INDIAN INSTITUTE OF TECHNOLOGY KANPUR

SEMESTER PROJECT 2023

Mentors
Dwija Kakkad
Devanshi Rastogi
Shruthi Munnur

PROJECT REPORT

PORTFOLIO MANAGEMENT AND RISK ANALYSIS

FINANCE AND ANALYTICS CLUB

Contents

Chapter 1

Introduction

1.1 About the project

Project Portfolio Management (PPM) and Risk Analysis are two essential practices in the field of project management. PPM involves the centralized management of a collection of projects and programs to achieve strategic objectives, while risk analysis focuses on identifying, assessing, and mitigating potential risks associated with these projects. Together, they form a crucial framework for organizations to make informed decisions and optimize their project investments.Risk analysis is the systematic process of identifying, assessing, and managing risks associated with projects. It involves evaluating potential threats and opportunities that can affect project objectives, and developing strategies to mitigate or exploit them. This is going to be the basic framework of this project.

Chapter 2

Contents

2.1 Week 0

In Week 0, we focused on mastering the fundamental aspects of Python syntax and exploring key modules such as NumPy and Pandas. We learned about variables, data types, conditionals, loops, and functions, which are essential building blocks of any programming language. Additionally, we delved into NumPy, which equipped us with the ability to work with multidimensional arrays and perform complex mathematical operations efficiently. We also explored Pandas, which enabled us to manipulate and analyze large datasets using data frames, which is the basic requirement for project dealing with large amount of data.

2.2 Week 1

2.2.1 Sharpe ratio

The Sharpe ratio compares the return of an investment with its risk. It's a mathematical expression of the insight that excess returns over a period of time may signify more volatility and risk, rather than investing skill.

Formula for Sharpe Ratio:
Sharpe ratio= $(\mathrm{R}_p - R_f)/\sigma_p$

The Sharpe ratio can be used to evaluate a portfolio's risk-adjusted performance. The Sharpe ratio can help explain whether a portfolio's excess returns are attributable to smart investment decisions or simply luck and risk.

2.2.2 Common methods for measurements of risks

Risk management is a crucial process used to make investment decisions.Some common measurements of risk include standard deviation, Sharpe ratio, beta, value at risk (VaR), conditional value at risk (CVaR), and R-squared.
R squared-R-squared is a statistical measure that represents the percentage of a fund portfolio or a security's movements that can be explained by movements in a benchmark index. R-squared values range from zero to one and are commonly stated as a percentage(0 to 100).Rest of the measures are discussed ahead.

2.2.3 Measuring portfolio performance

There are various methods which look at returns, some look on risk only. Today, there are three sets of performance measurement tools to assist with portfolio evaluations. The Treynor, Sharpe, and Jensen ratios combine risk and return performance into a single value.but each is slightly different from one another

2.2.4 Alpha and Beta

One measure of the relative volatility of a particular stock to the market is its beta (). A beta approximates the overall volatility of a security's returns against the returns of a relevant benchmark (usually the SP 500 is used).

2.2.5 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. Volatility is often measured from either the standard deviation or variance between returns from that same security or market index. Volatility is often calculated using variance and standard deviation:

Volatility over some interval of time$(\sigma_{\mathrm{T}}) = \sigma.\sqrt{T}$.

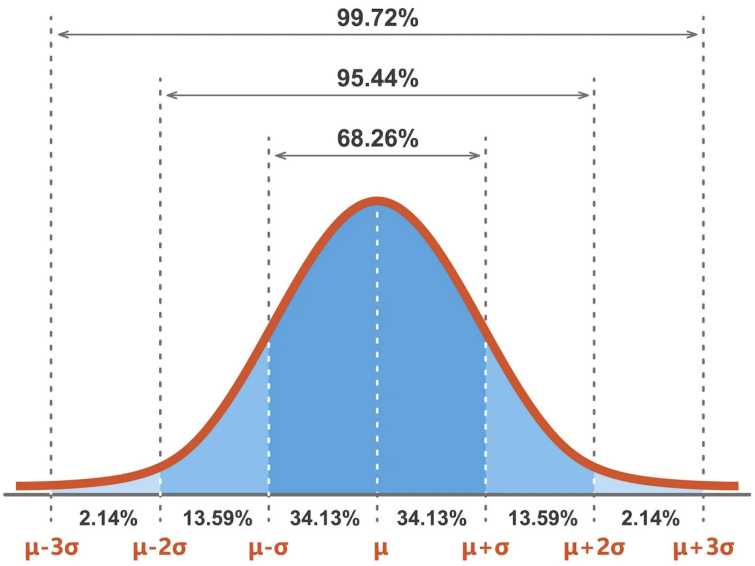
2.3 Week 2

2.3.1 Probability distribution and Expected Values

In week 2 we started first by learning about probability distributions which describes the likelihood of different outcomes or events occurring in a given set of circumstances. It provides a way to quantify and analyze uncertainty or randomness associated with a particular situation.We had learnt about three fundamental probability distributions-

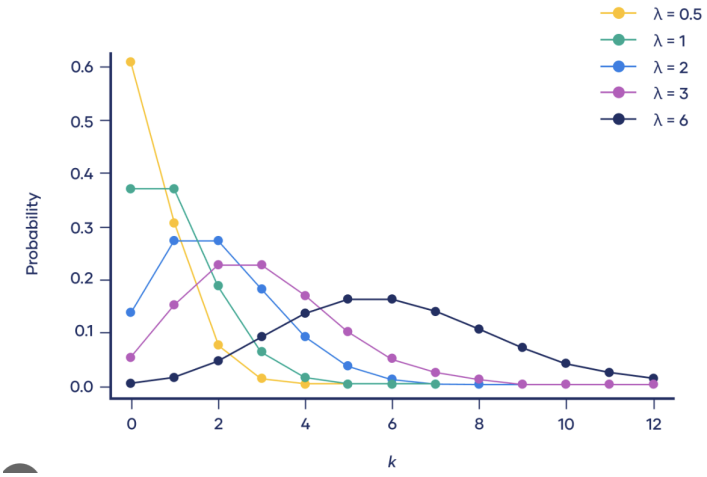
1.Normal Distribution-A probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean, and is also known as the Gaussian distribution.

Formula for normal distribution is $f(x) = \dfrac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$



2. Poisson's Distribution- It is a discrete probability distribution. The Poisson probability distribution is often used as a model of the number of arrivals at a facility within a given period of time. It is commonly used to model rare events or count data.

$$f(k;\lambda) = \Pr(X{=}k) = \frac{\lambda^k e^{-\lambda}}{k!}$$



3.Binomial Distribution- It is also a discrete probability distribution that models the number of successes in a fixed number of independent Bernoulli trials. It is commonly used to analyze situations where there are only two possible outcomes.

2.3.2 Kurtosis

Kurtosis describes the fatness of the tails found in probability distributions.
There are three kurtosis categories—

–Mesokurtic (normal),

–Platykurtic (less than normal), and

–Leptokurtic (more than normal).

Kurtosis risk is a measurement of how often an investment's price moves dramatically.

A curve's kurtosis characteristic tells you how much kurtosis risk the investment

you're evaluating has.

```
#function for calculating moment
def moment(series,degree):
    mean=sum(series)/len(series)
    moment=0
    for x in series:
        moment=moment+(x-mean)**degree
    moment=moment/len(series)
    return moment
def kurtosis(returns):
    kurt=moment(returns,4)/moment(returns,2)**2-3
    return kurt
```

2.3.3 Skewness

Skewness is the degree of asymmetry observed in a probability distribution. Distributions can exhibit right (positive) skewness or left (negative) skewness to varying degrees. A normal distribution (bell curve) exhibits zero skewness. Investors note right-skewness when judging a return distribution because it better represents the extremes of the data set rather than focusing solely on the average. Skewness informs users of the direction of outliers, though it does not tell users the number that occurs. Skewness is often found in stock market returns as well as the distribution of average individual income.

```
#function for calculatig moment
def moment(series,degree):
    mean=sum(series)/len(series)
    moment=0
    for x in series:
        moment=moment+(x-mean)**degree
    moment=moment/len(series)
    return moment
def skewness(returns):
    n=len(returns)
    skew=moment(returns,3)/(moment(returns,2)**(3/2))
    return skew
```

2.3.4 Jarque-bera test

As the Normal distribution has many benefits, we can check that the given distribution is normal or not by using Jarque-bera test. The Jarque-Bera Test is a test used to determine if a set of data values follows the normal distribution based on the data's skewness and kurtosis.

$$JB = \frac{n}{6}\left(S^2 + \frac{1}{4}(K-3)^2\right)$$

It is based on hypothesis testing which assumes null hypothesis as distribution is normal. And than we calculate p-value and test statistic. We take test statistic to be

large. If p-value is less than significance level which we generally assume to be 0.05 we reject the null hypothesis and the curve is not normal or vice versa.

```
#python function for applying jarque-bera test
import scipy.stats as st

def jarque_bera_test(returns):
    statistics , p_value = st.jarque_bera(returns)
    #scipy.stats has inbuilt for jarque-bera test

    if p_value > 0.05:
        return 1 #implies normal distribution
    else:
        return 0 #implies non-normal distribution
```

We generally get non-normal distribution as real life distributions are not normal

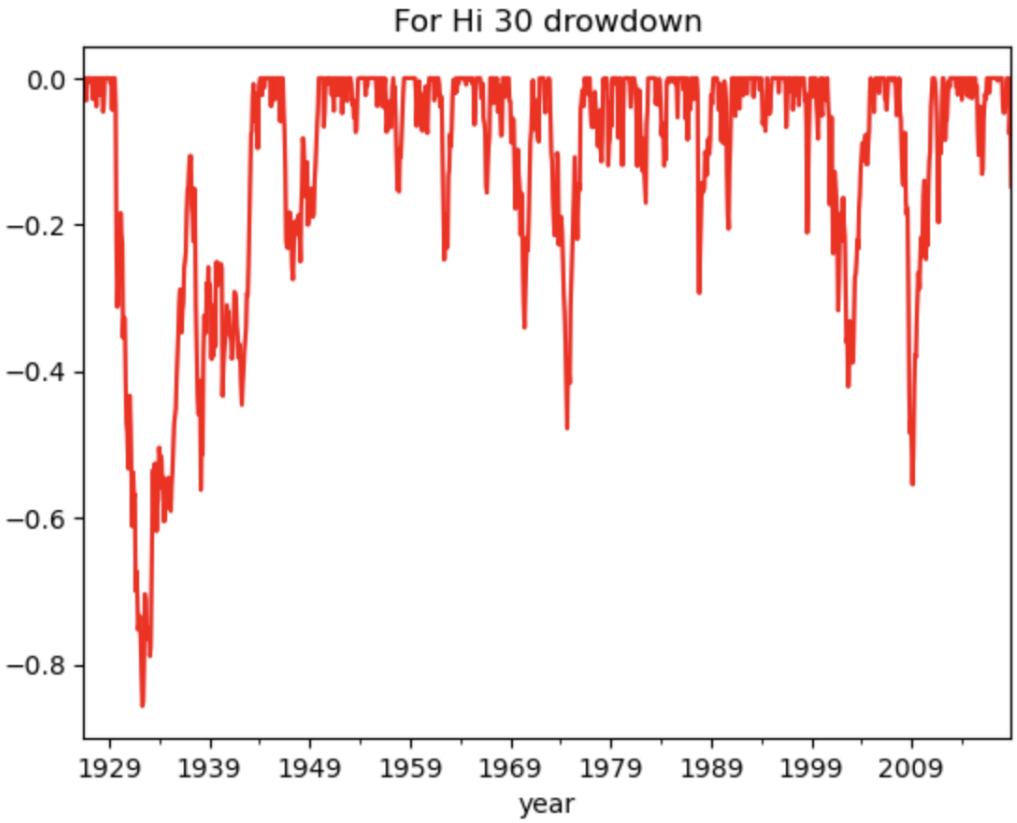
2.4 Week 3

2.4.1 Drawdown

The max drawdown is the maximum loss from the previous high to subsequent low. It measures the extent of loss from the highest point to the lowest point.
For calculating drawdown we have to first construct the wealth index which is basically done by hypothetical buy and hold investment in the asset. and than calculate difference between peaks in the specific range which gives drawdowns.

```
#python function for calculating drawdown for return_series
def drawdown(return_series):
    return_series=return_series/100
    wealth_index=1000*(1+return_series).cumprod()
    previous_peaks=wealth_index.cummax()
    drawdowns=(wealth_index-previous_peaks)/previous_peaks
    return drawdowns
```

And we obtain this graph as output when we plotted Drawdown from the above function for high 30 stocks, which are the 30 highest stocks in terms of market capitalization on the S&P500.



2.4.2 Covariance

Covariance measures the directional relationship between the returns on two assets. A positive covariance means asset returns move together, while a negative covariance means they move inversely.
Covariance is calculated by analyzing at-return surprises (standard deviations from the expected return) or multiplying the correlation between the two random variables by the standard deviation of each variable.Covariances have significant applications in finance and modern portfolio theory.

For determining covariance there is inbuilt function in python **.cov()**

2.4.3 Correlation Coefficient

The correlation coefficient is a statistical measure of the strength of a linear relationship between two variables. Its values can range from -1 to 1.
A correlation coefficient of -1 describes a perfect negative, or inverse, correlation, with values in one series rising as those in the other decline, and vice versa. A coefficient of 1 shows a perfect positive correlation, or a direct relationship. A correlation coefficient of 0 means there is no linear relationship.
By far the most common is the Pearson coefficient, which measures the strength and direction of a linear relationship between two variables.
To calculate the Pearson correlation ($\rho_{(X,Y)}$) :

$$\rho_{(X,Y)} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$

2.4.4 Value at Risk(VaR)

Value at risk (VaR) is a statistic that quantifies the extent of possible financial losses. This metric can be computed in three ways: the historical, Gaussian, and Monte Carlo methods

1. historical VaR-The historical method looks at one's prior returns history and orders them from worst losses to greatest gains—following from the premise that past returns experience will inform future outcomes

```
"""This is python implementation for calculating historical
VaR which takes confidence level and returns as input"""

def historical_VaR(returns,confidence):
    returns = sorted(returns)
    temp = int((1 - confidence) * len(returns))
    var = returns[temp]
    return -var
```

2. Gausian Method- It is a parametric approach .In this method we assume distribution to be normal. This way, potential losses can be framed in terms of standard deviation events from the mean.

```
"""This is python implementation for calculating gaussian
VaR which takes confidence level and returns as input"""

def gaussian_VaR(returns,confidence):
    n=len(returns)
    mean_return=sum(returns)/n
    stdv_return=(moment(returns,2))**(1/2)
    z_value = st.norm.ppf(1 - confidence)
    var=(mean_return+z_value*stdv_return)
    return -var
```

3. Cornish Fisher method-It is a semi parametric approach. This technique uses computational models to simulate projected returns over hundreds or thousands of possible iterations. Then, it takes the chances that a loss will occur—say, 5 percent of the time—and reveals the impact.

Formula for calculating z-score is-

$$\text{zvalue}_\alpha = z_\alpha + \frac{z_\alpha^2-1}{6}\cdot(S) + \frac{z_\alpha^3-3\cdot z_\alpha}{24}\cdot(K-3) - \frac{2z_\alpha^3-5\cdot z_\alpha}{36}\cdot(S^2)$$

and then corresponding Value at risk is-

$$\text{VaR}=-(\mu + zvalue_\alpha \cdot \sigma)$$

```
def cornish_fisher_VaR(returns,confidence):
    n=len(returns)
    mean=sum(returns)/n
    stdv=(moment(returns,2)**(1/2))
    z = st.norm.ppf(1-confidence)
    s = skewness(returns)
    k = kurtosis(returns)
    z = (z+(z**2-1)*s/6 +(z**3-3*z)*(k-3)/24 -(2*z**3-5*z)*(s**2)/36)
    var = (mean + z*stdv)
    return -var
```

2.4.5 Conditional Value at risk(CVaR)

CVaR is derived by taking a weighted average of the "extreme" losses in the tail of the distribution of possible returns, beyond the value at risk (VaR) cutoff point.It is a measure that quantifies the amount of tail risk an investment portfolio has.

It can be measure also by following two methods-

1. Historical CVaR- It is calculated by weighted average of the extreme loses beyond VaR

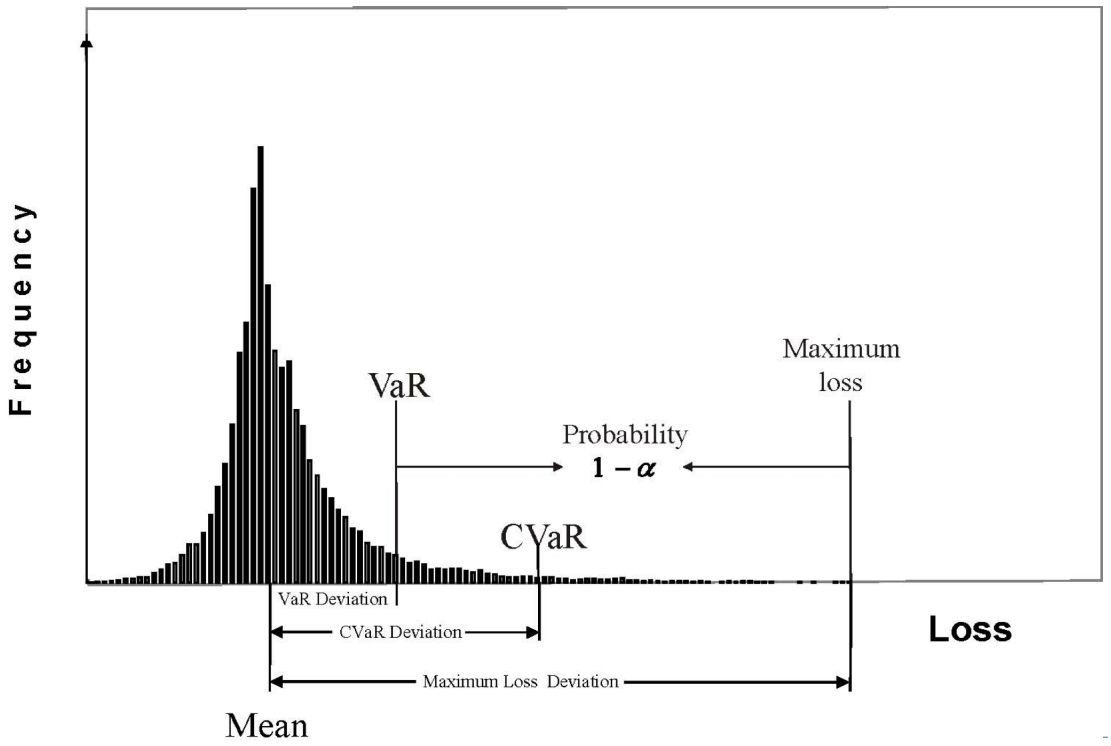
The python code for calculating historical CVaR is

```
def historical_CVaR(returns,confidence):
    temp = int((1 - confidence) * len(returns))
    cvar=0
    count=0
    for x in range(temp):
        cvar=cvar+returns[x]
        count+=1
    if count==0:
        return 0
    cvar=cvar/count
    return cvar
```

2. Gaussian CVaR-The advantage of Gaussian CVaR is that it assumes a normal distribution for portfolio returns, making it relatively simple to calculate.

The python code for calculating Gaussian CVaR is-

```
def gaussian_CVaR(returns,confidence):
    sums=0
    count=0
    for x in returns:
        if(x<gaussian_VaR(returns,confidence)):
            sums=sums+x
            count+=1
    if count==0:
        return 0
    cvar=sums/count
    return cvar
```



2.5 Week 4

In week 4 Our main aim was to apply learnt concepts so far to develop a 10 stocks portfolio which would provide maximum returns for a given risk, for which we have plotted Efficient frontier and Capital Market Line using Markowitz Portfolio Optimization.

2.5.1 Efficient Frontier

The efficient frontier is the set of optimal portfolios that offer the highest expected return for a defined level of risk or the lowest risk for a given level of expected return. Portfolios that lie below the efficient frontier are sub-optimal because they do not provide enough return for the level of risk. Portfolios that cluster to the right of the efficient frontier are sub-optimal because they have a higher level of risk for the defined rate of return.

2.5.2 Capital Market Line

The capital market line (CML) represents portfolios that optimally combine risk free returns and risky returns. The slope of the CML is the Sharpe ratio of the market portfolio. Portfolios that fall on the capital market line (CML), in theory, optimize the risk/return relationship, thereby maximizing performance. The Markowitz model is a method of maximizing returns within a calculated risk. It is also called the Markowitz portfolio theory or modern portfolio theory.

2.5.3 Portfolio Return and Volatility

Portfolio Returns are calculated by multiplying each stock returns with its weights and Portfolio Volatility is obtained by multiply weights transpose matrix with covariance matrix and than multiplying by weights matrix

Python Code for Return and Volatility functions-

```
def portfolio_return(weights, returns):
    return weights.T @ returns

def portfolio_vol(weights, covmat):
    return (weights.T @ covmat @ weights) ** 0.5
```

2.5.4 Python Implementation of Efficient frontier and CML

We have taken ten stocks with tickers and there historical data for 15 years-

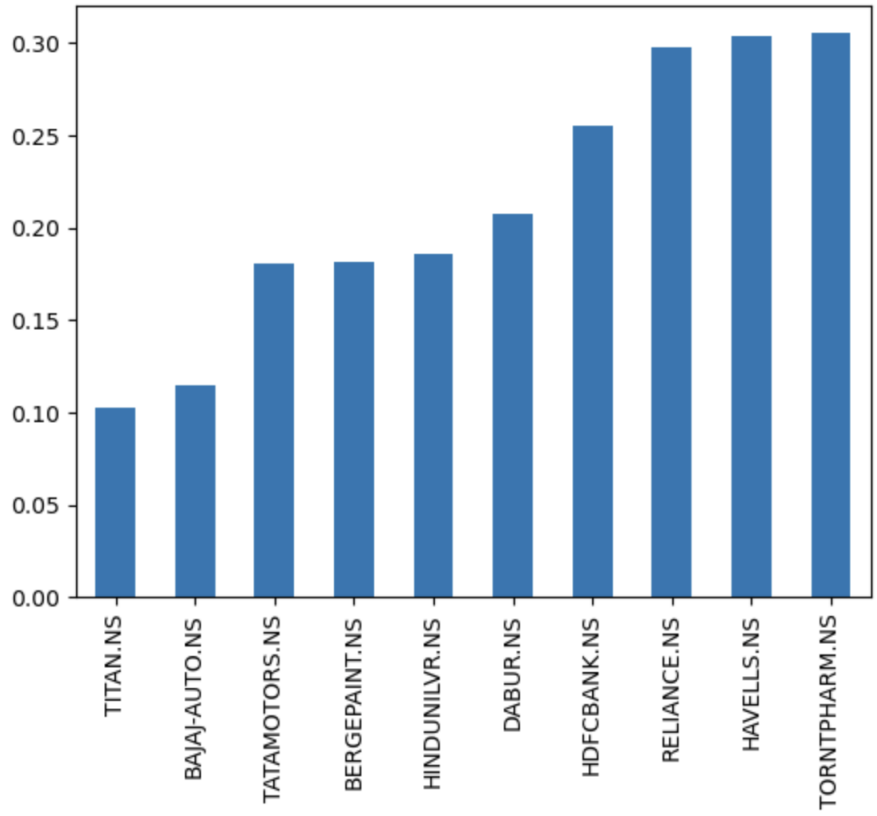
```
tickers = ['TATAMOTORS.NS', 'DABUR.NS', 'BAJAJ-AUTO.NS', 'BERGEPAINT.NS',
'HAVELLS.NS', 'HDFCBANK.NS', 'TORNTPHARM.NS', 'HINDUNILVR.NS', 'RELIANCE.NS',
'TITAN.NS']
```

Then Calculated Cov matrix and Expected returns

```
cov_matrix = returns.cov()"""where returns is a dataframe containning
returns of 15 stocks and cov_matrix is the covariance matrix obtained"""
```

We obtain following bar graph for expected returns

```
er_df.sort_values().plot.bar()"""where er_df is dataframe containning
expected returns"""
```



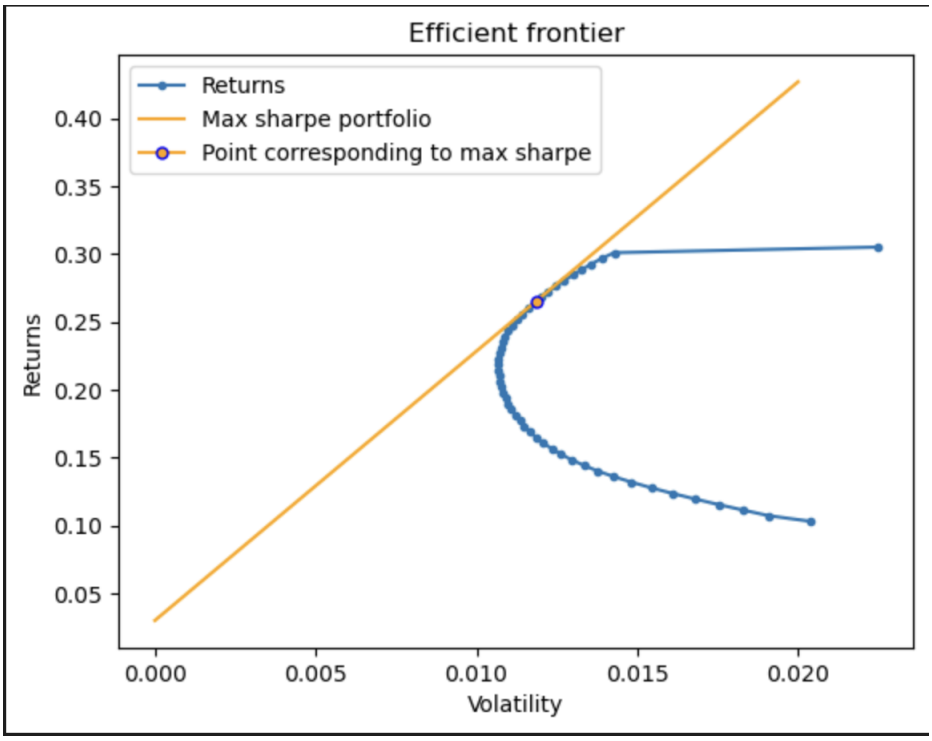
Now we have define various python functions for minimizing volatility, optimizing weights, finding weights for max sharpe ratio and plotting of efficient frontier which follows as-

```
def minimize_vol(target_return, er, cov):
    n = er.shape[0]
    init_guess = np.repeat(1 / n, n)
    bounds = ((0.0, 1.0),) * n

    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1}

    return_is_target = {'type': 'eq',
                        'args': (er,),
                        'fun': lambda weights, er:
                        target_return - portfolio_return(weights, er)}

    weights = minimize(portfolio_vol, init_guess,
                       args=(cov,), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1, return_is_target),
                       bounds=bounds)
    return weights.x

def optimal_weights(n_points, er, cov):
    target_rs = np.linspace(er.min(), er.max(), n_points)
    weights = [minimize_vol(target_return, er, cov)
               for target_return in target_rs]
    return weights

def neg_sharpe(weights, riskfree_rate, er, cov):
    returns = portfolio_return(weights, er)
    volatility =portfolio_vol(weights, cov)
    return -(returns - riskfree_rate)/volatility

def msr(risk_free_rate, er, cov):
    n = er.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n

    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }

    weights = minimize(neg_sharpe, init_guess,
                       args=(risk_free_rate, er, cov), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1,),
                       bounds=bounds)
    return weights.x
```

```
def plot_ef(n_points, er, cov):
    weights = optimal_weights(n_points, er, cov)
    rets = [portfolio_return(w, er) for w in weights]
    vols = [portfolio_vol(w, cov) for w in weights]
    ef = pd.DataFrame({
        "Returns": rets,
        "Volatility": vols
    })
    return ef.plot.line(x="Volatility", y="Returns",
                        style='.-',title="Efficient frontier")
```

Now using these function for obtaining Efficient frontier and CML

```
#finding weights,returns and volatility corresponding to max sharpe
weights=rk.msr(0.03,er_df, cov_matrix)
r=rk.portfolio_return(weights,er_df)
vol=rk.portfolio_vol(weights,cov_matrix)

#plotting efficient frontier and CML
rk.plot_ef(50, er_df, cov_matrix)
x=[0,0.020]
y=[0.03,0.020*(r-0.03)/vol+0.03]
plt.plot(x,y,label='Max sharpe portfolio',color="orange")
plt.plot(vol,r, color='orange', marker='.',markersize=10,
         markeredgecolor= 'blue',label="Point corresponding to max sharpe")
plt.ylabel("Returns")
plt.legend()
plt.show()
```



Chapter 3

Risk-kit

During the project we have also made the module of python functions for different financial and statistical tools which are used in this project.For the easier use of these functions directly later wherever we need.
Risk kit includes function such as skewness,kurtosis, sharpe-ratio, VaR ,CVaR , functions for optimizing weights and plotting efficient frontier and many more.

One of the risk kit made during project is as follows

```
import numpy as np
import pandas as pd
import scipy.stats as st
import matplotlib.pyplot
import hvplot.pandas
import hvplot as hv
from scipy.optimize import minimize

#function for calculatig moment
def moment(series,degree):
    mean=sum(series)/len(series)
    moment=0
    for x in series:
        moment=moment+(x-mean)**degree
    moment=moment/len(series)
    return moment

#function for skewness
def skewness(returns):
    n=len(returns)
    skew=moment(returns,3)/(moment(returns,2)**(3/2))
    return skew

#function for calculating kurtosis
def kurtosis(returns):
    kurt=moment(returns,4)/moment(returns,2)**2-3
    return kurt

def ind_returns(df):
    df.index = pd.to_datetime(df.index, format="%Y%m").to_period("M")
```

```
    return df

#annualised return
def annualized_return(df):
    annualized_returns = []
    for column in df.columns:
        returns = df[column]
        total_returns=1
        for cumm_return in returns:
            total_returns *= (1 + cumm_return)
        total_returns = (total_returns) ** (252/ len(returns)) - 1
        annualized_returns.append(total_returns)
    return annualized_returns

"""function for calculating annualised returns for year to year
gives list as a output"""

def annualized_returns(df):
    df['Year'] = df.index.year
    yearly_returns = df.groupby('Year')[
        'Close'].last().pct_change().dropna()
    return yearly_returns
#daily returns

#annualised volatility
def Annualized_volatility(returns):
    stdv=(moment(returns,2)**(1/2))
    volatility=stdv*(12**(1/2))
    return volatility

#function converting price dataframe to return data frame
def convert_to_returns(df):
    return df.pct_change().dropna()

#sharpe ratio
def sharpe_ratio(returns,risk_free_rate):
    n=len(returns)
    stdv=moment(returns,2)**(1/2)
    mean_returns = sum(returns)/n
    sharpe=(mean_returns-risk_free_rate)/((n**(1/2))*stdv)
    return sharpe

#Jarque bera test gives 1 for true else 0

def jarque_bera_test(returns):
    #scipy.stats has inbuilt for jarque-bera test

    statistics , p_value = st.jarque_bera(returns)
    if p_value > 0.05:
        return 1 #implies normal distribution
```

```
    else:
        return 0 #implies non-normal distribution

def drawdown(return_series):
    """
       This function is used to calculate drawdowns
       for a given return series
    """
    return_series=return_series/100
    wealth_index=1000*(1+return_series).cumprod()
    previous_peaks=wealth_index.cummax()
    drawdowns=(wealth_index-previous_peaks)/previous_peaks
    return drawdowns

#function for semideviation
def semi_deviation(returns):
    n=len(returns)
    mean=sum(returns)
    sums=0
    for x in returns:
        if(x<mean):
            sums=sums+(mean-x)**2
    semideviation=(sum/n)**(1/2)
    return semideviation

def historical_VaR(returns,confidence):
    returns = sorted(returns)
    temp = int((1 - confidence) * len(returns))
    var = returns[temp]
    return -var

def historical_CVaR(returns,confidence):
    temp = int((1 - confidence) * len(returns))
    cvar=0
    count=0
    for x in range(temp):
        cvar=cvar+returns[x]
        count+=1
    if count==0:
        return 0
    cvar=cvar/count
    return cvar

def gaussian_VaR(returns,confidence):
    n=len(returns)
    mean_return=sum(returns)/n
    stdv_return=(moment(returns,2))**(1/2)
    z_value = st.norm.ppf(1 - confidence)
    var=(mean_return+z_value*stdv_return)
    return -var
```

```
def gaussian_CVaR(returns,confidence):
    sums=0
    count=0
    for x in returns:
        if(x<gaussian_VaR(returns,confidence)):
            sums=sums+x
            count+=1
    if count==0:
        return 0
    cvar=sums/count
    return cvar

def cornish_fisher_VaR(returns,confidence):
    n=len(returns)
    mean=sum(returns)/n
    stdv=(moment(returns,2)**(1/2))
    z = st.norm.ppf(1-confidence)
    s = skewness(returns)
    k = kurtosis(returns)
    z = (z + (z**2 - 1)*s/6 + (z**3 -3*z)*(k-3)/24
        - (2*z**3 - 5*z)*(s**2)/36)
    var = (mean + z*stdv)
    return -var

def portfolio_return(weights, returns):
    return weights.T @ returns

def portfolio_vol(weights, covmat):
    return (weights.T @ covmat @ weights) ** 0.5

def minimize_vol(target_return, er, cov):
    n = er.shape[0]
    init_guess = np.repeat(1 / n, n)
    bounds = ((0.0, 1.0),) * n

    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1}

    return_is_target = {'type': 'eq',
                        'args': (er,),
                        'fun': lambda weights, er: target_return -
                                    portfolio_return(weights, er)}

    weights = minimize(portfolio_vol, init_guess,
                       args=(cov,), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1, return_is_target),
                       bounds=bounds)
    return weights.x
```

```
def optimal_weights(n_points, er, cov):
    target_rs = np.linspace(er.min(), er.max(), n_points)
    weights = [minimize_vol(target_return, er, cov) for target_return in
               target_rs]
    return weights

def plot_ef2(n_points,er,cov_matrix):
    weights=[np.array([w,1-w]) for w in linspace(0,1,n_points)]
    returns=[portfolio_return(w,er) for w in weights]
    volatility=[portfolio_vol(w,cov_matrix)for w in weights]
    ef=pd.dataframe({
        "returns":returns,
        "volatilty":volatility
    })
    return ef.plot.line(x="volatility",y="returns",style=".-")

def plot_ef(n_points, er, cov):
    weights = optimal_weights(n_points, er, cov)
    rets = [portfolio_return(w, er) for w in weights]
    vols = [portfolio_vol(w, cov) for w in weights]
    ef = pd.DataFrame({
        "Returns": rets,
        "Volatility": vols
    })
    return ef.plot.line(x="Volatility", y="Returns",style='.-',
title="Efficient_frontier")



def neg_sharpe(weights, riskfree_rate, er, cov):
    returns = portfolio_return(weights, er)
    volatility =portfolio_vol(weights, cov)
    return -(returns - riskfree_rate)/volatility

def msr(risk_free_rate, er, cov):
    n = er.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n

    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }

    weights = minimize(neg_sharpe, init_guess,
                       args=(risk_free_rate, er, cov), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1,),
                       bounds=bounds)
    return weights.x
```

Chapter 4

Limitations

Markowitz method provide some nice insights about portfolio management and risk analysis in a simpler way but it have several limitations such as-

1.This method highly depends on historical data for portfolio returns and volatility. It assumes that the given historical return will remain to be continuous in the future, which not be necessarily true as the financial market keeps on changing continuously.This can be overcome to some extent by using future prediction data from various financial model instead of historical data.

2.The Markowitz method assumes that asset returns follow a normal distribution. However, in reality, asset returns often exhibit non-normal distributions, with fat tails and skewness. This assumption can lead to inaccurate estimates of risk and return, as well as misallocation of portfolio weights.

3.Markowitz Portfolio Theory also assumes that the correlation between assets is constant, which may not always be accurate. It may vary with time

4.Inability to Capture Behavioral Factors: The Markowitz model assumes that investors are solely focused on maximizing expected returns and minimizing risk. It does not consider psychological factors, such as investor sentiment, behavioral biases, or market irrationality, which can influence investment decisions.

5. Single-Period Focus: The Markowitz model is a single-period model that assumes a static investment horizon. It does not consider dynamic changes in the portfolio over time, changing risk preferences, or the impact of rebalancing decisions.

6.Difficulty in Estimating Covariance Matrix: The theory requires estimating the covariance matrix of asset returns, which represents the interdependencies among the assets. Estimating a reliable covariance matrix, especially when the number of assets is large, can be challenging due to limited data or instability of correlations over time.