# Portfolio Optimization

October 9, 2023

## 1 Description: This program attempts to optimize a users portfolio using the Efficient Frontier & Python

[ ]:
```
#install pandas_datareader
# pip install pandas_datareader (you only do it once)
```

[ ]:
```
# Import the python libraries
from pandas_datareader import data as web
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
```

### 1.0.1 Create The Fictional Portfolio

**Get the stock symbols / tickers for the fictional portfolio. I am going to use the five most popular and best performing American technology companies known as FAANG, which is an acronym for Facebook, Amazon , Apple, Netflix , & Alphabet (formerly known as Google)**

[ ]:
```
assets =  ["FB", "AMZN", "AAPL", "NFLX", "GOOG"]
```

**Next I will assign equivalent weights to each stock within the portfolio, meaning 20% of this portfolio will have shares in Facebook (FB), 20% in Amazon (AMZN), 20% in Apple (AAPL) , 20% in Netflix (NFLX), and 20% in Google (GOOG).**

[ ]:
```
weights = np.array([0.2, 0.2, 0.2, 0.2, 0.2])
```

**Now I will get the stocks starting date which will be January 1st 2013, and the ending date which will be the current date (today)**

[ ]:
```
#Get the stock starting date
stockStartDate = '2013-01-01'

# Get the stocks ending date aka todays date and format it in the form␣
↪YYYY-MM-DD
```

```
today = datetime.today().strftime('%Y-%m-%d')                    #the today()␣
↪method only prints the date, while the now() method prints the day and the␣
↪time
```

**Time to create the data frame that will hold the stocks Adjusted Close price**

[ ]:
```
#Create a dataframe to store the adjusted close price of the stocks
df = pd.DataFrame()

#Store the adjusted close price of stock into the data frame
for i in assets:
   df[i] = web.DataReader(i,data_source='yahoo',start=stockStartDate ,␣
↪end=today)['Adj Close']    #we can use Stock instead of i to give the␣
↪exercise more meaning


# The formula is
# df["stock"] = web.DataReader("stock", data_source='yahoo', start,␣
↪end)['column selected']
```

**Show the data frame and the adjusted close price of each stock.**

[ ]:
```
df.head()
```

[ ]:

| Date | FB | AMZN | AAPL | NFLX | GOOG |
|---|---|---|---|---|---|
| 2013-01-02 | 28.000000 | 257.309998 | 68.378807 | 13.144286 | 360.274597 |
| 2013-01-03 | 27.770000 | 258.480011 | 67.515701 | 13.798572 | 360.483826 |
| 2013-01-04 | 28.760000 | 259.149994 | 65.635078 | 13.711429 | 367.607117 |
| 2013-01-07 | 29.420000 | 268.459991 | 65.249001 | 14.171429 | 366.003143 |
| 2013-01-08 | 29.059999 | 266.380005 | 65.424622 | 13.880000 | 365.280823 |

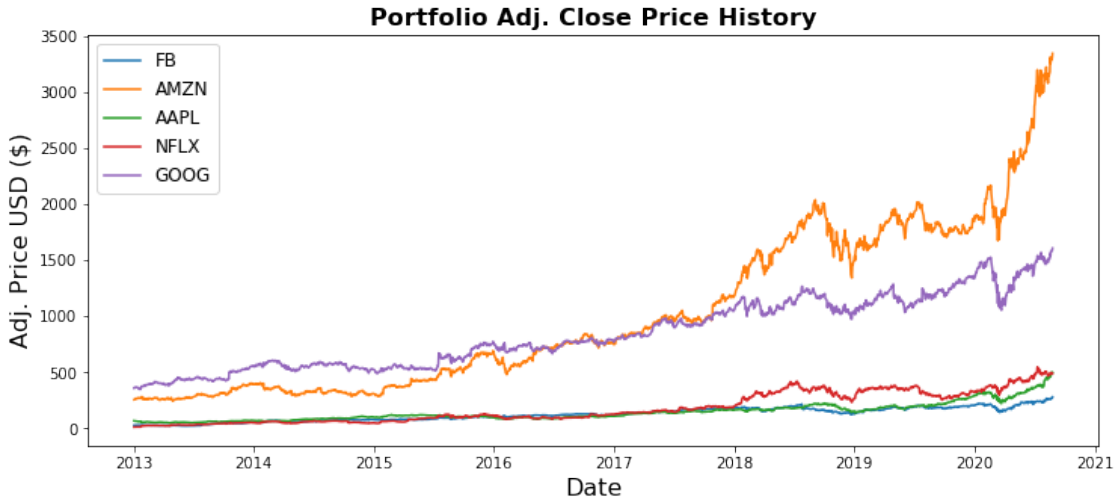
**Visually show the stock prices.**

[ ]:
```
my_stocks = df

plt.figure(figsize=(12,5))

for c in my_stocks:
    plt.plot(my_stocks[c], label=c)

#plt.plot(my_stocks['FB'], label='FB')
plt.title('Portfolio Adj. Close Price History', fontsize=16, fontweight =␣
↪'bold')
plt.xlabel('Date', fontsize=16)
plt.ylabel('Adj. Price USD ($)', fontsize=16)
plt.legend(my_stocks.columns.values, loc='upper left', fontsize=12)
plt.show()
```




### 1.0.2 Financial Calculations

**Now I want to show the daily simple returns which is a calculation of the (new_price + -old_price)/ old_price or (new_price / old_price)-1.**

[ ]:
```
returns = df.pct_change()   #Basically I'm creating a new data frame that is
 ↪called 'return' and to that data frame I'm applying the pct_change function
returns.head()
```

[ ]:

| Date | FB | AMZN | AAPL | NFLX | GOOG |
|---|---|---|---|---|---|
| 2013-01-02 | NaN | NaN | NaN | NaN | NaN |
| 2013-01-03 | -0.008214 | 0.004547 | -0.012622 | 0.049777 | 0.000581 |
| 2013-01-04 | 0.035650 | 0.002592 | -0.027855 | -0.006315 | 0.019760 |
| 2013-01-07 | 0.022949 | 0.035925 | -0.005882 | 0.033549 | -0.004363 |
| 2013-01-08 | -0.012237 | -0.007748 | 0.002692 | -0.020565 | -0.001974 |

**Now we want to create and show the annualized co-variance matrix. The co-variance matrix is used to determine how much two random variables vary or move together.** The diagonal of the matrix are the variances and the other entries are the co-variances.

If you take the square root of variance you get the volatility also known as the standard deviation.

To show the annualized co-variance matrix we must multiply the co-variance matrix by the number of trading days for the current year. In this case the number of trading days will be 252 for this year.

[ ]:
```
cov_matrix_annual = returns.cov()*252
cov_matrix_annual
```

[ ]:

| | FB | AMZN | AAPL | NFLX | GOOG |
|---|---|---|---|---|---|
| FB | 0.117058 | 0.052641 | 0.042310 | 0.054584 | 0.048707 |
| AMZN | 0.052641 | 0.092613 | 0.036611 | 0.061649 | 0.046758 |
| AAPL | 0.042310 | 0.036611 | 0.078267 | 0.032411 | 0.037223 |
| NFLX | 0.054584 | 0.061649 | 0.032411 | 0.211262 | 0.048909 |
| GOOG | 0.048707 | 0.046758 | 0.037223 | 0.048909 | 0.064855 |

**Now calculate and show the portfolio variance using the formula :** Expected portfolio variance= WT * (Covariance Matrix) * W

[ ]:
```
port_variance = np.dot(weights.T ,np.dot(weights, cov_matrix_annual))
#### Básicamente lo que se esta haciendo es multiplicar las ponderaciones de␣
 ↪cada covarianza con el peso de esa accíon, para luego sumarlas y a eso␣
 ↪multiplicarla por el peso de la acción en la cartera de acciones.
#### La columna de pesos esta de manera vertical, y para multiplicarla por el␣
 ↪resultado de la suma de productos anterior, hay que transponerla

port_variance
```

[ ]:
```
0.05950645978785821
```

**Now calculate and show the portfolio volatility using the formula :** Expected portfolio volatility= SQRT (WT * (Covariance Matrix) * W)

[ ]:
```
port_volatility = np.sqrt(port_variance)
port_volatility
```

[ ]:
```
0.24393945926778268
```

**Now we calculate the portfolio annual simple return**

[ ]:
```
port_return = np.dot(returns.mean(), weights)*252   #### you could have also␣
 ↪done np.sum(returns.mean()*weights) * 252
port_return
```

[ ]:
```
0.3691356676227962
```

**Show the expected annual return, volatility or risk, and variance**

[ ]:
```
percent_var = str(round(port_variance*100, 2)) + '%'
percent_vol = str(round(port_volatility*100, 2)) + '%'
percent_ret = str(round(port_return*100, 2)) + '%'

print("Expected annual return : "+ percent_ret)
print('Annual volatility/standard deviation/risk : '+ percent_vol)
print('Annual variance : '+ percent_var)
```

```
Expected annual return : 36.91%
Annual volatility/standard deviation/risk : 24.39%
Annual variance : 5.95%
```

So, now I can see the expected annual return on the investments which is 32% and the amount of risk for this portfolio which is 23%, but can I do better ? I think I can.

### 1.0.3 Optimize The Portfolio

It's now time to optimize this portfolio, meaning I want to optimize for the maximum return with the least amount of risk. Luckily their is a very nice package that can help with this created by Robert Ansrew Martin. #### I will install the package that he created called pyportfolioopt.

[ ]:
```
pip install PyPortfolioOpt
```

```
Requirement already satisfied: PyPortfolioOpt in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (1.2.4)
Requirement already satisfied: pandas>=0.19 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
PyPortfolioOpt) (1.0.1)
Requirement already satisfied: numpy<2.0,>=1.12 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
PyPortfolioOpt) (1.18.1)
Requirement already satisfied: scipy<2.0,>=1.3 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
PyPortfolioOpt) (1.4.1)
Requirement already satisfied: cvxpy<1.1,>=1.0 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
PyPortfolioOpt) (1.0.31)
Requirement already satisfied: pytz>=2017.2 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
pandas>=0.19->PyPortfolioOpt) (2019.3)
Requirement already satisfied: python-dateutil>=2.6.1 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
pandas>=0.19->PyPortfolioOpt) (2.8.1)
Requirement already satisfied: multiprocess in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
cvxpy<1.1,>=1.0->PyPortfolioOpt) (0.70.10)
Requirement already satisfied: ecos>=2 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
cvxpy<1.1,>=1.0->PyPortfolioOpt) (2.0.7.post1)
Requirement already satisfied: osqp>=0.4.1 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
cvxpy<1.1,>=1.0->PyPortfolioOpt) (0.6.1)
Requirement already satisfied: scs>=1.1.3 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
cvxpy<1.1,>=1.0->PyPortfolioOpt) (2.1.2)
Requirement already satisfied: six>=1.5 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from python-
```

```
dateutil>=2.6.1->pandas>=0.19->PyPortfolioOpt) (1.14.0)
Requirement already satisfied: dill>=0.3.2 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
multiprocess->cvxpy<1.1,>=1.0->PyPortfolioOpt) (0.3.2)
Requirement already satisfied: future in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
osqp>=0.4.1->cvxpy<1.1,>=1.0->PyPortfolioOpt) (0.18.2)
Note: you may need to restart the kernel to use updated packages.
```

Next, I will import the necessary libraries.

[ ]:
```
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
```

Calculate the expected returns and the annualised sample covariance matrix of asset returns

[ ]:
```
mu = expected_returns.mean_historical_return(df)
S = risk_models.sample_cov(df)
```

**Optimize for maximal Sharpe Ratio**

[ ]:
```
ef = EfficientFrontier(mu, S)

weights = ef.max_sharpe()

cleaned_weights = ef.clean_weights()
```

[ ]:
```
print(cleaned_weights)
```

```
OrderedDict([('FB', 0.1283), ('AMZN', 0.29292), ('AAPL', 0.28847), ('NFLX',
0.2903), ('GOOG', 0.0)])
```

[ ]:
```
ef.portfolio_performance(verbose=True)
```

```
Expected annual return: 41.2%
Annual volatility: 26.0%
Sharpe Ratio: 1.51
```

[ ]:
```
(0.4117680663846699, 0.2597042155274632, 1.5085163927315657)
```

**Now we see that we can optimize this portfolio by having about 12.83% of the portfolio in Facebook, 29.29% in Amazon , 28.84% in Apple, 29.03% in Netflix and 0% in Google.** Also, we can see that the expected annual return has increased to 41.2% (before it was 36.9%) with this optimziation and the annual volatility / risk is now 26.0% (before it was 24.39%).

This portfolio has a sharpe ratio of 1.51 which is good.

**Discrete Allocations** I want to get the discrete allocation of each share of the stock, meaning I want to know exactly how many of each stock I should buy given some amount that I am willing to put into this portfolio.

So, for example I am willing to put in $15,000 USD into this portfolio, and need to know how much of each stock I can purchase in the portfolio to give me the optimal results.

First I will install pulp.

[ ]:
```
pip install pulp
```

```
Collecting pulp
  Downloading PuLP-2.3-py3-none-any.whl (40.6 MB)
     |                  | 40.6 MB 13.7 MB/s eta 0:00:01
Collecting amply>=0.1.2
  Downloading amply-0.1.2.tar.gz (26 kB)
  Installing build dependencies … done
  Getting requirements to build wheel … done
    Preparing wheel metadata … done
Requirement already satisfied: pyparsing in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
amply>=0.1.2->pulp) (2.4.6)
Requirement already satisfied: docutils>=0.3 in
/Users/jjtfernandez/opt/anaconda3/lib/python3.7/site-packages (from
amply>=0.1.2->pulp) (0.16)
Building wheels for collected packages: amply
  Building wheel for amply (PEP 517) … done
  Created wheel for amply: filename=amply-0.1.2-py3-none-any.whl
size=16573
sha256=cd010878bd04f9923fc01be60a95346a17f281c2a5993152967030331b2aafd6
  Stored in directory: /Users/jjtfernandez/Library/Caches/pip/wheels/79/c3/09/00
48ee46d04fd5b56f7e3bead9dcef92b2443e529e1f932e6b
Successfully built amply
Installing collected packages: amply, pulp
Successfully installed amply-0.1.2 pulp-2.3
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
from pypfopt.discrete_allocation import DiscreteAllocation, get_latest_prices
```

[ ]:
```
latest_prices = get_latest_prices(df)
weights = cleaned_weights
da = DiscreteAllocation(weights, latest_prices, total_portfolio_value=15000)
allocation, leftover = da.lp_portfolio()

print("Discrete allocation:", allocation)
print("Funds remaining: ${:.2f}".format(leftover))
```

```
Discrete allocation: {'FB': 8.0, 'AMZN': 1.0, 'AAPL': 9.0, 'NFLX': 9.0}
Funds remaining: $498.03
```

**Looks like I can buy 8 shares of Facebook, 1 share of Amazon, 9 shares of Apple, and 9 shares of NetFlix for this optimized portfolio and still have about 498.03 USD leftover from my initial investment of 15,000 USD.**