MATH 628 FINAL PROJECT

PCA Analysis

Chunlin Shi

Noah Collins

1. Abstract

In this project, we found 1-year daily trading data for the following stocks: AAL, AMD, AVGO, BAC, BRK-A, CVX, DAL, EOG, INTC, JPM, LUV, NVDA, SLB, UAL, WFC, XOM. We repeated the analysis of Sections 2.1-2.2 of the paper by Avellaneda and Lee. We then discussed the signs of the second and the third eigenvectors partition stocks into stocks and discovered that stocks in the same group have the same sign on the second and third eigenvectors, which corresponds to the grouping of the stocks by industries.

2. Introduction

Principal Component Analysis is a powerful statistical tool that allows us to explore multidimensional relationships within datasets. In our paper's case, our dataset was 15 stocks and their trading information over a year. PCA analysis helps capture the most significant sources of variation, which in the case of stocks, unveils patterns and correlations that may not be apparent to the naked eye. We seek to delineate groups of stocks that share common trends, behaviors, or underlying factors and to compare these results with official industry groupings to demonstrate the efficacy of PCA as a tool for financial professionals, researchers and investors.

3. Data Collection Methodology

To conduct this research, we gathered data from the website of Center for Research in Security Prices (CRSP) of Wharton Research Data Services. The time range of the daily trading data is from December 31, 2021, to December 30, 2022. We collected following variables for our study:

Names Date: Date of trading.

Ticker Symbol: Ticker for stocks accordingly.

North American Industry Classification System: Code used to proxy belonging industry of the stock.

Price or Bid/Ask Average, Shares Outstanding: Variables used to construct market cap weighted portfolios.

Returns without Dividends: Daily return data used for PCA analysis and eigen portfolio construction.

4. Implementation of the Analysis in the Paper by Avellaneda and Lee.

a) Section 2.1

According to the Paper, the daily return is calculated by:

$$R_{ik} = \frac{S_{i(t_0-(k-1)\Delta t)} - S_{i(t_0-k\Delta t)}}{S_{i(t_0-k\Delta t)}}, \; k = 1, \cdots, M, \; i = 1, \cdots, N$$

Where $M = 252$ and $N = 16$, where $S_{it}$is the price of stock i at time t adjusted for dividends and $\Delta t = \frac{1}{252}$

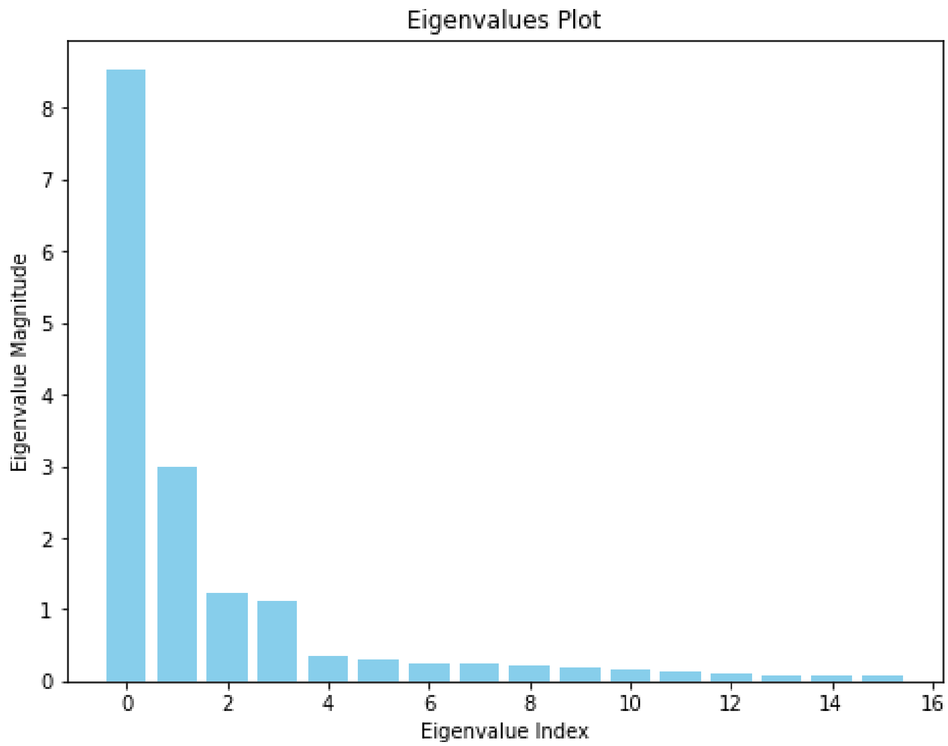
The return was standardized as follows:

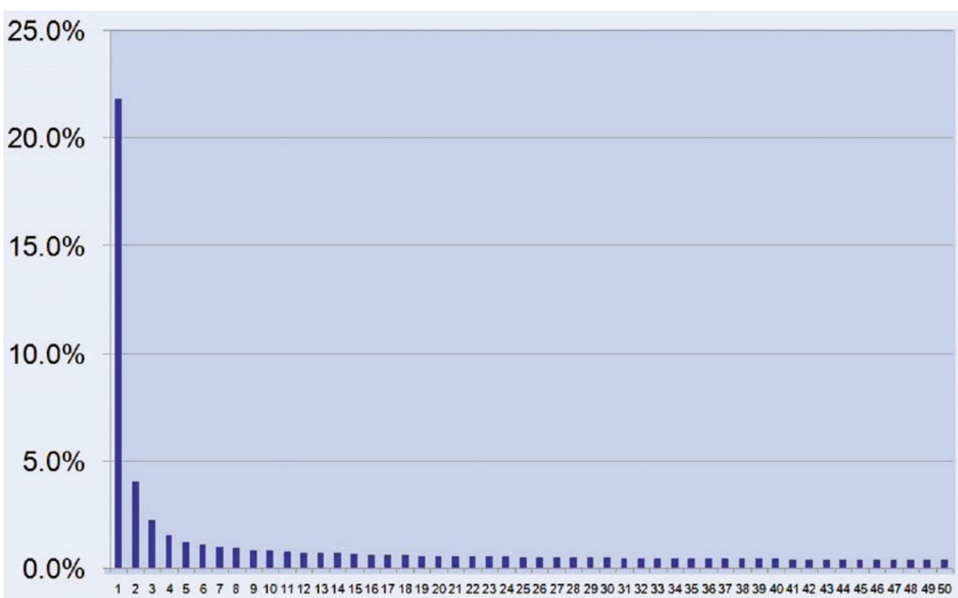
$$Y_{ik} = \frac{R_{ik} - \overline{R_i}}{\overline{\sigma_i}}$$

Where,

$$\overline{R_i} = \frac{1}{M} \sum_{k=1}^{M} R_{ik}$$

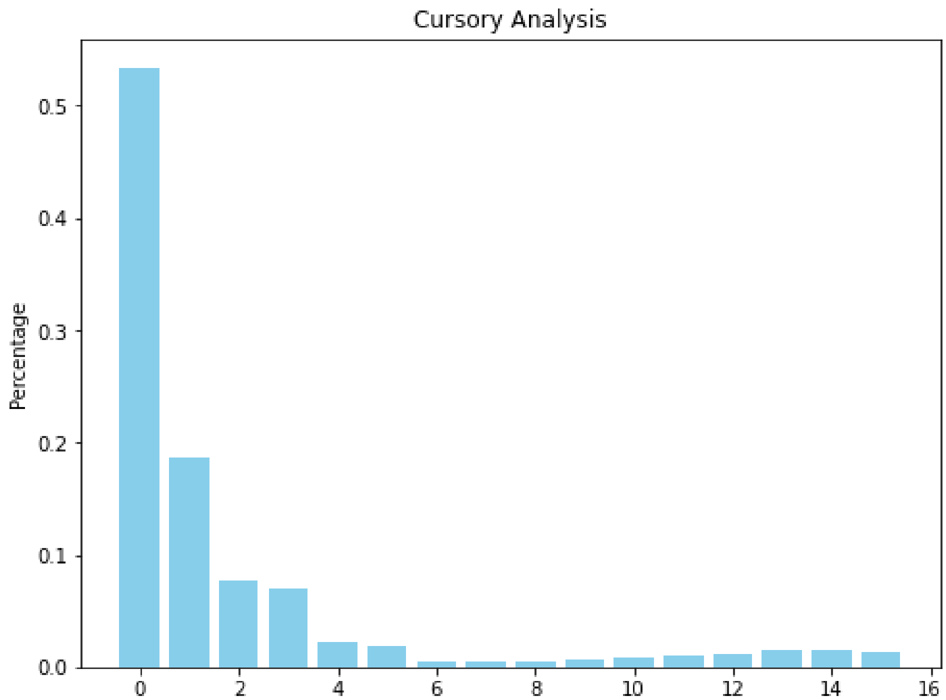
In our procedures, we used daily return adjusted for dividends from CRSP and used *StandardScalar in sklearn* in Python to calculate standardized return matrix. Then, we applied PCA analysis with 3 principal components on the standardized return matrix. We also calculated the correlation matrix based on the standardized return matrix, and generated eigenvalues and plotted them from the largest value to the lowest value. Below is the graph we plot and the graph in the paper:

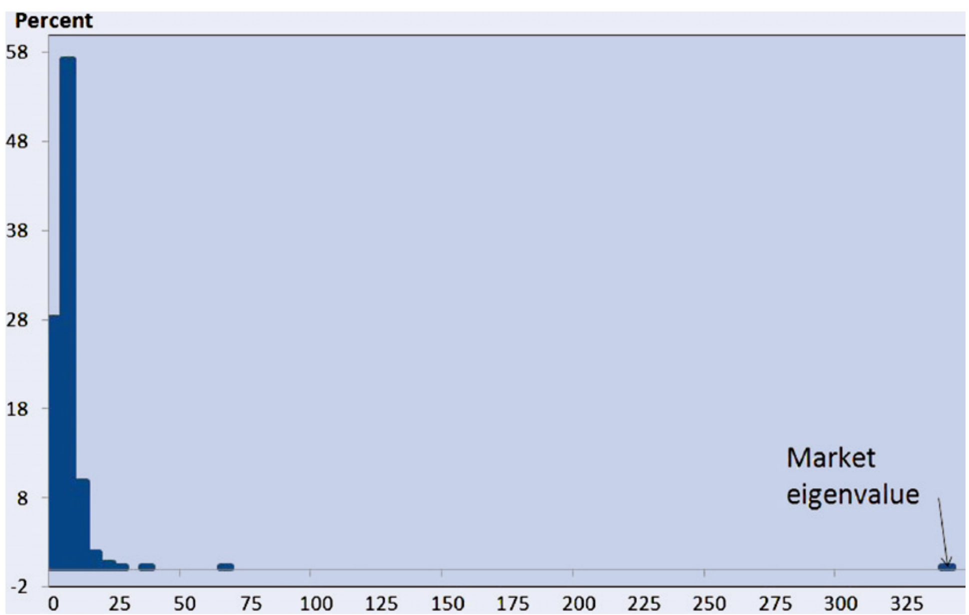




From graphs above, we can see that both graphs show a decreasing trend of the magnitude of eigenvalues.

We later conducted a cursory analysis on eigenvalues. Below is the graph we plot and the graph in the paper:

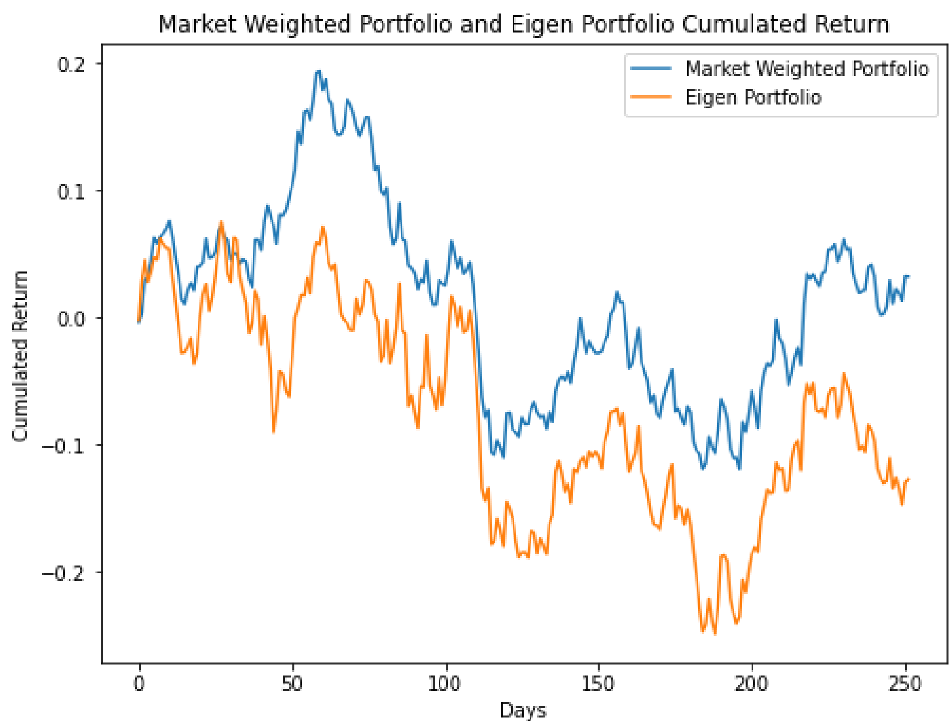


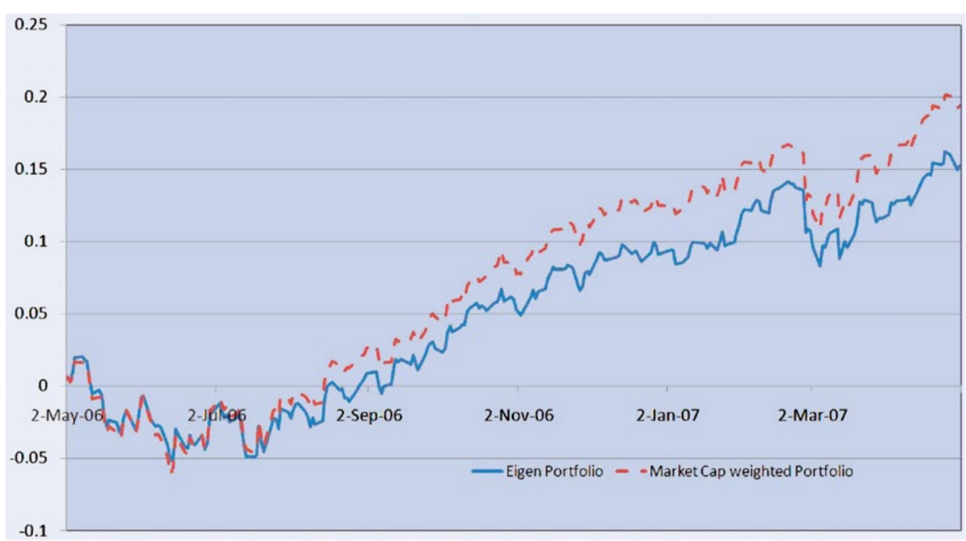


From graphs above, we can see that both graphs show similar trends that high density is shown on the left side of the graph while a minor increase on the right side.

b) Section 2.2

In this section, we constructed the eigen portfolio and the market cap weighted portfolio to calculate their cumulative returns. Below is the cumulative return graph we plot and the graph in the paper:





We discovered that our graph showed high similarity with graph provided in the paper that both portfolios showed a comparative evolution, and the cumulative return of the eigen portfolio is slightly lower than market cap weighted portfolio. Hence, this result matches the view in the paper that two portfolios are not identical but are good proxies for each other.

c) The relationship between Signs of the Second and Third Eigenvectors and the Belonging Industry of the Stock

In this section, we used the result of PCA analysis with 3 principal components on the standardized return matrix. We retrieved the second and third eigenvectors of each stock and merged them with the industry code. Below is the table:

Stock	Second Eigenvector	Third Eigenvector	North American Industry Classification System
EOG	-0.450143	-0.049374	211120
SLB	-0.423821	-0.103848	213112
XOM	-0.468001	-0.032569	324110
AMD	0.073633	0.412608	334413
AVGO	0.069199	0.376026	334413
INTC	0.034908	0.417784	334413
NVDA	0.098801	0.420549	334413
CVX	- 0.451722	- 0.026942	447190
AAL	0.218933	- 0.210610	481111
DAL	0.203492	- 0.259594	481111
LUV	0.174883	- 0.220278	481111
UAL	0.218353	- 0.266802	481111
BAC	0.000662	- 0.165386	522110
JPM	0.025836	- 0.158595	522110
WFC	0.032731	- 0.194900	522110
BRK	- 0.064327	0.001756	524126

From the table above, we can see that stocks in the same industry have the same signs of the second and third eigenvectors.

5. Conclusion

Through replication of the original results using a smaller dataset, our project aimed to validate the robustness and generalizability of the findings to a more constrained context. Despite the scale reduction, our results closely mirror those reported in the original paper, demonstrating the consistency of PCA and its ability to group stocks in a matter that corresponds to industries.

6. References

Marco Avellaneda & Jeong-Hyun Lee (2010) Statistical arbitrage in the US equities market, Quantitative Finance, 10:7, 761-782, DOI: 10.1080/14697680903124632

Wharton Research Data Services. (2022). Wharton Financial Data Services. Retrieved from https://wrds-www.wharton.upenn.edu/

Pandas Development Team. (2023). pandas (1.3.3). Retrieved from https://pandas.pydata.org/

NumPy Community. (2023). NumPy (1.21.2). Retrieved from https://numpy.org/

Matplotlib Development Team. (2023). Matplotlib (3.4.3). Retrieved from https://matplotlib.org/

Scikit-learn Developers. (2023). scikit-learn (0.24.2). Retrieved from https://scikit-learn.org/

MATH 628 FINAL PROJECT

Chunlin Shi Noah Collins

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
data = pd.read_excel('data.xlsx')
data
```

Out[1]:

	Names Date	Ticker Symbol	North American Industry Classification System	Price or Bid/Ask Average	Shares Outstanding	Returns without Dividends
0	2021-12-31	XOM	324110	61.189999	4233567	0.006580
1	2022-01-03	XOM	324110	63.540001	4233567	0.038405
2	2022-01-04	XOM	324110	65.930000	4233567	0.037614
3	2022-01-05	XOM	324110	66.750000	4233567	0.012437
4	2022-01-06	XOM	324110	68.320000	4233567	0.023521
...	...	...	...	...	...	...
4279	2022-12-23	AVGO	334413	552.429993	418000	-0.001193
4280	2022-12-27	AVGO	334413	553.539978	418000	0.002009
4281	2022-12-28	AVGO	334413	544.890015	418000	-0.015627
4282	2022-12-29	AVGO	334413	557.809998	418000	0.023711
4283	2022-12-30	AVGO	334413	559.130005	418000	0.002366

4284 rows × 6 columns

In [2]:
```
data_ret = data[['Ticker Symbol','Names Date','Returns without Dividends']]
data_ret
```

Out[2]:

	Ticker Symbol	Names Date	Returns without Dividends
0	XOM	2021-12-31	0.006580
1	XOM	2022-01-03	0.038405
2	XOM	2022-01-04	0.037614
3	XOM	2022-01-05	0.012437
4	XOM	2022-01-06	0.023521
...	...	...	...
4279	AVGO	2022-12-23	-0.001193
4280	AVGO	2022-12-27	0.002009
4281	AVGO	2022-12-28	-0.015627
4282	AVGO	2022-12-29	0.023711
4283	AVGO	2022-12-30	0.002366

4284 rows × 3 columns

In [3]:

```
data_ret = data_ret.pivot_table(index='Names Date', columns='Ticker Symbol', values='Re
data_ret
```

Out[3]:

Ticker Symbol	AAL	AMD	AVGO	BAC	BRK	CVX	DAL	EOG	INT
Names Date									
2021-12-31	-0.006087	-0.008612	0.000496	-0.000898	-0.003884	-0.000681	0.001025	-0.003925	-0.0046
2022-01-03	0.043987	0.044058	-0.003141	0.037986	0.007030	0.016276	0.030962	0.026230	0.0332
2022-01-04	0.014400	-0.038738	0.011457	0.039194	0.025440	0.018196	0.007446	0.045963	-0.0013
2022-01-05	-0.017876	-0.057264	-0.041614	-0.016879	0.003916	0.006506	-0.007637	-0.018353	0.0137
2022-01-06	-0.005889	0.000588	-0.009285	0.020136	0.011614	0.008509	-0.004220	0.020513	0.0025
...	...	...	...	...	...	...	...	...	
2022-12-23	0.011943	0.010335	-0.001193	0.002470	0.011400	0.030916	0.007290	0.034125	0.0046
2022-12-27	-0.014162	-0.019374	0.002009	0.001848	-0.003093	0.012571	-0.007841	0.011255	-0.0057
2022-12-28	-0.016760	-0.011064	-0.015627	0.007378	-0.005802	-0.014753	-0.027660	-0.035433	-0.0154
2022-12-29	0.030844	0.035960	0.023711	0.011291	0.018983	0.007572	0.023132	0.009655	0.0262
2022-12-30	0.001575	-0.000771	0.002366	-0.000604	-0.000274	0.006561	0.003972	0.006919	0.0083

252 rows × 16 columns

In [4]:

```
data_ret.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 252 entries, 2021-12-31 to 2022-12-30
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AAL     252 non-null    float64
 1   AMD     252 non-null    float64
 2   AVGO    252 non-null    float64
 3   BAC     252 non-null    float64
 4   BRK     252 non-null    float64
 5   CVX     252 non-null    float64
 6   DAL     252 non-null    float64
 7   EOG     252 non-null    float64
 8   INTC    252 non-null    float64
 9   JPM     252 non-null    float64
 10  LUV     252 non-null    float64
 11  NVDA    252 non-null    float64
 12  SLB     252 non-null    float64
 13  UAL     252 non-null    float64
 14  WFC     252 non-null    float64
 15  XOM     252 non-null    float64
dtypes: float64(16)
memory usage: 33.5 KB
```

Section 2.1 of the Paper

From above, we can see that there is no null values for the return data

We then standardize the return data

In [5]:

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_ret)
scaled_data = pd.DataFrame(scaled_data)
scaled_data.columns = data_ret.columns

scaled_data
```

Out[5]:

Ticker Symbol	AAL	AMD	AVGO	BAC	BRK	CVX	DAL	EOG	INT
0	-0.150503	-0.160448	0.038864	0.003356	-0.293354	-0.124574	0.044680	-0.205590	-0.0942
1	1.266021	1.213171	-0.116372	1.907877	0.485604	0.693871	1.082119	0.861953	1.4810
2	0.429058	-0.946130	0.506723	1.967067	1.799625	0.786510	0.267204	1.560554	0.0440
3	-0.483981	-1.429267	-1.758524	-0.779349	0.263372	0.222309	-0.255488	-0.716404	0.6706
4	-0.144880	0.079468	-0.378600	1.033580	0.812832	0.319004	-0.137078	0.659554	0.2069
...	...	...	...	...	...	...	...	...	
247	0.359542	0.333677	-0.033247	0.168326	0.797544	1.400451	0.261814	1.141461	0.2911
248	-0.378923	-0.441117	0.103450	0.137853	-0.236843	0.515021	-0.262534	0.331802	-0.1405
249	-0.452407	-0.224393	-0.649296	0.408711	-0.430191	-0.803724	-0.949329	-1.321064	-0.5431
250	0.894238	1.001959	1.029743	0.600364	1.338716	0.273746	0.810786	0.275149	1.1908
251	0.066252	0.044025	0.118693	0.017790	-0.035659	0.224983	0.146815	0.178303	0.4482

252 rows × 16 columns

Now we can do PCA analysis

In [6]:

```
pca = PCA(n_components=3)
pca.fit(scaled_data)
```

Out[6]:

```
PCA(n_components=3)
```

We extract the second and the third eigenvector from PCA

In [7]:

```
eigenvectors = pca.components_
second_eigenvector = eigenvectors[1]
third_eigenvector = eigenvectors[2]
```

In [8]:

```
correlation_matrix = scaled_data.corr()
correlation_matrix
```

Out[8]:

Ticker Symbol Ticker Symbol	AAL	AMD	AVGO	BAC	BRK	CVX	DAL	EOG	INTC	
AAL	1.000000	0.606493	0.594144	0.550496	0.529382	0.157999	0.924746	0.132858	0.514597	0.54
AMD	0.606493	1.000000	0.780958	0.559184	0.588605	0.303841	0.576782	0.264088	0.741271	0.51
AVGO	0.594144	0.780958	1.000000	0.554185	0.613924	0.296872	0.593796	0.273659	0.750160	0.56
BAC	0.550496	0.559184	0.554185	1.000000	0.707732	0.365815	0.595608	0.344822	0.507697	0.89
BRK	0.529382	0.588605	0.613924	0.707732	1.000000	0.469095	0.568930	0.412894	0.596912	0.70
CVX	0.157999	0.303841	0.296872	0.365815	0.469095	1.000000	0.190125	0.815066	0.296874	0.30
DAL	0.924746	0.576782	0.593796	0.595608	0.568930	0.190125	1.000000	0.156082	0.508122	0.59
EOG	0.132858	0.264088	0.273659	0.344822	0.412894	0.815066	0.156082	1.000000	0.261002	0.30
INTC	0.514597	0.741271	0.750160	0.507697	0.596912	0.296874	0.508122	0.261002	1.000000	0.51
JPM	0.548905	0.519383	0.564049	0.895891	0.709166	0.303311	0.594361	0.300045	0.519479	1.00
LUV	0.842430	0.552735	0.573072	0.537038	0.519312	0.200572	0.865357	0.180979	0.500612	0.52
NVDA	0.622287	0.887174	0.824741	0.547887	0.578304	0.273685	0.596280	0.238775	0.747413	0.52
SLB	0.185075	0.248181	0.261785	0.338610	0.399552	0.774093	0.191830	0.748056	0.276862	0.30
UAL	0.928827	0.554975	0.552298	0.548971	0.520238	0.139055	0.923427	0.132629	0.476877	0.54
WFC	0.576018	0.527571	0.531020	0.865869	0.698938	0.307397	0.619314	0.290887	0.519873	0.80
XOM	0.125324	0.259137	0.265218	0.315299	0.431941	0.873425	0.157380	0.837306	0.303884	0.27

In [9]:

```
eigenvalues = np.linalg.eigvals(correlation_matrix)
eigenvalues
```

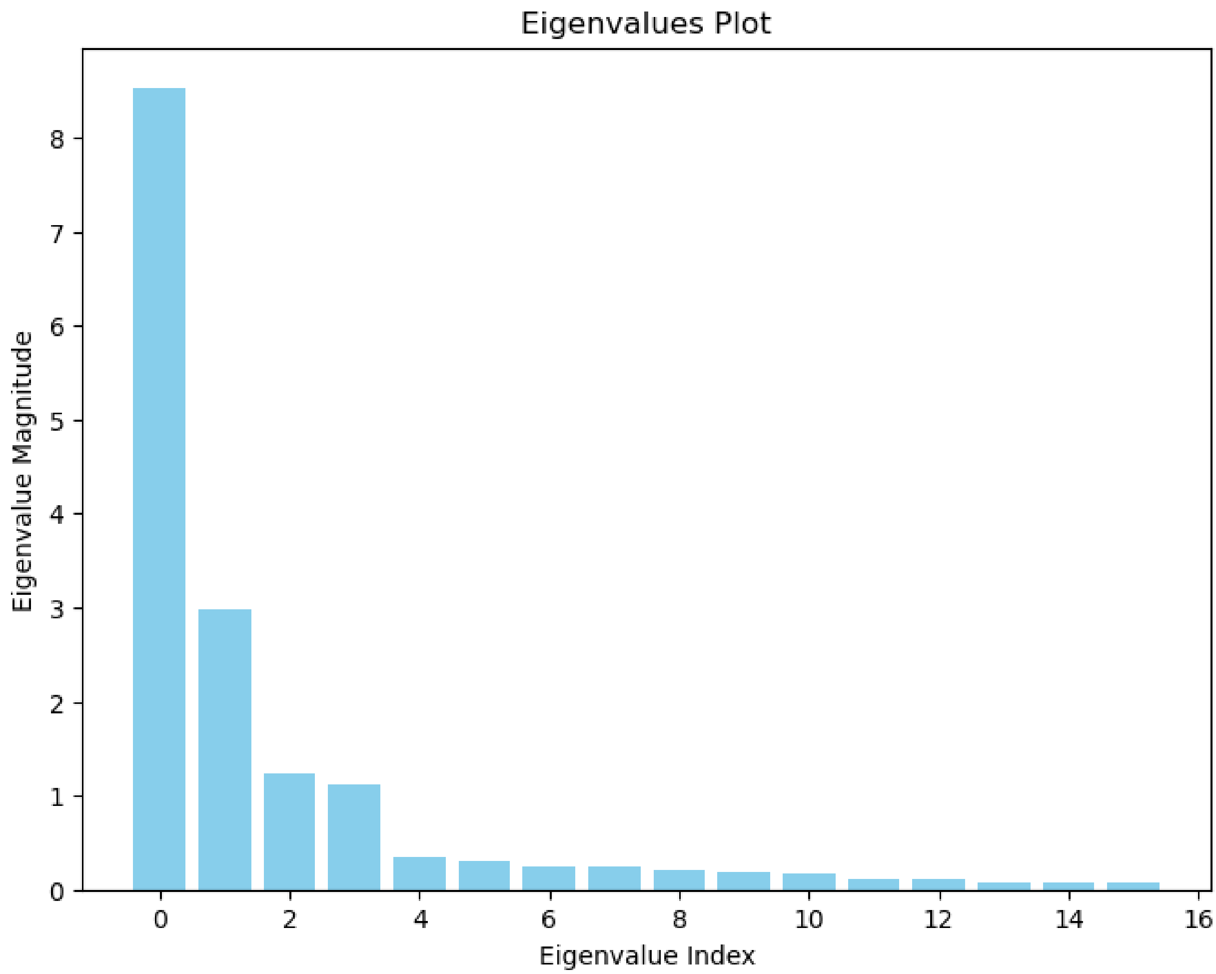
Out[9]:

```
array([8.52702392, 2.98514649, 1.23028626, 1.12078152, 0.34025521,
       0.30711004, 0.06357353, 0.06739485, 0.07830166, 0.10535933,
       0.11858138, 0.1726956 , 0.18402044, 0.24722549, 0.23754924,
       0.21469504])
```

In [10]:

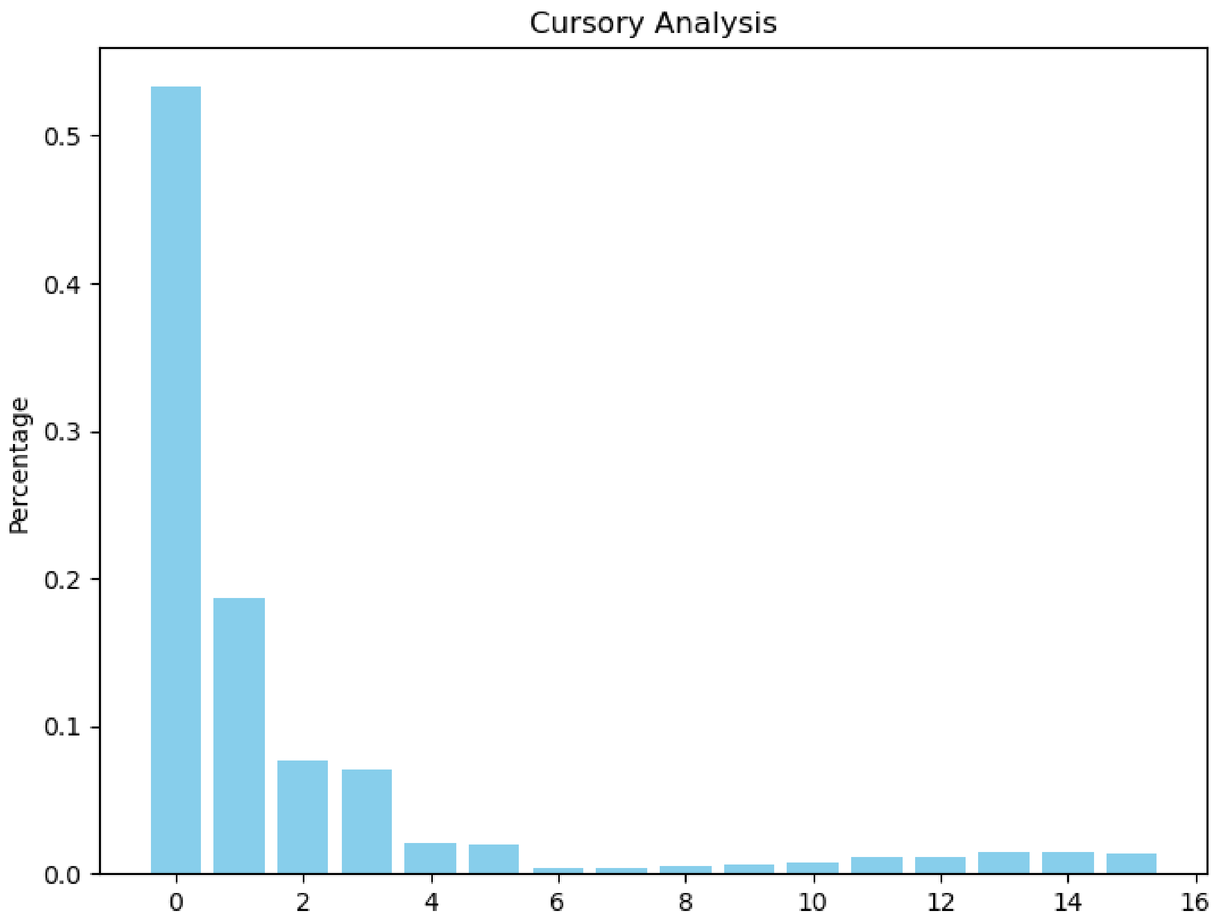
```
sorted_eigenvalues = np.sort(eigenvalues)[::-1]  # Reverse the order

# Plotting the eigenvalues
plt.figure(figsize=(8, 6))
plt.bar(range(len(sorted_eigenvalues)), sorted_eigenvalues, color='skyblue')
plt.xlabel('Eigenvalue Index')
plt.ylabel('Eigenvalue Magnitude')
plt.title('Eigenvalues Plot')
plt.show()
```



In [11]:
```
total_variance = np.sum(eigenvalues)
explained_variance_ratio = eigenvalues / total_variance

# Plotting the eigenvalues
plt.figure(figsize=(8, 6))
plt.bar(range(len(explained_variance_ratio)), explained_variance_ratio, color='skyblue')
plt.ylabel('Percentage')
plt.title('Cursory Analysis')
plt.show()
```



Section 2.3 of the Paper

We calculate the cumulative return of eigenportfolio

In [12]:
```
cov_matrix = np.cov(scaled_data, rowvar=False)

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Select top eigenportfolio (e.g., first eigenvector)
top_eigenvector = eigenvectors[:, 0]  # Replace '0' with the index of the desired eigenv

# Construct eigenportfolio by normalizing weights
eigenportfolio = top_eigenvector / np.sum(top_eigenvector)
eigenportfolio_returns = np.dot(data_ret, eigenportfolio)

cumulative_return = np.cumprod(1 + eigenportfolio_returns) - 1

cumulative_return = pd.DataFrame(cumulative_return)
cumulative_return
```

Out[12]:

	0
0	-0.002287
1	0.028602
2	0.045073
3	0.027664
4	0.035953
...	...
247	-0.126009
248	-0.134971
249	-0.147446
250	-0.130102
251	-0.127446

252 rows × 1 columns

Now we calculate the cumulative return of market cap weighted portfolio

In [13]:
```
mkt_cap = data[['Ticker Symbol','Names Date','Price or Bid/Ask Average','Shares Outstan
mkt_cap
```

Out[13]:

	Ticker Symbol	Names Date	Price or Bid/Ask Average	Shares Outstanding	Returns without Dividends
0	XOM	2021-12-31	61.189999	4233567	0.006580
1	XOM	2022-01-03	63.540001	4233567	0.038405
2	XOM	2022-01-04	65.930000	4233567	0.037614
3	XOM	2022-01-05	66.750000	4233567	0.012437
4	XOM	2022-01-06	68.320000	4233567	0.023521
...	...	...	...	...	...
4279	AVGO	2022-12-23	552.429993	418000	-0.001193
4280	AVGO	2022-12-27	553.539978	418000	0.002009
4281	AVGO	2022-12-28	544.890015	418000	-0.015627
4282	AVGO	2022-12-29	557.809998	418000	0.023711
4283	AVGO	2022-12-30	559.130005	418000	0.002366

4284 rows × 5 columns

In [14]:
```
shr = pd.Series(mkt_cap.groupby('Ticker Symbol')['Shares Outstanding'].sum()/252)
shr
```

Out[14]:
```
Ticker Symbol
AAL     6.494009e+05
AMD     1.567547e+06
AVGO    4.087312e+05
BAC     8.070363e+06
BRK     1.296293e+06
CVX     1.951552e+06
DAL     6.407667e+05
EOG     5.857664e+05
INTC    4.100060e+06
JPM     2.942416e+06
LUV     5.928831e+05
NVDA    2.491785e+06
SLB     1.412687e+06
UAL     3.261760e+05
WFC     3.826443e+06
XOM     4.201088e+06
Name: Shares Outstanding, dtype: float64
```

In [15]:
```
mkt_ret = mkt_cap[['Ticker Symbol','Names Date','Price or Bid/Ask Average']]
mkt_ret = mkt_ret.pivot_table(index='Names Date', columns='Ticker Symbol', values='Pric
mkt_ret
```

Out[15]:

Ticker Symbol	AAL	AMD	AVGO	BAC	BRK	CVX	DAL	EC
Names Date								
2021-12-31	17.959999	143.899994	665.409973	44.490002	225480.500000	117.349998	39.080002	88.8300
2022-01-03	18.750000	150.240005	663.320007	46.180000	227300.395004	119.260002	40.290001	91.1600
2022-01-04	19.020000	144.419998	670.919983	47.990002	233016.764999	121.430000	40.590000	95.3499
2022-01-05	18.680000	136.149994	643.000000	47.180000	233792.085007	122.220001	40.279999	93.5999
2022-01-06	18.570000	136.229996	637.030029	48.130001	236733.110001	123.260002	40.110001	95.5199
...	...	...	...	...	...	...	...	
2022-12-23	12.710000	64.519997	552.429993	32.470001	231853.244995	177.399994	33.160000	130.6100
2022-12-27	12.530000	63.270000	553.539978	32.529999	231130.274994	179.630005	32.900002	132.0800
2022-12-28	12.320000	62.570000	544.890015	32.770000	230051.714996	176.979996	31.990000	127.4000
2022-12-29	12.700000	64.820000	557.809998	33.139999	234517.029999	178.320007	32.730000	128.6300
2022-12-30	12.720000	64.769997	559.130005	33.119999	234509.934372	179.490005	32.860001	129.5200

252 rows × 16 columns

In [16]:

```
market_caps = mkt_ret * shr
market_caps
```

Out[16]:

Ticker Symbol	AAL	AMD	AVGO	BAC	BRK	CVX	
Names Date							
2021-12-31	1.166324e+07	2.255700e+08	2.719738e+08	3.590505e+08	2.922889e+11	2.290147e+08	2.5041
2022-01-03	1.217627e+07	2.355083e+08	2.711196e+08	3.726894e+08	2.946480e+11	2.327421e+08	2.5816
2022-01-04	1.235161e+07	2.263852e+08	2.742259e+08	3.872967e+08	3.020581e+11	2.369770e+08	2.6008
2022-01-05	1.213081e+07	2.134215e+08	2.628142e+08	3.807597e+08	3.030631e+11	2.385187e+08	2.5810
2022-01-06	1.205937e+07	2.135469e+08	2.603740e+08	3.884266e+08	3.068756e+11	2.405483e+08	2.5701
...	...	...	...	...	...	...	
2022-12-23	8.253885e+06	1.011381e+08	2.257954e+08	2.620447e+08	3.005498e+11	3.462054e+08	2.1247
2022-12-27	8.136993e+06	9.917871e+07	2.262491e+08	2.625289e+08	2.996126e+11	3.505573e+08	2.1081
2022-12-28	8.000619e+06	9.808142e+07	2.227135e+08	2.644658e+08	2.982145e+11	3.453857e+08	2.0498
2022-12-29	8.247391e+06	1.016084e+08	2.279943e+08	2.674518e+08	3.040029e+11	3.480008e+08	2.0972
2022-12-30	8.260380e+06	1.015300e+08	2.285339e+08	2.672904e+08	3.039937e+11	3.502841e+08	2.1055

252 rows × 16 columns

In [17]:

```
weights = market_caps.div(market_caps.sum(axis=1), axis=0)
weights
```

Out[17]:

Ticker Symbol	AAL	AMD	AVGO	BAC	BRK	CVX	DAL	EOG	INTC	
Names Date										
2021-12-31	0.000039	0.000764	0.000921	0.001215	0.989482	0.000775	0.000085	0.000176	0.000715	0.00
2022-01-03	0.000041	0.000791	0.000910	0.001251	0.989274	0.000781	0.000087	0.000179	0.000732	0.00
2022-01-04	0.000040	0.000742	0.000898	0.001269	0.989426	0.000776	0.000085	0.000183	0.000714	0.00
2022-01-05	0.000040	0.000697	0.000858	0.001243	0.989721	0.000779	0.000084	0.000179	0.000721	0.00
2022-01-06	0.000039	0.000689	0.000840	0.001253	0.989717	0.000776	0.000083	0.000180	0.000714	0.00
...	...	...	...	...	...	...	...	...	...	...
2022-12-23	0.000027	0.000334	0.000745	0.000864	0.991307	0.001142	0.000070	0.000252	0.000353	0.00
2022-12-27	0.000027	0.000328	0.000749	0.000869	0.991334	0.001160	0.000070	0.000256	0.000352	0.00
2022-12-28	0.000027	0.000326	0.000740	0.000879	0.991369	0.001148	0.000068	0.000248	0.000348	0.00
2022-12-29	0.000027	0.000331	0.000744	0.000872	0.991401	0.001135	0.000068	0.000246	0.000350	0.00
2022-12-30	0.000027	0.000331	0.000745	0.000872	0.991361	0.001142	0.000069	0.000247	0.000353	0.00

252 rows × 16 columns

In [18]:

```
daily_mkt_ret = data[['Ticker Symbol','Names Date','Returns without Dividends']]
daily_mkt_ret = daily_mkt_ret.pivot_table(index='Names Date', columns='Ticker Symbol',
daily_mkt_ret = (daily_mkt_ret * weights).sum(axis=1)
daily_mkt_ret
```

Out[18]:

```
Names Date
2021-12-31   -0.003867
2022-01-03    0.007260
2022-01-04    0.025291
2022-01-05    0.003626
2022-01-06    0.011630
                ...
2022-12-23    0.011400
2022-12-27   -0.003113
2022-12-28   -0.005826
2022-12-29    0.018955
2022-12-30   -0.000231
Length: 252, dtype: float64
```

In [19]:
```
daily_mkt_ret = daily_mkt_ret.reset_index(drop=True)
daily_mkt_ret = np.cumprod(1 + daily_mkt_ret) - 1
daily_mkt_ret = pd.DataFrame(daily_mkt_ret)
daily_mkt_ret
```

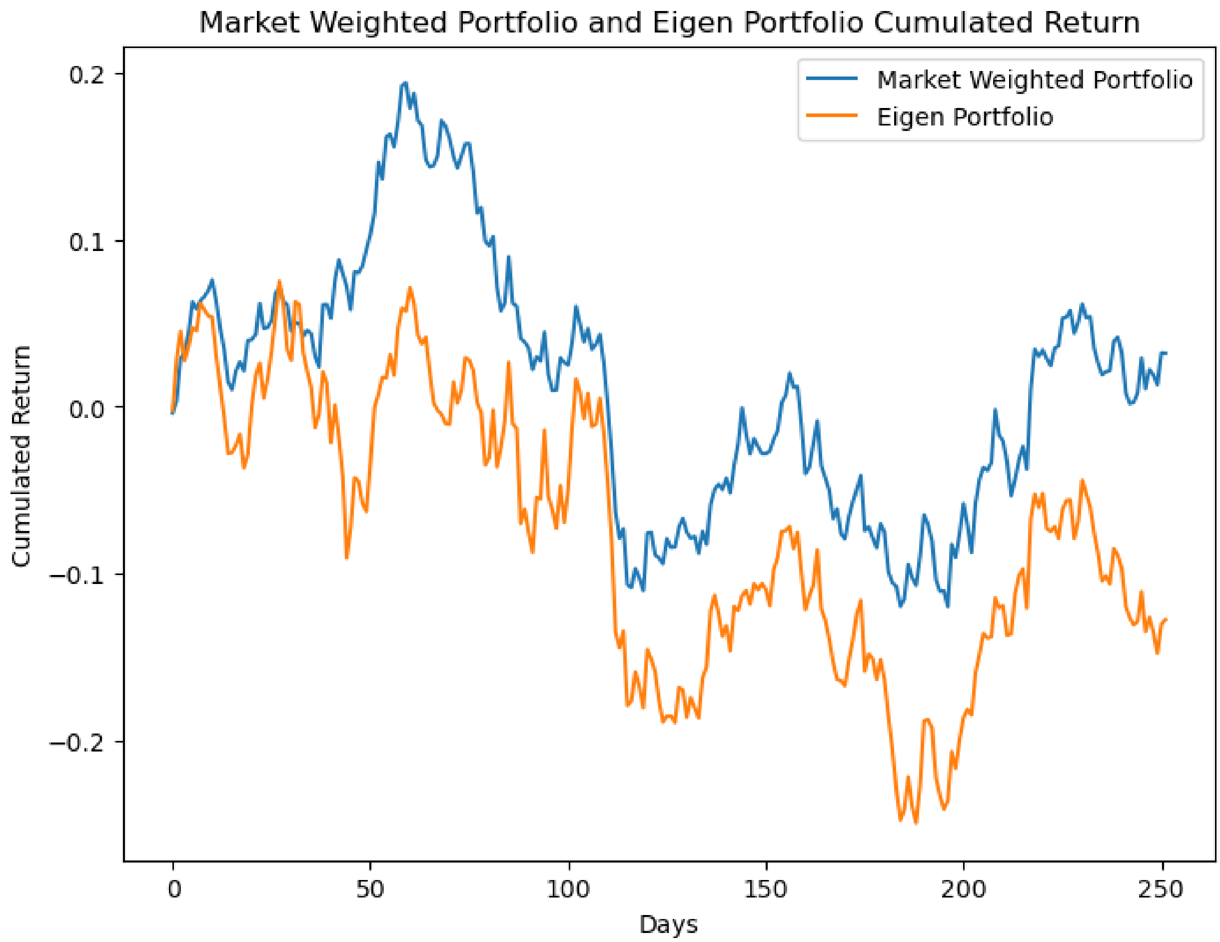
Out[19]:

	0
0	-0.003867
1	0.003365
2	0.028741
3	0.032471
4	0.044479
...	...
247	0.022213
248	0.019031
249	0.013095
250	0.032298
251	0.032059

252 rows × 1 columns

We plot the cumulative return of eigenportfolio and market cap weighted portfolio in the same graph

In [20]:
```
plt.figure(figsize=(8, 6))
plt.plot(daily_mkt_ret, label='Market Weighted Portfolio')
plt.plot(cumulative_return, label='Eigen Portfolio')
plt.xlabel('Days')
plt.ylabel('Cumulated Return')
plt.title('Market Weighted Portfolio and Eigen Portfolio Cumulated Return')
plt.legend()
plt.show()
```



The relationship between Signs of the Second and Third Eigenvectors and the Belonging Industry of the Stock

In [21]:
```
signs = pd.DataFrame({'Stock': scaled_data.columns,
                      'Second_Eigenvector': second_eigenvector,
                      'Third_Eigenvector': third_eigenvector})
```

In [22]:
```
signs
```

Out[22]:

	Stock	Second_Eigenvector	Third_Eigenvector
0	AAL	0.218933	-0.210610
1	AMD	0.073633	0.412608
2	AVGO	0.069199	0.376026
3	BAC	0.000662	-0.165386
4	BRK	-0.064327	-0.001756
5	CVX	-0.451722	-0.026942
6	DAL	0.203492	-0.259594
7	EOG	-0.450143	-0.049374
8	INTC	0.034908	0.417784
9	JPM	0.025836	-0.158595
10	LUV	0.174883	-0.220278
11	NVDA	0.098801	0.420549
12	SLB	-0.423821	-0.103848
13	UAL	0.218353	-0.266802
14	WFC	0.032731	-0.194900
15	XOM	-0.468001	-0.032569

We extract industry code from the original table

In [23]:

```
industry_data = data[['Ticker Symbol', 'North American Industry Classification System']]
industry_data = industry_data.rename(columns={'Ticker Symbol': 'Stock'})
industry_data = industry_data.sort_values(by='North American Industry Classification Sy
industry_data = industry_data.reset_index(drop=True)
industry_data
```

Out[23]:

	Stock	North American Industry Classification System
0	EOG	211120
1	SLB	213112
2	XOM	324110
3	INTC	334413
4	AMD	334413
5	NVDA	334413
6	AVGO	334413
7	CVX	447190
8	AAL	481111
9	LUV	481111
10	UAL	481111
11	DAL	481111
12	WFC	522110
13	JPM	522110
14	BAC	522110
15	BRK	524126

In [24]:

```
signs_with_industry = signs.merge(industry_data, on='Stock')
signs_with_industry = signs_with_industry.sort_values('North American Industry Classifi
signs_with_industry
```

Out[24]:

	Stock	Second_Eigenvector	Third_Eigenvector	North American Industry Classification System
0	EOG	-0.450143	-0.049374	211120
1	SLB	-0.423821	-0.103848	213112
2	XOM	-0.468001	-0.032569	324110
3	AMD	0.073633	0.412608	334413
4	AVGO	0.069199	0.376026	334413
5	INTC	0.034908	0.417784	334413
6	NVDA	0.098801	0.420549	334413
7	CVX	-0.451722	-0.026942	447190
8	AAL	0.218933	-0.210610	481111
9	DAL	0.203492	-0.259594	481111
10	LUV	0.174883	-0.220278	481111
11	UAL	0.218353	-0.266802	481111
12	BAC	0.000662	-0.165386	522110
13	JPM	0.025836	-0.158595	522110
14	WFC	0.032731	-0.194900	522110
15	BRK	-0.064327	-0.001756	524126

From table above, we can see that within the same industry group, the signs of second eigenvector and the third eigenvector are the same