Portfolio Optimization Project

By: Aryan Gandhi

Project Description

In this project the optimal portfolio will be found based on the portfolio's Sharpe ratio. The stocks that make up the portfolio will be chosen from a basket of 20 stocks that includes the top four US companies in each sector based on market cap. These sectors include technology, gold, healthcare, energy, and utility. The portfolio creation is split up into two parts:

1. Sector Grouping Method
    - This process includes grouping the stocks based on the industry sector that they are in. For example, all tech stocks will be in one group, all the healthcare stocks will be in another group and so on.
    - Based on the groups, one stock from each industry sector/group will be selected to include in the portfolio. This is to encourage portfolio diversification in a manual way so that the portfolio is comprised of many of stocks in the same sector. Thus, each valid portfolio will include a total of 5 companies (one per each sector).
2. K-means Clustering Method
    - This process relies on an unsupervised machine learning algorithm called "K-means clustering". This algorithm groups the data together based on data points that have similar features by minimizing the variances within-cluster.
    - The number of groups will not be determined manually but rather by the algorithm itself.

In both processes the dataset used includes stock price data which ranges from dates of 2012-01-01 to 2022-01-01 (i.e., about 10 years). The dataset will be split up into three datasets:

- Training Dataset
    - This dataset includes stock price data from 2012-01-01 to 2016-12-31 (i.e., about 5 years)
    - The purpose of this dataset is to use the data in this timeframe to build a portfolio. We then test how well this portfolio performed in the other datasets
- Pre-covid Dataset
    - This dataset includes stock price data from 2017-01-01 to 2019-12-31 (i.e., about 3 years)
    - The purpose of this dataset is to test how well the portfolio created using the training dataset performed in a timeframe that is not influenced by the effects of COVID-19 on the stock market. Furthermore, the portfolio's performance in this dataset may indicate whether the portfolio created was over-fitted to data in the training dataset.
- Covid Dataset
    - This dataset includes stock price data from 2020-01-01 to 2022-01-01 (i.e., about 2 years)
    - The purpose of this dataset is to test how well the portfolio created using the training dataset performed in a timeframe during COVID-19

List of Companies

Energy:

- Chevron Corporation (CVX)
- Shell PLC (SHEL)
- Exxon Mobile Corp (XOM)
- ConocoPhillips (COP)

Gold:

- Newmont Corporation (NEM)
- Royal Gold (RGLD)
- Hecla Mining (HL)
- Coeur Mining (CDE)

Healthcare:

- UnitedHealth (UNH)
- CVS Health (CVS)
- Elevance Health (ELV)
- HCA Healthcare (HCA)

Tech:

- Apple (AAPL)
- Microsoft (MSFT)
- Alphabet Inc. (GOOG)
- Amazon (AMZN)

Utilities:

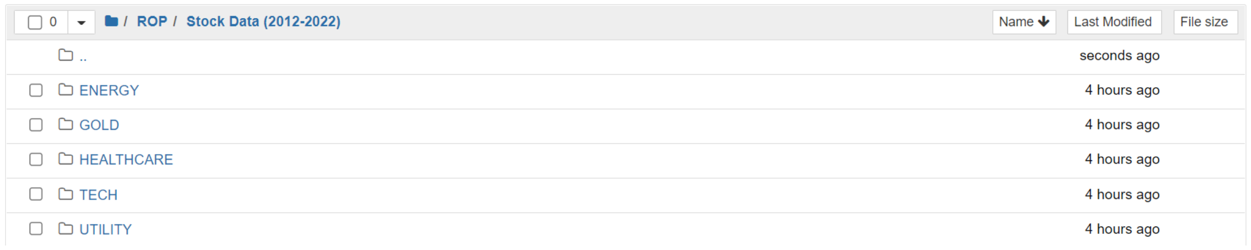
- Nextera Energy (NEE)
- Duke Energy (DUK)
- Southern Company (SO)
- Dominion Energy (D)

Importing Data

In order to obtain the stock price data, we can use the library “yfinance” which allows us to import historical stock data from yahoo finance.

Follow these steps to import the data:

1. Open JupyterHub and create a folder called “Stock Data (2012-2022)”
2. Within the newly created folder create subdirectories/folders for each industry sector.



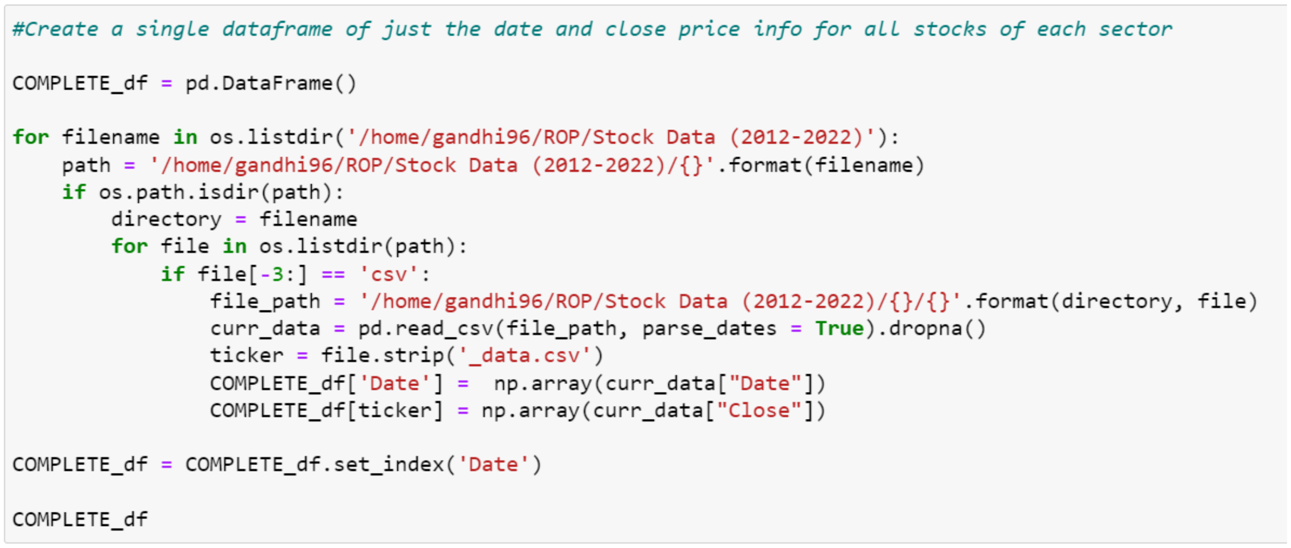
0 / ROP / Stock Data (2012-2022)	Last Modified	File size
..	seconds ago	
ENERGY	4 hours ago	
GOLD	4 hours ago	
HEALTHCARE	4 hours ago	
TECH	4 hours ago	
UTILITY	4 hours ago	

3. Go back to the main directory (in my case the ROP folder) and create a new python notebook called “Data Import”
4. In order to efficiently get historical data for each stock and store it as csv in the correct location, use the following code:

```python
#Data import

import pandas as pd
import numpy as np
import math
import os
import yfinance as yf

tickers_dict = {'ENERGY': ['COP', 'CVX', 'SHEL', 'XOM'], 'GOLD': ['CDE', 'HL', 'NEM', 'RGLD'],
           'HEALTHCARE': ['CVS', 'ELV', 'HCA', 'UNH'], 'TECH': ['AAPL', 'AMZN', 'GOOG', 'MSFT'],
          'UTILITY': ['D', 'DUK', 'NEE', 'SO']}

for sector in tickers_dict:
    for symbol in tickers_dict[sector]:

        ticker = yf.Ticker(symbol)

        data = ticker.history(start='2012-01-01', end='2022-01-01')

        path = 'Stock Data (2012-2022)/{}/{}_data.csv'.format(sector, ticker.ticker)

        data.to_csv(path)
```

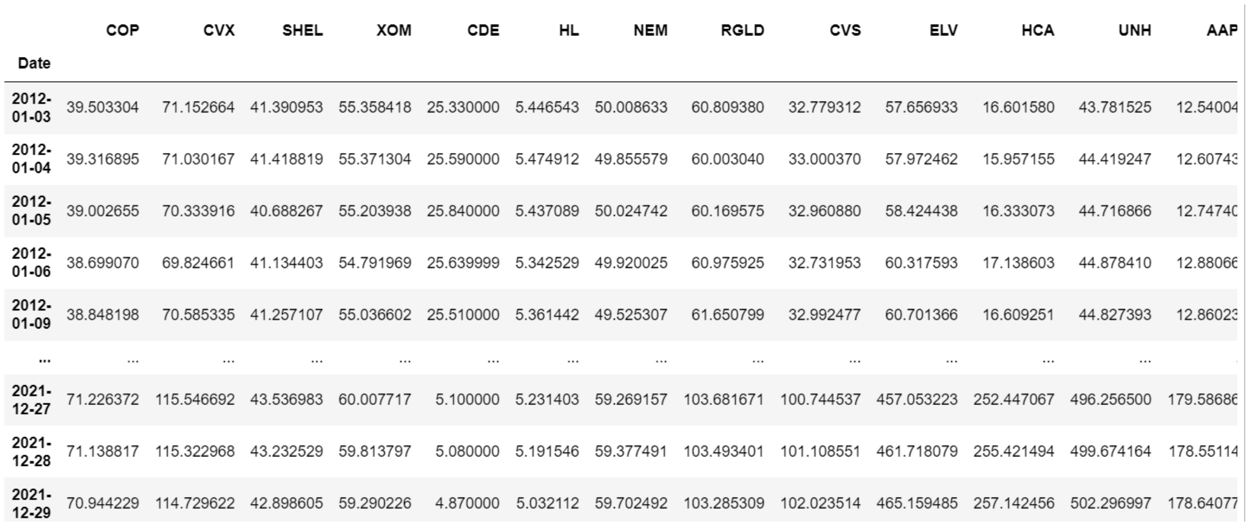
This will create a dictionary where each key is the industry sector, and the value is a list of stock tickers in that sector. Then the yfinance library will be used to the import the data and store it as a csv file in the correct folder while looping through the dictionary.

5. In order to organize the data such that all closing stock prices are in one dataframe use the following code:



```
#Create a single dataframe of just the date and close price info for all stocks of each sector

COMPLETE_df = pd.DataFrame()

for filename in os.listdir('/home/gandhi96/ROP/Stock Data (2012-2022)'):
    path = '/home/gandhi96/ROP/Stock Data (2012-2022)/{}'.format(filename)
    if os.path.isdir(path):
        directory = filename
        for file in os.listdir(path):
            if file[-3:] == 'csv':
                file_path = '/home/gandhi96/ROP/Stock Data (2012-2022)/{}/{}'.format(directory, file)
                curr_data = pd.read_csv(file_path, parse_dates = True).dropna()
                ticker = file.strip('_data.csv')
                COMPLETE_df['Date'] =  np.array(curr_data["Date"])
                COMPLETE_df[ticker] = np.array(curr_data["Close"])

COMPLETE_df = COMPLETE_df.set_index('Date')

COMPLETE_df
```

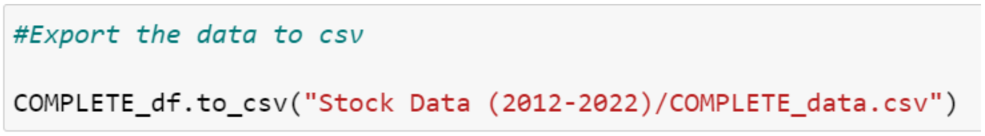
Output:



	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAP
Date													
2012-01-03	39.503304	71.152664	41.390953	55.358418	25.330000	5.446543	50.008633	60.809380	32.779312	57.656933	16.601580	43.781525	12.54004
2012-01-04	39.316895	71.030167	41.418819	55.371304	25.590000	5.474912	49.855579	60.003040	33.000370	57.972462	15.957155	44.419247	12.60743
2012-01-05	39.002655	70.333916	40.688267	55.203938	25.840000	5.437089	50.024742	60.169575	32.960880	58.424438	16.333073	44.716866	12.74740
2012-01-06	38.699070	69.824661	41.134403	54.791969	25.639999	5.342529	49.920025	60.975925	32.731953	60.317593	17.138603	44.878410	12.88066
2012-01-09	38.848198	70.585335	41.257107	55.036602	25.510000	5.361442	49.525307	61.650799	32.992477	60.701366	16.609251	44.827393	12.86023
...	...	...	...	...	...	...	...	...	...	...	...	...	
2021-12-27	71.226372	115.546692	43.536983	60.007717	5.100000	5.231403	59.269157	103.681671	100.744537	457.053223	252.447067	496.256500	179.58686
2021-12-28	71.138817	115.322968	43.232529	59.813797	5.080000	5.191546	59.377491	103.493401	101.108551	461.718079	255.421494	499.674164	178.55114
2021-12-29	70.944229	114.729622	42.898605	59.290226	4.870000	5.032112	59.702492	103.285309	102.023514	465.159485	257.142456	502.296997	178.64077

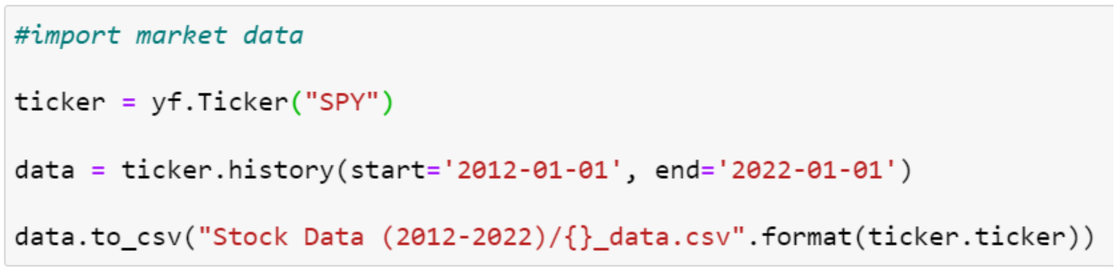
This creates a pandas dataframe that includes closing prices for all the stocks throughout the entire timeframe.

6. Now store this new dataframe as a csv using the following code:



```
#Export the data to csv

COMPLETE_df.to_csv("Stock Data (2012-2022)/COMPLETE_data.csv")
```

7. Import S&P 500 index fund data (i.e., market data) using the following code



```
#import market data

ticker = yf.Ticker("SPY")

data = ticker.history(start='2012-01-01', end='2022-01-01')

data.to_csv("Stock Data (2012-2022)/{}_data.csv".format(ticker.ticker))
```

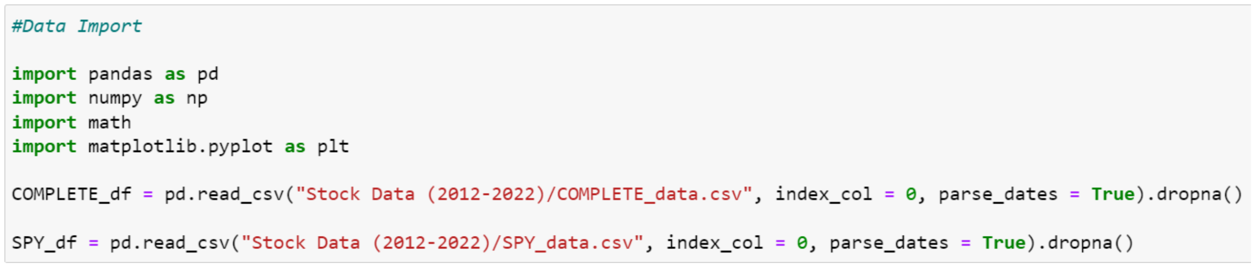
We are done importing the necessary data for this project. Make sure to run all the code before proceeding to next part.

Creating a Portfolio using the Sector Grouping Process

In this section we will create a portfolio using sector group process described in the project description section

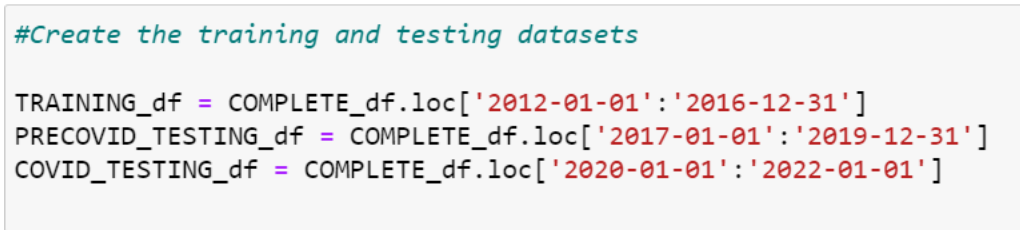
<u>Follow these Steps:</u>

1. Import the stock price data and market data from csv files into pandas dataframes using the following code:



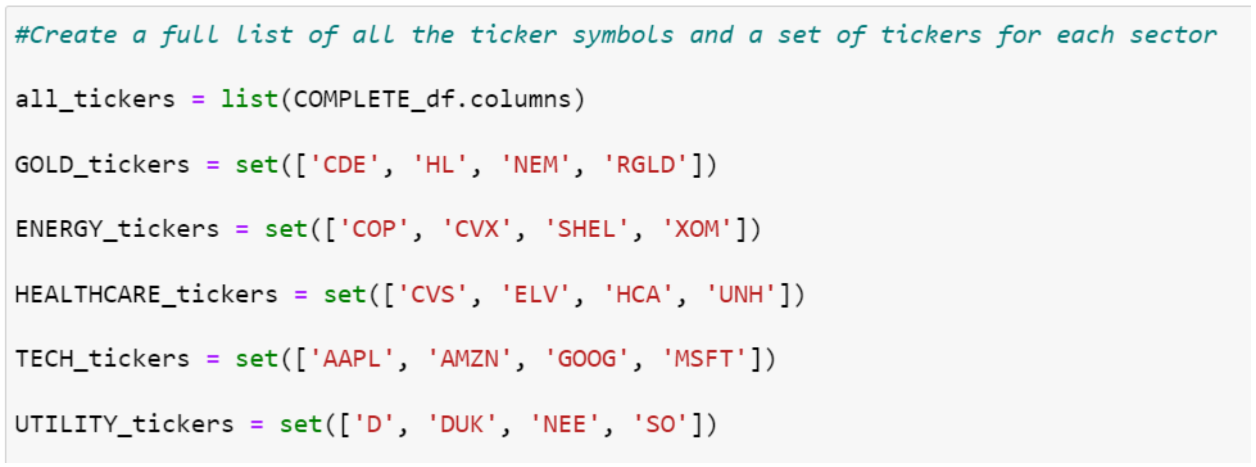
```
#Data Import

import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt

COMPLETE_df = pd.read_csv("Stock Data (2012-2022)/COMPLETE_data.csv", index_col = 0, parse_dates = True).dropna()

SPY_df = pd.read_csv("Stock Data (2012-2022)/SPY_data.csv", index_col = 0, parse_dates = True).dropna()
```

2. Create the training, pre-covid and covid datasets



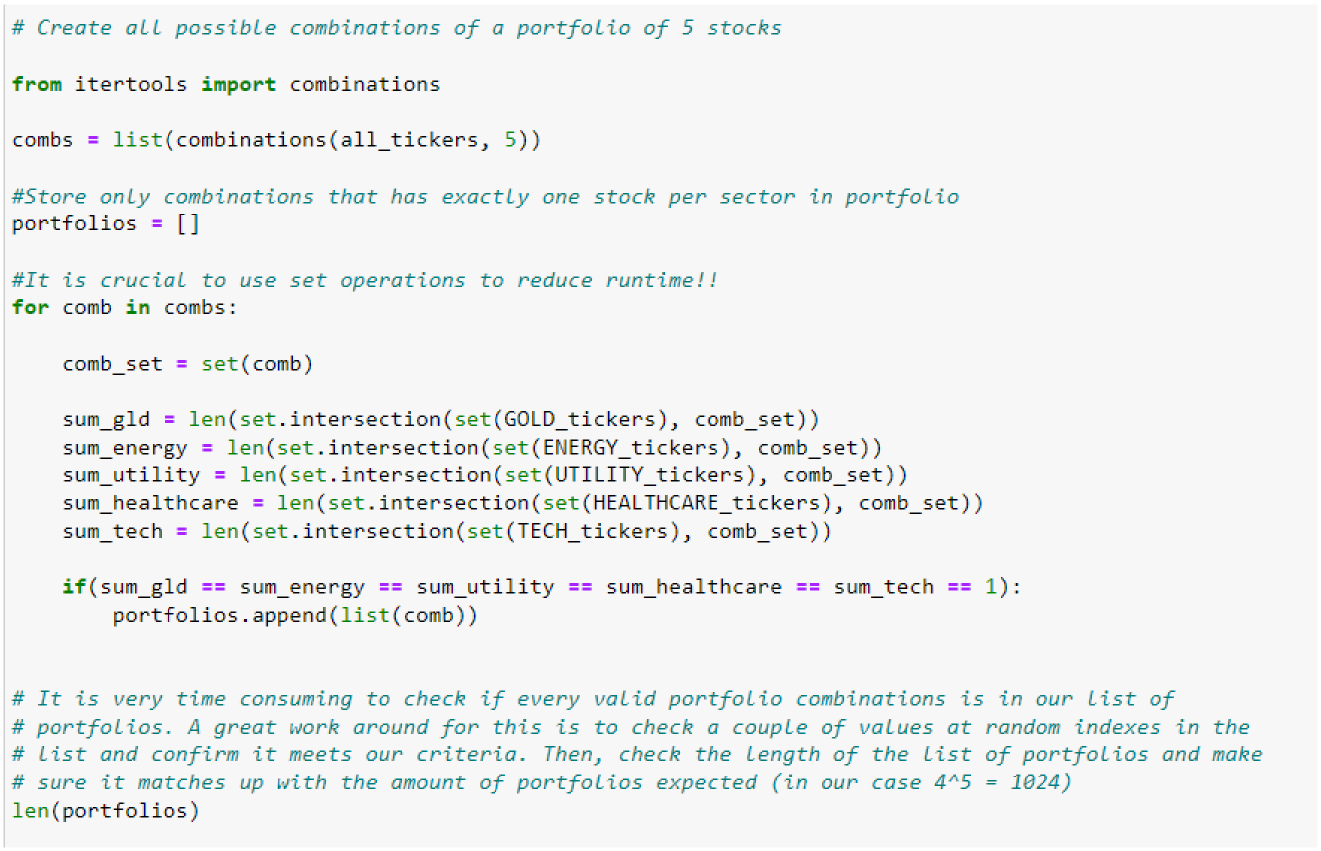
```
#Create the training and testing datasets

TRAINING_df = COMPLETE_df.loc['2012-01-01':'2016-12-31']
PRECOVID_TESTING_df = COMPLETE_df.loc['2017-01-01':'2019-12-31']
COVID_TESTING_df = COMPLETE_df.loc['2020-01-01':'2022-01-01']
```

3. Create a full list of all the ticker symbols and a set of tickers for each sector. These sets will be utilized when creating the portfolios.



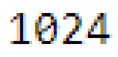
```
#Create a full list of all the ticker symbols and a set of tickers for each sector

all_tickers = list(COMPLETE_df.columns)

GOLD_tickers = set(['CDE', 'HL', 'NEM', 'RGLD'])

ENERGY_tickers = set(['COP', 'CVX', 'SHEL', 'XOM'])

HEALTHCARE_tickers = set(['CVS', 'ELV', 'HCA', 'UNH'])

TECH_tickers = set(['AAPL', 'AMZN', 'GOOG', 'MSFT'])

UTILITY_tickers = set(['D', 'DUK', 'NEE', 'SO'])
```

4. Now we loop through all the possible portfolios with 5 stocks and store the ones that include exactly one stock per sector in a list called “portfolios”. Since there are 4 stocks in each sector and there are 5 sectors, the number of possible combinations is 4^5 = 1024. We can use this information to check whether all valid portfolio combinations are stored in the list “portfolios”.



```
# Create all possible combinations of a portfolio of 5 stocks

from itertools import combinations

combs = list(combinations(all_tickers, 5))

#Store only combinations that has exactly one stock per sector in portfolio
portfolios = []

#It is crucial to use set operations to reduce runtime!!
for comb in combs:

    comb_set = set(comb)

    sum_gld = len(set.intersection(set(GOLD_tickers), comb_set))
    sum_energy = len(set.intersection(set(ENERGY_tickers), comb_set))
    sum_utility = len(set.intersection(set(UTILITY_tickers), comb_set))
    sum_healthcare = len(set.intersection(set(HEALTHCARE_tickers), comb_set))
    sum_tech = len(set.intersection(set(TECH_tickers), comb_set))

    if(sum_gld == sum_energy == sum_utility == sum_healthcare == sum_tech == 1):
        portfolios.append(list(comb))


# It is very time consuming to check if every valid portfolio combinations is in our list of
# portfolios. A great work around for this is to check a couple of values at random indexes in the
# list and confirm it meets our criteria. Then, check the length of the list of portfolios and make
# sure it matches up with the amount of portfolios expected (in our case 4^5 = 1024)
len(portfolios)
```

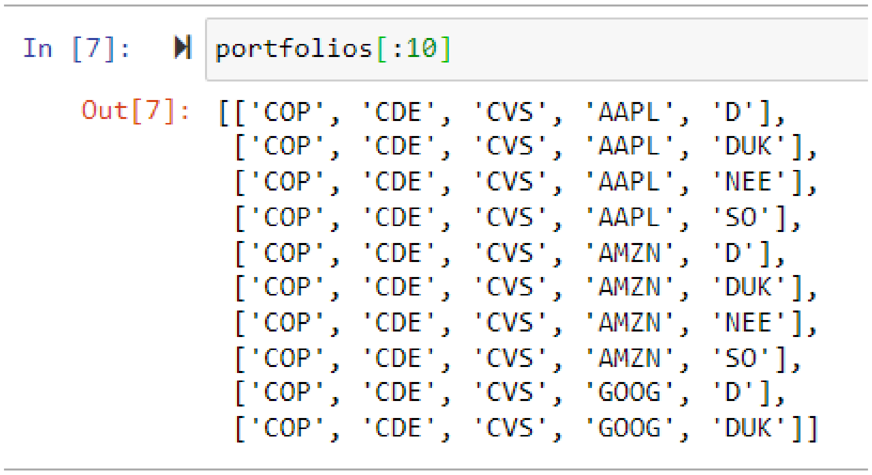
Output:



```
1024
```

The output for the length of portfolios is 1024 which is exactly what we expect.

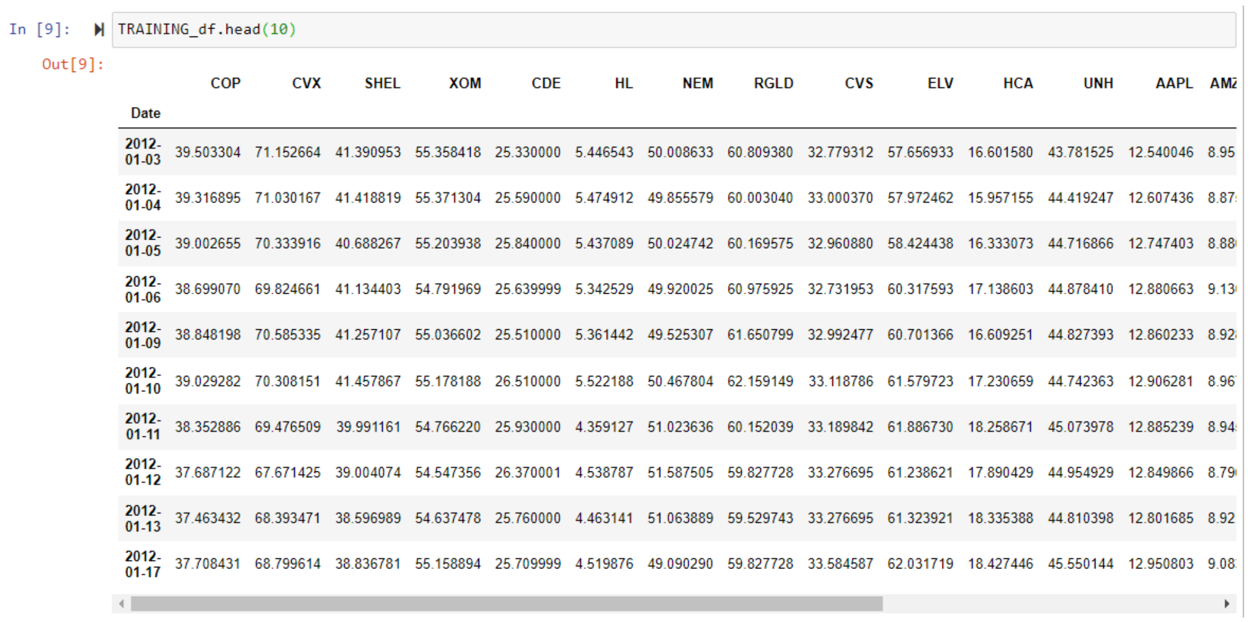
5. Let’s take a look at the first 10 portfolios in the list of portfolios



```
In [7]:    portfolios[:10]

Out[7]: [['COP', 'CDE', 'CVS', 'AAPL', 'D'],
         ['COP', 'CDE', 'CVS', 'AAPL', 'DUK'],
         ['COP', 'CDE', 'CVS', 'AAPL', 'NEE'],
         ['COP', 'CDE', 'CVS', 'AAPL', 'SO'],
         ['COP', 'CDE', 'CVS', 'AMZN', 'D'],
         ['COP', 'CDE', 'CVS', 'AMZN', 'DUK'],
         ['COP', 'CDE', 'CVS', 'AMZN', 'NEE'],
         ['COP', 'CDE', 'CVS', 'AMZN', 'SO'],
         ['COP', 'CDE', 'CVS', 'GOOG', 'D'],
         ['COP', 'CDE', 'CVS', 'GOOG', 'DUK']]
```

CREATING THE PORTFOLIO IN THE TRAINING DATASET

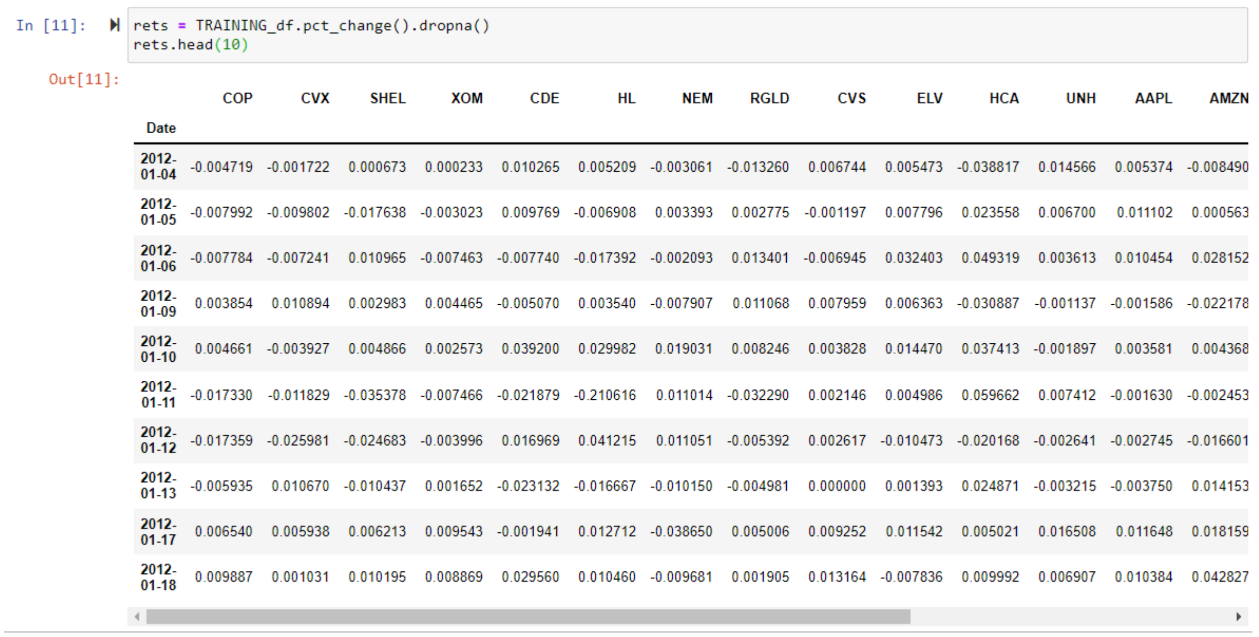
6. Now let’s take a look the training dataset



```
In [9]:   TRAINING_df.head(10)
Out[9]:
```

Date	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAPL	AM
2012-01-03	39.503304	71.152664	41.390953	55.358418	25.330000	5.446543	50.008633	60.809380	32.779312	57.656933	16.601580	43.781525	12.540046	8.95
2012-01-04	39.316895	71.030167	41.418819	55.371304	25.590000	5.474912	49.855579	60.003040	33.000370	57.972462	15.957155	44.419247	12.607436	8.87
2012-01-05	39.002655	70.333916	40.688267	55.203938	25.840000	5.437089	50.024742	60.169575	32.960880	58.424438	16.333073	44.716866	12.747403	8.88
2012-01-06	38.699070	69.824661	41.134403	54.791969	25.639999	5.342529	49.920025	60.975925	32.731953	60.317593	17.138603	44.878410	12.880663	9.13
2012-01-09	38.848198	70.585335	41.257107	55.036602	25.510000	5.361442	49.525307	61.650799	32.992477	60.701366	16.609251	44.827393	12.860233	8.92
2012-01-10	39.029282	70.308151	41.457867	55.178188	26.510000	5.522188	50.467804	62.159149	33.118786	61.579723	17.230659	44.742363	12.906281	8.96
2012-01-11	38.352886	69.476509	39.991161	54.766220	25.930000	4.359127	51.023636	60.152039	33.189842	61.886730	18.258671	45.073978	12.885239	8.94
2012-01-12	37.687122	67.671425	39.004074	54.547356	26.370001	4.538787	51.587505	59.827728	33.276695	61.238621	17.890429	44.954929	12.849866	8.79
2012-01-13	37.463432	68.393471	38.596989	54.637478	25.760000	4.463141	51.063889	59.529743	33.276695	61.323921	18.335388	44.810398	12.801685	8.92
2012-01-17	37.708431	68.799614	38.836781	55.158894	25.709999	4.519876	49.090290	59.827728	33.584587	62.031719	18.427446	45.550144	12.950803	9.08

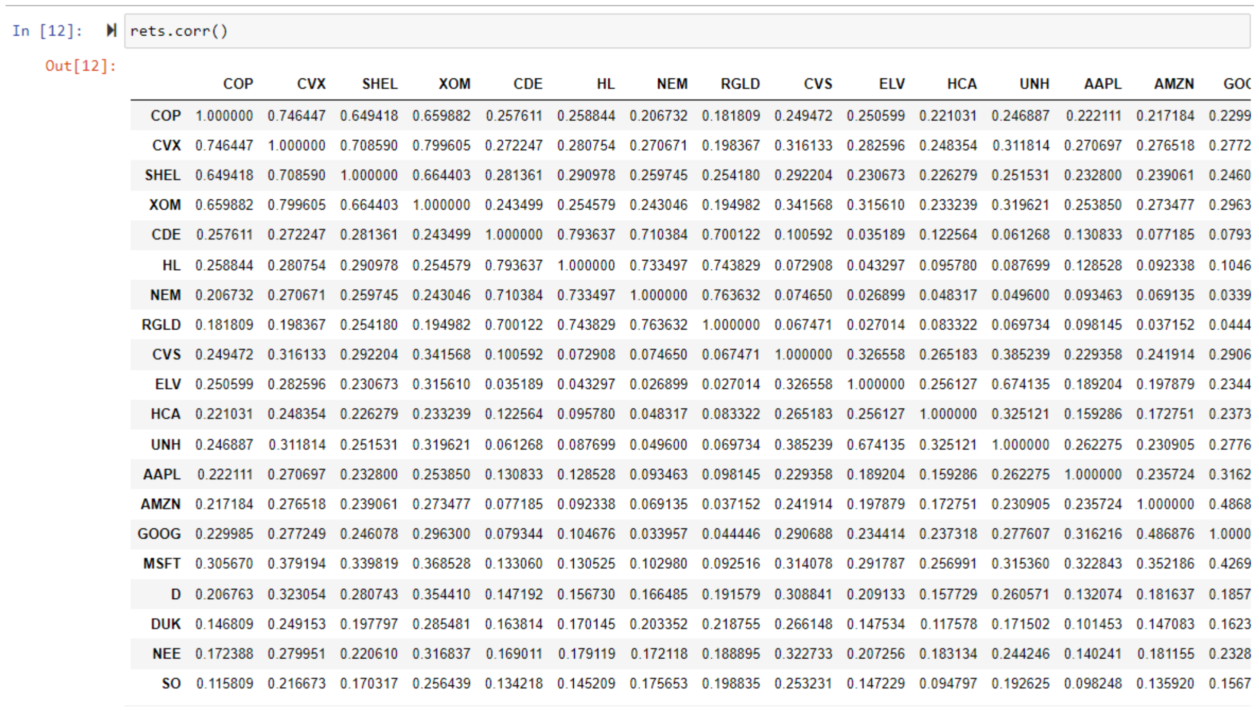
7. We can calculate the daily returns of the stocks by using the following code:



```
In [11]:   rets = TRAINING_df.pct_change().dropna()
           rets.head(10)
Out[11]:
```

Date	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAPL	AMZN
2012-01-04	-0.004719	-0.001722	0.000673	0.000233	0.010265	0.005209	-0.003061	-0.013260	0.006744	0.005473	-0.038817	0.014566	0.005374	-0.008490
2012-01-05	-0.007992	-0.009802	-0.017638	-0.003023	0.009769	-0.006908	0.003393	0.002775	-0.001197	0.007796	0.023558	0.006700	0.011102	0.000563
2012-01-06	-0.007784	-0.007241	0.010965	-0.007463	-0.007740	-0.017392	-0.002093	0.013401	-0.006945	0.032403	0.049319	0.003613	0.010454	0.028152
2012-01-09	0.003854	0.010894	0.002983	0.004465	-0.005070	0.003540	-0.007907	0.011068	0.007959	0.006363	-0.030887	-0.001137	-0.001586	-0.022178
2012-01-10	0.004661	-0.003927	0.004866	0.002573	0.039200	0.029982	0.019031	0.008246	0.003828	0.014470	0.037413	-0.001897	0.003581	0.004368
2012-01-11	-0.017330	-0.011829	-0.035378	-0.007466	-0.021879	-0.210616	0.011014	-0.032290	0.002146	0.004986	0.059662	0.007412	-0.001630	-0.002453
2012-01-12	-0.017359	-0.025981	-0.024683	-0.003996	0.016969	0.041215	0.011051	-0.005392	0.002617	-0.010473	-0.020168	-0.002641	-0.002745	-0.016601
2012-01-13	-0.005935	0.010670	-0.010437	0.001652	-0.023132	-0.016667	-0.010150	-0.004981	0.000000	0.001393	0.024871	-0.003215	-0.003750	0.014153
2012-01-17	0.006540	0.005938	0.006213	0.009543	-0.001941	0.012712	-0.038650	0.005006	0.009252	0.011542	0.005021	0.016508	0.011648	0.018159
2012-01-18	0.009887	0.001031	0.010195	0.008869	0.029560	0.010460	-0.009681	0.001905	0.013164	-0.007836	0.009992	0.006907	0.010384	0.042827

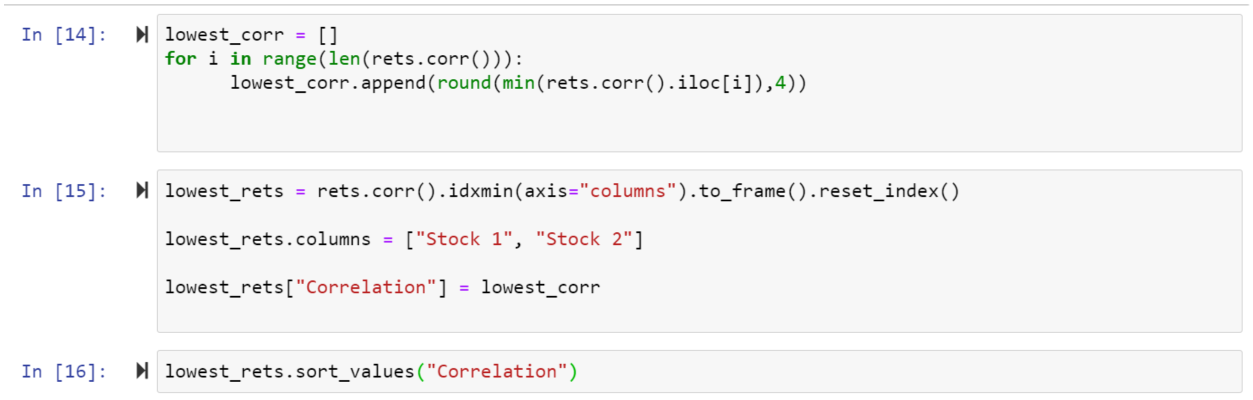
8. We can create a correlation matrix like so:



```
In [12]:  rets.corr()

Out[12]:
```

	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAPL	AMZN	GOO
COP	1.000000	0.746447	0.649418	0.659882	0.257611	0.258844	0.206732	0.181809	0.249472	0.250599	0.221031	0.246887	0.222111	0.217184	0.2299
CVX	0.746447	1.000000	0.708590	0.799605	0.272247	0.280754	0.270671	0.198367	0.316133	0.282596	0.248354	0.311814	0.270697	0.276518	0.2772
SHEL	0.649418	0.708590	1.000000	0.664403	0.281361	0.290978	0.259745	0.254180	0.292204	0.230673	0.226279	0.251531	0.232800	0.239061	0.2460
XOM	0.659882	0.799605	0.664403	1.000000	0.243499	0.254579	0.243046	0.194982	0.341568	0.315610	0.233239	0.319621	0.253850	0.273477	0.2963
CDE	0.257611	0.272247	0.281361	0.243499	1.000000	0.793637	0.710384	0.700122	0.100592	0.035189	0.122564	0.061268	0.130833	0.077185	0.0793
HL	0.258844	0.280754	0.290978	0.254579	0.793637	1.000000	0.733497	0.743829	0.072908	0.043297	0.095780	0.087699	0.128528	0.092338	0.1046
NEM	0.206732	0.270671	0.259745	0.243046	0.710384	0.733497	1.000000	0.763632	0.074650	0.026899	0.048317	0.049600	0.093463	0.069135	0.0339
RGLD	0.181809	0.198367	0.254180	0.194982	0.700122	0.743829	0.763632	1.000000	0.067471	0.027014	0.083322	0.069734	0.098145	0.037152	0.0444
CVS	0.249472	0.316133	0.292204	0.341568	0.100592	0.072908	0.074650	0.067471	1.000000	0.326558	0.265183	0.385239	0.229358	0.241914	0.2906
ELV	0.250599	0.282596	0.230673	0.315610	0.035189	0.043297	0.026899	0.027014	0.326558	1.000000	0.256127	0.674135	0.189204	0.197879	0.2344
HCA	0.221031	0.248354	0.226279	0.233239	0.122564	0.095780	0.048317	0.083322	0.265183	0.256127	1.000000	0.325121	0.159286	0.172751	0.2373
UNH	0.246887	0.311814	0.251531	0.319621	0.061268	0.087699	0.049600	0.069734	0.385239	0.674135	0.325121	1.000000	0.262275	0.230905	0.2776
AAPL	0.222111	0.270697	0.232800	0.253850	0.130833	0.128528	0.093463	0.098145	0.229358	0.189204	0.159286	0.262275	1.000000	0.235724	0.3162
AMZN	0.217184	0.276518	0.239061	0.273477	0.077185	0.092338	0.069135	0.037152	0.241914	0.197879	0.172751	0.230905	0.235724	1.000000	0.4868
GOOG	0.229985	0.277249	0.246078	0.296300	0.079344	0.104676	0.033957	0.044446	0.290688	0.234414	0.237318	0.277607	0.316216	0.486876	1.0000
MSFT	0.305670	0.379194	0.339819	0.368528	0.133060	0.130525	0.102980	0.092516	0.314078	0.291787	0.256991	0.315360	0.322843	0.352186	0.4269
D	0.206763	0.323054	0.280743	0.354410	0.147192	0.156730	0.166485	0.191579	0.308841	0.209133	0.157729	0.260571	0.132074	0.181637	0.1857
DUK	0.146809	0.249153	0.197797	0.285481	0.163814	0.170145	0.203352	0.218755	0.266148	0.147534	0.117578	0.171502	0.101453	0.147083	0.1623
NEE	0.172388	0.279951	0.220610	0.316837	0.169011	0.179119	0.172118	0.188895	0.322733	0.207256	0.183134	0.244246	0.140241	0.181155	0.2328
SO	0.115809	0.216673	0.170317	0.256439	0.134218	0.145209	0.175653	0.198835	0.253231	0.147229	0.094797	0.192625	0.098248	0.135920	0.1567

9. We use the correlation matrix and the following code to find out which stocks are least correlated with each other.



```
In [14]:  lowest_corr = []
          for i in range(len(rets.corr())):
              lowest_corr.append(round(min(rets.corr().iloc[i]),4))
```

```
In [15]:  lowest_rets = rets.corr().idxmin(axis="columns").to_frame().reset_index()

          lowest_rets.columns = ["Stock 1", "Stock 2"]

          lowest_rets["Correlation"] = lowest_corr
```

```
In [16]:  lowest_rets.sort_values("Correlation")
```

Output:

```
Out[16]:
```

	Stock 1	Stock 2	Correlation
9	ELV	NEM	0.0269
6	NEM	ELV	0.0269
7	RGLD	ELV	0.0270
14	GOOG	NEM	0.0340
4	CDE	ELV	0.0352
13	AMZN	RGLD	0.0372
5	HL	ELV	0.0433
10	HCA	NEM	0.0483
11	UNH	NEM	0.0496
8	CVS	RGLD	0.0675
15	MSFT	RGLD	0.0925
12	AAPL	NEM	0.0935
19	SO	HCA	0.0948
17	DUK	AAPL	0.1015
0	COP	SO	0.1158
16	D	AAPL	0.1321
18	NEE	AAPL	0.1402
2	SHEL	SO	0.1703
3	XOM	RGLD	0.1950
1	CVX	RGLD	0.1984

Note the output is a dataframe where for each stock in the "Stock 1" column, the ticker in the "Stock 2" column represents the stock that is least correlated with it. This information can be useful when constructing a diverse portfolio and while we do not utilize it to do so, it can often be a good starting place. For our purposes this is just to give us an idea of which stocks are least correlated with each other and is not used to actually create the portfolio.

10. Create functions to calculate the portfolio expected return and volatility for given a portfolio dataframe and weights.

```python
# Expected return and volatility for any portfolio weights

def port_ret(weights, portfolio_df):
    port_rets = np.log(portfolio_df/portfolio_df.shift(1))
    return np.sum(port_rets.mean() * weights) * 252

def port_vol(weights, portfolio_df):
    port_rets = np.log(portfolio_df/portfolio_df.shift(1))
    return np.sqrt(np.dot(weights.T, np.dot(port_rets.cov() * 252, weights)))
```

11. Create a function that can calculate the Sharpe ratio given a portfolio dataframe and weights

```python
# Optimal portfolio
import scipy.optimize as sco

noa = 5

def min_func_sharpe(weights, portfolio_df):
 return -port_ret(weights, portfolio_df) / port_vol(weights, portfolio_df)

# Equality constraint
cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

# Bounds for the parameters
bnds = tuple((0, 1) for x in range(noa))

# Starting parameter list
# Equal weights vector
eweights = np.array(noa * [1. / noa,])
```

Note the function "min_func_sharpe" returns a negative Sharpe ratio so the minimize function that is used in the next step can adjust the portfolio weights in order to minimize the negative Sharpe ratio. Thus by taking the absolute value of the most negative Sharpe ratio we have the maximum Sharpe ratio of portfolio.

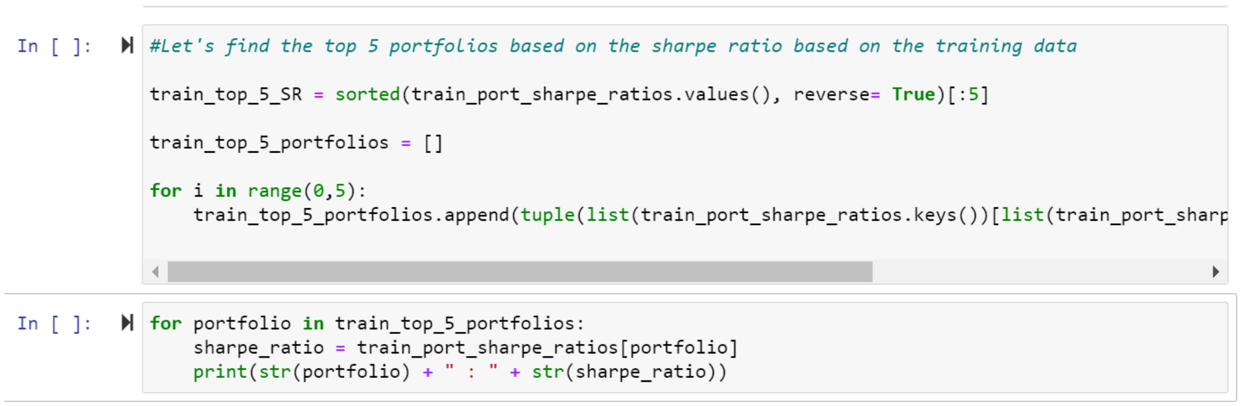
12. Find the minimum Sharpe ratio possible for each portfolio and store the information. In this part we are looping through all the valid portfolios and finding the maximum Sharpe ratio for each one using the "sco.minimize" function.

```python
# Optimal portfolio by maximizing the Sharpe Ratio
# Solve for the optimal weights with the maximum sharpe ratio

train_port_sharpe_ratios = {}

for comb in portfolios:
    portfolio_df = TRAINING_df[comb]
    opts = sco.minimize(min_func_sharpe, eweights, args=(portfolio_df), method = 'SLSQP',
                        bounds = bnds, constraints = cons)

    sharpe_ratio = -opts['fun']

    train_port_sharpe_ratios[tuple(comb)] = sharpe_ratio
```

Note: this chunk of code may take a while but be patient and wait until the code chunk is done running.

When the code is finished executing the variable “train_port_sharpe_ratios” will be a dictionary where the keys are a tuple of stocks that make up a portfolio and the values are the Sharpe ratios associated with that portfolio.

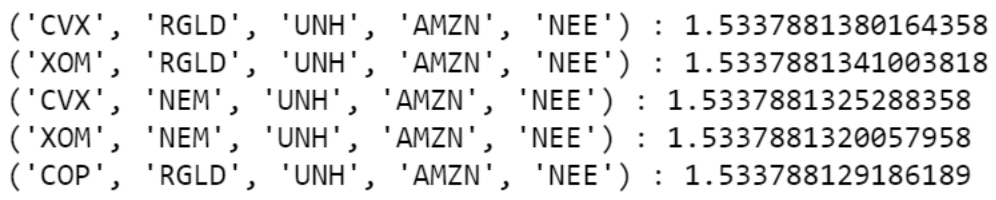
13. Let’s take a look at the top 5 portfolios with the highest Sharpe ratios



```
In [ ]:  #Let's find the top 5 portfolios based on the sharpe ratio based on the training data

         train_top_5_SR = sorted(train_port_sharpe_ratios.values(), reverse= True)[:5]

         train_top_5_portfolios = []

         for i in range(0,5):
             train_top_5_portfolios.append(tuple(list(train_port_sharpe_ratios.keys())[list(train_port_sharp
```

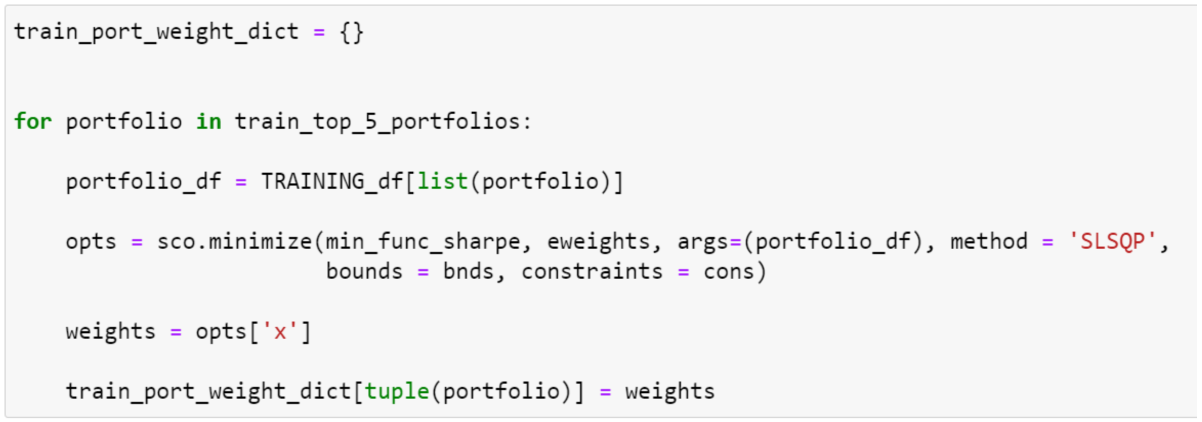
```
In [ ]:  for portfolio in train_top_5_portfolios:
             sharpe_ratio = train_port_sharpe_ratios[portfolio]
             print(str(portfolio) + " : " + str(sharpe_ratio))
```

Output:



```
('CVX', 'RGLD', 'UNH', 'AMZN', 'NEE') : 1.5337881380164358
('XOM', 'RGLD', 'UNH', 'AMZN', 'NEE') : 1.5337881341003818
('CVX', 'NEM', 'UNH', 'AMZN', 'NEE') : 1.5337881325288358
('XOM', 'NEM', 'UNH', 'AMZN', 'NEE') : 1.5337881320057958
('COP', 'RGLD', 'UNH', 'AMZN', 'NEE') : 1.533788129186189
```

14. Now let’s find the weights that allows each portfolio to achieve it’s maximum Sharpe ratio

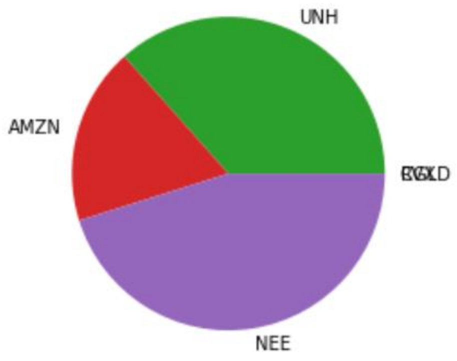


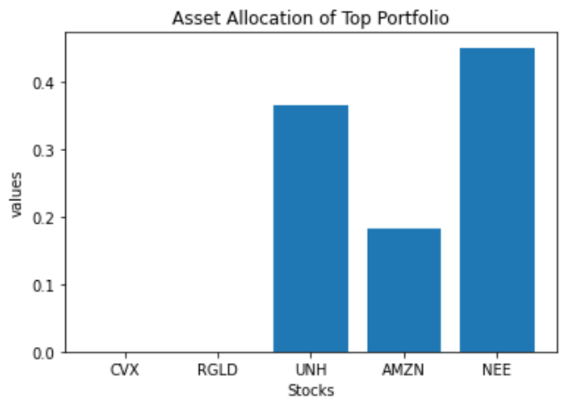
```
train_port_weight_dict = {}


for portfolio in train_top_5_portfolios:

    portfolio_df = TRAINING_df[list(portfolio)]

    opts = sco.minimize(min_func_sharpe, eweights, args=(portfolio_df), method = 'SLSQP',
                        bounds = bnds, constraints = cons)

    weights = opts['x']

    train_port_weight_dict[tuple(portfolio)] = weights
```

Note: the variable "train_port_weight_dict" is dictionary where the keys are tuples of the top 5 portfolios and the keys are the associated weights that maximize the portfolio's Sharpe ratio.

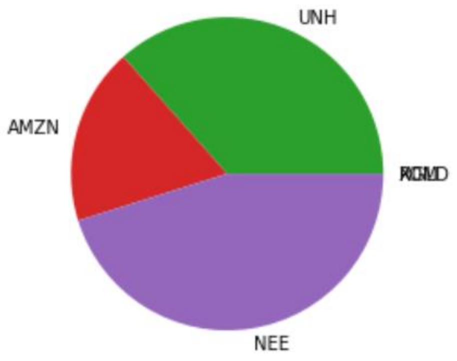
15. Let's visually examine the asset allocation of each of the top 5 portfolios with the highest Sharpe ratios

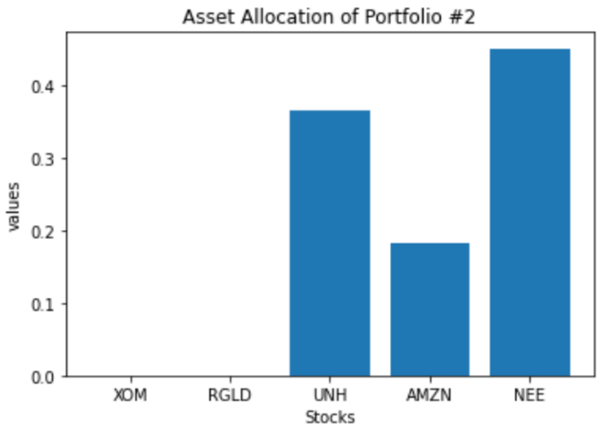
Portfolio #1:



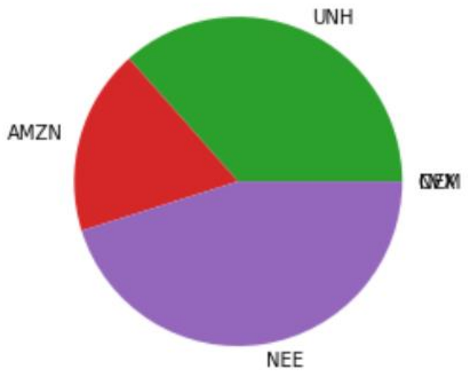


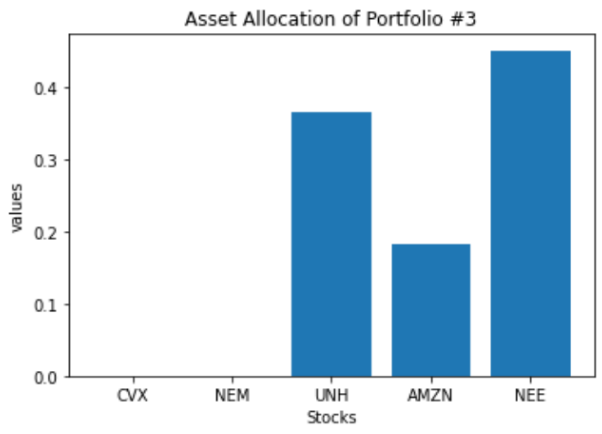
Portfolio #2:



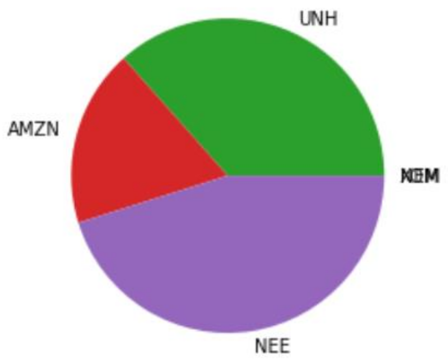


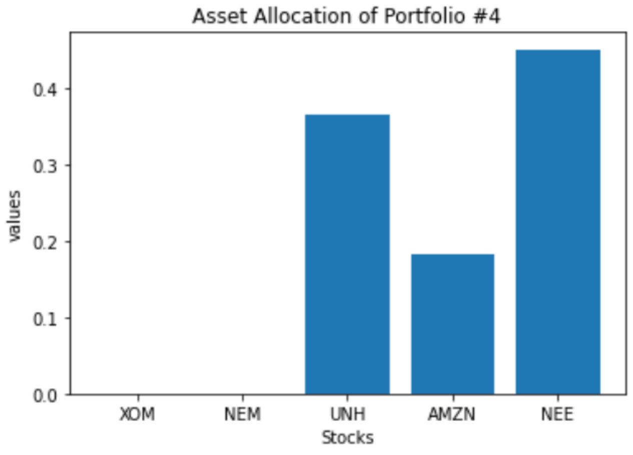
Portfolio #3:



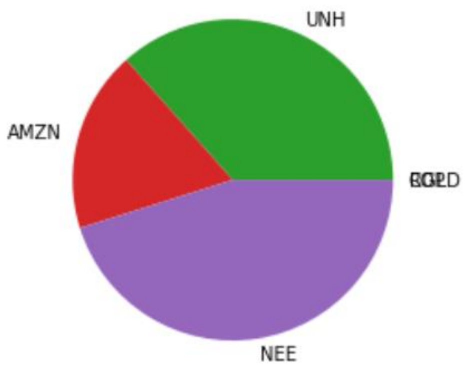


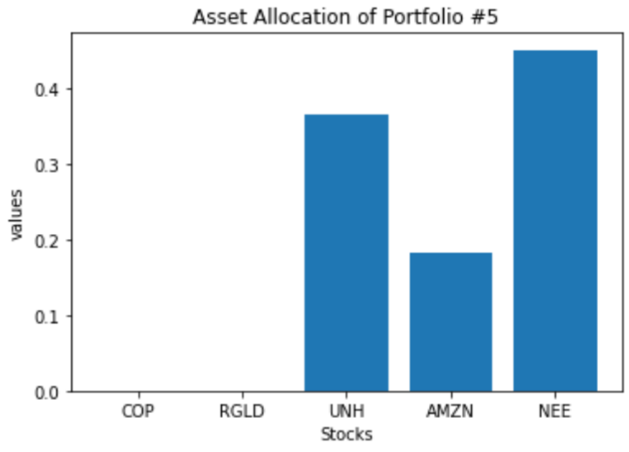
Portfolio #4:





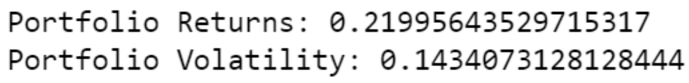
Portfolio #5:





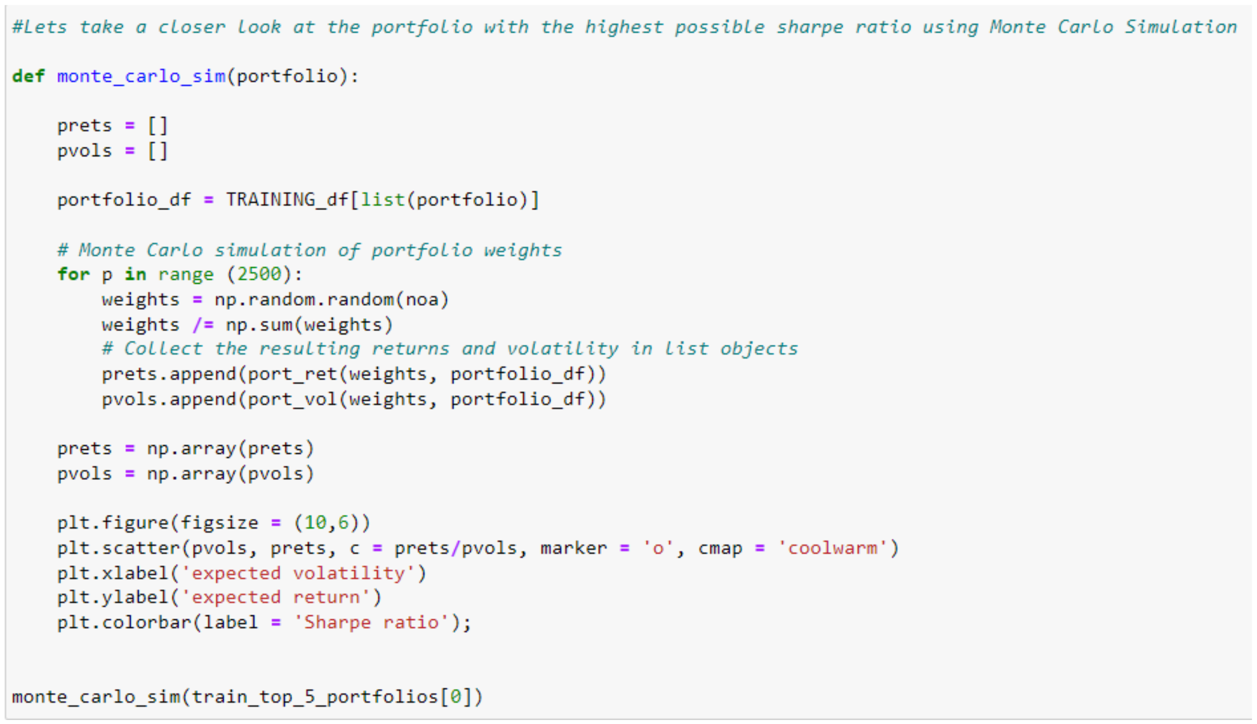
All 5 of the top portfolios with the highest Sharpe ratios are the same in that it is comprised of only UNH, AMZN, and NEE with the other two stocks having zero weight in the asset allocation of the portfolio. This implies that in order to have the highest Sharpe ratio possible it maybe best to exclude stocks from the gold and healthcare sectors. Thus, indicating the optimal portfolio only includes stocks that have stocks from the healthcare, tech and utility sectors and in particular should include UNH, AMZN, and NEE, respectively.

16. Since all portfolios are essential the same, let's focus on the top portfolio (highest Sharpe ratio). Let's examine the portfolio annual expected returns and volatility.



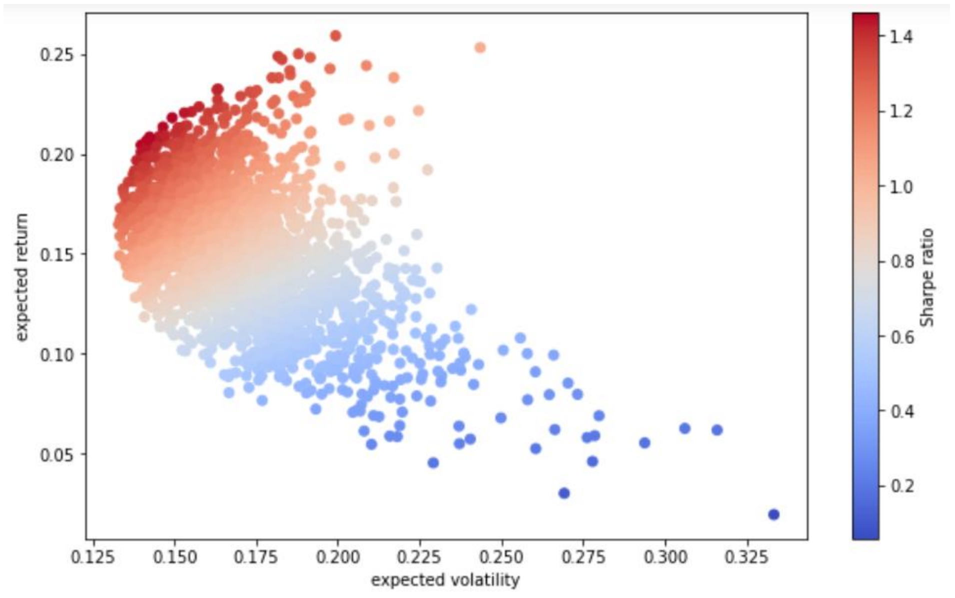
```
Portfolio Returns: 0.21995643529715317
Portfolio Volatility: 0.1434073128128444
```

17. Examine visually the expected returns and volatility of the top portfolio with different weights using a Monte Carlo Simulation.



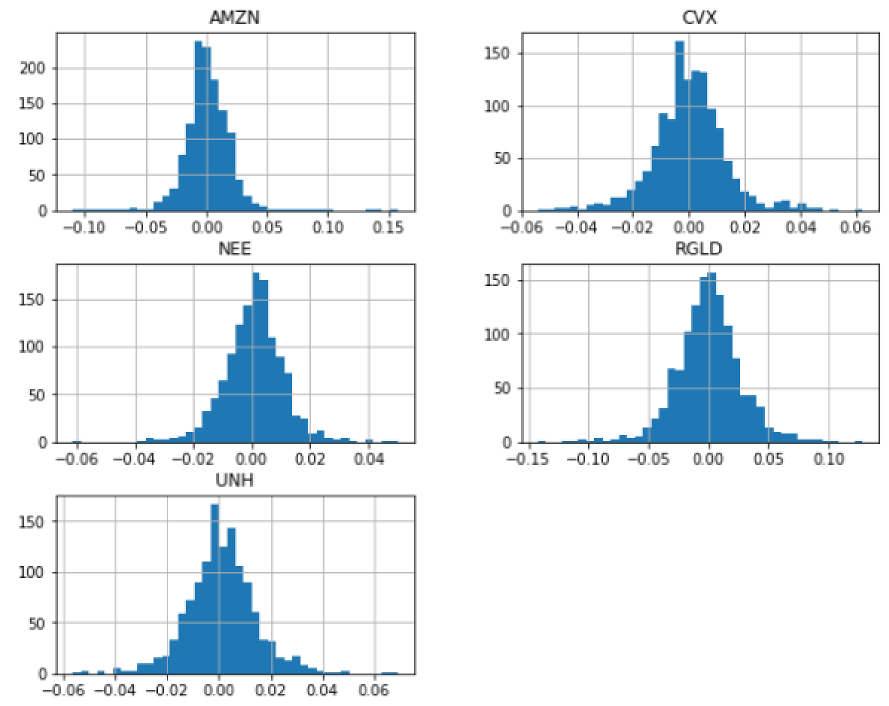
```
#Lets take a closer look at the portfolio with the highest possible sharpe ratio using Monte Carlo Simulation

def monte_carlo_sim(portfolio):

    prets = []
    pvols = []

    portfolio_df = TRAINING_df[list(portfolio)]

    # Monte Carlo simulation of portfolio weights
    for p in range (2500):
        weights = np.random.random(noa)
        weights /= np.sum(weights)
        # Collect the resulting returns and volatility in list objects
        prets.append(port_ret(weights, portfolio_df))
        pvols.append(port_vol(weights, portfolio_df))

    prets = np.array(prets)
    pvols = np.array(pvols)

    plt.figure(figsize = (10,6))
    plt.scatter(pvols, prets, c = prets/pvols, marker = 'o', cmap = 'coolwarm')
    plt.xlabel('expected volatility')
    plt.ylabel('expected return')
    plt.colorbar(label = 'Sharpe ratio');


monte_carlo_sim(train_top_5_portfolios[0])
```

Output:



Depending on the investor’s level of risk tolerance they may want to adjust the portfolio weights / asset allocation to have a portfolio with less volatility. However, for our purposes we are strictly interested in the portfolio that gives the best expected return to expected volatility ratio (Sharpe ratio).

18. Examine the distribution of returns of each of the stocks in the portfolio with the highest Sharpe ratio.



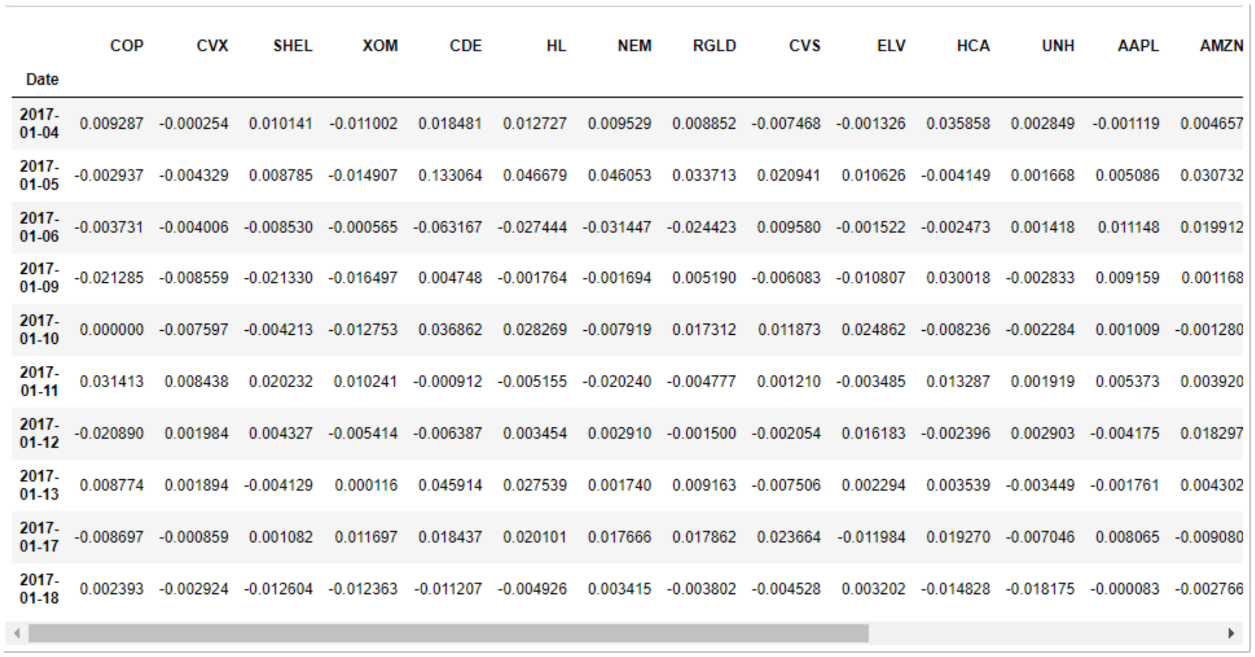
These histograms can give us an idea of how frequently we can expect certain returns when allocating resources to that stock and how the returns were distributed in the timeframe of the training dataset. Interestingly, we can see that all histograms are normally distributed.

BACKTESTING THE PORTFOLIO IN THE PRE-COVID DATASET

19. Let’s take a quick look at the daily returns of the stocks in the pre-covid testing dataset.

```
#Let's take a quick look at the returns in the PRE-COVID testing dataset

PRECOVID_rets = PRECOVID_TESTING_df.pct_change().dropna()
PRECOVID_rets.head(10)
```

Output:



Date	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAPL	AMZN
2017-01-04	0.009287	-0.000254	0.010141	-0.011002	0.018481	0.012727	0.009529	0.008852	-0.007468	-0.001326	0.035858	0.002849	-0.001119	0.004657
2017-01-05	-0.002937	-0.004329	0.008785	-0.014907	0.133064	0.046679	0.046053	0.033713	0.020941	0.010626	-0.004149	0.001668	0.005086	0.030732
2017-01-06	-0.003731	-0.004006	-0.008530	-0.000565	-0.063167	-0.027444	-0.031447	-0.024423	0.009580	-0.001522	-0.002473	0.001418	0.011148	0.019912
2017-01-09	-0.021285	-0.008559	-0.021330	-0.016497	0.004748	-0.001764	-0.001694	0.005190	-0.006083	-0.010807	0.030018	-0.002833	0.009159	0.001168
2017-01-10	0.000000	-0.007597	-0.004213	-0.012753	0.036862	0.028269	-0.007919	0.017312	0.011873	0.024862	-0.008236	-0.002284	0.001009	-0.001280
2017-01-11	0.031413	0.008438	0.020232	0.010241	-0.000912	-0.005155	-0.020240	-0.004777	0.001210	-0.003485	0.013287	0.001919	0.005373	0.003920
2017-01-12	-0.020890	0.001984	0.004327	-0.005414	-0.006387	0.003454	0.002910	-0.001500	-0.002054	0.016183	-0.002396	0.002903	-0.004175	0.018297
2017-01-13	0.008774	0.001894	-0.004129	0.000116	0.045914	0.027539	0.001740	0.009163	-0.007506	0.002294	0.003539	-0.003449	-0.001761	0.004302
2017-01-17	-0.008697	-0.000859	0.001082	0.011697	0.018437	0.020101	0.017666	0.017862	0.023664	-0.011984	0.019270	-0.007046	0.008065	-0.009080
2017-01-18	0.002393	-0.002924	-0.012604	-0.012363	-0.011207	-0.004926	0.003415	-0.003802	-0.004528	0.003202	-0.014828	-0.018175	-0.000083	-0.002766

20. Now let’s take a look at the Sharpe ratios of the top portfolios in the pre-covid testing dataset

```
#Now let's take a look at how the top portfolios found using the training dataset performed in the PRE-COVID testing dataset

for portfolio in train_top_5_portfolios:

    weights = train_port_weight_dict[portfolio]
    portfolio_df = PRECOVID_TESTING_df[list(portfolio)]
    sharpe_ratio = -min_func_sharpe(weights, portfolio_df)

    print(str(portfolio) + " : " + str(sharpe_ratio))
```

```
('CVX', 'RGLD', 'UNH', 'AMZN', 'NEE') : 1.9213678647677745
('XOM', 'RGLD', 'UNH', 'AMZN', 'NEE') : 1.9213542282791605
('CVX', 'NEM', 'UNH', 'AMZN', 'NEE') : 1.9213498929782922
('XOM', 'NEM', 'UNH', 'AMZN', 'NEE') : 1.9213483607528457
('COP', 'RGLD', 'UNH', 'AMZN', 'NEE') : 1.9213363061797535
```

Surprisingly, the Sharpe ratios in this dataset are even higher than the ones in the training dataset which is a good sign. This tells us confidence that the portfolio creation process was not over-fitted based on the data.

21. Let’s take a look at the expected returns and violated in this dataset

```
#Now let's take a look closer at how the top portfolio in the training dataset performed

weights = train_port_weight_dict[top_port]
portfolio_df = PRECOVID_TESTING_df[list(top_port)]
PRECOVID_port_ret = port_ret(weights, portfolio_df)
PRECOVID_port_vol = port_vol(weights, portfolio_df)
print("Portfolio Returns: " + str(PRECOVID_port_ret))
print("Portfolio Volatility: " + str(PRECOVID_port_vol))
```

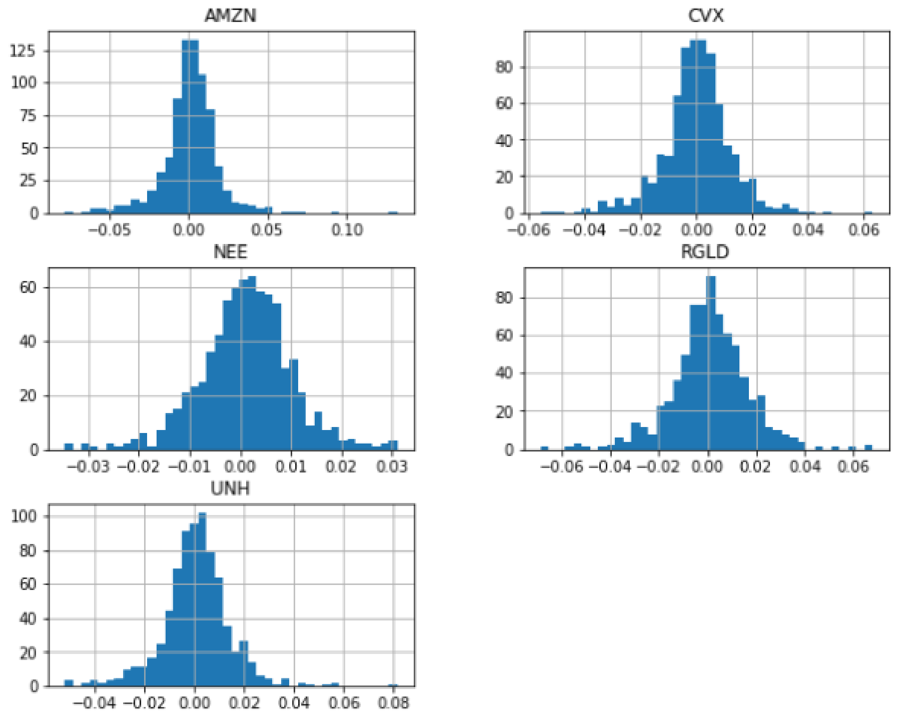
Output:

```
Portfolio Returns: 0.2532479060348971
Portfolio Volatility: 0.13180604853381672
```

22. Now let’s see how the individual stock returns are distributed in this dataset compared to the returns in the training dataset

```
port_rets = PRECOVID_rets[list(top_port)]
port_rets.hist(bins = 40, figsize = (10, 8));
```

Output:



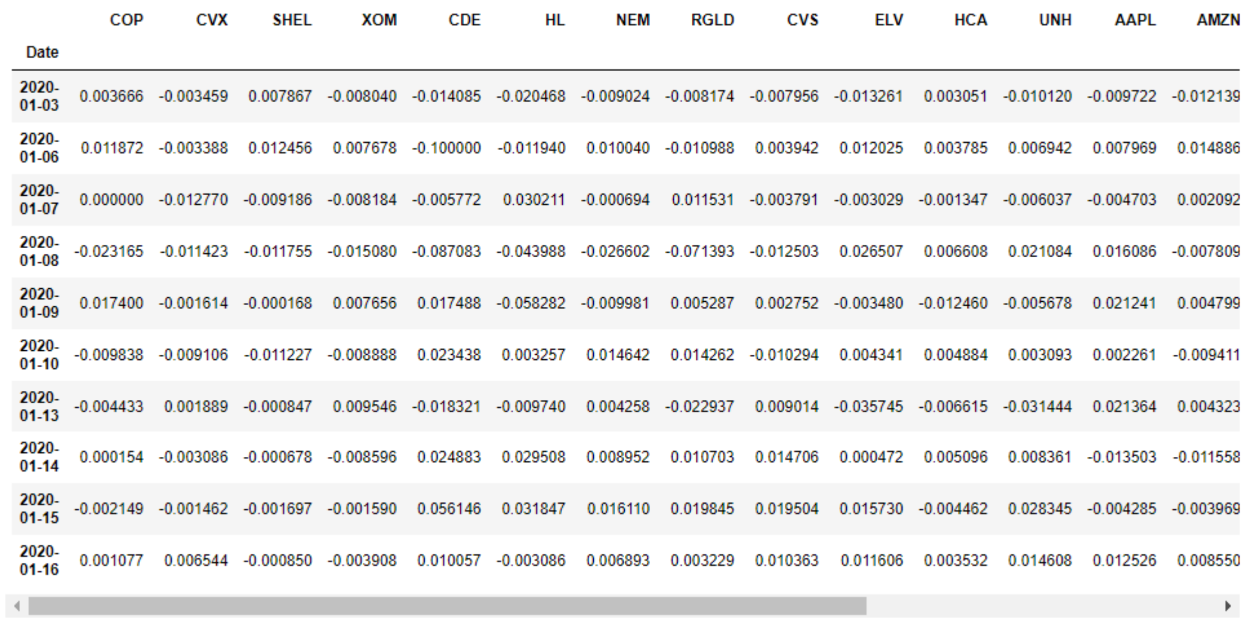
We see that for the most part the returns are distributed similarly to the returns in the training dataset.

BACKTESTING THE PORTFOLIO IN THE COVID DATASET

23. Let’s take a quick look at the daily returns of the stocks in the covid testing dataset.

```
#Let's take a quick look at the returns in the COVID testing dataset

COVID_rets = COVID_TESTING_df.pct_change().dropna()
COVID_rets.head(10)
```

Output:



	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAPL	AMZN
Date														
2020-01-03	0.003666	-0.003459	0.007867	-0.008040	-0.014085	-0.020468	-0.009024	-0.008174	-0.007956	-0.013261	0.003051	-0.010120	-0.009722	-0.012139
2020-01-06	0.011872	-0.003388	0.012456	0.007678	-0.100000	-0.011940	0.010040	-0.010988	0.003942	0.012025	0.003785	0.006942	0.007969	0.014886
2020-01-07	0.000000	-0.012770	-0.009186	-0.008184	-0.005772	0.030211	-0.000694	0.011531	-0.003791	-0.003029	-0.001347	-0.006037	-0.004703	0.002092
2020-01-08	-0.023165	-0.011423	-0.011755	-0.015080	-0.087083	-0.043988	-0.026602	-0.071393	-0.012503	0.026507	0.006608	0.021084	0.016086	-0.007809
2020-01-09	0.017400	-0.001614	-0.000168	0.007656	0.017488	-0.058282	-0.009981	0.005287	0.002752	-0.003480	-0.012460	-0.005678	0.021241	0.004799
2020-01-10	-0.009838	-0.009106	-0.011227	-0.008888	0.023438	0.003257	0.014642	0.014262	-0.010294	0.004341	0.004884	0.003093	0.002261	-0.009411
2020-01-13	-0.004433	0.001889	-0.000847	0.009546	-0.018321	-0.009740	0.004258	-0.022937	0.009014	-0.035745	-0.006615	-0.031444	0.021364	0.004323
2020-01-14	0.000154	-0.003086	-0.000678	-0.008596	0.024883	0.029508	0.008952	0.010703	0.014706	0.000472	0.005096	0.008361	-0.013503	-0.011558
2020-01-15	-0.002149	-0.001462	-0.001697	-0.001590	0.056146	0.031847	0.016110	0.019845	0.019504	0.015730	-0.004462	0.028345	-0.004285	-0.003969
2020-01-16	0.001077	0.006544	-0.000850	-0.003908	0.010057	-0.003086	0.006893	0.003229	0.010363	0.011606	0.003532	0.014608	0.012526	0.008550

24. Now let’s take a look at the Sharpe ratios of the top portfolios in the covid testing dataset.

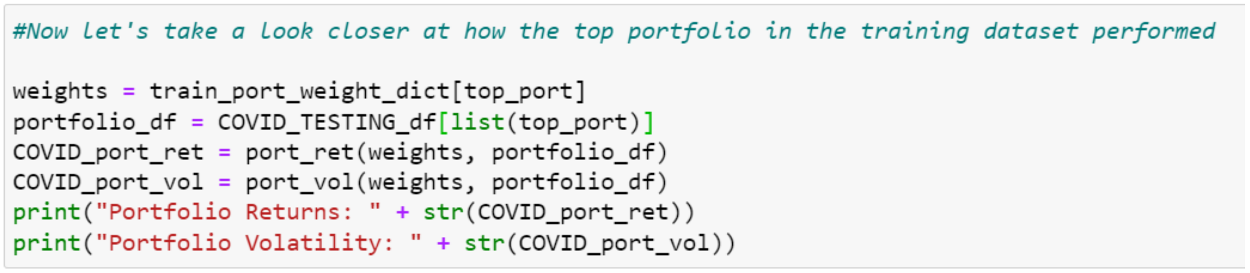
```
#Now let's take a look at how the top portfolios found usin

for portfolio in train_top_5_portfolios:

    weights = train_port_weight_dict[portfolio]
    portfolio_df = COVID_TESTING_df[list(portfolio)]
    sharpe_ratio = -min_func_sharpe(weights, portfolio_df)

    print(str(portfolio) + " : " + str(sharpe_ratio))
```

Output:

```
('CVX', 'RGLD', 'UNH', 'AMZN', 'NEE') : 0.9443305643406732
('XOM', 'RGLD', 'UNH', 'AMZN', 'NEE') : 0.944332667943389
('CVX', 'NEM', 'UNH', 'AMZN', 'NEE') : 0.9443342184146928
('XOM', 'NEM', 'UNH', 'AMZN', 'NEE') : 0.9443346091872103
('COP', 'RGLD', 'UNH', 'AMZN', 'NEE') : 0.9443328310190686
```

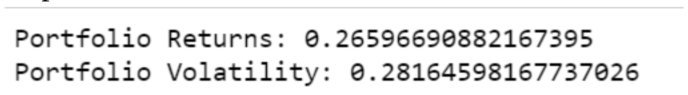
The Sharpe ratios are much smaller in this dataset which indicates that the portfolio may not perform well in times of extreme volatility in the market. However, COVID19 was a very uncertain time in the stock market as a whole with large variability, extreme rise of crypto and meme stocks. Overall, since the Sharpe ratio decreased so drastically during this time an investor may want to aim for a portfolio with less volatility to ensure better returns during this time.

25. Let’s take a look at the expected returns and violated in this dataset



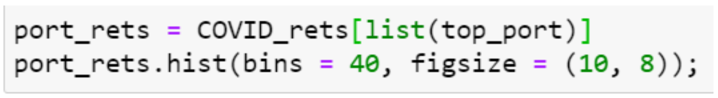
```python
#Now let's take a look closer at how the top portfolio in the training dataset performed

weights = train_port_weight_dict[top_port]
portfolio_df = COVID_TESTING_df[list(top_port)]
COVID_port_ret = port_ret(weights, portfolio_df)
COVID_port_vol = port_vol(weights, portfolio_df)
print("Portfolio Returns: " + str(COVID_port_ret))
print("Portfolio Volatility: " + str(COVID_port_vol))
```

Output:



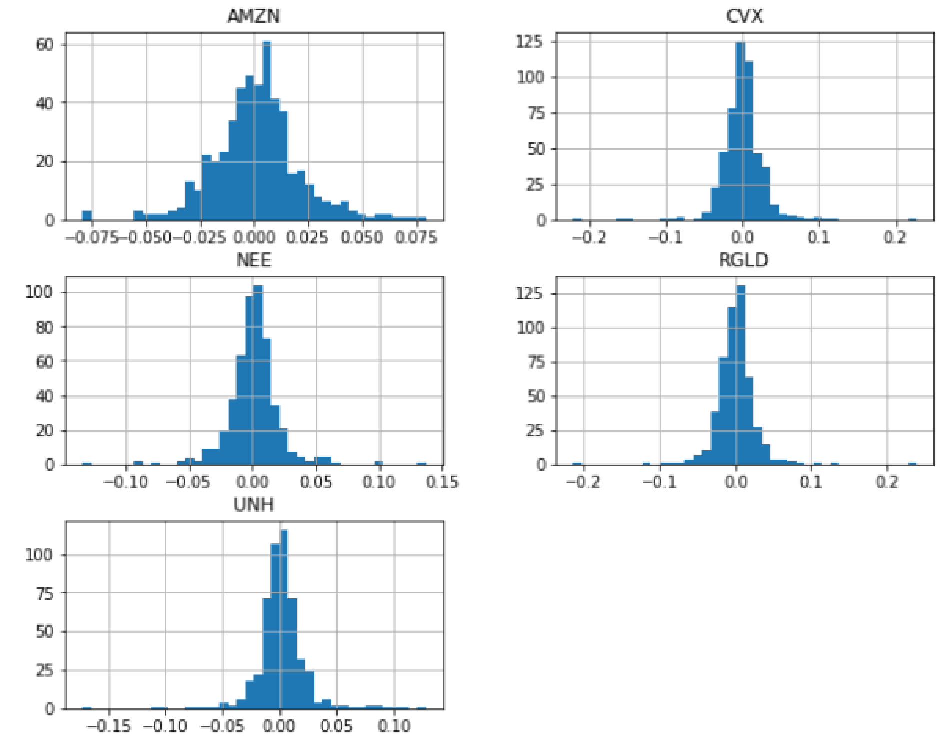
```
Portfolio Returns: 0.26596690882167395
Portfolio Volatility: 0.28164598167737026
```

26. Now let’s see how the individual stock returns are distributed in this dataset compared to the returns in the training dataset



```python
port_rets = COVID_rets[list(top_port)]
port_rets.hist(bins = 40, figsize = (10, 8));
```

Output:



There seems to be far more outliers for all of these distributions than in the previous datasets. This makes sense as during COVID19 there was extremely high volatility and daily returns were sometimes quite high and low.

Conclusion:

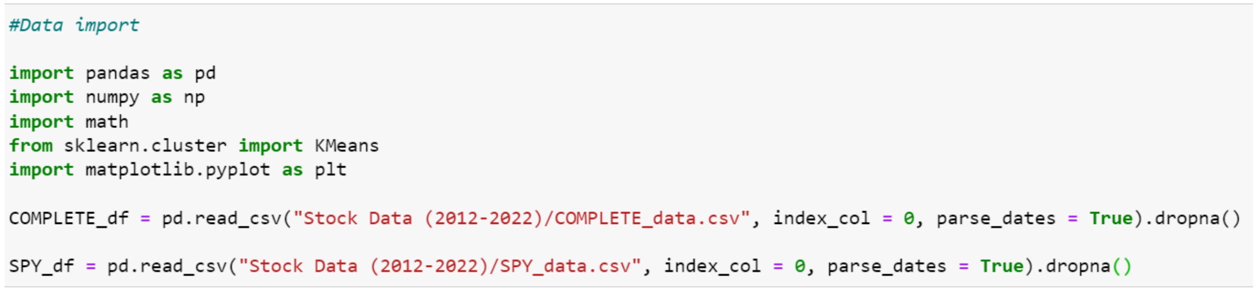
While the portfolios found in the training dataset performed well in the pre-covid dataset it performed relatively poorly in the covid dataset. This means that in times of volatility the portfolio may not perform well as the risk may out-weigh the return. However, an investor can use tools like the Monte Carlo Simulation to determine the portfolio that meets their risk tolerance and adjust the portfolio weighting that way.

Creating a Portfolio using the K-means Clustering Method

In this section we will use the K-means clustering algorithm to create a diverse portfolio with an optimum Sharpe ratio.

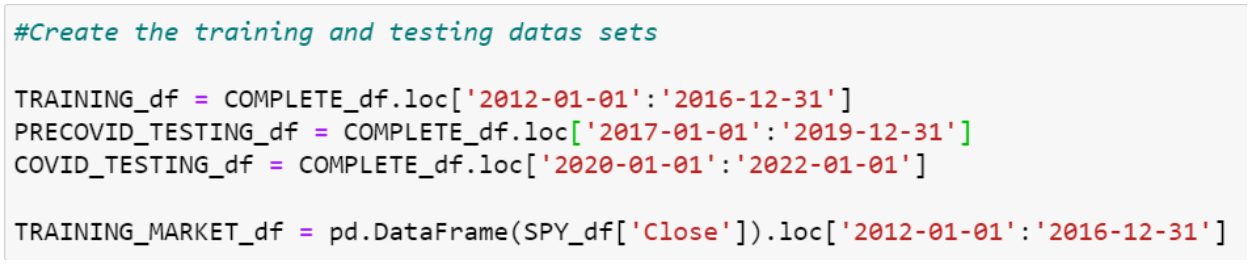
Follow these steps:

1. Import the stock price data and market data from csv files into pandas dataframes using the following code:



```
#Data import

import pandas as pd
import numpy as np
import math
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

COMPLETE_df = pd.read_csv("Stock Data (2012-2022)/COMPLETE_data.csv", index_col = 0, parse_dates = True).dropna()

SPY_df = pd.read_csv("Stock Data (2012-2022)/SPY_data.csv", index_col = 0, parse_dates = True).dropna()
```

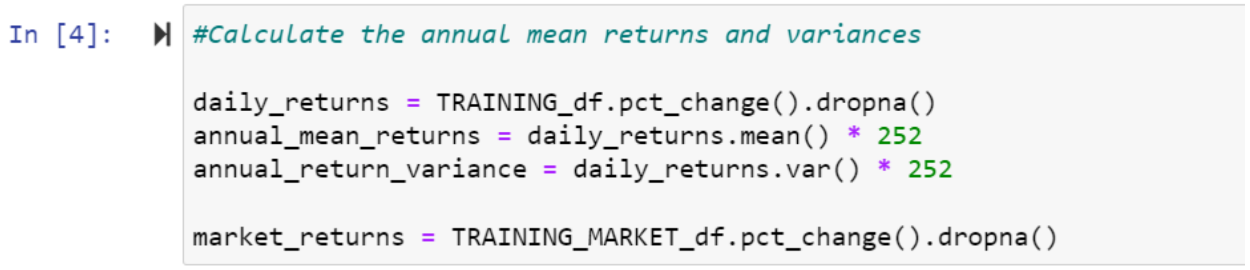
2. Create the training, pre-covid and covid testing datasets



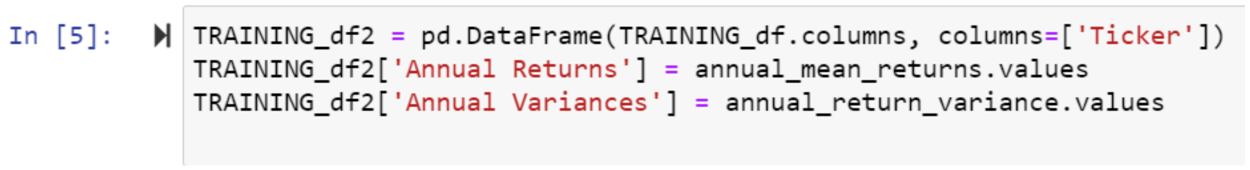
```
#Create the training and testing datas sets

TRAINING_df = COMPLETE_df.loc['2012-01-01':'2016-12-31']
PRECOVID_TESTING_df = COMPLETE_df.loc['2017-01-01':'2019-12-31']
COVID_TESTING_df = COMPLETE_df.loc['2020-01-01':'2022-01-01']

TRAINING_MARKET_df = pd.DataFrame(SPY_df['Close']).loc['2012-01-01':'2016-12-31']
```

CREATING THE PORTFOLIO IN THE TRAINING DATASET

3. Calculate the daily returns, annual mean returns, annual return variances for each stock as well as the market daily returns. Then create a data frame with that contains the stock tickers, annual returns, and annual variances.



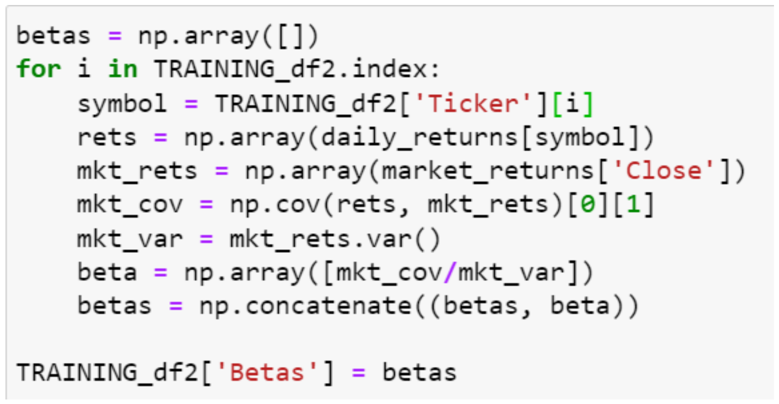
```
In [4]:  #Calculate the annual mean returns and variances

         daily_returns = TRAINING_df.pct_change().dropna()
         annual_mean_returns = daily_returns.mean() * 252
         annual_return_variance = daily_returns.var() * 252

         market_returns = TRAINING_MARKET_df.pct_change().dropna()
```



```
In [5]:  TRAINING_df2 = pd.DataFrame(TRAINING_df.columns, columns=['Ticker'])
         TRAINING_df2['Annual Returns'] = annual_mean_returns.values
         TRAINING_df2['Annual Variances'] = annual_return_variance.values
```

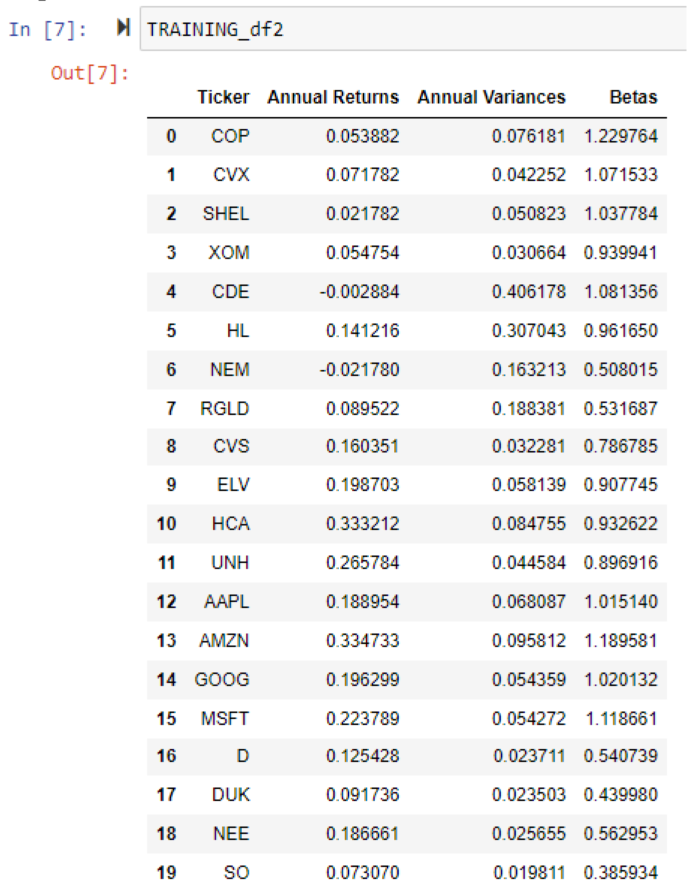
This data frame will be used to create group the data.

4. Calculate the beta for each stock and attach the beta values to the data frame created in the previous step



```
betas = np.array([])
for i in TRAINING_df2.index:
    symbol = TRAINING_df2['Ticker'][i]
    rets = np.array(daily_returns[symbol])
    mkt_rets = np.array(market_returns['Close'])
    mkt_cov = np.cov(rets, mkt_rets)[0][1]
    mkt_var = mkt_rets.var()
    beta = np.array([mkt_cov/mkt_var])
    betas = np.concatenate((betas, beta))

TRAINING_df2['Betas'] = betas
```

Output:

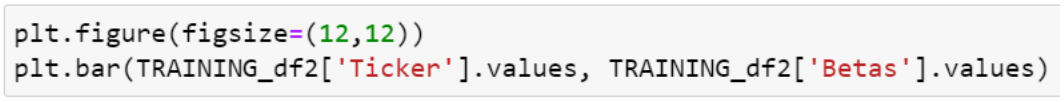


```
In [7]:  TRAINING_df2
Out[7]:
```

	Ticker	Annual Returns	Annual Variances	Betas
0	COP	0.053882	0.076181	1.229764
1	CVX	0.071782	0.042252	1.071533
2	SHEL	0.021782	0.050823	1.037784
3	XOM	0.054754	0.030664	0.939941
4	CDE	-0.002884	0.406178	1.081356
5	HL	0.141216	0.307043	0.961650
6	NEM	-0.021780	0.163213	0.508015
7	RGLD	0.089522	0.188381	0.531687
8	CVS	0.160351	0.032281	0.786785
9	ELV	0.198703	0.058139	0.907745
10	HCA	0.333212	0.084755	0.932622
11	UNH	0.265784	0.044584	0.896916
12	AAPL	0.188954	0.068087	1.015140
13	AMZN	0.334733	0.095812	1.189581
14	GOOG	0.196299	0.054359	1.020132
15	MSFT	0.223789	0.054272	1.118661
16	D	0.125428	0.023711	0.540739
17	DUK	0.091736	0.023503	0.439980
18	NEE	0.186661	0.025655	0.562953
19	SO	0.073070	0.019811	0.385934

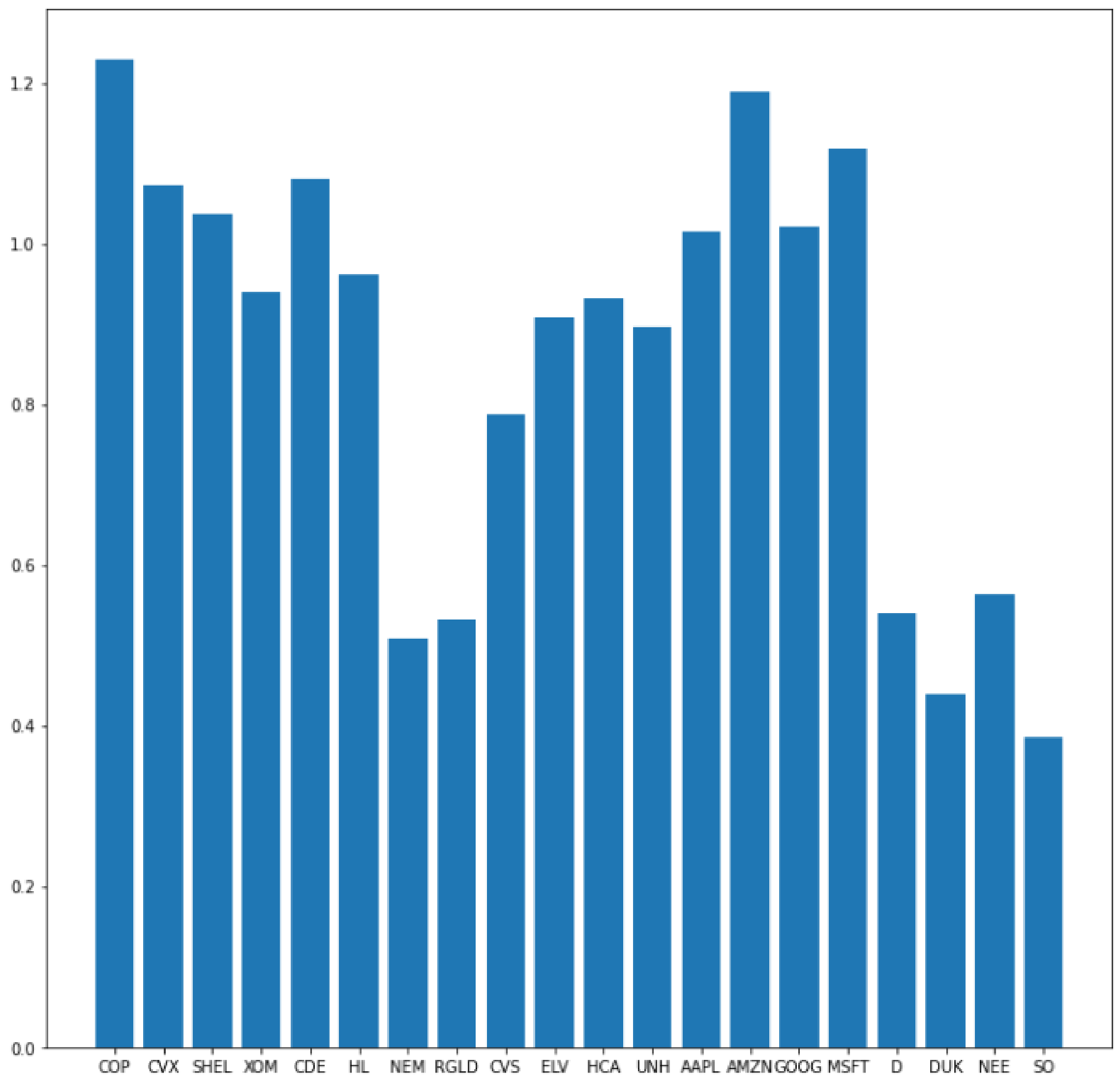
The beta values are a measure of the systematic risk of a stock compared to the market as a whole. The higher the beta value the more systematic risk of the stock. By grouping the stocks based on beta values and choosing a stock from each group we should reduce the overall diversification risk of the portfolio.

5. Let's compare the betas using a bar graph



```
plt.figure(figsize=(12,12))
plt.bar(TRAINING_df2['Ticker'].values, TRAINING_df2['Betas'].values)
```

Output:

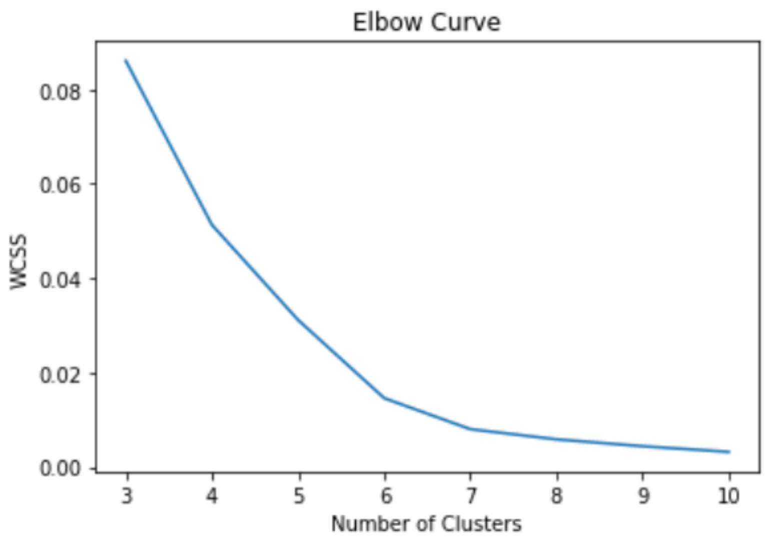


The bar graph allows us to visually see the beta values relative to each other.

6. We can now use the K-means clustering algorithm and pick the number of clusters based on the plot using the elbow method.

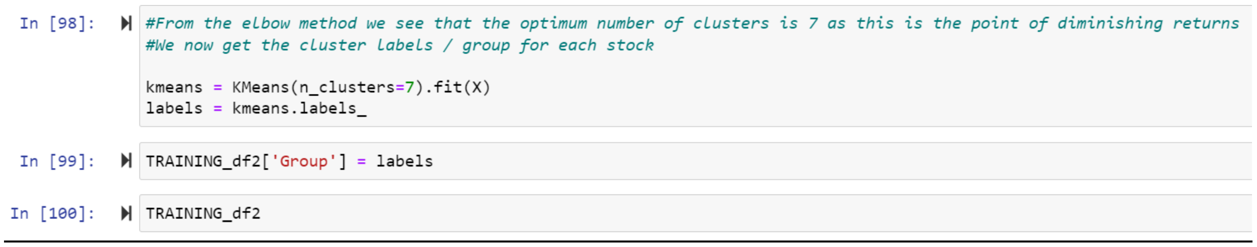
```
#Use the Elbow method to determine the number of clusters to use to group the stocks based on the beta values
#Note the "intertia" is the Within-cluster Sum of Squares (WCSS)

X = TRAINING_df2['Betas'].values.reshape(-1, 1)

inertia_list = []
for k in range(3,11):
    #Create and train the model
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    inertia_list.append(kmeans.inertia_)

#Plot the data
plt.plot(range(3,11), inertia_list)
plt.title('Elbow Curve')
plt.xlabel('Number of Clusters')
plt.xticks(range(3,11))
plt.ylabel('WCSS')
```

<u>Output:</u>



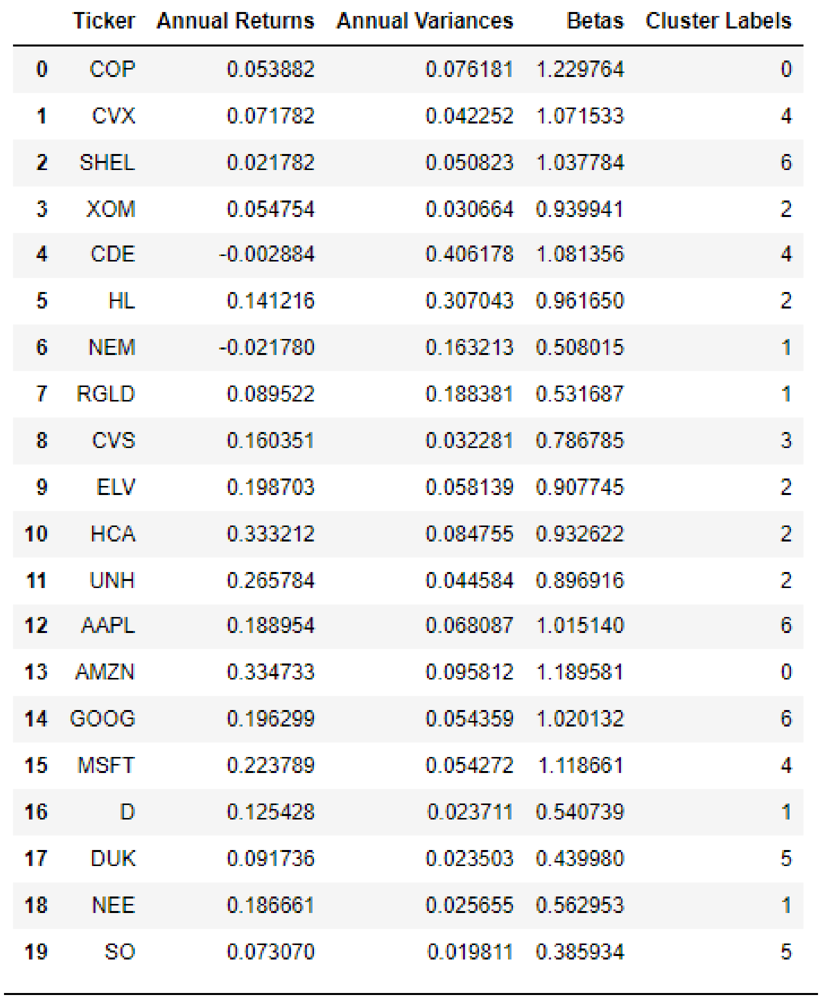
The K-means clustering method works by finding the within-cluster sum of squares (WCSS) for each number of clusters. The elbow method allows us to visually find the optimal number of clusters such that there are diminishing returns for values of WCSS as the number of clusters increases. From the plot we can see that this point is likely either 6 or 7. I have chosen 7 because it allows for a bit less WCSS and may group the stocks better.

7. Add a column to the data frame which has a label for each stock representing a group based on the results of the K-means clustering algorithm.



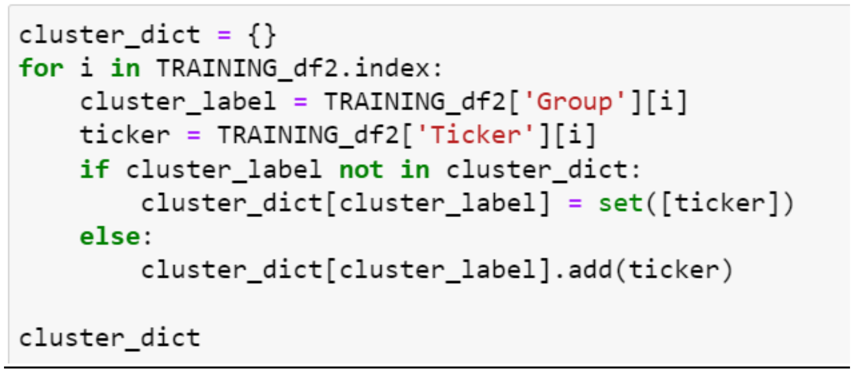
```
In [98]:  #From the elbow method we see that the optimum number of clusters is 7 as this is the point of diminishing returns
          #We now get the cluster labels / group for each stock

          kmeans = KMeans(n_clusters=7).fit(X)
          labels = kmeans.labels_

In [99]:  TRAINING_df2['Group'] = labels

In [100]: TRAINING_df2
```

Output:



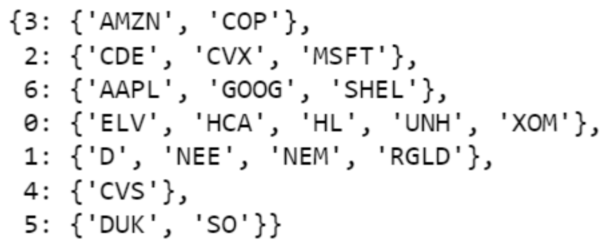
	Ticker	Annual Returns	Annual Variances	Betas	Cluster Labels
0	COP	0.053882	0.076181	1.229764	0
1	CVX	0.071782	0.042252	1.071533	4
2	SHEL	0.021782	0.050823	1.037784	6
3	XOM	0.054754	0.030664	0.939941	2
4	CDE	-0.002884	0.406178	1.081356	4
5	HL	0.141216	0.307043	0.961650	2
6	NEM	-0.021780	0.163213	0.508015	1
7	RGLD	0.089522	0.188381	0.531687	1
8	CVS	0.160351	0.032281	0.786785	3
9	ELV	0.198703	0.058139	0.907745	2
10	HCA	0.333212	0.084755	0.932622	2
11	UNH	0.265784	0.044584	0.896916	2
12	AAPL	0.188954	0.068087	1.015140	6
13	AMZN	0.334733	0.095812	1.189581	0
14	GOOG	0.196299	0.054359	1.020132	6
15	MSFT	0.223789	0.054272	1.118661	4
16	D	0.125428	0.023711	0.540739	1
17	DUK	0.091736	0.023503	0.439980	5
18	NEE	0.186661	0.025655	0.562953	1
19	SO	0.073070	0.019811	0.385934	5

8. Create a dictionary where the keys are the group / cluster labels, and the values are the set of stocks that fall under that label.



```python
cluster_dict = {}
for i in TRAINING_df2.index:
    cluster_label = TRAINING_df2['Group'][i]
    ticker = TRAINING_df2['Ticker'][i]
    if cluster_label not in cluster_dict:
        cluster_dict[cluster_label] = set([ticker])
    else:
        cluster_dict[cluster_label].add(ticker)

cluster_dict
```

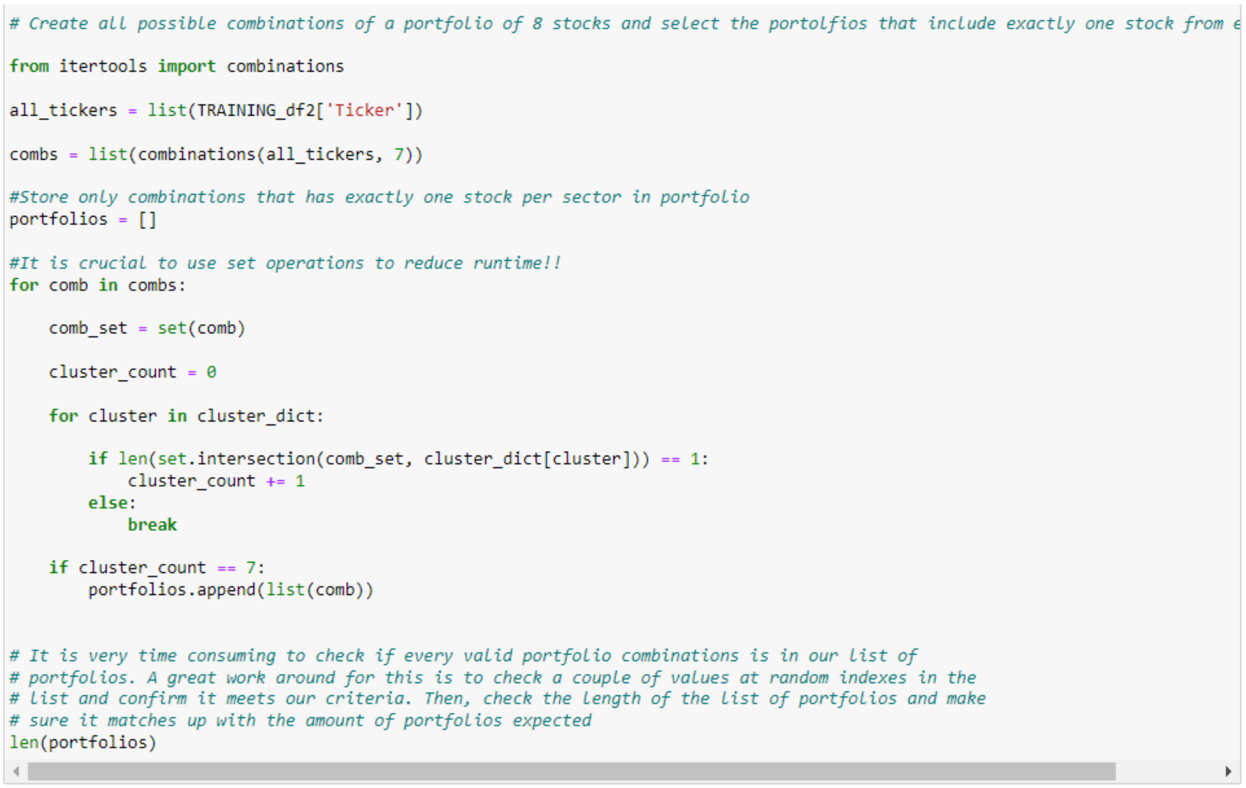
Output:



```
{3: {'AMZN', 'COP'},
 2: {'CDE', 'CVX', 'MSFT'},
 6: {'AAPL', 'GOOG', 'SHEL'},
 0: {'ELV', 'HCA', 'HL', 'UNH', 'XOM'},
 1: {'D', 'NEE', 'NEM', 'RGLD'},
 4: {'CVS'},
 5: {'DUK', 'SO'}}
```

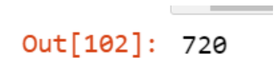
This dictionary will be useful in the upcoming steps to efficiently create portfolios.

9. Create a list of portfolios such that each portfolio includes exactly one stock from each group.



```python
# Create all possible combinations of a portfolio of 8 stocks and select the portolfios that include exactly one stock from e

from itertools import combinations

all_tickers = list(TRAINING_df2['Ticker'])

combs = list(combinations(all_tickers, 7))

#Store only combinations that has exactly one stock per sector in portfolio
portfolios = []

#It is crucial to use set operations to reduce runtime!!
for comb in combs:

    comb_set = set(comb)

    cluster_count = 0

    for cluster in cluster_dict:

        if len(set.intersection(comb_set, cluster_dict[cluster])) == 1:
            cluster_count += 1
        else:
            break

    if cluster_count == 7:
        portfolios.append(list(comb))


# It is very time consuming to check if every valid portfolio combinations is in our list of
# portfolios. A great work around for this is to check a couple of values at random indexes in the
# list and confirm it meets our criteria. Then, check the length of the list of portfolios and make
# sure it matches up with the amount of portfolios expected
len(portfolios)
```

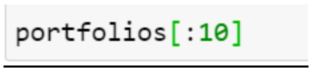
<u>Output:</u>



```
Out[102]:  720
```

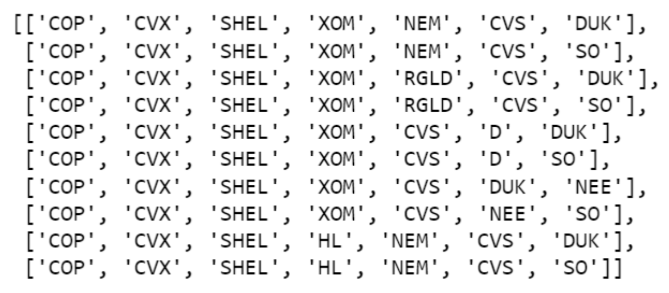
Since there are 3,4,2,5,2,1,3 for groups 0-6 respectively, there should be 3 x 4 x 2 x 5 x 2 x 1 x 3 = 720 portfolios. Since the number of portfolios, we got matches the number we expect we can be assured that all of the valid portfolios are in the list “portfolios”.

10. Let’s take a look at the first 10 portfolios in the list



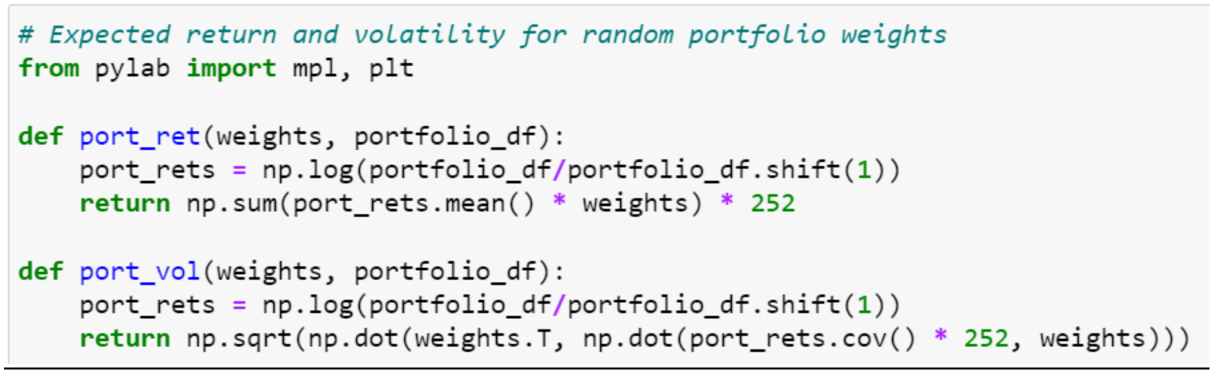
```python
portfolios[:10]
```

<u>Output:</u>

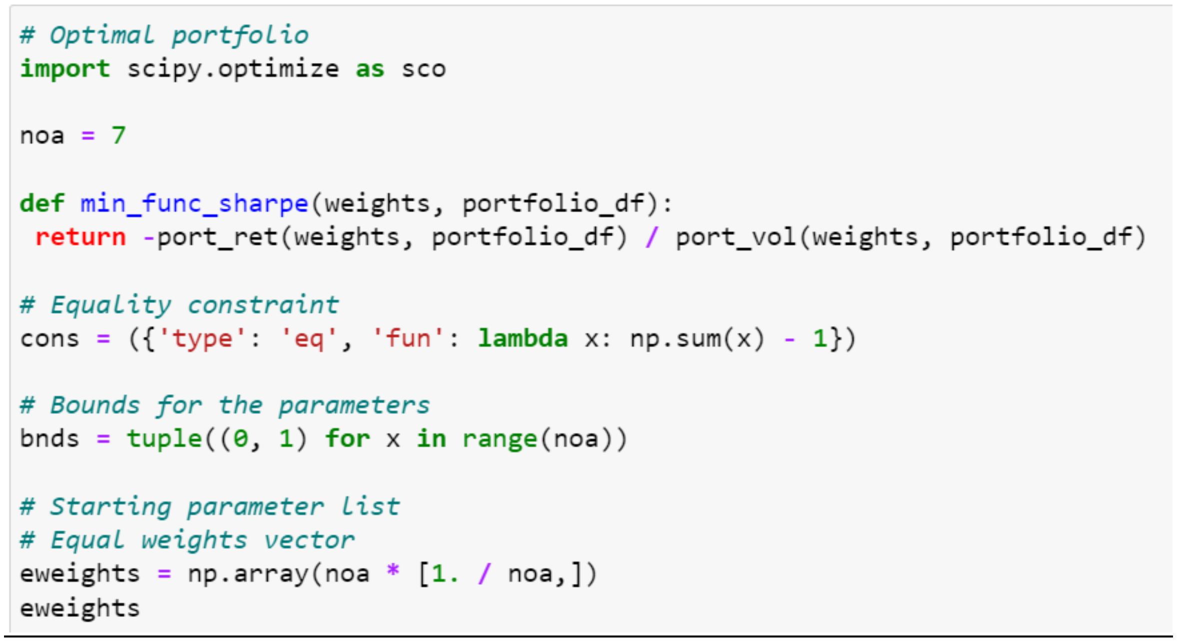


```
[['COP', 'CVX', 'SHEL', 'XOM', 'NEM', 'CVS', 'DUK'],
 ['COP', 'CVX', 'SHEL', 'XOM', 'NEM', 'CVS', 'SO'],
 ['COP', 'CVX', 'SHEL', 'XOM', 'RGLD', 'CVS', 'DUK'],
 ['COP', 'CVX', 'SHEL', 'XOM', 'RGLD', 'CVS', 'SO'],
 ['COP', 'CVX', 'SHEL', 'XOM', 'CVS', 'D', 'DUK'],
 ['COP', 'CVX', 'SHEL', 'XOM', 'CVS', 'D', 'SO'],
 ['COP', 'CVX', 'SHEL', 'XOM', 'CVS', 'DUK', 'NEE'],
 ['COP', 'CVX', 'SHEL', 'XOM', 'CVS', 'NEE', 'SO'],
 ['COP', 'CVX', 'SHEL', 'HL', 'NEM', 'CVS', 'DUK'],
 ['COP', 'CVX', 'SHEL', 'HL', 'NEM', 'CVS', 'SO']]
```

11. Create the functions for the portfolio expected return and volatility as we did previously. As well as the function to minimize the Sharpe ratio of a given portfolio.

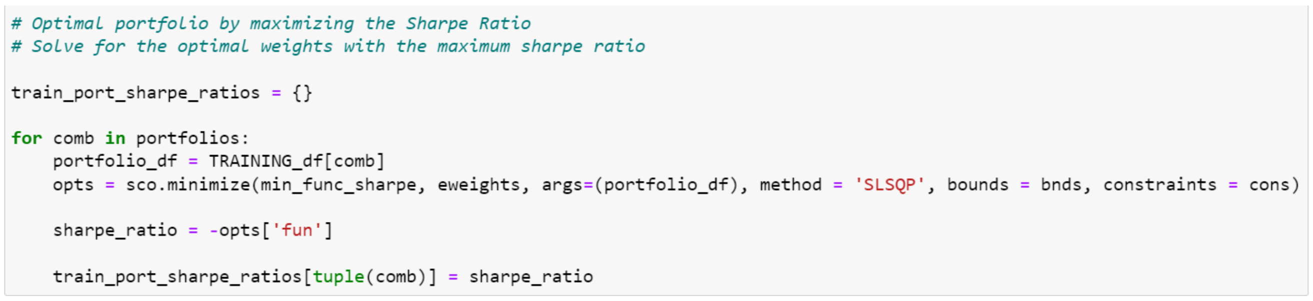


```python
# Expected return and volatility for random portfolio weights
from pylab import mpl, plt

def port_ret(weights, portfolio_df):
    port_rets = np.log(portfolio_df/portfolio_df.shift(1))
    return np.sum(port_rets.mean() * weights) * 252

def port_vol(weights, portfolio_df):
    port_rets = np.log(portfolio_df/portfolio_df.shift(1))
    return np.sqrt(np.dot(weights.T, np.dot(port_rets.cov() * 252, weights)))
```



```python
# Optimal portfolio
import scipy.optimize as sco

noa = 7

def min_func_sharpe(weights, portfolio_df):
 return -port_ret(weights, portfolio_df) / port_vol(weights, portfolio_df)

# Equality constraint
cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

# Bounds for the parameters
bnds = tuple((0, 1) for x in range(noa))

# Starting parameter list
# Equal weights vector
eweights = np.array(noa * [1. / noa,])
eweights
```

12. Create a dictionary as we did previously where the keys are a tuple of the stocks in a portfolio and the values are the maximum Sharpe ratio that portfolio can achieve.



```python
# Optimal portfolio by maximizing the Sharpe Ratio
# Solve for the optimal weights with the maximum sharpe ratio

train_port_sharpe_ratios = {}

for comb in portfolios:
    portfolio_df = TRAINING_df[comb]
    opts = sco.minimize(min_func_sharpe, eweights, args=(portfolio_df), method = 'SLSQP', bounds = bnds, constraints = cons)

    sharpe_ratio = -opts['fun']

    train_port_sharpe_ratios[tuple(comb)] = sharpe_ratio
```

13. Examine the top 5 portfolios with the highest Sharpe ratio

```
#Let's find the top 5 portfolios based on the sharpe ratio based on the training data

train_top_5_SR = sorted(train_port_sharpe_ratios.values(), reverse= True)[:5]
```

```
train_top_5_portfolios = []

for i in range(0,5):
    train_top_5_portfolios.append(tuple(list(train_port_sharpe_ratios.keys())[list(train_port_sharpe_ratios.values()).
                                                                                index(train_top_5_SR[i])]))
```

```
for portfolio in train_top_5_portfolios:
    sharpe_ratio = train_port_sharpe_ratios[portfolio]
    print(str(portfolio) + " : " + str(sharpe_ratio))
```

<u>Output:</u>

```
('CVS', 'UNH', 'AAPL', 'AMZN', 'MSFT', 'NEE', 'SO') : 1.549573913288882
('CVS', 'UNH', 'AAPL', 'AMZN', 'MSFT', 'DUK', 'NEE') : 1.549573618770525
('CVS', 'UNH', 'AMZN', 'GOOG', 'MSFT', 'NEE', 'SO') : 1.5469583256601187
('SHEL', 'CVS', 'UNH', 'AMZN', 'MSFT', 'NEE', 'SO') : 1.5469582844458225
('CVS', 'UNH', 'AMZN', 'GOOG', 'MSFT', 'DUK', 'NEE') : 1.5469580824006828
```

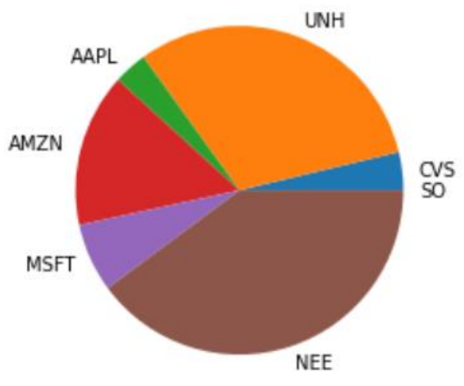
Again, we have very similar assets in each of the top 5 portfolios

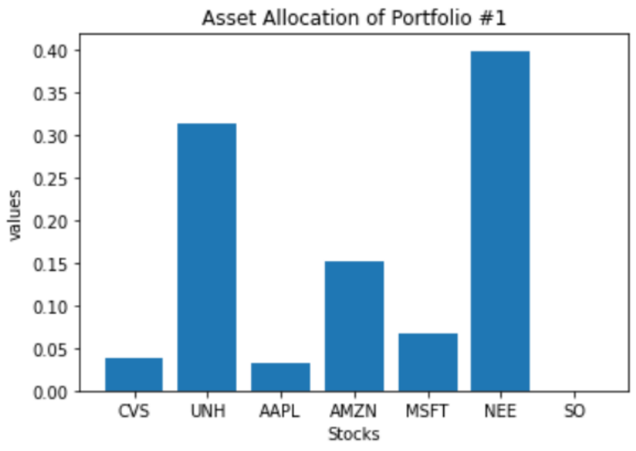
14. Create a dictionary as we did previously where the keys are tuples of stocks representing a portfolio and the keys are the portfolio weights.

```
train_port_weight_dict = {}

for portfolio in train_top_5_portfolios:

    portfolio_df = TRAINING_df[list(portfolio)]

    opts = sco.minimize(min_func_sharpe, eweights, args=(portfolio_df), method = 'SLSQP', bounds = bnds, constraints = cons)

    weights = opts['x']

    train_port_weight_dict[tuple(portfolio)] = weights
```

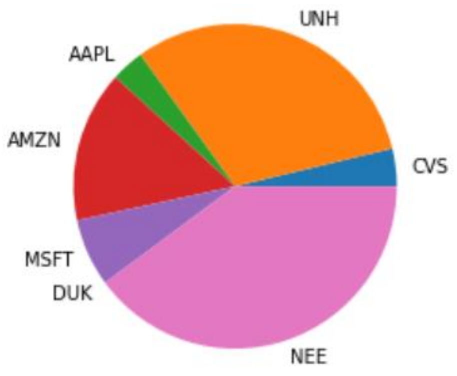
15. Let's visually examine the asset allocation of each of the top 5 portfolios with the highest Sharpe ratios

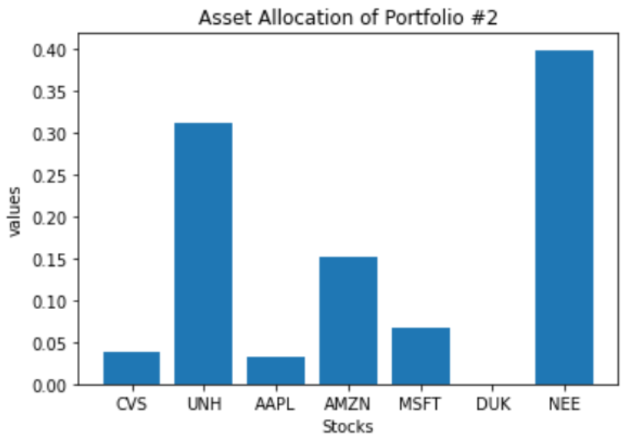
Portfolio #1:



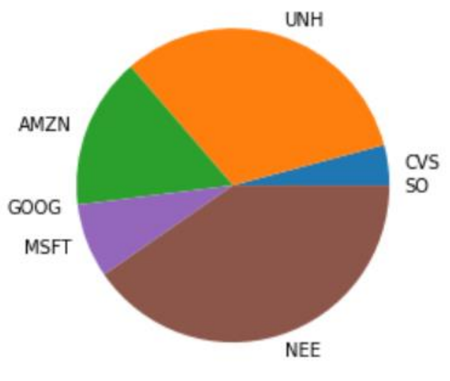


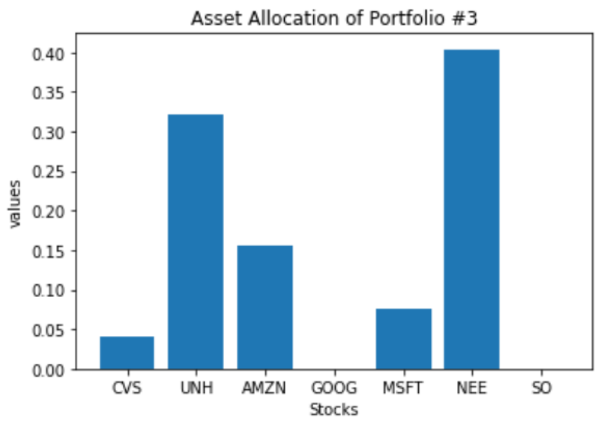
Portfolio #2:



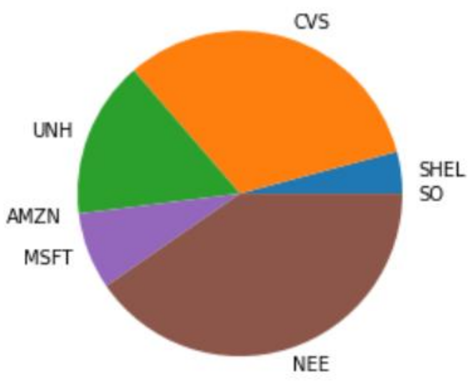


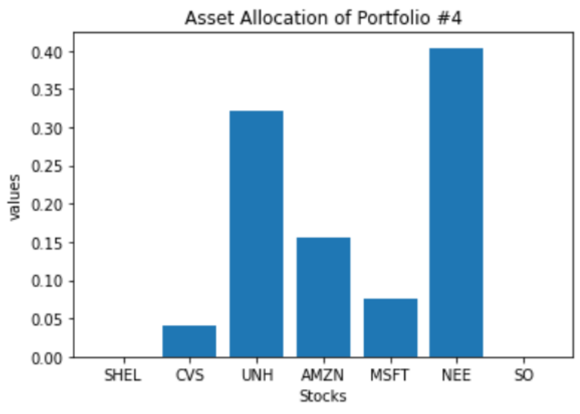
Portfolio #3:



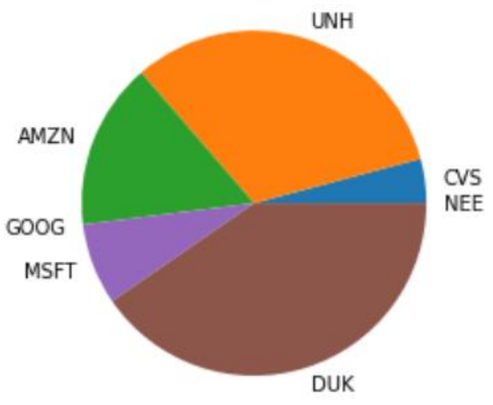


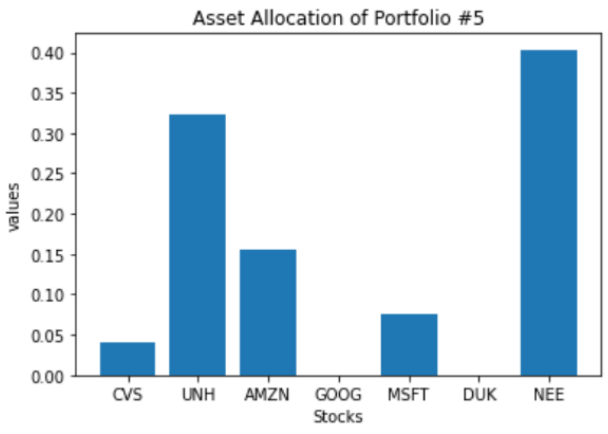
Portfolio #4:





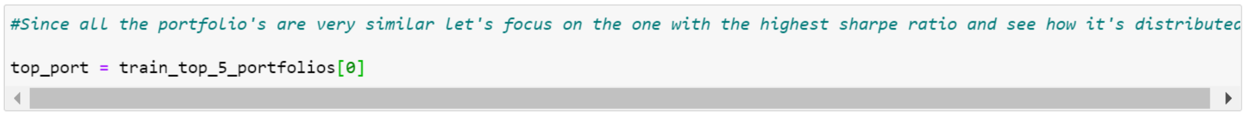
Portfolio #5:





All these portfolios are very similar and are essentially the same.

16. Since all the portfolios are very similar let's closely examine the top portfolio (highest Sharpe ratio)



```
#Since all the portfolio's are very similar let's focus on the one with the highest sharpe ratio and see how it's distributed

top_port = train_top_5_portfolios[0]
```

```
weights = train_port_weight_dict[top_port]
portfolio_df = TRAINING_df[list(top_port)]
TRAINING_port_ret = port_ret(weights, portfolio_df)
TRAINING_port_vol = port_vol(weights, portfolio_df)
print("Portfolio Returns: " + str(TRAINING_port_ret))
print("Portfolio Volatility: " + str(TRAINING_port_vol))
```

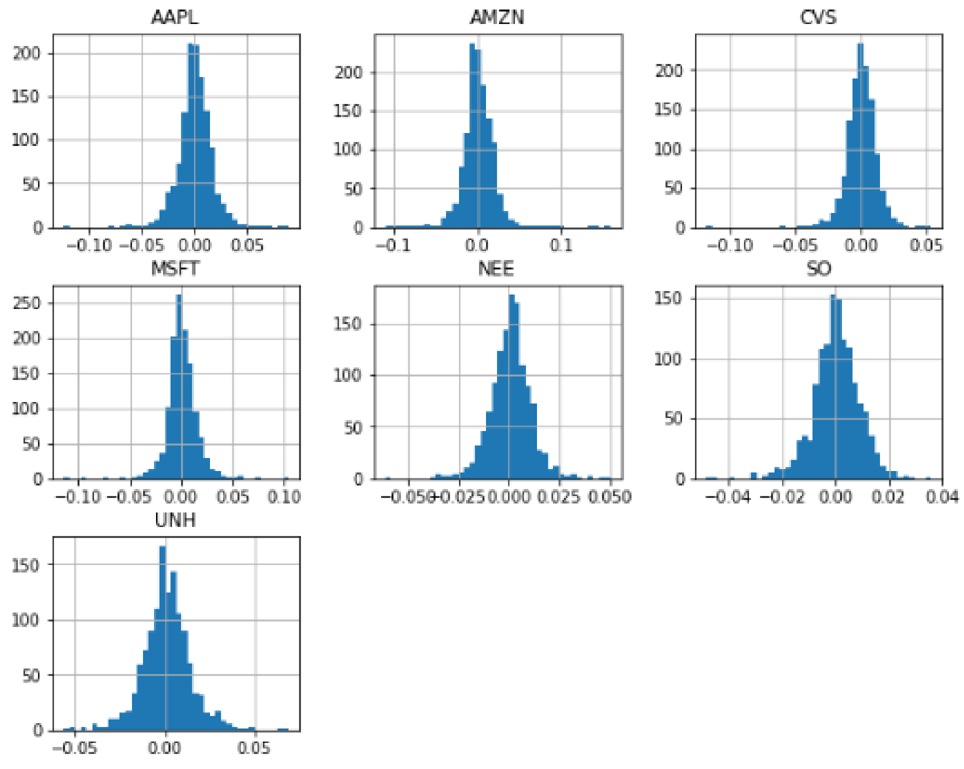
Output:

```
Portfolio Returns: 0.21248279285555638
Portfolio Volatility: 0.13712336729041458
```

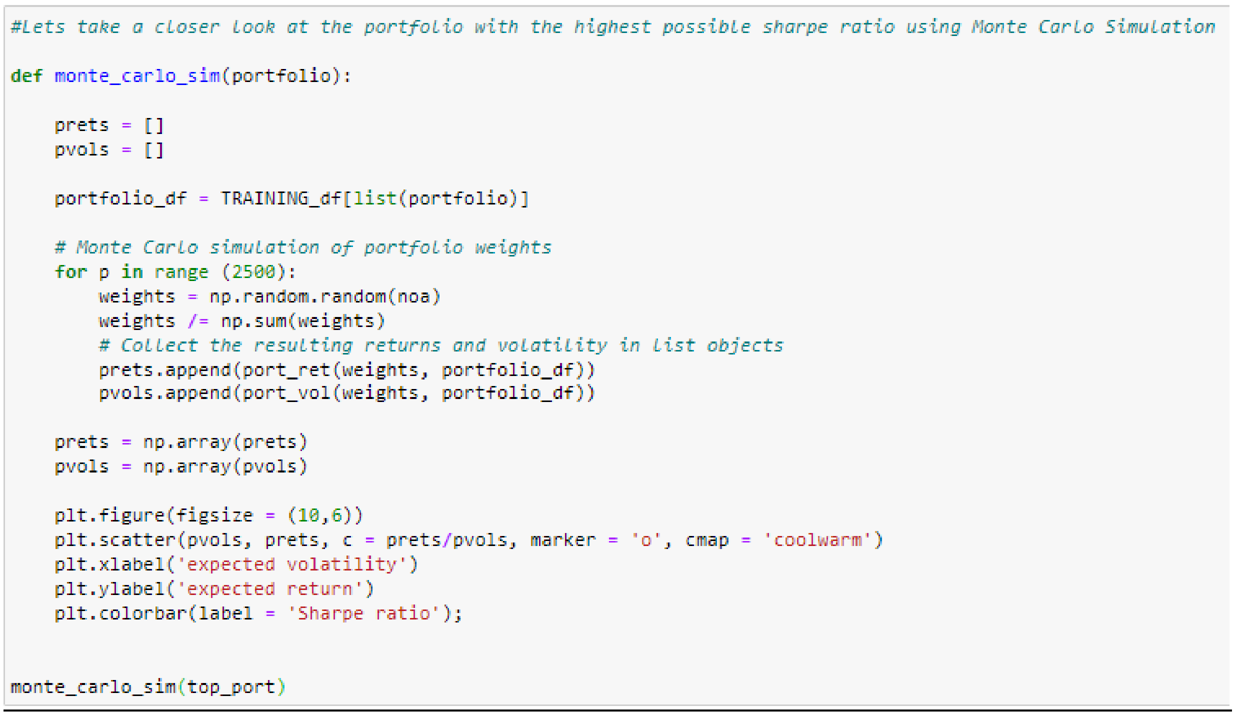
17. Let’s look at how each stock in the portfolio is distributed

```
port_rets = daily_returns[list(top_port)]
port_rets.hist(bins = 40, figsize = (10, 8));
```

Output:

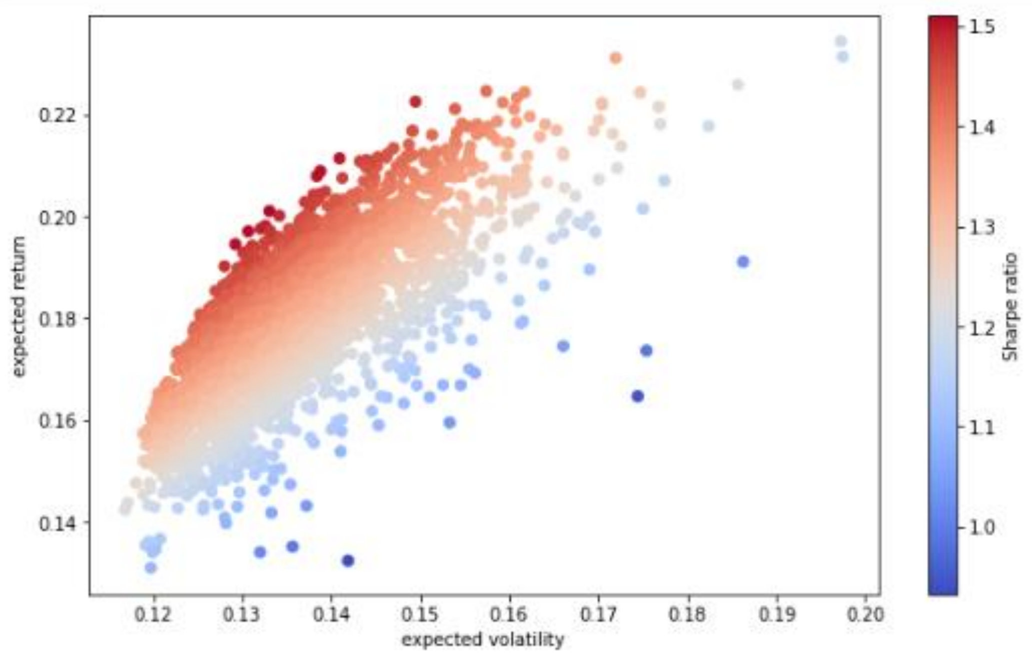


18. Examine visually the expected returns and volatility of the top portfolio with different weights using a Monte Carlo Simulation.



```
#Lets take a closer look at the portfolio with the highest possible sharpe ratio using Monte Carlo Simulation

def monte_carlo_sim(portfolio):

    prets = []
    pvols = []

    portfolio_df = TRAINING_df[list(portfolio)]

    # Monte Carlo simulation of portfolio weights
    for p in range (2500):
        weights = np.random.random(noa)
        weights /= np.sum(weights)
        # Collect the resulting returns and volatility in list objects
        prets.append(port_ret(weights, portfolio_df))
        pvols.append(port_vol(weights, portfolio_df))

    prets = np.array(prets)
    pvols = np.array(pvols)

    plt.figure(figsize = (10,6))
    plt.scatter(pvols, prets, c = prets/pvols, marker = 'o', cmap = 'coolwarm')
    plt.xlabel('expected volatility')
    plt.ylabel('expected return')
    plt.colorbar(label = 'Sharpe ratio');


monte_carlo_sim(top_port)
```

Output:



BACKTESTING THE PORTFOLIO IN THE PRE-COVID DATASET

19. Let’s take a quick look at the daily returns of the stocks in the pre-covid testing dataset.

```
#Let's take a quick look at the returns in the PRE-COVID testing dataset

PRECOVID_rets = PRECOVID_TESTING_df.pct_change().dropna()
PRECOVID_rets.head(10)
```

Output:

Date	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAPL	AMZN
2017-01-04	0.009287	-0.000254	0.010141	-0.011002	0.018481	0.012727	0.009529	0.008852	-0.007468	-0.001326	0.035858	0.002849	-0.001119	0.004657
2017-01-05	-0.002937	-0.004329	0.008785	-0.014907	0.133064	0.046679	0.046053	0.033713	0.020941	0.010626	-0.004149	0.001668	0.005086	0.030732
2017-01-06	-0.003731	-0.004006	-0.008530	-0.000565	-0.063167	-0.027444	-0.031447	-0.024423	0.009580	-0.001522	-0.002473	0.001418	0.011148	0.019912
2017-01-09	-0.021285	-0.008559	-0.021330	-0.016497	0.004748	-0.001764	-0.001694	0.005190	-0.006083	-0.010807	0.030018	-0.002833	0.009159	0.001168
2017-01-10	0.000000	-0.007597	-0.004213	-0.012753	0.036862	0.028269	-0.007919	0.017312	0.011873	0.024862	-0.008236	-0.002284	0.001009	-0.001280
2017-01-11	0.031413	0.008438	0.020232	0.010241	-0.000912	-0.005155	-0.020240	-0.004777	0.001210	-0.003485	0.013287	0.001919	0.005373	0.003920
2017-01-12	-0.020890	0.001984	0.004327	-0.005414	-0.006387	0.003454	0.002910	-0.001500	-0.002054	0.016183	-0.002396	0.002903	-0.004175	0.018297
2017-01-13	0.008774	0.001894	-0.004129	0.000116	0.045914	0.027539	0.001740	0.009163	-0.007506	0.002294	0.003539	-0.003449	-0.001761	0.004302
2017-01-17	-0.008697	-0.000859	0.001082	0.011697	0.018437	0.020101	0.017666	0.017862	0.023664	-0.011984	0.019270	-0.007046	0.008065	-0.009080
2017-01-18	0.002393	-0.002924	-0.012604	-0.012363	-0.011207	-0.004926	0.003415	-0.003802	-0.004528	0.003202	-0.014828	-0.018175	-0.000083	-0.002766

20. Now let’s take a look at the Sharpe ratios of the top portfolios in the pre-covid testing dataset

```
#Now let's take a look at how the top portfolios found using the training dataset performed in the PRE-COVID testing dataset

for portfolio in train_top_5_portfolios:

    weights = train_port_weight_dict[portfolio]
    portfolio_df = PRECOVID_TESTING_df[list(portfolio)]
    sharpe_ratio = -min_func_sharpe(weights, portfolio_df)

    print(str(portfolio) + " : " + str(sharpe_ratio))
```

Output:

```
('CVS', 'UNH', 'AAPL', 'AMZN', 'MSFT', 'NEE', 'SO') : 1.927663382902878
('CVS', 'UNH', 'AAPL', 'AMZN', 'MSFT', 'DUK', 'NEE') : 1.927857134817072
('CVS', 'UNH', 'AMZN', 'GOOG', 'MSFT', 'NEE', 'SO') : 1.8993387983903505
('SHEL', 'CVS', 'UNH', 'AMZN', 'MSFT', 'NEE', 'SO') : 1.8995667892847548
('CVS', 'UNH', 'AMZN', 'GOOG', 'MSFT', 'DUK', 'NEE') : 1.8989248528103253
```

Again, we see that the portfolio Sharpe ratios have increased significantly in the pre-covid dataset

21. Let’s take a closer look at how the top portfolio performed in pre-covid dataset

```
#Now let's take a look closer at how the top portfolio in the training dataset performed

weights = train_port_weight_dict[top_port]
portfolio_df = PRECOVID_TESTING_df[list(top_port)]
PRECOVID_port_ret = port_ret(weights, portfolio_df)
PRECOVID_port_vol = port_vol(weights, portfolio_df)
print("Portfolio Returns: " + str(PRECOVID_port_ret))
print("Portfolio Volatility: " + str(PRECOVID_port_vol))
```

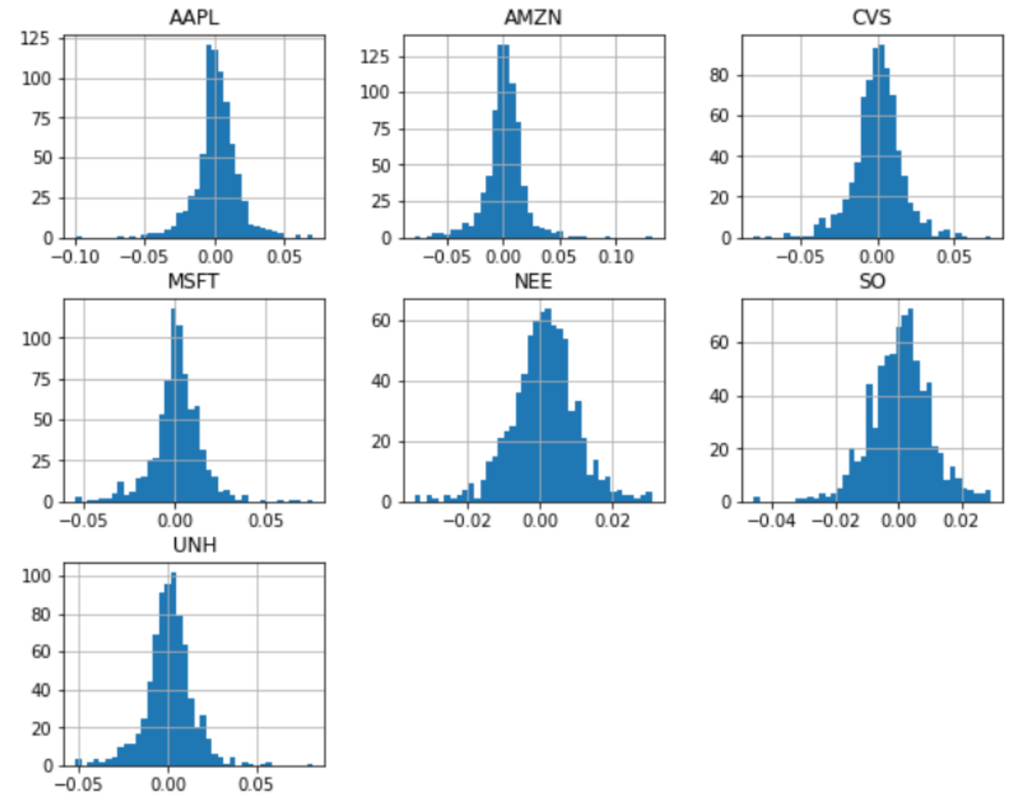
Output:

```
Portfolio Returns: 0.2512289234629903
Portfolio Volatility: 0.13032821274255021
```

22. Let’s look at how each stock in the portfolio is distributed in this dataset

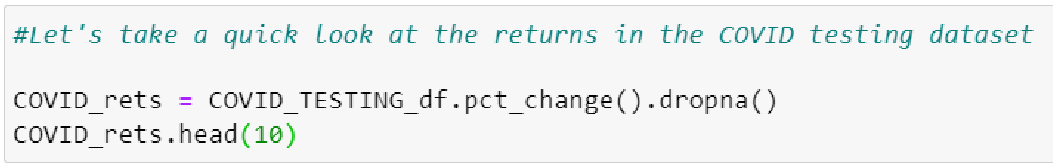
```
port_rets = PRECOVID_rets[list(top_port)]
port_rets.hist(bins = 40, figsize = (10, 8));
```

Output:



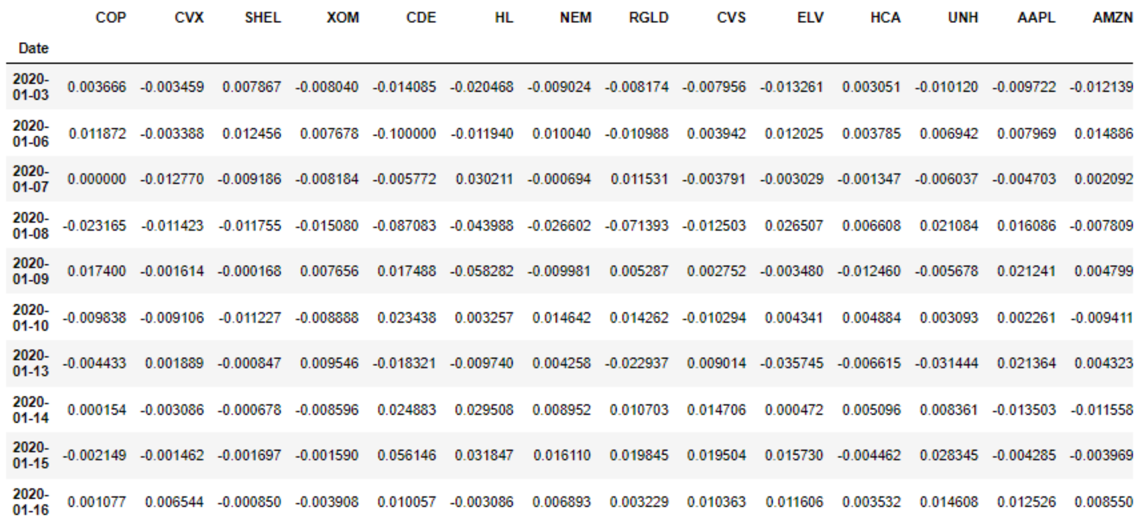
BACKTESTING THE PORTFOLIO IN THE COVID DATASET

23. Let’s take a quick look at the daily returns of the stocks in the covid testing dataset.



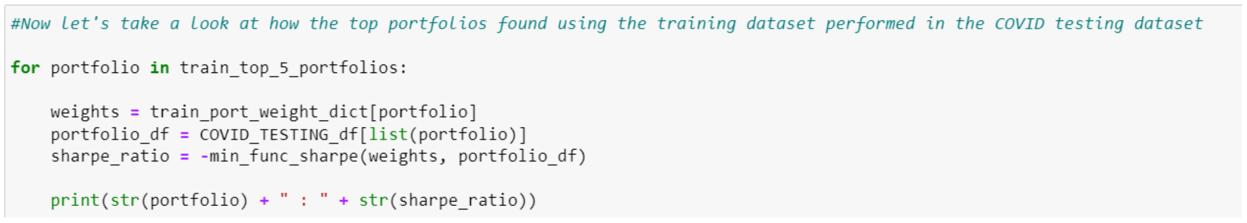
```python
#Let's take a quick look at the returns in the COVID testing dataset

COVID_rets = COVID_TESTING_df.pct_change().dropna()
COVID_rets.head(10)
```

Output:



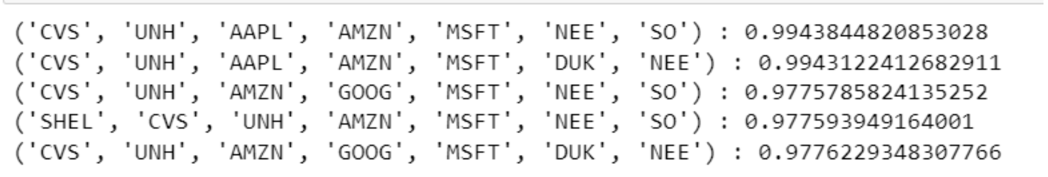
Date	COP	CVX	SHEL	XOM	CDE	HL	NEM	RGLD	CVS	ELV	HCA	UNH	AAPL	AMZN
2020-01-03	0.003666	-0.003459	0.007867	-0.008040	-0.014085	-0.020468	-0.009024	-0.008174	-0.007956	-0.013261	0.003051	-0.010120	-0.009722	-0.012139
2020-01-06	0.011872	-0.003388	0.012456	0.007678	-0.100000	-0.011940	0.010040	-0.010988	0.003942	0.012025	0.003785	0.006942	0.007969	0.014886
2020-01-07	0.000000	-0.012770	-0.009186	-0.008184	-0.005772	0.030211	-0.000694	0.011531	-0.003791	-0.003029	-0.001347	-0.006037	-0.004703	0.002092
2020-01-08	-0.023165	-0.011423	-0.011755	-0.015080	-0.087083	-0.043988	-0.026602	-0.071393	-0.012503	0.026507	0.006608	0.021084	0.016086	-0.007809
2020-01-09	0.017400	-0.001614	-0.000168	0.007656	0.017488	-0.058282	-0.009981	0.005287	0.002752	-0.003480	-0.012460	-0.005678	0.021241	0.004799
2020-01-10	-0.009838	-0.009106	-0.011227	-0.008888	0.023438	0.003257	0.014642	0.014262	-0.010294	0.004341	0.004884	0.003093	0.002261	-0.009411
2020-01-13	-0.004433	0.001889	-0.000847	0.009546	-0.018321	-0.009740	0.004258	-0.022937	0.009014	-0.035745	-0.006615	-0.031444	0.021364	0.004323
2020-01-14	0.000154	-0.003086	-0.000678	-0.008596	0.024883	0.029508	0.008952	0.010703	0.014706	0.000472	0.005096	0.008361	-0.013503	-0.011558
2020-01-15	-0.002149	-0.001462	-0.001697	-0.001590	0.056146	0.031847	0.016110	0.019845	0.019504	0.015730	-0.004462	0.028345	-0.004285	-0.003969
2020-01-16	0.001077	0.006544	-0.000850	-0.003908	0.010057	-0.003086	0.006893	0.003229	0.010363	0.011606	0.003532	0.014608	0.012526	0.008550

24. Now let’s take a look at the Sharpe ratios of the top portfolios in the covid testing dataset



```python
#Now let's take a look at how the top portfolios found using the training dataset performed in the COVID testing dataset

for portfolio in train_top_5_portfolios:

    weights = train_port_weight_dict[portfolio]
    portfolio_df = COVID_TESTING_df[list(portfolio)]
    sharpe_ratio = -min_func_sharpe(weights, portfolio_df)

    print(str(portfolio) + " : " + str(sharpe_ratio))
```

Output:



```
('CVS', 'UNH', 'AAPL', 'AMZN', 'MSFT', 'NEE', 'SO') : 0.9943844820853028
('CVS', 'UNH', 'AAPL', 'AMZN', 'MSFT', 'DUK', 'NEE') : 0.9943122412682911
('CVS', 'UNH', 'AMZN', 'GOOG', 'MSFT', 'NEE', 'SO') : 0.9775785824135252
('SHEL', 'CVS', 'UNH', 'AMZN', 'MSFT', 'NEE', 'SO') : 0.977593949164001
('CVS', 'UNH', 'AMZN', 'GOOG', 'MSFT', 'DUK', 'NEE') : 0.9776229348307766
```

Once again the portfolio performed far worse during the covid dataset.

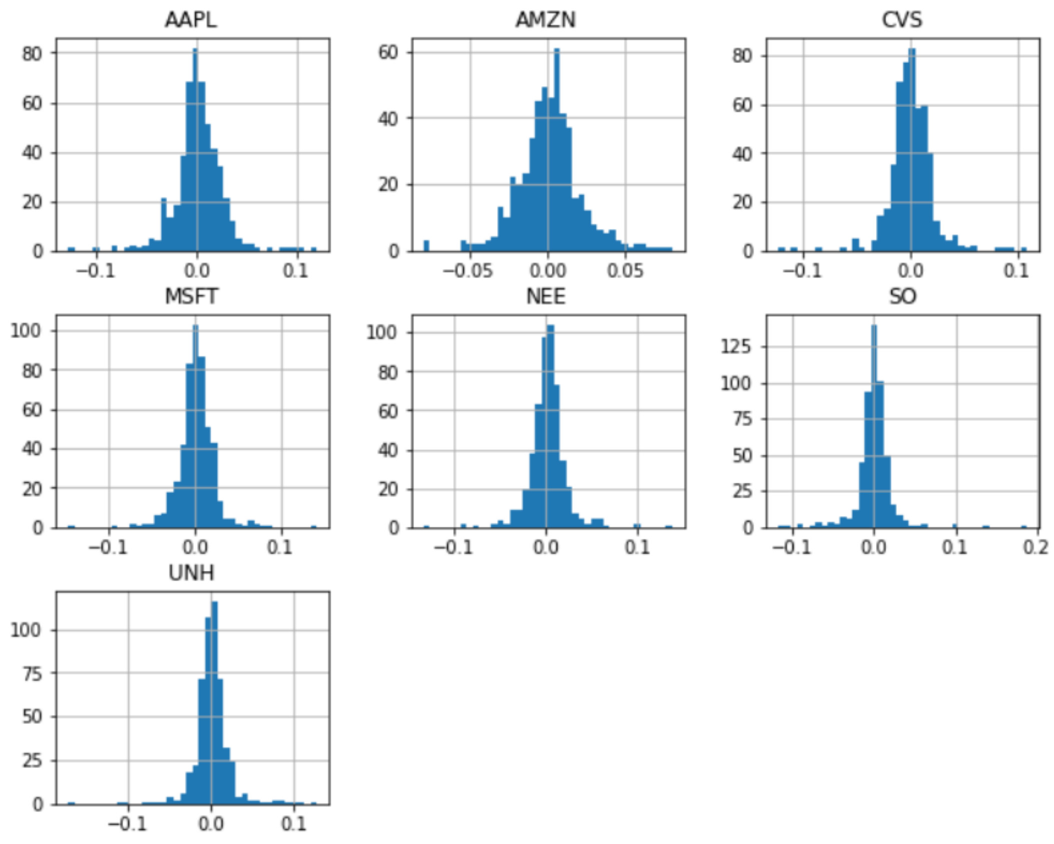
25. Let's take a look closer at how the top portfolio in this dataset

```
#Now let's take a look closer at how the top portfolio in the training dataset performed

weights = train_port_weight_dict[top_port]
portfolio_df = COVID_TESTING_df[list(top_port)]
COVID_port_ret = port_ret(weights, portfolio_df)
COVID_port_vol = port_vol(weights, portfolio_df)
print("Portfolio Returns: " + str(COVID_port_ret))
print("Portfolio Volatility: " + str(COVID_port_vol))
```

Output:

```
Portfolio Returns: 0.2761959038778191
Portfolio Volatility: 0.27775564568205496
```

26. Let’s look at how each stock in the portfolio is distributed in this dataset



Again, there seems to be many more outliers than previously.

Conclusions

The portfolios found using Sector Grouping Process and K-means Clustering Method both performed very similarly even though the asset allocation is different. Both methods seem to agree that to have a diverse portfolio with a high Sharpe ratio, the portfolio must include the stocks UNH, AMZN, and NEE. What I find particularly interesting is that the portfolio found using the K-means Clustering Method had three stocks in the tech sector. This is interesting because the clustering algorithm maybe telling us that despite the fact that the companies are in the same sector, they move differently enough with the market that all three can be included and still have a somewhat diverse portfolio. An interesting project idea could be interesting is to create a portfolio using the K-means Clustering Method but with 100-500 of the biggest companies in the US regardless of the industry sector. This may lead to more interesting results as that could allow us to group the stocks and find different combinations in ways we had never thought of as humans. The unsupervised learning aspect of the algorithm allows us to come to findings that we might have never imagined but this comes with limitations as each portfolio that is suggested by the method should be closely examined by the investor. In particular, the investor should pay close attention to how the portfolio performs when the market is the most volatile, we saw this with the COVID19 dataset. The portfolio performed much differently throughout this dataset, and this may influence to investor to settle for a portfolio with lower Sharpe ratio depending on the risk tolerance of the investor. Overall, both methods seem to have come to a similar conclusion which means that for the most part, the optimal portfolio should include UNH, AMZN, and NEE. However, the investor can adjust the weights or incorporate periodic rebalancing to reduce the volatility. In conclusion, both methods could be a good place to start when deciding what stocks to include in their portfolio, however, the investor much examine each portfolio closely and assess the level of risk they are comfortable with taking on. Furthermore, extensive research should be done for each potential company that the investor is interesting in investing in.