Portfolio Optimization with Monte Carlo Simulation and Modern Portfolio Theory

Abstract

This project will follow the classic Modern Portfolio Theory of Harry Markowitz. The project aims to utilize Monte Carlo simulation and Modern Portfolio Theory (MPT) to determine the optimal weights of stocks in a portfolio. The goal is to construct an efficient frontier using historical stock data and MPT, allowing for maximized returns while minimizing risks. A Monte Carlo simulation will be performed to test various stock weights in the portfolio of specific stocks in order to find the optimal allocation.

Introduction

1. What is Modern Portfolio Theory

Modern Portfolio Theory (MPT) is a theory of investment that aims to maximize expected return while minimizing risk by carefully choosing the proportion of various assets in a portfolio. At its core, MPT provides a quantitative approach to the concept of diversification that aims to help investors achieve their financial goals by constructing portfolios that balance risk and reward.

Advantages and Disadvantages of MPT

Advantages of Modern Portfolio Theory:

By diversifying investments across multiple asset classes, MPT aims to optimize the risk-return tradeoff of a portfolio, potentially leading to better risk-adjusted returns. MPT encourages investors to assess their risk tolerance, goals, and investment horizon, which can lead to a more structured investment plan. The theory provides a framework for understanding portfolio construction and risk management.

Disadvantages of Modern Portfolio Theory:

MPT relies on statistical data, which can be unreliable and calculated with assumptions that do not match reality. The theory assumes that the returns of assets are normally distributed, which can lead to errors when applied to non-normal asset classes. MPT places a greater emphasis on maximizing returns rather than minimizing losses or considering downside risk, which may not be suitable for all investors.

MPT formula

- Expected return:

$$\mathrm{E}\left(R_p\right) = \sum_i w_i \mathrm{E}\left(R_i\right)$$

where R_p is the return on the portfolio, R_i is the return on asset i and w_i is the weighting of component asset i (that is, the proportion of asset "i" in the portfolio, so that $\sum_i w_i = 1$).

- Portfolio return variance:

$$\sigma_p^2 = \sum_i w_i^2 \sigma_i^2 + \sum_i \sum_{j \neq i} w_i w_j \sigma_i \sigma_j \rho_{ij}$$

where σ_i is the (sample) standard deviation of the periodic returns on an asset i, and ρ_{ij} is the correlation coefficient between the returns on assets i and j. Alternatively the expression can be written as:

$$\sigma_p^2 = \sum_i \sum_j w_i w_j \sigma_i \sigma_j \rho_{ij}$$

where $\rho_{ij} = 1$ for $i = j$, or

$$\sigma_p^2 = \sum_i \sum_j w_i w_j \sigma_{ij}$$

where $\sigma_{ij} = \sigma_i \sigma_j \rho_{ij}$ is the (sample) covariance of the periodic returns on the two assets, or alternatively denoted as $\sigma(i, j)$, cov_{ij} or $\operatorname{cov}(i, j)$.

- Portfolio return volatility (standard deviation):

$$\sigma_p = \sqrt{\sigma_p^2}$$

3. Optimize MPT with Monte Carlo

Monte Carlo simulation is a mathematical technique used to predict the probability of a range of outcomes when dealing with potential random variables. It involves using computer programs to run random experiments and analyze the results to gain insights into the likelihood of certain outcomes. Monte

By simulating a large number of potential market scenarios and running them through the portfolio optimization process such as MPT, the two methods can be combined to identify the most robust and efficient portfolio weights that yield the maximum returns at the lowest risk level.

Implementation

Extract Data

Extract data from Financial Modelling Prep API

In [ ]:
```
%pip install python-dotenv
%pip install scipy
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: python-dotenv in /Users/anhhoang.chu/Library/Python/3.9/lib/python/site-packages (1.0.0)
WARNING: You are using pip version 21.2.4; however, version 23.0.1 is available.
You should consider upgrading via the '/Library/Developer/CommandLineTools/usr/bin/python3 -m pip install --upgrade pip' comman
d.
Note: you may need to restart the kernel to use updated packages.
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scipy in /Users/anhhoang.chu/Library/Python/3.9/lib/python/site-packages (1.10.1)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in /Users/anhhoang.chu/Library/Python/3.9/lib/python/site-packages (from s
cipy) (1.24.2)
WARNING: You are using pip version 21.2.4; however, version 23.0.1 is available.
You should consider upgrading via the '/Library/Developer/CommandLineTools/usr/bin/python3 -m pip install --upgrade pip' comman
d.
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
import os
from dotenv import load_dotenv
import ssl
from urllib.request import urlopen
import numpy as np
import datetime as dt
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
import json

load_dotenv()
API_KEY = os.getenv('API_KEY')
base_url = "https://financialmodelingprep.com/api/v3"
```

In [ ]:
```
def get_jsonparsed_data(url):
    """
    Receive the content of ``url``, parse it as JSON and return the object.

    Parameters
    ----------
    url : str

    Returns
    -------
    dict
    """
    context = ssl.create_default_context()
    response = urlopen(url, context=context)
    data = response.read().decode("utf-8")
    return json.loads(data)
```

In [ ]:
```
def get_historical_price_full(stickers, file_path):
    """
    Extract historical 1yr daily price for stock stickers, and save to json file with file_path

    Parameters:
      stickers (list): list of stock stickers
      file_path (str): json data file

    Returns:
      Json object of historical stock prices of all stocks in the list
    """
    stickers_str = ','.join(stickers)

    url = (f"{base_url}/historical-price-full/{stickers_str}?apikey={API_KEY}")

    data = get_jsonparsed_data(url)

    with open(file_path, "w") as f:
        if len(stickers) == 1:
          json.dump(data, f)
        else:
          json.dump(data["historicalStockList"], f)

    return data

def get_historical_price(stock, start_date, end_date, file_path):
    """
    Extract historical daily price for 1 stock, and save to json file with file_path

    Parameters:
      stocks (list): list of stocks
      file_path (str): json data file

    Returns:
      Json object of historical stock prices of all stocks in the list
    """
```

```
    url = (f"{base_url}/historical-price-full/{stock}?from={start_date}&to={end_date}&apikey={API_KEY}")

    data = get_jsonparsed_data(url)

    with open(file_path, "w") as f:
      json.dump(data, f)

    return data

def get_quote(stocks, file_path):
    """
    Extract current price for stock stocks, and save to json file with file_path

    Parameters:
      stocks (list): list of stock stocks
      file_path (str): json data file

    Returns:
      Json object of historical stock prices of all stocks in the list
    """
    stocks_str = ','.join(stocks)

    url = (f"{base_url}/quote/{stocks_str}?apikey={API_KEY}")

    data = get_jsonparsed_data(url)

    with open(file_path, "w") as f:
        json.dump(data, f)
    return data
```

Process Data

In [ ]:

```
def get_json_data(file_path):
  # open the JSON file
  with open(file_path, 'r') as f:
      # load the JSON object into a Python object
      json_obj = json.load(f)

  columns = list(json_obj[0]['historical'][0].keys())
  df = pd.DataFrame(columns=['symbol'] + columns)
  for stock in json_obj:
    symbol = stock['symbol']
    historical = stock['historical']
    data = pd.DataFrame(historical, columns=columns)
    data.insert(0, 'symbol', symbol)
    df = pd.concat([df, data])
  return df

def get_data(file_path):
  """
  Return a cleaned pandas dataframe of historical full stock price information from json file
  """
  df = pd.read_json(file_path)
  # use explode to split a list to multiple rows
  explode_df = df.explode('historical')
  # use apply and pd.Series to split dictionary column into multiple columns
  normalize_df = explode_df['historical'].apply(pd.Series)
  # concatenate 'symbol' column to the normalized_df by column (axis=1)
  df_final = pd.concat([explode_df['symbol'], normalize_df], axis=1)
  return df_final
```

Modern Portfolio Theory

In [ ]:

```
def cal_return(df):
  """
  Return a pandas dataframe of adjusted close price for each stock sticker
  """
  pivot_df = df.pivot(index = 'date', columns='symbol', values = 'adjClose')
  returns = pivot_df.pct_change()
  mean_returns = returns.mean()
  cov_matrix = returns.cov()
  return pivot_df, mean_returns, cov_matrix

def cal_portfolio_performance(weights, mean_returns, cov_matrix):
  """
  Given porfolio weight, calculate portfolio return and standard deviatzion based on modern portfolio theory

  Parameters:
    mean_returns
    cov_matrix
    weights (numpy array): array of weights for each stock sticker

  Returns:
    portfolio_return (float):  Sum(mean_returns * weights) * trading_days
    porfolio_std (float):  weights_transposed * cov_matrix * weights
  """
  trading_days = 252
  portfolio_returns = round(np.sum(mean_returns * weights) * trading_days, 4)
```

```
  portfolio_std = round(np.sqrt( np.dot(weights.T, np.dot(cov_matrix, weights)) ), 4)
  return portfolio_returns, portfolio_std
```

Monte Carlo Simulation

In [ ]:
```
def cal_portfolio_metrics(weights, mean_returns, cov_matrix, risk_free_rate = 0.0, index=0):

    '''
    This function generates the relative performance metrics that will be reported and will be used
    to find the optimal weights.

    Parameters
    ---
    weights (numpy array): initialized weights or optimal weights for performance reporting
    cov_matrix (pd dataframe): covariance matrix of stock ,
    risk_free_rate (float): risk free rate such as t-bill, default is 0.0

    Returns
    ---
    pandas dataframe of a portfolio performance
    '''
    portfolio_returns, portfolio_std = cal_portfolio_performance(weights, mean_returns, cov_matrix)
    sharpe = (portfolio_returns - risk_free_rate)/portfolio_std
    df = pd.DataFrame({"Expected Return": portfolio_returns,
                       "Portfolio Variance":portfolio_std**2,
                       'Portfolio Std': portfolio_std,
                       'Sharpe Ratio': sharpe}, index=[index])
    return df
```

Verify

In [ ]:
```
file_name = "historical.json"
file_path = f"{os.getcwd()}/data/{file_name}"
```

In [ ]:
```
stickers = ['AAPL', 'MSFT', 'TSLA']
end_date = '2023-04-01'
start_date = '2013-04-01'

def process_all(stocks, start_date, end_date):
    # write stock data to files
    for stock in stocks:
        file_path = f"{os.getcwd()}/data/{stock}_10yr.json"
        price = get_historical_price(stickers, start_date, end_date, file_path)
```

In [ ]:
```
df = get_data(file_path)
stocks = df['symbol'].unique()
```

In [ ]:
```
stocks
```

Out[ ]:
```
array(['AAPL', 'MSFT', 'TSLA', 'LCID', 'PFE'], dtype=object)
```

In [ ]:
```
pivot_df, mean_returns, cov_matrix = cal_return(df)
```

In [ ]:
```
print(pivot_df.index.min())
print(pivot_df.index.max())
```

```
2022-04-01
2023-03-31
```

In [ ]:
```
def simulate_portfolios( mean_returns, cov_matrix, risk_free_rate=0.0, n=100):
    """
    Given the historical mean_returns and cov_matrix of the portfolio, as well as the risk_free_rate
    Simulate n portfolios with different weights and performance

    Parameters
    ---
    mean_returns (float): historical mean_returns
    cov_matrix (pd dataframe): covariance matrix of stock ,
    risk_free_rate (float): risk free rate such as t-bill, default is 0.0
    n: number of simulations, default is 100

    Returns
    ---
    portfolios (pandas dataframe): pandas dataframe of n portfolios and their expected performances
    """
    np.random.seed(42)
    #Empty Container
    portfolios = pd.DataFrame(columns=[*stocks, "Expected Return","Portfolio Variance", "Portfolio Std", "Sharpe Ratio"])
    #Loop
    for i in range(n):
        weights = np.random.random(len(stocks))
        weights /= np.sum(weights)
        portfolios.loc[i, stocks] = weights
        metrics = cal_portfolio_metrics(weights, mean_returns, cov_matrix, risk_free_rate = risk_free_rate, index=i)
        # print(metrics)
        portfolios.loc[i, ["Expected Return","Portfolio Variance", "Portfolio Std", "Sharpe Ratio"]] = metrics.loc[i,["Expected

    return portfolios
```

In [ ]:
```
portfolios = simulate_portfolios(mean_returns, cov_matrix, risk_free_rate=0.0, n=10000)
portfolios[portfolios["Sharpe Ratio"]==portfolios["Sharpe Ratio"].max()]
```

Out[ ]:

	AAPL	MSFT	TSLA	LCID	PFE	Expected Return	Portfolio Variance	Portfolio Std	Sharpe Ratio
4222	0.552807	0.009441	0.389236	0.044609	0.003906	-0.0105	0.000412	0.0203	-0.517241

In [ ]: