data of the 24 stocks. This is used as before to compute the covariance matrix. For the stock mean returns, however, we assume there is a good prediction model – for illustration, suppose these predicted returns are identical to the mean returns based on the 224 returns observations per stock in 2021. (This is of course forward looking – but is used as an illustration here.) This mean vector is then used together with the historical covariance to compute the optimal portfolio weights.

The returns in the first 111 observations of 2021 form the test data set which is a subset of the 224 observations. The test data set return means are therefore randomly close to that based on the 224 observations, but they are not identical.

[+]Chapter1-7.ipynb

Using a More Accurate Forward Predictor

After the optimal weights for every (required return) k are computed using the training set, suppose we apply these weights to fresh data that is out-of-sample in the test data set of 111 observations. This test set is used to construct the out-of-sample mean and covariance matrix of the portfolio returns. The results are shown below. Clearly, for whichever set of optimal weight chosen (depending on the target required returns), the performance is close to the ex-ante frontier (for volatility < 0.16.



[+]Chapter1-7.ipynb. Test set frontier using regular cov from training set but 'informed' mean returns

- The machine learning approach could attempt to reduce the generalized portfolio performance problem by attempting to form the expected returns and estimated covariances based on predictions rather than based simply on historical data. We can use factor models to try to find better estimates of future returns. We can also use Neural Networks or other ML prediction methods to predict next period expected stock returns and use these for constructing optimal weights with the training set data.
- We can of course choose only stocks with predictably good performances to form the portfolio.
- The covariance used for computing the optimal portfolio weights to use in the test data set could also use predictions of variances of individual stock returns from ML models or econometrics model such as the generalized autoregressive conditional heteroskedasticity (GARCH). We can assume the historical correlations are more stable, and use these together with the estimated variances to form a covariance matrix for the computation of optimal portfolio weights.
- In testing, a one period ahead test with a rolling training data set could be more accurate than testing over a time series of test data. For financial investment or trading, it is not sufficient to produce small mean-squared errors in the test; the effect test is to back-test a trading strategy and see if there is significant profit.

In-Class Practice Exercise (not graded):

Try running Chapter1-7.ipynb

Use same data set of 24 stocks in Chapter1-2.ipynb for this exercise.

Assume there is a good prediction model – for illustration, suppose these forward predicted returns are identical to the mean returns based on the 224 returns observations per stock in 2021, Use this mean vector together with the historical covariance to compute the optimal portfolio weights.

Then use these optimal weights on the same 100 required returns in Chapter1-2 to perform a portfolio performance on the test data set of 111 observations as in Chapter1-2.

(1) Show the ex-post efficient frontier

(2) Show the mean returns of the 24 stocks on (i) the training data set (ii) the forward prediction values (iii) the test set

(For this exercise, you should get the same results as in Chapter1-7.ipynb of slide #38.)

In [26]:
```
origmean=np.mean(trainingset,axis=0)*252
origmean1=pd.Series(origmean)
fwdmu1=pd.Series(fwdmu)
testr1=pd.Series(testr)
# Combine 3 series.
df2=pd.concat([origmean1,fwdmu1,testr1],axis=1)
df2.columns =['TrainingSet_Means', 'Forward_Predicted_Means', 'TestSet_Means']
print(df2)
```

	TrainingSet_Means	Forward_Predicted_Means	TestSet_Means
BroadR	0.286417	0.470443	0.204632
IntelR	0.116410	0.016049	0.225270
MicroCR	0.253299	0.243650	0.071833
MicronR	0.384098	0.250944	0.262756
QualR	0.289594	0.285015	-0.021806
SamR	0.277415	-0.010463	-0.051485
SKR	0.306904	0.167307	0.081670
SMICR	0.066511	-0.228100	0.093388
AliR	0.224909	-0.690943	-0.199983
AmazR	0.358371	0.021768	0.021125
EbayR	0.131242	0.320447	0.756070
RakR	0.025977	0.078155	0.446730
SunR	-0.112776	-0.766110	-0.906423
WayR	0.486958	-0.226671	0.379485
ZalR	0.274734	-0.157747	0.271807
JDR	0.328025	-0.221995	-0.364407
TysonR	0.049259	0.329792	0.284453
PepsicoR	0.131077	0.127670	-0.040782
NestleR	0.163392	0.137228	0.123623
MondelezR	0.090088	0.099645	0.108762
KweichowR	0.458848	-0.059354	-0.059943
DiageoR	0.129241	0.300348	0.378728
DanoneR	0.007296	-0.047896	0.239929
AnheuserR	-0.108380	-0.225020	0.021732

In [27]:
```
from sklearn.metrics import mean_squared_error
mse1 = mean_squared_error(df2['TrainingSet_Means'], df2['TestSet_Means'])
mse2 = mean_squared_error(df2['Forward_Predicted_Means'], df2['TestSet_Means'])
print(mse1,mse2)
```

```
0.11644235124989104 0.07040596164238468
```

End of Class