FE 535 Lab 1

Authors: Tejas Appana, Akshay Parate

I pledge my honor that I have abided by the Stevens Honor System

Task 1.

Tejas Appana is in the last semester of his Quantitative Finance Undergraduate degree. He interned this past summer in a Market Liquidity Risk Management role. Tejas has always been passionate about applying mathematial techniques to real world scenarios, and given his creative problem solving skills, took it upon himself to self-teach the nuances of computer science and advanced programming techniques. Now, he enjoys challenging his capabilities to the highest extent and continues to broaden his scope in financial modeling, forecasting, data science, machine learning, time series analysis and any form of alpha-generation.

Akshay Parate, currently in the second semester of his Data Science graduate degree, brings a wealth of practical experience with 2 years in IT as a senior consultant at a reputable firm. Passionate about coding, finance, and statistics, he has successfully deployed two algorithmic strategies, showcasing his expertise in both data science and financial analytics. With a strong academic foundation and a proven track record in the industry, Akshay is a dedicated professional who seamlessly blends theoretical knowledge with hands-on experience in the field of data science and finance.

Task 2.

Our team's preferred programming languages in the following order are:

1. Python
2. R

We chose Python for the purpose of this class as it may be easier to collaborate on the labs through Google Colab.

Task 3.

Here we are trying to prove: $E[R_A] = 52 * E[R_w]$. Since $R_A = \sum_{n=1}^{52} R_w$, then $E[R_A] = E[\sum_{n=1}^{52} R_w]$. Under the assumption that R_w is i.i.d., we can pull the expectation inside the summation and get the expression $E[R_A] = \sum_{n=1}^{52} E[R_w]$. Then, since R_w is i.i.d., the expected value of each R_w is equal to every R_w making $\sum_{n=1}^{52} E[R_w] = 52 * E[R_w]$ and by the transitive property, $E[R_A] = 52 * E[R_w]$.

Here we are trying to prove: $\sqrt{V[R_A]} = \sqrt{52V[R_w]}$. Since $R_A = \sum_{n=1}^{52} R_w$, then $\sqrt{V[R_A]} = \sqrt{V[\sum_{n=1}^{52} R_w]}$. Under the assumption that R_w is i.i.d., we can pull the variance inside the summation and get the expression $V[R_A] = \sqrt{\sum_{n=1}^{52} V[R_w]}$. Then, since R_w is i.i.d., the variance value of each R_w is equal to every R_w making $\sqrt{\sum_{n=1}^{52} V[R_w]} = \sqrt{52V[R_w]}$ and by the transitive property, $\sqrt{V[R_A]} = \sqrt{52V[R_w]}$.

2. Also, **what is the assumption behind this to hold true**? (1 point)

The above scales assume that the stock returns are normally distributed, which in conjunction assumes that the stock prices are log-normal.

Task 4.

Go to Yahoo Finance and download weekly historical data for SPY and GLD ETFs, dating from 2010 to 2022. Compute the weekly log returns using the adjusted close price column. This should result in two time series (two columns). Based on this, report the annual mean return and volatility for each ETF.

1. The first approach to answer this is to scale weekly average returns and volatility (standard deviation) to reflect annual values - same as above (1 point)
2. The other approach is to compute annual rather than weekly returns. Given the annual return time series, compute the average and volatility (1 point)
3. How do both results compare? (1 point)

In [ ]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore")
```

Approach 1.

In [ ]:
```
df = yf.download(["SPY", "GLD"], start="2010-01-01", end="2022-12-31", interval="1w
df = df["Adj Close"]
```

In [ ]:
```
df = np.log1p(df.pct_change().dropna())
```

In [ ]:
```
np.mean(df, axis=0) * 52
```

Out[ ]:
```
Ticker
GLD    0.032655
SPY    0.111708
dtype: float64
```

The annualized mean of GLD is 3.265%, and for SPY it is 11.17%

In [ ]:
```
np.std(df, axis=0) * np.sqrt(52)
```

Out[ ]:
```
Ticker
GLD    0.158665
SPY    0.163628
dtype: float64
```

The annualized volatility of GLD is 15.86%, and for SPY it is 16.36%

Approach 2.

In [ ]:
```
df = yf.download(["SPY", "GLD"], start="2010-01-01", end="2022-12-31", interval="3m
df = df["Adj Close"].resample("Y").first()
```

In [ ]:
```
df = np.log1p(df.pct_change().dropna())
```

In [ ]:
```
np.mean(df, axis=0)
```

Out[ ]:
```
Ticker
GLD    0.042139
SPY    0.131870
dtype: float64
```

In [ ]:
```
np.std(df, axis=0)
```

Out[ ]:
```
Ticker
GLD    0.124377
SPY    0.117470
dtype: float64
```

The annualized volatility of GLD is 12.44%, and for SPY it is 11.74%

Compare Results

We found that by using weekly data and annualizing the moments based on that data, there were some discrepancies than by calculating the moments straight from the annual time series.

The annual means for example, were a little higher when calculated from the annual time series. The volatilities however, were lower for both ETFs when calculated from the annual time series. Overall though, the values were fairly similar.

Task 5

Follow the first approach, compute the correlation coefficient between the two ETFs, and report the covariance matrix. (1 point)

In [ ]:
```
df = yf.download(["SPY", "GLD"], start="2010-01-01", end="2022-12-31", interval="1w
df = df["Adj Close"]
df = np.log1p(df.pct_change().dropna())
```

In [ ]:
```
df.corr()
```

Out[ ]:

Ticker	GLD	SPY
Ticker		
GLD	1.000000	0.190306
SPY	0.190306	1.000000

In [ ]:
```
df.cov()
```

Out[ ]:

Ticker	GLD	SPY
Ticker		
GLD	0.000485	0.000095
SPY	0.000095	0.000516

Task 6

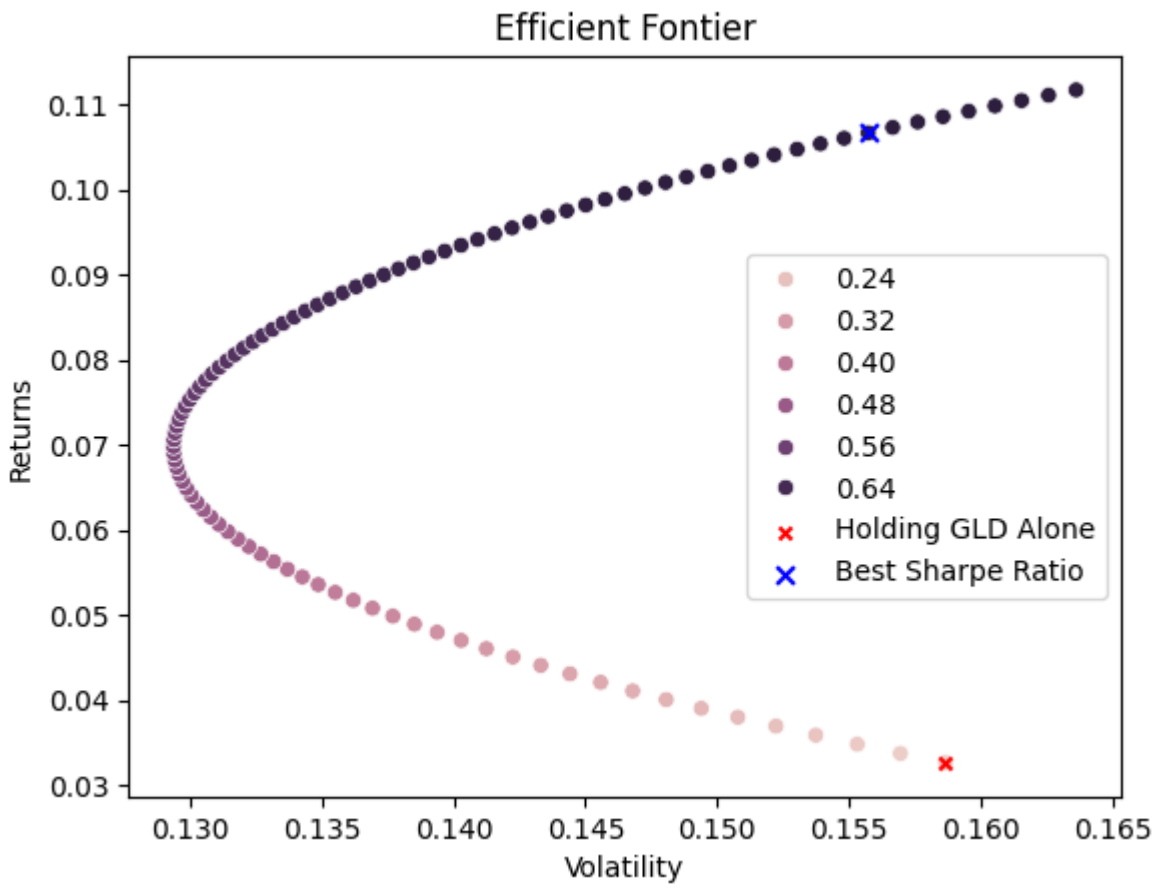
In [ ]:
```
#Downloading the data
df = yf.download(["SPY", "GLD"], start="2010-01-01", end="2022-12-31", interval="1w
df = df["Adj Close"]
portfolioDict = {"SPYW":[],"GLDW":[],"portfolioRet":[],"portfolioVol":[],"sharpe_ra
```

In [ ]:
```
#Simulation of 100 portfolios
numPortfolio = 100
interval = 1/numPortfolio
#Sequentially increasing the weights by predefined interval
for i in range(0,numPortfolio+1):
  tempDf = pd.DataFrame()
  SPYW = i*interval
  GLDW = 1-SPYW
  #Assigining weights to the asset values
  tempDf["GLD"] = df["GLD"] * GLDW
  tempDf["SPY"] = df["SPY"] * SPYW
  #Calculating the Total Profit by adding the asset values
  tempDf["total_profit"] = tempDf["GLD"] + tempDf["SPY"]
  portfolioDict["SPYW"].append(SPYW)
  portfolioDict["GLDW"].append(GLDW)
  #Calculating the percent change using log returns
  tempDf["pct_change"] = np.log1p(tempDf["total_profit"].pct_change().dropna())
  #Calculating annualized average mean and standard deviation
  portfolioDict["portfolioRet"].append(np.mean(tempDf["pct_change"]) * 52)
  portfolioDict["portfolioVol"].append(np.std(tempDf["pct_change"]) * np.sqrt(52))
  #Assuming risk free rate as 0
  portfolioDict["sharpe_ratio"].append((np.mean(tempDf["pct_change"]) * 52)/(np.std
```

In [ ]:
```
portfolioDf = pd.DataFrame.from_dict(portfolioDict)
# print(portfolioDf)
```

In [ ]:
```
#Plot the efficient frontier graph
# plt.scatter(list(portfolioDf["portfolioVol"]),list(portfolioDf["portfolioRet"]))
# plt.scatter(portfolioDf["portfolioVol"][0], portfolioDf["portfolioRet"][0], color
import seaborn as sns
sns.scatterplot(x='portfolioVol', y='portfolioRet', data=portfolioDf, hue='sharpe_r
plt.scatter(portfolioDf["portfolioVol"][0], portfolioDf["portfolioRet"][0], color='
plt.scatter(portfolioDf.iloc[92]["portfolioVol"], portfolioDf.iloc[92]["portfolioRe
plt.title('Efficient Fontier')
plt.xlabel('Volatility')
```

```
plt.ylabel('Returns')
plt.legend();
```



In [ ]:
```
portfolioDf.iloc[92]
```

Out[ ]:
```
SPYW            0.920000
GLDW            0.080000
portfolioRet    0.106666
portfolioVol    0.155735
sharpe_ratio    0.684922
Name: 92, dtype: float64
```

What does the frontier tell us about holding the GLD alone versus a combination of the two?

Analyzing the aforementioned investment frontier reveals that holding GLD alone (marked by the red X) would result in a portfolio characterized by high risk and volatility, coupled with relatively low returns. Specifically, it exhibits a 15% volatility and meager 1.5% returns, resulting in a Sharpe ratio of 0.2, indicative of poor performance. However, by strategically combining GLD and SPY, with appropriate adjustments in weights, one can attain a balanced portfolio marked by stable returns and consistent volatility. This optimized combination yields a Sharpe ratio as high as 0.68, which is considered average within a weekly timeframe, showcasing a more favorable risk-return profile.

Which point from the frontier would you choose?

Based on my analysis, I would opt for the weightings that resulted in the highest Sharpe ratio. Specifically, when allocating 92% to SPY and 8% to GLD, the portfolio achieved a Sharpe ratio of 0.68.

Task7

Note that in the case of two assets any value of $\omega \in (0, 1)$ results in an efficient portfolio. This is mainly due to the fact that the budget constraint is binding in this case. Nonetheless, given the mean-variance optimization solution, we know that the optimal portfolio can be written as an optional combination between a GMV portfolio w_0 and a self-financing portfolio w_1 (see slides from Session 01). Compute these two portfolios and consider arbitrary values for the risk aversion level to create 100 mean-variance efficient portfolios. For each portfolio, compute their mean and variance. As a summary, plot the former as a function of the latter. How does the plot compare with the previous task? (2 points)

In [ ]:
```
df1 = yf.download(["GLD", "SPY"], start="2010-01-01", end="2022-12-31", interval="1
df = np.log1p(df1["Adj Close"].pct_change().dropna())
```

Calculating GMV portfolio and Sharpe Ratio portfolio

In [ ]:
```
siginv = np.linalg.inv(df.cov())
ones = np.ones((2,1), dtype=int)
w_gmv = siginv @ ones / (ones.T @ siginv @ ones)
w_gmv
```

Out[ ]:
```
array([[0.51901284],
       [0.48098716]])
```

In [ ]:
```
mu = df.mean().to_numpy().reshape(2, 1)
w_sr = siginv @ mu / (ones.T @ siginv @ mu)
w_sr
```

Out[ ]:
```
array([[0.10843069],
       [0.89156931]])
```

$$w_{MV}(\alpha) = \alpha \cdot w_{GMV} + (1 - \alpha) \cdot w_{SR}$$

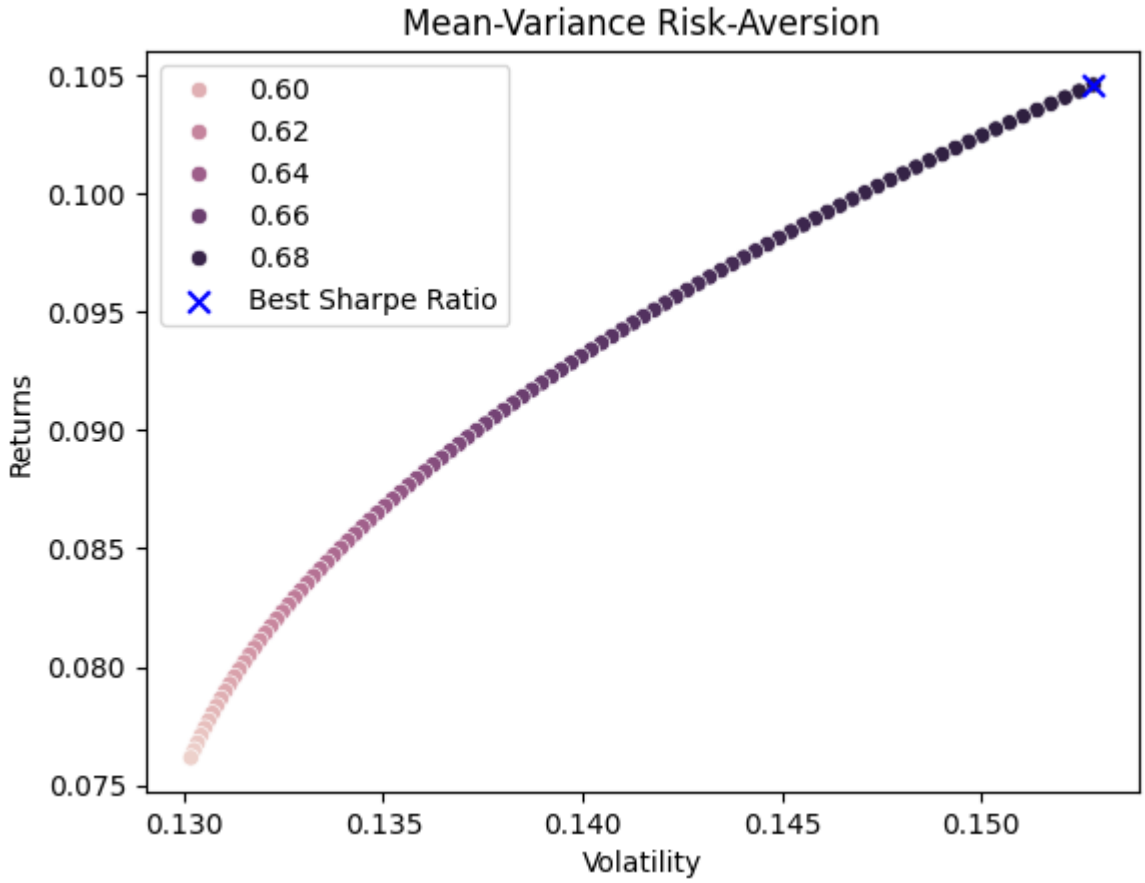
In [ ]:
```
alpha = np.arange(0.01, 1.01, 0.01)
MV = pd.DataFrame(index=alpha, columns=["GLD Weight", "SPY Weight", "Mean", "Volati
MV["GLD Weight"], MV["SPY Weight"] = alpha * w_gmv + (1-alpha) * w_sr
# MV.reset_index(inplace = True)
```

In [ ]:
```
df = df1["Adj Close"]
SR = []
for i in range(0,len(MV["Mean"])):
  tempDf = pd.DataFrame()
  SPYW = MV.iloc[i]["SPY Weight"]
  GLDW = MV.iloc[i]["GLD Weight"]
  #Assigining weights to the asset values
  tempDf["GLD"] = df["GLD"] * GLDW
  tempDf["SPY"] = df["SPY"] * SPYW
  #Calculating the Total Profit by adding the asset values
  tempDf["total_profit"] = tempDf["GLD"] + tempDf["SPY"]
  #Calculating the percent change using log returns
  tempDf["pct_change"] = np.log1p(tempDf["total_profit"].pct_change().dropna())
  #Calculating annualized average mean and standard deviation
  MV.iloc[i,2] = np.mean(tempDf["pct_change"]) * 52
  MV.iloc[i,3] = np.std(tempDf["pct_change"]) * np.sqrt(52)
  #Assuming risk free rate as 0
  SR.append((np.mean(tempDf["pct_change"]) * 52)/(np.std(tempDf["pct_change"]) * np
MV["sharpe_ratio"] = SR
```

In [ ]:
```
print(MV.head())
print("Max Sharpe Ratio",MV.max()["sharpe_ratio"])
print("Index of max value of sharpe ratio",MV[["sharpe_ratio"]].idxmax())
```

```
      GLD Weight  SPY Weight      Mean Volatility  sharpe_ratio
0.01    0.112537    0.887463  0.104579   0.152813      0.684361
0.02    0.116642    0.883358  0.104314   0.152456      0.684227
0.03    0.120748    0.879252  0.104049   0.152101      0.684078
0.04    0.124854    0.875146  0.103784   0.151749      0.683915
0.05    0.128960    0.871040  0.103517   0.151399      0.683737
Max Sharpe Ratio 0.6843605628907762
Index of max value of sharpe ratio sharpe_ratio    0.01
dtype: float64
```

In [ ]:
```
import seaborn as sns
sns.scatterplot(x='Volatility', y='Mean', data=MV, hue='sharpe_ratio')
# plt.scatter(portfolioDf["portfolioVol"][0], portfolioDf["portfolioRet"][0], color
plt.scatter(MV.iloc[0]["Volatility"], MV.iloc[0]["Mean"], color='blue', marker='x',
plt.title('Mean-Variance Risk-Aversion')
plt.xlabel('Volatility')
plt.ylabel('Returns')
plt.legend();
```



We see that the Mean-Variance portfolio when measured at different risk-aversion levels creates a plot of 100 efficient portfolios with varying levels of risk-adjusted return, mimicing the Markowitz Efficient Frontier. Here, we just see the top half of the curve since, instead of exploring all possible allocations between the assets GLD and SPY, we are just exploring all possible combinations between the min-variance and sharpe-ratio weightings.