AF3214 Week 9 Introduction to Machine Learning in Accounting and Finance

Portfolio Optimization using Eigen Portfolio - Unsupervised Learning

In this study, we will use dimensionality reduction techniques (e.g., Principle Component Analysis, or PCA) for portfolio management and allocation.

Note: This set of scripts demonstrates a machine learning based algorithmic trading model to help you construct a near optimal portfolio and test its performance via backtesting. By applying specific trading strategy or model to historical market data, we try to assess how it would have performed in the past. You may change the size of training and testing samples to see the different results in backtesting.

You may find the scripts difficult to understand and it's okay. The purpose of this set of scripts is to let you know how machine learns from unlabled data and what factors may affect the training or testing outcome.

1. The Problem

The goal in this study is to maximize risk-adjusted returns using dimensionality reduction-based (e.g., PCA) algorithm on 30 Dow Jones component stocks to allocate capital into different asset classes.

The dataset used for this study is Dow Jones Industrial Average (DJIA) index and its 30 component stocks from year 2000-2019. We use Alpha Vantage to download the data.

1.1 Getting the Data

In [40]:
```
from alpha_vantage.timeseries import TimeSeries
import time
import pandas as pd
import numpy as np

stock_data = {}
ts = TimeSeries(key='Your key here', output_format='pandas')

tickers = ['MMM', 'AXP', 'AMGN', 'AAPL', 'BA', 'CAT', 'CVX', 'CSCO', 'KO', '
           'HD', 'HON', 'IBM', 'INTC', 'JNJ', 'JPM', 'MCD', 'MRK', 'MSFT', '
           'CRM', 'TRV', 'UNH', 'VZ', 'V', 'WBA', 'WMT', 'DIS'] ### 30 Dow J
'''
for ticker in tickers:
    data, meta_data = ts.get_daily_adjusted(symbol=ticker, outputsize='full'
    stock_data[ticker] = data
    time.sleep(10) ### Free api key only allows 5 calls per minute, so we ne
'''
## remove comments to change your dataset
```

Out[40]:
```
"\nfor ticker in tickers: \n    data, meta_data = ts.get_daily_adjusted(sym
bol=ticker, outputsize='full')\n    stock_data[ticker] = data\n    time.sle
ep(10) ### Free api key only allows 5 calls per minute, so we need to set t
he waiting time to be long enough.\n"
```

In [41]:
```
'''
stock_final_data = pd.DataFrame()
for ticker in tickers:
    stock_final_data[ticker] = stock_data[ticker].loc['2019': '2000','5. adj
stock_final_data = stock_final_data.sort_values(by='date')
stock_final_data.to_csv("Dow_Adjusted.csv")
stock_final_data.tail()
'''
## remove comments to change your dataset
```

Out[41]:
```
'\nstock_final_data = pd.DataFrame()\nfor ticker in tickers:\n    stock_fin
al_data[ticker] = stock_data[ticker].loc[\'2019\': \'2000\',\'5. adjusted c
lose\']\nstock_final_data = stock_final_data.sort_values(by=\'date\')\nstoc
k_final_data.to_csv("Dow_Adjusted.csv")\nstock_final_data.tail()\n'
```

1.2. Loading the data and python packages

1.2.1. Loading Python Packages for Machine Learning

In [42]:
```
# Load libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas import read_csv, set_option
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
```

```python
from numpy.linalg import inv, eig, svd
from sklearn.manifold import TSNE
from sklearn.decomposition import KernelPCA
```

In [43]:
```python
#Diable the warnings
import warnings
warnings.filterwarnings('ignore')
```

1.2.2. Loading our Stock Data

In [44]:
```python
# load dataset
dataset = read_csv('Dow_Adjusted.csv',index_col=0)
```

In [45]:
```python
type(dataset)
```

Out[45]:
```
pandas.core.frame.DataFrame
```

In [46]:
```python
dataset.head()
```

Out[46]:

date	MMM	AXP	AMGN	AAPL	BA	CAT	CVX	CSCO	
2000-01-03	27.1839	34.0973	49.1135	0.8568	25.8976	13.5570	18.8615	39.6146	1
2000-01-04	26.1038	32.8072	45.3601	0.7845	25.8589	13.3813	18.8615	37.3791	1
2000-01-05	27.4345	32.6391	46.7725	0.7960	27.6696	13.8859	19.2697	37.6723	1
2000-01-06	29.0330	32.6391	47.7011	0.7271	27.7469	14.3933	20.0162	36.6462	1
2000-01-07	29.6091	33.0952	53.0619	0.7616	28.5524	14.8616	20.3680	38.8083	1

5 rows × 30 columns

2. Data Inspection and Analysis

2.1. Descriptive Statistics

In [47]:
```python
# shape
dataset.shape
```

Out[47]:
```
(5031, 30)
```

In [48]:
```python
# types
dataset.dtypes
```

Out[48]:
```
MMM     float64
AXP     float64
AMGN    float64
AAPL    float64
BA      float64
CAT     float64
CVX     float64
CSCO    float64
KO      float64
DOW     float64
GS      float64
HD      float64
HON     float64
IBM     float64
INTC    float64
JNJ     float64
JPM     float64
MCD     float64
MRK     float64
MSFT    float64
NKE     float64
PG      float64
CRM     float64
TRV     float64
UNH     float64
VZ      float64
V       float64
WBA     float64
WMT     float64
DIS     float64
dtype: object
```

In [49]:
```
# describe data
pd.options.display.precision = 4 # round to 4 decimal

dataset.describe()
```

Out[49]:

	MMM	AXP	AMGN	AAPL	BA	CAT	
count	5031.0000	5031.0000	5031.0000	5031.0000	5031.0000	5031.0000	5031.
mean	82.9086	50.7264	79.2684	13.7446	95.5525	56.0575	56.
std	51.3350	25.3160	48.1227	15.6120	94.6691	34.9889	29.
min	22.8970	8.3743	24.2446	0.2008	16.9701	8.6098	15.
25%	47.4205	33.2869	44.7437	1.0517	37.0341	25.2551	28.
50%	59.3137	41.4290	52.8200	6.1992	58.1266	51.1641	51.
75%	122.3240	68.7116	124.9957	22.7017	115.0154	73.9313	82.
max	229.4977	124.3932	231.5281	72.3372	430.3480	155.3298	112.

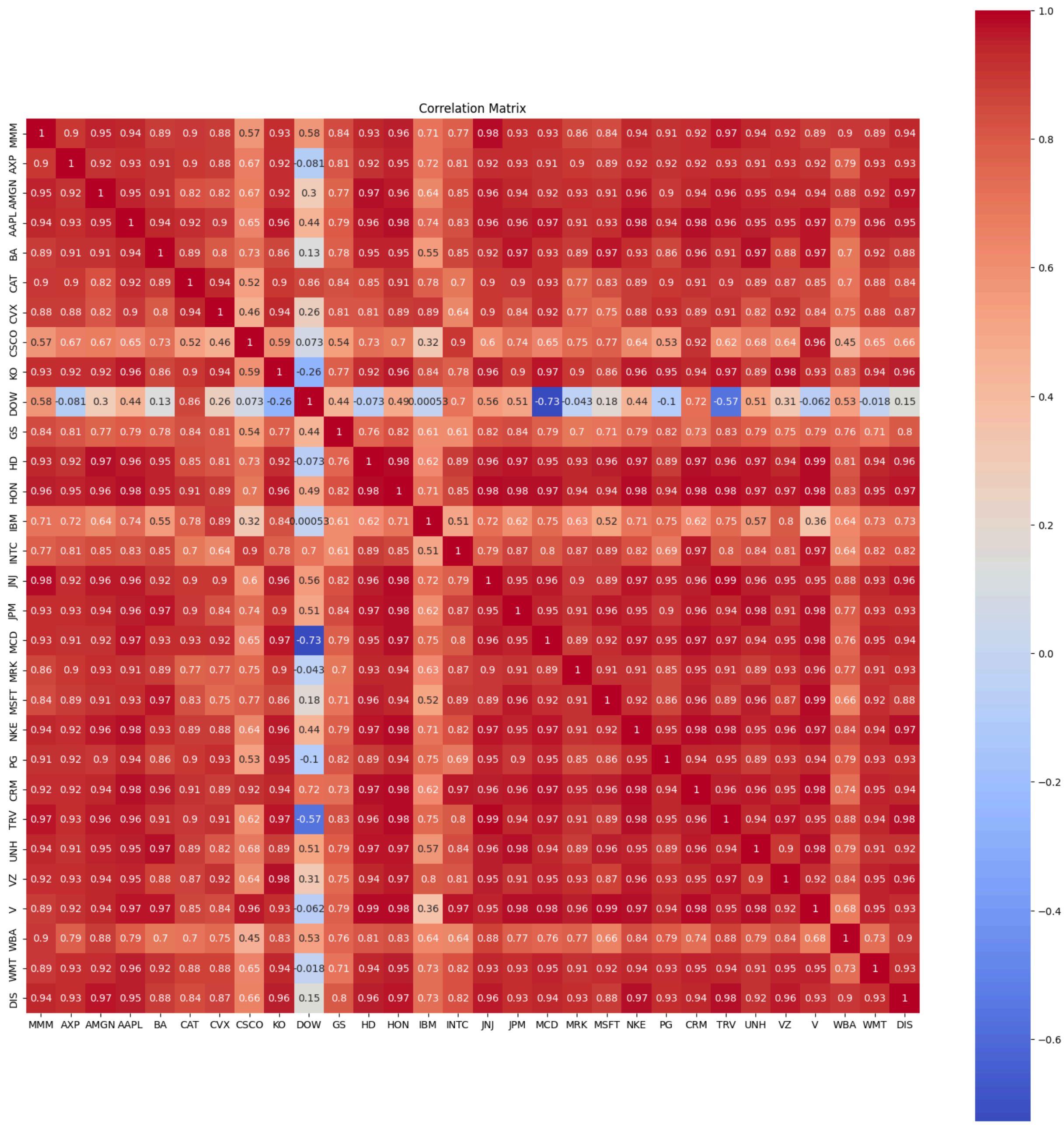
8 rows × 30 columns

2.2. Visualize the Data by Correlation Matrix using a Heatmap

Taking a first look at the correlation matrix. We will be back to this matrix after implementing the Dimensionality Reduction Models.

In [50]:
```python
# correlation matrix
correlation = dataset.corr()
plt.figure(figsize=(20,20))
plt.title('Correlation Matrix')
sns.heatmap(correlation, vmax=1, square=True, annot=True, cmap='coolwarm')
```

Out[50]:
```
<Axes: title={'center': 'Correlation Matrix'}>
```



From above correlation matrix, it seems that these 30 stocks have a significant positive correlation between each other.

3. Data Processing

3.1. Data Cleaning

Let us check for all null values in the data, we can either drop them or fill them with the mean of the column

In [51]:
```
# Check for any null values and remove them
print('Null Values =',dataset.isnull().values.any())
print(dataset.shape)
```

```
Null Values = True
(5031, 30)
```

If a column has more than 20% missing values, we will drop this stock.

In [52]:
```
missing_cell = dataset.isnull().mean().sort_values(ascending=False)

print(missing_cell.head(15))

drop_list = sorted(list(missing_cell[missing_cell > 0.2].index))

dataset.drop(labels=drop_list, axis=1, inplace=True)
dataset.shape
```

```
DOW     0.9604
V       0.4101
CRM     0.2230
AAPL    0.0000
AMGN    0.0000
MMM     0.0000
CAT     0.0000
CVX     0.0000
CSCO    0.0000
KO      0.0000
GS      0.0000
HD      0.0000
BA      0.0000
AXP     0.0000
IBM     0.0000
dtype: float64
```

Out[52]: (5031, 27)

In [53]:
```
dataset.head(5)
```

Out[53]:

date	MMM	AXP	AMGN	AAPL	BA	CAT	CVX	CSCO	
2000-01-03	27.1839	34.0973	49.1135	0.8568	25.8976	13.5570	18.8615	39.6146	1
2000-01-04	26.1038	32.8072	45.3601	0.7845	25.8589	13.3813	18.8615	37.3791	1
2000-01-05	27.4345	32.6391	46.7725	0.7960	27.6696	13.8859	19.2697	37.6723	1
2000-01-06	29.0330	32.6391	47.7011	0.7271	27.7469	14.3933	20.0162	36.6462	1
2000-01-07	29.6091	33.0952	53.0619	0.7616	28.5524	14.8616	20.3680	38.8083	1

5 rows × 27 columns

In [54]:

```
# Drop the rows containing NA
dataset= dataset.dropna(axis=0)

dataset.head(5)
```

Out[54]:

date	MMM	AXP	AMGN	AAPL	BA	CAT	CVX	CSCO	
2000-01-03	27.1839	34.0973	49.1135	0.8568	25.8976	13.5570	18.8615	39.6146	1
2000-01-04	26.1038	32.8072	45.3601	0.7845	25.8589	13.3813	18.8615	37.3791	1
2000-01-05	27.4345	32.6391	46.7725	0.7960	27.6696	13.8859	19.2697	37.6723	1
2000-01-06	29.0330	32.6391	47.7011	0.7271	27.7469	14.3933	20.0162	36.6462	1
2000-01-07	29.6091	33.0952	53.0619	0.7616	28.5524	14.8616	20.3680	38.8083	1

5 rows × 27 columns

Computing Daily Return

In [55]:

```
# Log Returns (in %)
#data_returns = np.log(dataset / dataset.shift(1))

# Simple Daily Returns (in %)
data_returns = dataset.pct_change(1)

# Let's remove "outliers" that beyong 3 standard deviation
# If you remember in Week 8, when we discuss standard deviation
# 99.7% of data observed following a normal distribution lies within 3 stanc
```

```
# for those beyond 3 standard deviation, we consider them as "outliers".
data_returns= data_returns[data_returns.apply(lambda x :(x-x.mean()).abs()<(
data_returns
```

Out[55]:

	MMM	AXP	AMGN	AAPL	BA	CAT	CVX	CSCO	
date									
2000-01-20	-0.0372	0.0168	-0.0026	0.0651	-0.0237	-0.0442	-5.0034e-03	0.0009	0.(
2000-02-02	-0.0173	-0.0284	-0.0239	-0.0144	0.0201	0.0058	-6.7316e-03	-0.0331	-0.(
2000-02-03	-0.0088	-0.0079	0.0039	0.0455	-0.0267	-0.0260	-1.4402e-02	0.0342	-0.(
2000-02-04	-0.0287	-0.0092	0.0098	0.0454	0.0129	0.0000	-3.5363e-02	0.0280	0.(
2000-03-02	-0.0008	-0.0140	0.0038	-0.0638	-0.0102	-0.0126	5.8472e-03	0.0084	-0.(
...	...	...	...	...	...	...	...	...	
2019-12-24	-0.0100	0.0020	-0.0029	0.0010	-0.0135	-0.0069	8.3105e-05	-0.0067	-0.(
2019-12-26	-0.0005	0.0054	-0.0018	0.0198	-0.0092	0.0050	2.1605e-03	0.0015	0.(
2019-12-27	0.0038	-0.0018	-0.0015	-0.0004	0.0007	0.0004	-2.4876e-03	-0.0017	0.(
2019-12-30	-0.0081	-0.0071	-0.0052	0.0059	-0.0113	-0.0051	-3.7406e-03	-0.0038	-0.(
2019-12-31	0.0034	0.0015	0.0033	0.0073	-0.0020	0.0011	5.5069e-03	0.0078	0.(

4071 rows × 27 columns

3.2. Data Transformation

Standardization in statistics is a useful technique to transform attributes to a standard Normal distribution with a mean of 0 and a standard deviation of 1.

In this study, we need to keep all variables in the same scale before applying PCA. If not, a feature with large values will dominate the result.

We use StandardScaler in sklearn to standardize the dataset’s features onto unit scale (mean = 0 and standard deviation = 1).

In [56]:

```
%pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /Library/Frameworks/Python.fr
amework/Versions/3.13/lib/python3.13/site-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /Library/Frameworks/Python.f
ramework/Versions/3.13/lib/python3.13/site-packages (from scikit-learn) (2.
2.3)
Requirement already satisfied: scipy>=1.6.0 in /Library/Frameworks/Python.fr
amework/Versions/3.13/lib/python3.13/site-packages (from scikit-learn) (1.1
5.2)
Requirement already satisfied: joblib>=1.2.0 in /Library/Frameworks/Python.f
ramework/Versions/3.13/lib/python3.13/site-packages (from scikit-learn) (1.
4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /Library/Frameworks/P
ython.framework/Versions/3.13/lib/python3.13/site-packages (from scikit-lear
n) (3.5.0)

[notice] A new release of pip is available: 24.2 -> 25.0.1
[notice] To update, run: pip3 install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [57]:
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(data_returns)
standardzied_Dataset = pd.DataFrame(scaler.fit_transform(data_returns),colum

# Let's take a look at the standardized data
data_returns.dropna(how='any', inplace=True)
standardzied_Dataset.dropna(how='any', inplace=True)
standardzied_Dataset.head(5)
```

Out[57]:

	MMM	AXP	AMGN	AAPL	BA	CAT	CVX	CSCO	
date									
2000-01-20	-3.5062	1.0959	-0.2078	3.3667	-1.6600	-2.8405	-0.4501	0.0171	0.98
2000-02-02	-1.6592	-1.9520	-1.6746	-0.8270	1.2910	0.3140	-0.5868	-2.0498	-3.26
2000-02-03	-0.8657	-0.5704	0.2452	2.3334	-1.8578	-1.6924	-1.1932	2.0446	-2.19
2000-02-04	-2.7128	-0.6575	0.6540	2.3258	0.8048	-0.0512	-2.8505	1.6660	1.88
2000-03-02	-0.1294	-0.9822	0.2397	-3.4335	-0.7489	-0.8470	0.4078	0.4707	-2.92

5 rows × 27 columns

In [58]:
```python
standardzied_Dataset.describe()
```

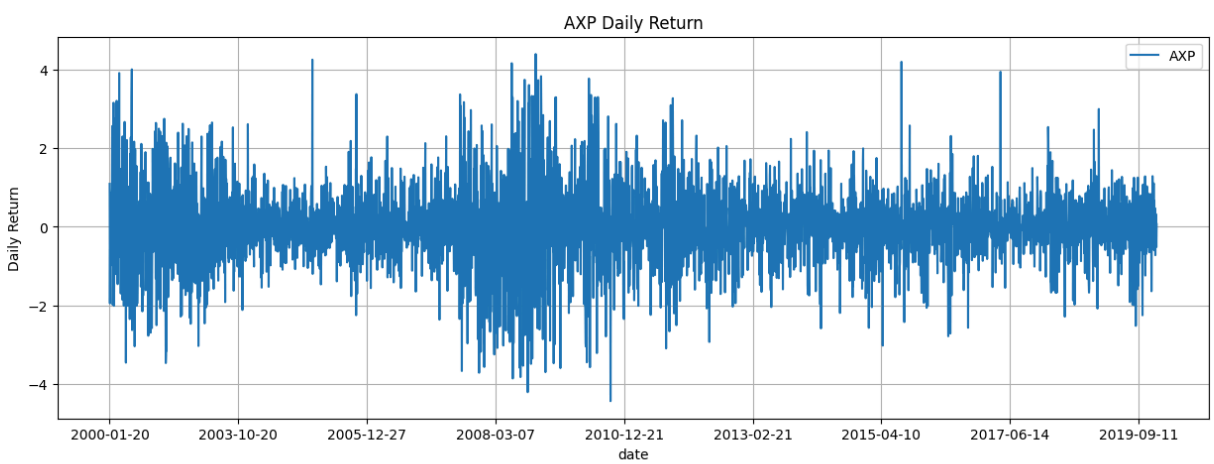
Out[58]:

	MMM	AXP	AMGN	AAPL	BA	
count	4.0710e+03	4.0710e+03	4.0710e+03	4.0710e+03	4.0710e+03	4.071(
mean	1.8326e-17	7.8542e-18	-6.9815e-18	-6.8070e-17	1.0472e-17	-6.196
std	1.0001e+00	1.0001e+00	1.0001e+00	1.0001e+00	1.0001e+00	1.000:
min	-3.9100e+00	-4.4434e+00	-4.0056e+00	-3.8069e+00	-3.6524e+00	-3.697(
25%	-5.4541e-01	-4.9203e-01	-5.7435e-01	-5.3082e-01	-6.1017e-01	-5.684
50%	-1.8649e-03	-5.1296e-03	-2.1100e-02	-2.2063e-02	-1.0500e-02	-1.473
75%	5.7714e-01	5.3338e-01	5.8458e-01	5.4479e-01	6.0454e-01	5.814
max	4.0153e+00	4.4049e+00	4.0816e+00	3.9992e+00	3.6863e+00	3.741(

8 rows × 27 columns

In [59]:

```
# Let's take a look at the Returns for American Express
plt.figure(figsize=(15, 5))
plt.title("AXP Daily Return")
plt.ylabel("Daily Return")
standardzied_Dataset.AXP.plot()
plt.grid(True);
plt.legend()
plt.show()
```



4. Algorithms and Models Evaluation

4.1. Train Test Split

Now we need to divide the portfolio into training sample and testing sample (e.g., train test split) to perform the analysis regarding the best porfolio and backtesting shown later.

In [60]:
```
# the length of our cleaned dataset
len(standardzied_Dataset)
```
Out[60]: 4071

In [61]:
```
# the % allocate to training sample
t = 0.8
percentage = int(len(standardzied_Dataset) * t)
percentage
```
Out[61]: 3256

In [62]:
```
# Dividing the dataset into training and testing samples

percentage = int(len(standardzied_Dataset) * t)
X_training = standardzied_Dataset[:percentage]
X_testing = standardzied_Dataset[percentage:]
print("Training sample: ",len(X_training))
print("Testing sample: ",len(X_testing))

X_train_raw = data_returns[:percentage]
X_test_raw = data_returns[percentage:]
print("Raw Data Training sample: ", len(X_train_raw))
print("Raw Data Testing sample: ", len(X_test_raw))

stock_tickers = standardzied_Dataset.columns.values
n_tickers = len(stock_tickers)
print("Number of tickers after data cleaning", n_tickers)
```
```
Training sample:  3256
Testing sample:  815
Raw Data Training sample:  3256
Raw Data Testing sample:  815
Number of tickers after data cleaning 27
```

4.2. Model Evaluation by Applying Principle Component Analysis (PCA)

Below we create a function to compute PCA from sklearn library using the training sample. This function will compute an inversed elbow chart that shows the amount of principle components and how many of them explain the variance threshold.

In [63]:
```
pca = PCA()
Principal_Component=pca.fit(X_training)
```

First Principal Component / Eigenvector

In [64]:
```
print(pca.components_[0])
print(len(pca.components_[0]))
```

```
[0.22527844 0.23148387 0.16932139 0.15712119 0.19157568 0.20525556
 0.17997341 0.20040529 0.17093388 0.21374475 0.2087245  0.23550339
 0.20438752 0.19918021 0.17139448 0.23519283 0.15771199 0.17288519
 0.20033565 0.17798975 0.17036088 0.20201904 0.14341466 0.17181737
 0.16785209 0.17413498 0.21413697]
27
```

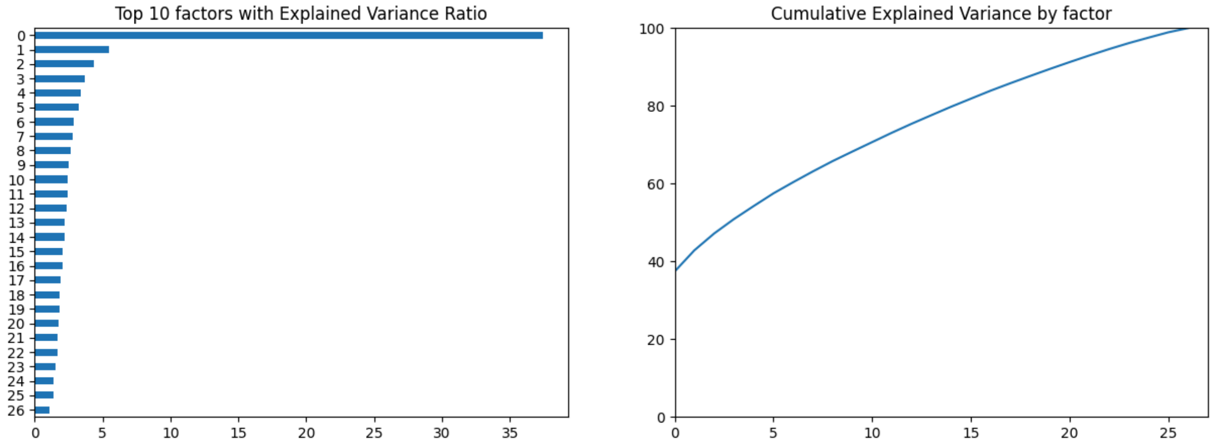
4.2.1. Explained Variance using PCA

In [76]:

```
Num_Eigenvalues=100
fig, axes = plt.subplots(ncols=2, figsize=(15,5))

Series1 = pd.Series(pca.explained_variance_ratio_[:Num_Eigenvalues]).sort_va
Series2 = pd.Series(pca.explained_variance_ratio_[:Num_Eigenvalues]).cumsum(

Series1.plot.barh(ylim=(0,27), label="woohoo",title='Top 10 factors with Exp
Series2.plot(ylim=(0,100),xlim=(0,27),ax=axes[1], title='Cumulative Explaine

# explained_variance
pd.Series(np.cumsum(pca.explained_variance_ratio_)).to_frame('Explained Vari
```

Out[76]:

	Explained Variance
0	37.41%
1	42.86%
2	47.18%
3	50.88%
4	54.23%
5	57.49%
6	60.37%
7	63.15%
8	65.80%
9	68.26%
10	70.68%
11	73.09%
12	75.40%
13	77.61%
14	79.78%
15	81.85%
16	83.91%
17	85.83%
18	87.68%
19	89.50%
20	91.26%
21	92.97%
22	94.61%
23	96.15%
24	97.55%
25	98.92%
26	100.00%



We can see from above two plots that factor 0 explains around 39%-40% of the daily return variation. We call such factor as the dominant factor, which is usually interpreted as ‘The Market’, depending on the results of closer inspection.

The plot on the right hand side shows the cumulative explained variance in a curve and indicates that 10 factors explain around 68% of the returns of our cross-section of stocks.

4.2.2. Portfolio Weights

Now we will compute and determine the weights of each principle component, and then we can visualize a scatterplot such that we can see an organized descending plot with the respective weight of every stock at the current chosen principle component.

In [66]:
```
def PCWeights():

    # 27 weights for each Principal Component
    weights = pd.DataFrame()

    for i in range(len(pca.components_)):
        weights["weights_{}".format(i)] = pca.components_[i] / sum(pca.compo
        #print(weights)

    weights = weights.values.T
    return weights

weights=PCWeights()
print(weights, '\n\n', len(weights))
```

```
[[ 4.37252600e-02  4.49296987e-02  3.28643154e-02  3.04963275e-02
   3.71837470e-02  3.98389333e-02  3.49318114e-02  3.88975230e-02
   3.31772911e-02  4.14866364e-02  4.05122346e-02  4.57098649e-02
   3.96704514e-02  3.86597416e-02  3.32666909e-02  4.56495857e-02
   3.06109971e-02  3.35560296e-02  3.88840060e-02  3.45467946e-02
   3.30660745e-02  3.92107425e-02  2.78359681e-02  3.33487707e-02
   3.25791337e-02  3.37986065e-02  4.15627637e-02]
 [ 7.37404850e-02 -2.25718542e-01  1.95092687e-01 -1.01651553e+00
  -1.16802566e-02 -3.52649753e-01  1.57861167e-01 -1.03672525e+00
   8.90246599e-01 -5.21163241e-01  1.68967195e-01 -1.96919335e-01
  -4.70362622e-01 -9.68566941e-01  1.00803363e+00 -3.90317766e-01
   5.04354724e-01  8.73423251e-01 -5.84724102e-01  8.42458299e-02
   9.91220024e-01  2.41383125e-01  4.95712902e-01  3.85676880e-01
   4.52697791e-01  4.69264155e-01 -2.16577103e-01]
 [-2.66393506e-01 -7.29287495e-01  7.77515578e-01  7.39360827e-01
  -5.55629778e-01 -6.53742855e-01 -4.78252262e-01  7.39966650e-01
   4.36309093e-01 -9.19679827e-01 -3.01381032e-02 -5.02498649e-01
   6.05391443e-01  8.43990754e-01  3.44864063e-01 -9.29033334e-01
   2.22548834e-01  2.79899806e-01  7.93825075e-01 -1.88549801e-01
   4.06231621e-01 -4.75561225e-01 -1.57533201e-01  1.80072614e-01
   1.85138265e-01  4.70840469e-01 -1.39655056e-01]
 [ 1.36738014e+00 -1.73364936e+00  1.93488022e+00  1.07942993e+00
   1.43194492e+00  2.18649617e+00  4.51075596e+00  4.64721834e-02
   4.24234327e-01 -8.15510162e-01 -4.35583839e+00  1.69083154e+00
   5.97431852e-01  1.49959126e-01  2.03417777e+00 -1.37779202e+00
  -1.52241387e+00  2.18537466e+00  4.29251673e-01 -2.83081334e+00
   3.23452846e-02 -7.93415176e-01  2.43371746e+00 -9.83042805e-01
  -1.50864168e+00 -4.95207375e+00 -6.61492657e-01]
 [ 3.07323116e+00 -3.18300160e+00 -4.63264127e+00  1.81839436e+00
   3.90667280e+00  4.89238396e+00  3.68106678e+00 -3.86544139e-01
   2.40467192e+00 -5.00986404e+00  4.82393556e-01  3.07008241e+00
   3.50849715e-01 -6.10630528e-01 -2.89190715e+00 -5.17921491e+00
   5.99720096e+00 -5.59892958e+00 -1.39240076e+00  3.99812082e+00
   1.89651702e+00 -2.20019941e+00 -3.50375890e+00 -1.74621358e+00
   1.49304515e-01  1.30190039e+00  3.12515496e-01]
 [ 1.10718634e-02 -3.75098305e-01  1.25162805e+00  7.53510724e-01
   2.44127354e-01  7.32751774e-02 -4.26084101e-01 -3.33660694e-01
  -1.23897653e+00 -1.61535444e-01  1.04843933e+00  1.77434266e-01
  -4.22853268e-01 -3.34525961e-01 -1.44496915e-01 -4.47417741e-01
   3.34578819e-01 -1.10413966e-01 -3.56617629e-01  1.11481398e+00
  -1.02449976e+00 -8.44366990e-01  2.87951717e+00 -2.09143221e+00
   1.19601730e+00  3.53825143e-01 -1.26259665e-01]
 [-5.62535905e-01  6.08946317e-01  9.08799526e-02  1.39586650e+00
  -4.33951032e-01 -1.04788554e+00 -3.40612069e-01  3.66793167e-01
   9.64222290e-01  7.57666918e-01 -1.06835789e+00 -1.04042524e+00
  -4.13516125e-01 -2.15261841e-01 -7.13355682e-01  6.26471902e-01
   3.49715099e+00 -7.32357579e-01 -3.80428984e-01 -4.84097316e-01
   5.47263229e-01  1.66099463e+00  1.05439468e+00 -5.23528074e-01
  -6.64123101e-01 -1.42805780e+00 -5.22156394e-01]
 [ 1.40919252e-01 -9.12420557e-01 -6.72809085e+00  6.42946972e-01
  -2.32175572e+00  1.16506520e-01  5.53222689e-01  5.76334803e-02
   5.50251423e+00 -8.91354793e-01 -1.12721218e-02 -8.95164278e-01
   2.31627163e+00  1.20874716e+00 -1.45397567e+00  6.85135630e-01
  -6.37249087e+00 -4.92291909e+00  1.72514363e+00 -2.22796452e-01
   2.05905306e+00  2.78525808e+00  7.93665412e+00  7.86056027e-01
   1.41271235e+00  9.72974760e-01 -3.16950919e+00]
```

```
[ 5.72997095e-02  9.01657184e-01 -2.17933267e+00  6.47325815e+00
 -2.59521871e+00  1.09737056e+00  3.02624752e+00 -1.46908291e+00
 -5.53138909e-01  2.24380174e+00  3.78668190e-01 -2.35969598e+00
 -7.43474705e-01 -5.98420945e-01  1.84550561e+00  1.43409007e+00
 -1.47799844e+00  2.11487939e+00 -1.91726131e+00 -1.08710303e+00
  2.34174472e+00 -6.72011302e-01 -4.48262696e+00 -3.68885171e+00
  6.05149139e+00  5.81000175e-01 -3.72279683e+00]
[-3.96385892e-01  3.65892029e-01 -5.55050736e-01  3.13981988e+00
  3.76161685e-01 -1.69729893e-01 -6.92101823e-01 -1.05706691e+00
 -2.24229266e-01 -3.11781026e-01  7.37139009e-03  1.64558521e-01
 -5.72501229e-01 -1.08883613e+00 -3.47412671e-01 -4.92576143e-01
 -5.53585985e-01  6.92949679e-01 -7.47232480e-01  7.55607518e-01
 -2.01007613e-01 -8.94606584e-01  7.31191072e-01  3.09675311e+00
 -5.24466597e-01 -5.11597183e-01  1.00986326e+00]
[ 1.04958610e+00 -1.90106482e+00  3.58166680e+00  1.08383760e+00
  5.26291303e-01  1.26994448e+00  1.64836368e+00 -1.93094192e-02
 -5.41312371e+00 -1.17090576e+00 -6.64863664e-02  2.32532922e-01
 -5.77573786e-01 -6.13990319e-01 -3.87926386e-01 -6.72735088e-01
 -1.87804565e-01 -3.31203790e+00 -1.34745073e+00 -2.91407536e+00
  5.45265547e-01  5.57600778e+00  1.92661680e-01  3.81519161e+00
  1.38155479e-01  3.66139025e+00 -3.73641103e+00]
[-1.18496600e-01  7.01703581e-01 -2.70804026e-01 -4.90015273e-01
 -8.47052078e+00  1.72485718e+00  8.50286063e+00 -1.95414340e+00
 -1.80619554e+00  2.68270447e-01  2.32142627e+00 -4.24108648e+00
  3.00050341e+00 -8.99774041e-01 -3.33525642e-01  6.94461911e-01
  2.43669738e+00  1.80064983e+00  1.62644371e+00  3.96493337e+00
 -2.55006919e+00 -2.07917510e+00  1.46360844e+00  1.02102678e+00
 -7.38920768e+00  2.98993095e+00 -9.14360149e-01]
[-9.35354946e-01  1.55318472e-01 -3.92772690e+00 -3.22132888e+00
 -6.57078734e-01 -2.05819216e-01  2.02090396e+00  1.05373159e+00
 -4.42326193e+00  2.66535271e-02 -1.72489593e+00 -1.10587054e+00
  1.87644088e+00  1.30511589e-01 -1.26456359e+00 -1.73500907e-01
  4.03921385e+00  5.26901522e-01  2.07925164e+00 -2.89032148e+00
 -6.98439825e-01 -3.11115098e+00  2.16156112e+00  4.14504451e+00
  6.92447748e+00 -1.20467277e+00  1.40397651e+00]
[ 7.65396296e-02 -9.64117201e-01 -6.30124501e-01 -6.76124171e-01
  9.71742245e-01 -9.29460702e-01 -2.85228046e-01  1.34677687e+00
 -1.73241800e+00  7.96058683e-01 -3.12095880e+00 -1.01156894e+00
 -6.96136676e-01  5.45754930e-01  2.39131623e+00  2.50296191e-01
 -3.47662109e-01 -2.67158835e-01  8.16928608e-01  6.00382587e+00
  3.02887125e-01  5.86228694e-01 -3.46416870e-01  1.01276706e+00
  4.64621514e-01 -1.09382284e+00 -2.46454596e+00]
[-3.41340067e+00  3.51834831e+00  2.23864458e+01 -3.06790553e+00
 -1.61965058e+00 -1.66817001e-01  5.48456582e+00 -1.14993846e+01
  1.10212592e+01  5.18557894e+00 -2.22686019e+00 -2.04162722e+00
  3.56334695e+00 -5.52542804e+00 -9.79446553e+00  4.07375016e+00
 -3.57282581e+00 -1.69363096e+01  6.66380058e+00  5.45243464e+00
  1.73718784e+00 -9.98012152e+00 -5.23096702e+00  6.93173005e+00
  1.04915151e+01 -7.32474088e+00 -3.10945914e+00]
[-2.02578338e+00  5.59548333e-01 -9.79549583e-01 -4.93482614e-02
  5.24890447e+00  1.13291917e+00 -1.38953649e+00 -1.44584947e+00
  1.36198926e+00  2.40445786e+00 -1.70218339e-01  1.61557504e+00
 -1.00308715e+00  1.78544511e+00 -1.48991462e+00  2.13718896e+00
  2.24912667e+00  2.54357348e+00  2.41490112e+00 -2.32601509e+00
 -3.62401844e-01 -5.63283165e+00  1.13869540e+00  1.21543677e+00
 -2.93275911e+00  4.02615433e+00 -9.02662100e+00]
```

```
[-3.22086935e+00 -1.91042964e+00 -3.87473712e-02 -4.57470399e-02
 -1.61332271e-01  1.88566575e+00  1.34632763e+00 -2.06847194e+00
  5.47343145e+00 -3.45794714e+00  3.45844420e-01  1.14540696e+00
 -1.86520890e+00  1.57350766e+00 -4.64159473e+00 -1.49302162e+00
  6.45540987e-01  6.62247412e+00  2.56164712e+00  2.29906173e+00
 -1.22320226e+01  8.09700578e+00 -2.30583768e+00  1.32580961e+00
  4.90101823e+00 -1.06857955e+00 -2.71293164e+00]
[ 7.82903091e+00  1.65547816e+00  3.15184545e+00 -3.89978517e+00
 -7.71107613e+00  5.83538285e+00  3.00966438e+00  1.53471783e+01
  6.56516220e+00 -9.82452530e-01  8.69405028e+00 -4.07402018e+00
 -2.27001444e+01  1.01039179e+01  9.13228556e-01  4.39907041e-01
 -2.12407507e-02 -2.90130653e+00 -1.05822966e+01 -1.33807406e+00
 -2.70696144e+00 -8.05996291e+00  2.24177223e+00  8.73393923e+00
  1.93514641e+00 -6.08543988e+00 -4.39294338e+00]
[-5.65870882e-01 -8.54687905e-01  2.54837036e-02  9.64591371e-01
  7.43700339e-01 -1.19603556e+00  2.12367196e+00  1.25267635e+00
  1.74139575e+00  1.00271373e+00 -4.31256582e+00 -6.71990644e-01
 -3.54863454e+00 -1.58504377e+00  1.10378526e+00  1.03347461e+00
 -2.96666617e-01 -1.47972185e+00  1.20712928e+00 -8.89271553e-01
 -2.27518442e+00 -8.40601665e-01  2.87298515e-01 -6.94504648e-01
  2.78827111e-01  5.16019325e+00  3.28583865e+00]
[-2.68797981e+00 -2.92910706e+00  6.71504632e-01 -6.82568633e-01
  2.04276293e+00 -3.80365545e+00  4.71304579e+00  1.44093286e+00
 -1.67167291e+00  2.60296343e-01  5.61119755e-01 -2.73583985e-02
 -3.55883153e+00  1.84826057e+00 -9.38618733e+00  6.10007575e-01
 -2.30880931e+00  4.55180408e+00  1.03736828e-01  2.07878149e+00
  6.93106503e+00  9.95697043e-01  5.31226884e-01 -1.05738197e+00
 -3.19916490e-01  2.77839253e-01  1.81538782e+00]
[-3.39416090e+00 -1.07437050e+00  3.16743866e-01 -2.09146488e-01
  3.53902761e+00 -3.97898439e+00  3.48296774e+00  3.22811803e+00
  1.71299382e+00  5.56431808e-01  1.61074032e+00 -5.77231290e-01
  5.18150365e+00  1.42254704e-01  1.22958689e+00  8.36052951e-01
 -4.43299118e-01 -7.75698373e-01 -7.48606142e+00  1.57310140e-01
 -2.74619642e+00 -9.15108976e-01 -1.22039579e-01  8.26199229e-01
  2.70156151e-01  2.95724114e-01 -6.63513551e-01]
[ 4.80957734e+00 -9.11143849e+00  4.41483844e+00 -2.81755812e+00
 -1.26474694e+01  1.46699565e+01 -1.24582785e+01  1.11226442e+00
  3.70944891e+00  8.27109890e+00 -1.77808898e+01  3.63139540e+00
  9.76402939e+00  3.09055304e+00 -1.12612152e+01  2.42776049e+00
  1.25620317e+00  7.94783589e+00 -1.80087373e+01  3.59480224e+00
  2.28269507e+00 -2.64758770e+00  2.20581217e+00  1.64672679e+00
  2.63122586e+00  8.59345101e+00  1.67349960e+00]
[ 1.74050733e+00 -5.77141340e+00 -4.59309939e-01  1.44859191e-01
  2.73851628e+00  1.07587388e+00 -4.57743641e-01 -3.74772652e+00
  2.50593771e-01  3.54561560e+00  1.95280497e+00 -5.35580527e+00
 -8.94669258e-02  3.33310931e+00  8.83418981e-01  4.97223177e-01
  4.35749785e-01 -4.82764254e-01  3.04849307e-01 -7.77633238e-01
 -6.98633399e-01  3.96983474e-01  5.86989887e-03  6.01824573e-01
 -4.01357144e-01 -8.57396066e-01  2.19145028e+00]
[ 2.25506287e+01  4.66934899e+01  5.26587016e+00 -3.04604727e+00
  2.40524599e+01  2.92731033e+00  5.21128585e-01 -9.64676575e+00
 -1.79303644e+00 -2.71002843e+01 -1.97531331e+01 -4.36049276e+01
  7.08337968e+00  1.88954757e+01 -1.35843829e+01 -6.71221430e+00
 -7.73865818e+00  6.52370178e+00 -1.14848108e+01  1.43975171e+00
 -2.48991410e+00  8.12782031e-01  3.66035906e+00 -5.18850156e+00
  9.32875288e-01  1.00167793e+01  1.76668423e+00]
```

```
[-3.51506094e+00 -1.10398083e-01  4.21661647e-01 -2.99289550e-01
  1.52371487e+00  4.74554869e+00 -8.43503251e-01  3.21917137e+00
  3.37785318e-01  3.78072581e-02  8.75165510e-01 -3.44869428e+00
  2.85015892e-01 -3.62203158e+00 -3.70398433e-02 -5.01612328e-01
 -5.83832208e-01  4.99947723e-01  9.33989820e-01 -3.61826621e-03
  9.80821015e-01  4.29056138e-01  1.62654564e-01 -1.83839580e-01
 -2.14975584e-01 -3.41228855e-01  2.52784536e-01]
[-7.80327184e+00  1.80537187e+00  9.21804431e-02 -2.87581336e-01
 -7.66727811e-01  3.04620023e+00  7.02847623e-01 -3.27808518e+00
 -8.60101389e-01 -1.17236412e+00 -1.09093570e+00  1.53271564e+00
 -1.56097542e+00  6.08923756e+00  2.69726323e+00 -2.30533055e-01
  4.22095900e-01 -1.78459021e+00 -1.59721682e+00  4.63631318e-01
  1.52724739e+00  5.16222386e-01  5.65625563e-01 -2.02900704e-01
 -5.48627685e-02  3.84887051e-01  1.84462014e+00]
[-3.59651418e-01  3.97195067e+00 -3.37212607e-01 -9.52956125e-01
 -1.16867756e-01 -1.02728957e+00  1.13455337e+00  4.04841088e-01
  7.16442880e-01  1.01790829e+01  3.44306201e-02  4.79603933e-01
 -2.98577933e-01  6.36128268e-01 -6.43172041e-01 -1.37899547e+01
 -8.72406311e-01 -9.63830509e-02 -1.75006264e-02 -2.11488418e-01
 -3.35765902e-01  1.36862149e+00  3.64047868e-01  2.70057295e-01
 -3.96101970e-02  1.13548403e+00 -5.96407720e-01]]

27
```

In [67]:

```
weights[0]
```

Out[67]:

```
array([0.04372526, 0.0449297 , 0.03286432, 0.03049633, 0.03718375,
       0.03983893, 0.03493181, 0.03889752, 0.03317729, 0.04148664,
       0.04051223, 0.04570986, 0.03967045, 0.03865974, 0.03326669,
       0.04564959, 0.030611  , 0.03355603, 0.03888401, 0.03454679,
       0.03306607, 0.03921074, 0.02783597, 0.03334877, 0.03257913,
       0.03379861, 0.04156276])
```

In [68]:

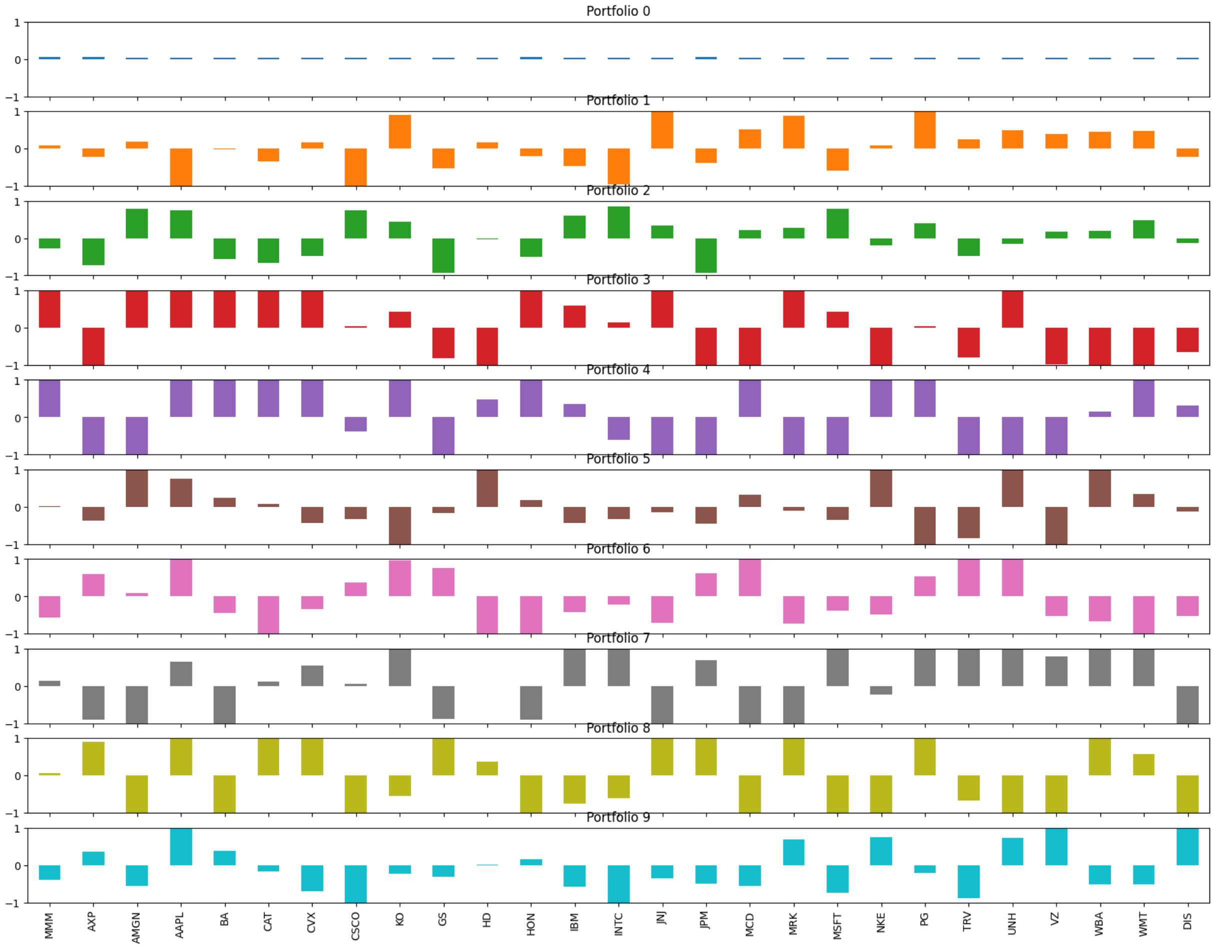
```
print(pca.components_[0])# component loadings (which represent the contribut
print(sum(pca.components_[0])) #This makes the normalized loadings for a sir
print(pca.components_[0]/sum(pca.components_[0])) #weights
```

```
[0.22527844 0.23148387 0.16932139 0.15712119 0.19157568 0.20525556
 0.17997341 0.20040529 0.17093388 0.21374475 0.2087245  0.23550339
 0.20438752 0.19918021 0.17139448 0.23519283 0.15771199 0.17288519
 0.20033565 0.17798975 0.17036088 0.20201904 0.14341466 0.17181737
 0.16785209 0.17413498 0.21413697]
5.152134928134312
[0.04372526 0.0449297  0.03286432 0.03049633 0.03718375 0.03983893
 0.03493181 0.03889752 0.03317729 0.04148664 0.04051223 0.04570986
 0.03967045 0.03865974 0.03326669 0.04564959 0.030611   0.03355603
 0.03888401 0.03454679 0.03306607 0.03921074 0.02783597 0.03334877
 0.03257913 0.03379861 0.04156276]
```

In [69]:

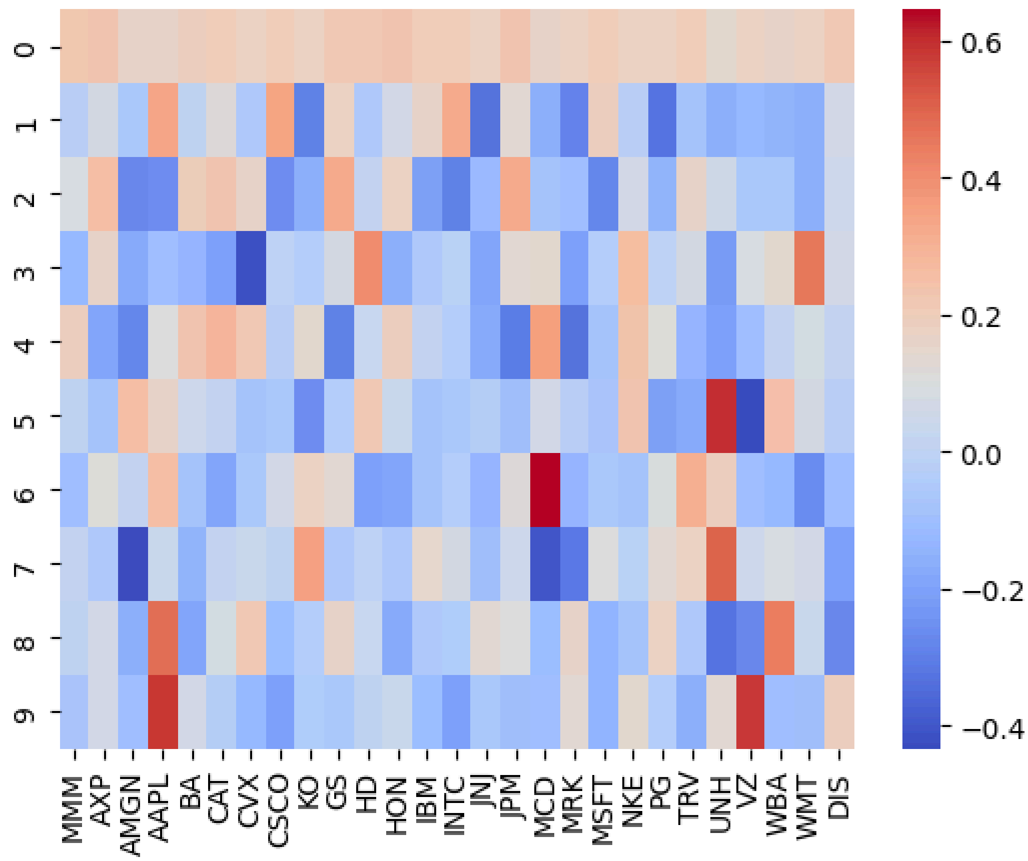
```
Num_Components = 10 # num of top portfolio

top_Portfolios = pd.DataFrame(pca.components_[:Num_Components], columns=data
eigen_portfolios = top_Portfolios.div(top_Portfolios.sum(1), axis=0)
eigen_portfolios.index = [f'Portfolio {i}' for i in range(Num_Components)]
np.sqrt(pca.explained_variance_)
eigen_portfolios.T.plot.bar(subplots=True, layout=(int(Num_Components),1), f
```

Out[69]:
```
array([[<Axes: title={'center': 'Portfolio 0'}>],
       [<Axes: title={'center': 'Portfolio 1'}>],
       [<Axes: title={'center': 'Portfolio 2'}>],
       [<Axes: title={'center': 'Portfolio 3'}>],
       [<Axes: title={'center': 'Portfolio 4'}>],
       [<Axes: title={'center': 'Portfolio 5'}>],
       [<Axes: title={'center': 'Portfolio 6'}>],
       [<Axes: title={'center': 'Portfolio 7'}>],
       [<Axes: title={'center': 'Portfolio 8'}>],
       [<Axes: title={'center': 'Portfolio 9'}>]], dtype=object)
```



In [70]:
```
# plotting heatmap
sns.heatmap(top_Portfolios, cmap='coolwarm')
```

Out[70]: <Axes: >



The plots and the heatmap above shown the contributions of different stocks in each eigenvector.

4.2.3. Finding the Best Eigen Portfolio

In order to find the best eigen portfolios and perform backtesting in the next step, we use the sharpe ratio. A higher sharpe ratio explains higher returns and lower risk for one particular portfolio.

The annualized sharpe ratio is computed by dividing the annualized log returns against the annualized risk. For annualized log return we apply the geometric average of all the returns in respect to the number of trading days per year. Annualized risk is computed by taking the standard deviation of the returns and multiplying it by the square root of the number of trading days per year.

In [71]:
```
# Sharpe Ratio

def sharpe_ratio(daily_returns, trading_days=252):

    # Sharpe ratio is the average return earned in excess of the risk-free r
    # It calculares the annualized return, annualized volatility, and annual
    # daily_returns is returns of a signle eigen portfolio.

    n_years = daily_returns.shape[0] / trading_days
    annualized_return = np.power(np.prod(1 + daily_returns), (1/n_years)) -
```

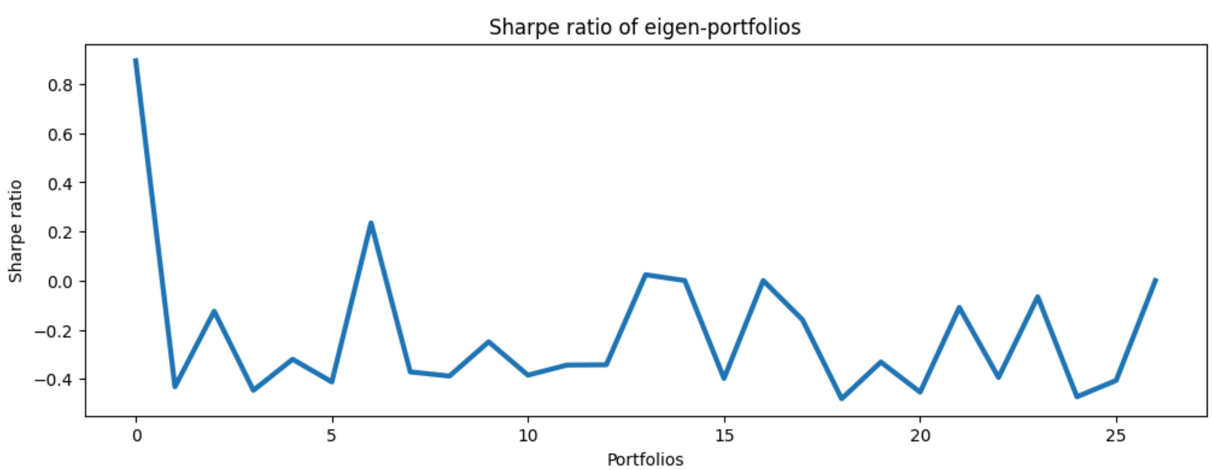
```
    annualized_vol = daily_returns.std() * np.sqrt(trading_days)
    annualized_sharpe = annualized_return / annualized_vol

    return annualized_return, annualized_vol, annualized_sharpe
```

We construct a loop to compute the principle component's weights for each eigen portfolio, which then uses the sharpe ratio function to look for the portfolio with the highest sharpe ratio. Once we know which portfolio has the highest sharpe ratio, we can visualize its performance against the DJIA Index for comparison.

In [72]:
```
def optimized_Portfolio():
    n_portfolios = len(pca.components_)
    #print(n_portfolios)
    annualized_ret = np.array([0.] * n_portfolios)
    sharpe_metric = np.array([0.] * n_portfolios)
    annualized_vol = np.array([0.] * n_portfolios)
    highest_sharpe = 0
    stock_tickers = standardzied_Dataset.columns.values
    n_tickers = len(stock_tickers)
    pcs = pca.components_

    for i in range(n_portfolios):

        pc_w = pcs[i] / sum(pcs[i])
        eigen_prtfi = pd.DataFrame(data ={'weights': pc_w.squeeze()*100}, ir
        eigen_prtfi.sort_values(by=['weights'], ascending=False, inplace=Tru
        eigen_prti_returns = np.dot(X_train_raw.loc[:, eigen_prtfi.index], p
        eigen_prti_returns = pd.Series(eigen_prti_returns.squeeze(), index=X
        er, vol, sharpe = sharpe_ratio(eigen_prti_returns)
        #print(er)
        #print(vol)
        #print(sharpe)
        annualized_ret[i] = er
        annualized_vol[i] = vol
        sharpe_metric[i] = sharpe

        sharpe_metric= np.nan_to_num(sharpe_metric.astype(float))

    # find portfolio with the highest Sharpe ratio
    highest_sharpe = np.argmax(sharpe_metric)

    print('Eigen portfolio #%d with the highest Sharpe. Return %.2f%%, vol =
          (highest_sharpe,
           annualized_ret[highest_sharpe]*100,
           annualized_vol[highest_sharpe]*100,
           sharpe_metric[highest_sharpe]))


    fig, ax = plt.subplots()
    fig.set_size_inches(12, 4)
    ax.plot(sharpe_metric, linewidth=3)
    ax.set_title('Sharpe ratio of eigen-portfolios')
    ax.set_ylabel('Sharpe ratio')
```

```
    ax.set_xlabel('Portfolios')

    results = pd.DataFrame(data={'Return': annualized_ret, 'Vol': annualized
    results.dropna(inplace=True)
    results.sort_values(by=['Sharpe'], ascending=False, inplace=True)
    print(results.head(25))

    plt.show()

optimized_Portfolio()
```

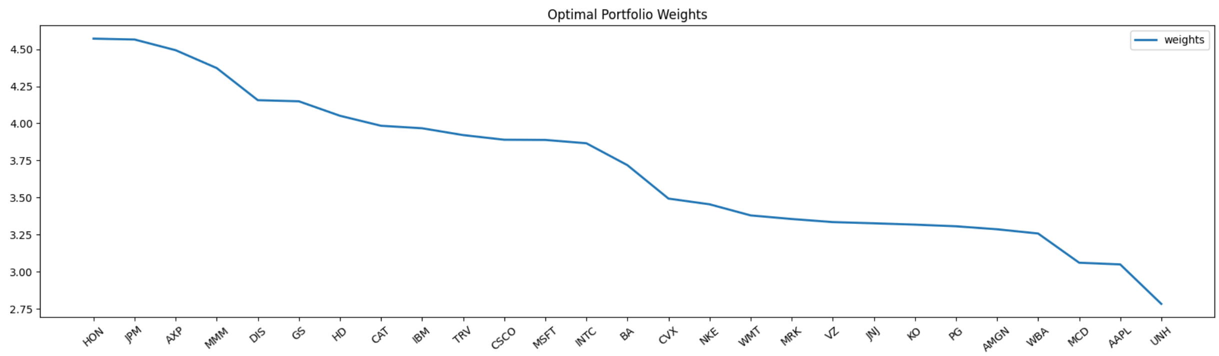
```
Eigen portfolio #0 with the highest Sharpe. Return 12.09%, vol = 13.50%, Sha
rpe = 0.89
    Return      Vol  Sharpe
0   0.1209   0.1350  0.8950
6   0.2274   0.9687  0.2347
13  0.0296   1.2507  0.0237
23 -1.0000  15.1406 -0.0660
21 -1.0000   9.1370 -0.1094
2  -0.0569   0.4554 -0.1249
17 -1.0000   6.2795 -0.1592
9  -0.2404   0.9645 -0.2493
4  -0.9992   3.1187 -0.3204
19 -0.9936   2.9956 -0.3317
12 -0.8865   2.5870 -0.3427
11 -0.9949   2.8924 -0.3440
7  -0.9715   2.6129 -0.3718
10 -0.7664   1.9902 -0.3851
8  -0.9889   2.5432 -0.3888
22 -0.7920   2.0083 -0.3944
15 -0.9637   2.4185 -0.3985
25 -0.9580   2.3568 -0.4065
5  -0.3664   0.8880 -0.4126
1  -0.2603   0.6021 -0.4324
3  -0.8614   1.9277 -0.4469
20 -0.9631   2.1198 -0.4544
24 -0.7896   1.6685 -0.4732
18 -0.8056   1.6741 -0.4812
```



In [73]:
```
weights = PCWeights()
portfolio = pd.DataFrame()
```

```
def optimal_port(weights, plot=False, portfolio=portfolio):
    portfolio = pd.DataFrame(data ={'weights': weights.squeeze()*100}, index
    portfolio.sort_values(by=['weights'], ascending=False, inplace=True)
    if plot:
        print('Sum of Portfolio Weights is: ', np.sum(portfolio))
        portfolio.plot(title='Optimal Portfolio Weights', figsize=(20,5), xt
            linewidth=2
            )
        plt.show()

    return portfolio

# Weights are stored in arrays, where 0 is the first PC's weights.
optimal_port(weights=weights[0], plot=True)
```

```
Sum of Portfolio Weights is:  weights    100.0
dtype: float64
```



Out[73]:

	weights
HON	4.5710
JPM	4.5650
AXP	4.4930
MMM	4.3725
DIS	4.1563
GS	4.1487
HD	4.0512
CAT	3.9839
IBM	3.9670
TRV	3.9211
CSCO	3.8898
MSFT	3.8884
INTC	3.8660
BA	3.7184
CVX	3.4932
NKE	3.4547
WMT	3.3799
MRK	3.3556
VZ	3.3349
JNJ	3.3267
KO	3.3177
PG	3.3066
AMGN	3.2864
WBA	3.2579
MCD	3.0611
AAPL	3.0496
UNH	2.7836

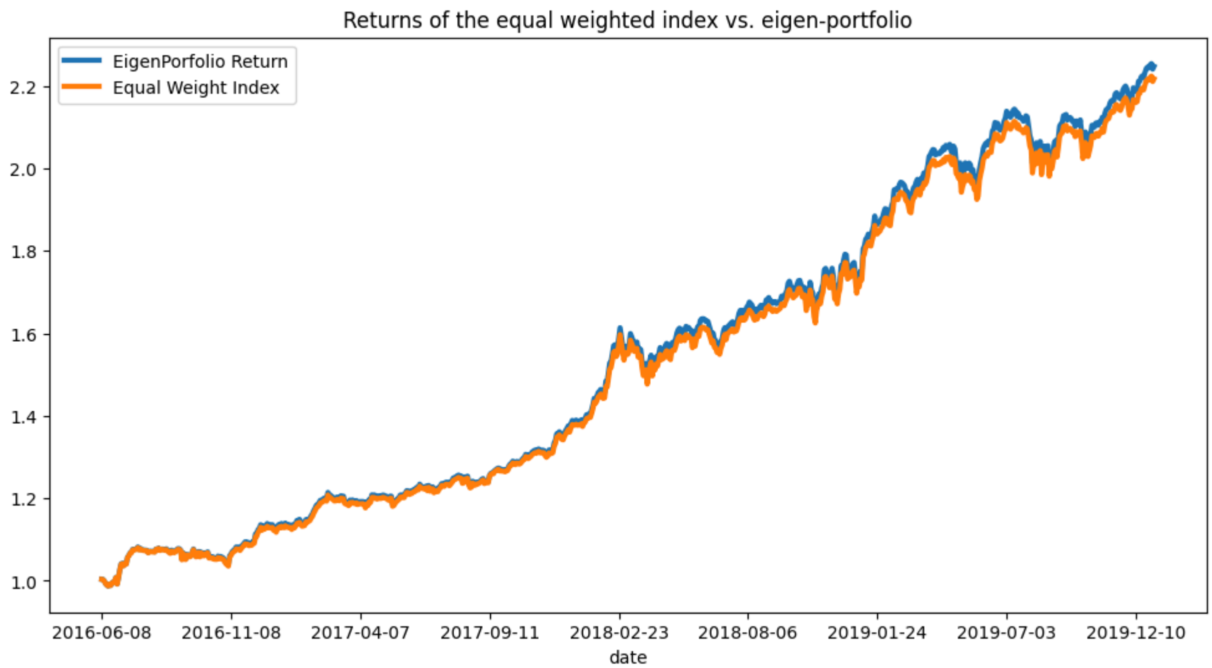
The chart shows the allocation of the best portfolio. The weights in the chart are in percentages.

4.2.4. Backtesting Eigenportfolio

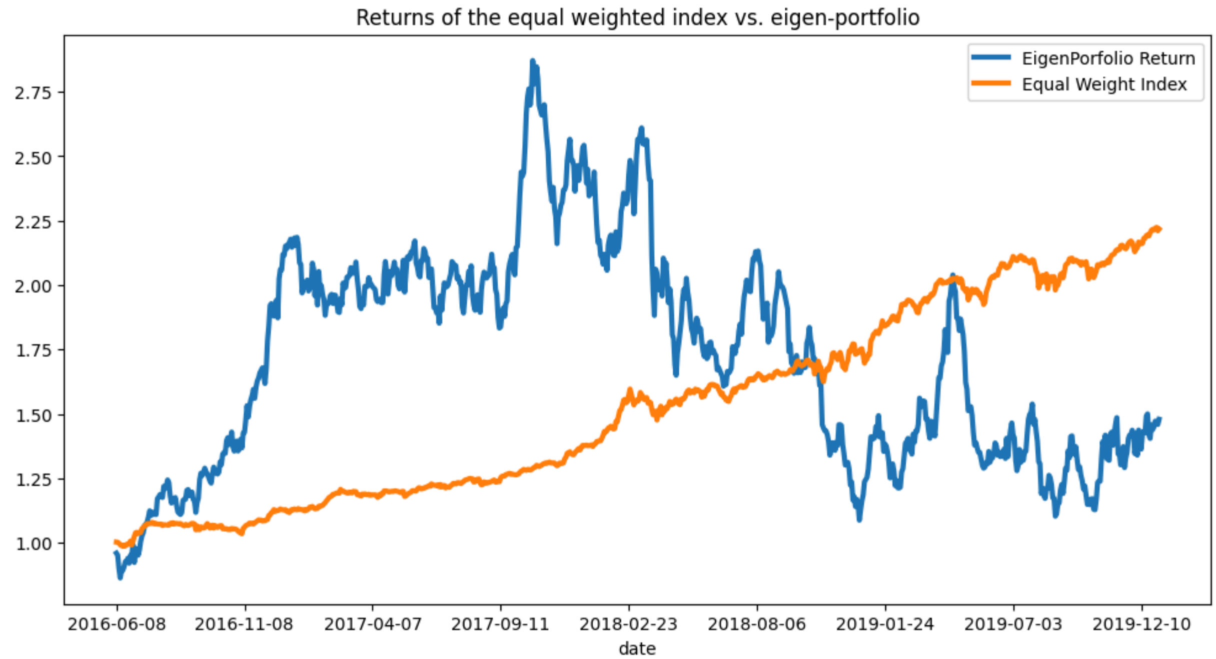
We will now try to backtest our model on the test set, by looking at few top and bottom portfolios.

In [74]:
```
def Backtest(eigen):

    # Plots Principle components returns against real return
    eigen_prtfi = pd.DataFrame(data ={'weights': eigen.squeeze()}, index = s
    eigen_prtfi.sort_values(by=['weights'], ascending=False, inplace=True)

    eigen_prti_returns = np.dot(X_test_raw.loc[:, eigen_prtfi.index], eigen)
    eigen_portfolio_returns = pd.Series(eigen_prti_returns.squeeze(), index=
    returns, vol, sharpe = sharpe_ratio(eigen_portfolio_returns)
    print('Return = %.2f%%\nVolatility = %.2f%%\nSharpe = %.2f' % (returns*1
    equal_weight_return=(X_test_raw * (1/len(pca.components_))).sum(axis=1)
    df_plot = pd.DataFrame({'EigenPorfolio Return': eigen_portfolio_returns,
    np.cumprod(df_plot + 1).plot(title='Returns of the equal weighted index
                          figsize=(12,6), linewidth=3)
    plt.show()

for j in range(len(pca.components_)):
    print('Eigen-Portfolio',j)
    Backtest(eigen=weights[j])


#Backtest(eigen=weights[0])
```

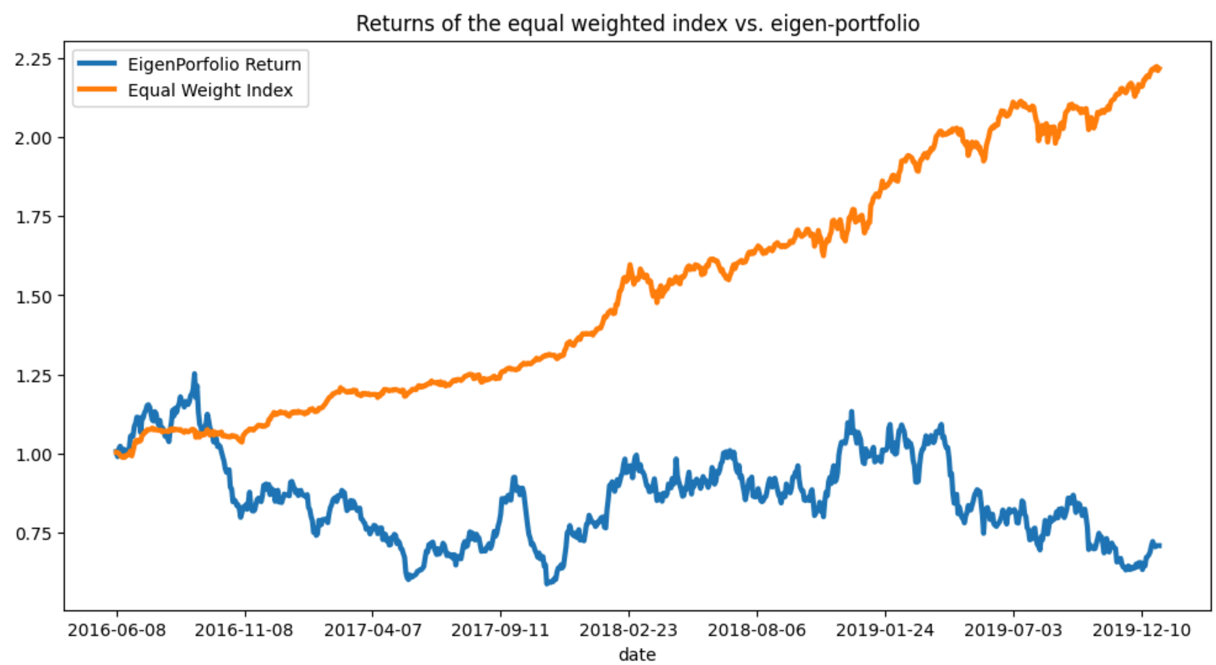
```
Eigen-Portfolio 0
Return = 28.46%
Volatility = 10.65%
Sharpe = 2.67
```



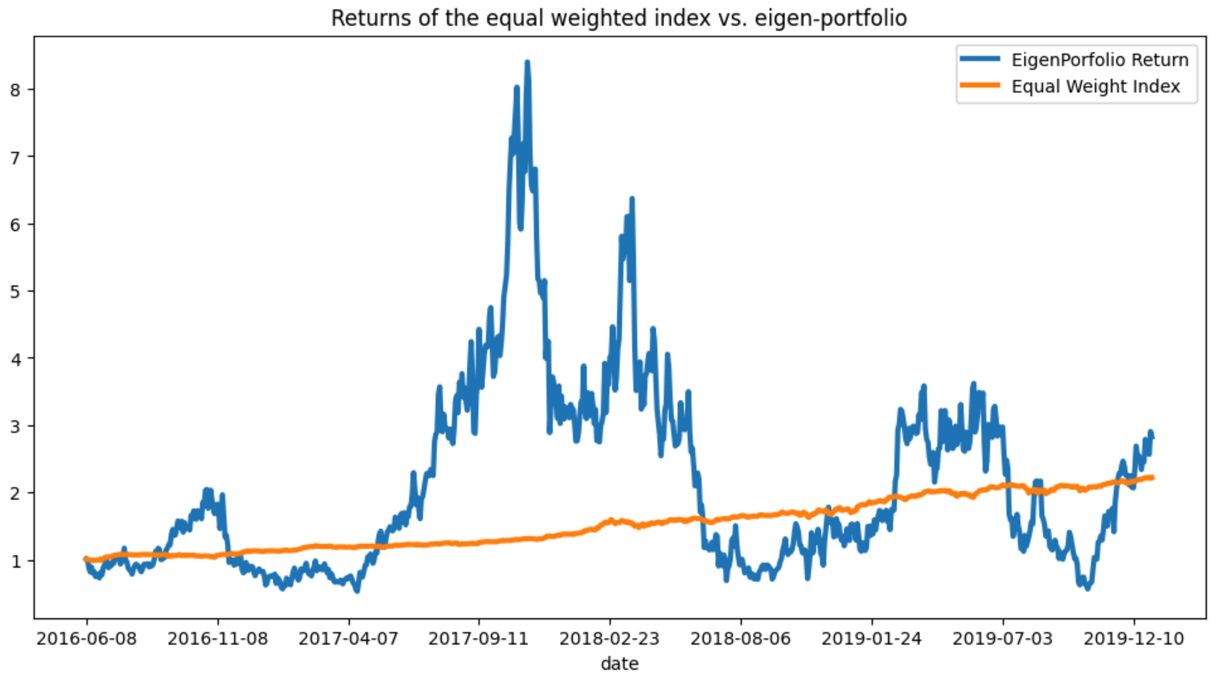

```
Eigen-Portfolio 1
Return = 12.89%
Volatility = 44.72%
Sharpe = 0.29
```



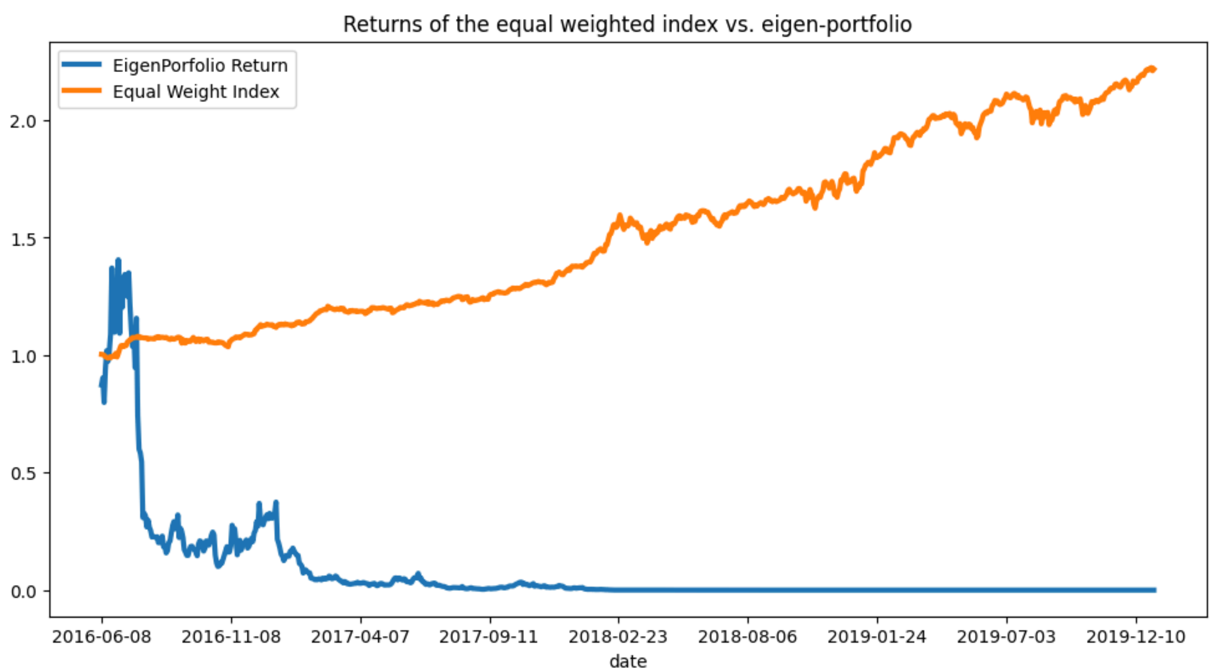

```
Eigen-Portfolio 2
Return = -10.12%
Volatility = 38.51%
Sharpe = -0.26
```



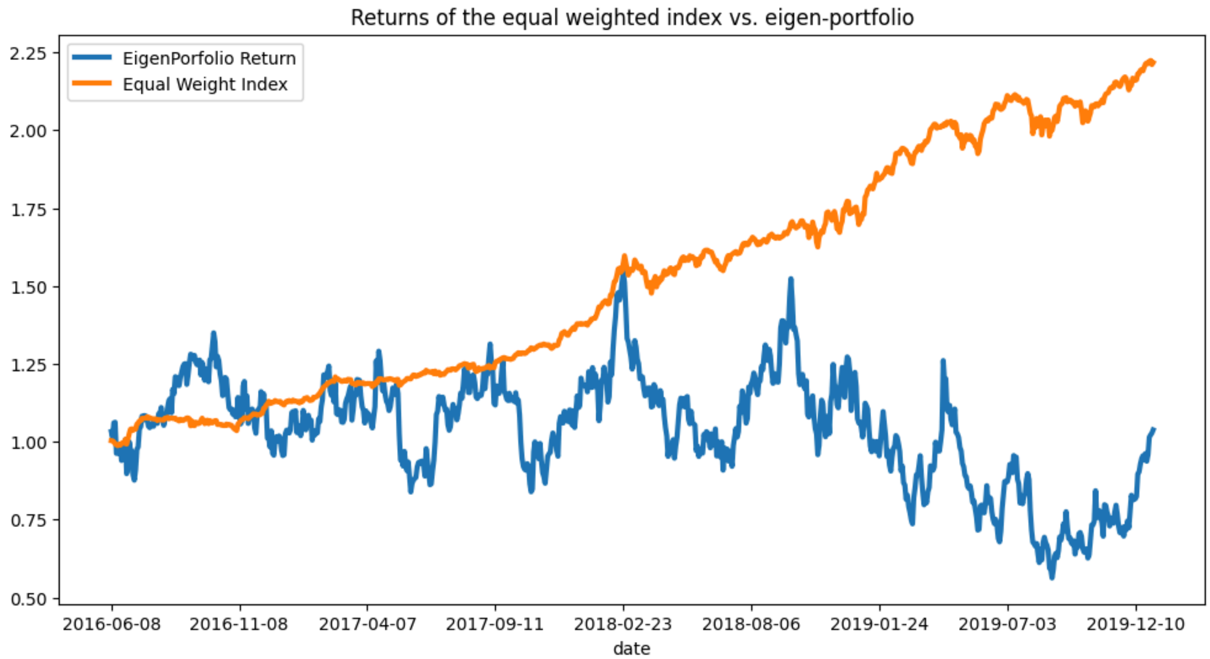

```
Eigen-Portfolio 3
Return = 37.84%
Volatility = 157.93%
Sharpe = 0.24
```



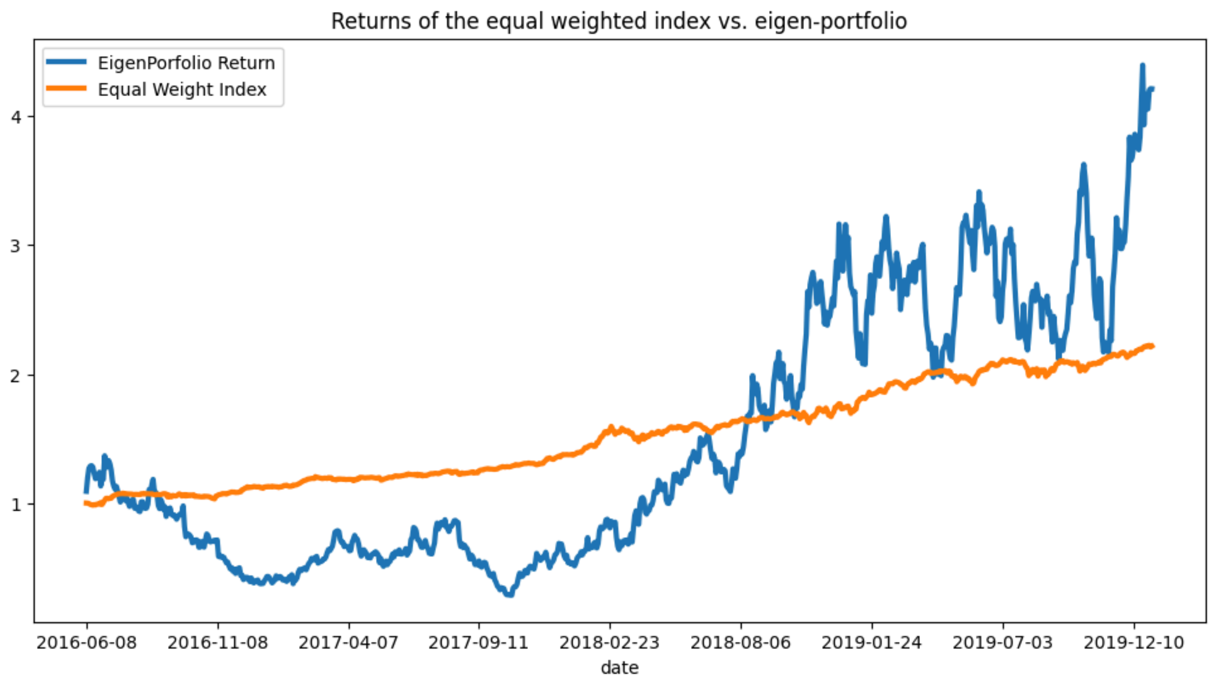

```
Eigen-Portfolio 4
Return = -99.89%
Volatility = 275.37%
Sharpe = -0.36
```



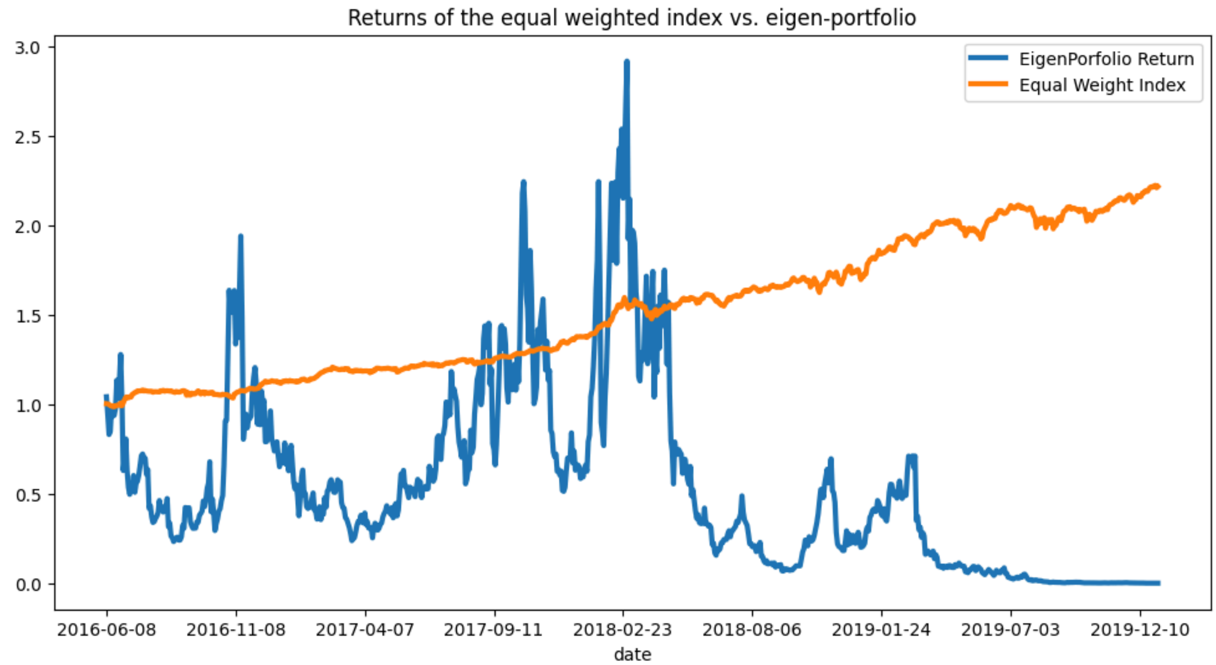

```
Eigen-Portfolio 5
Return = 1.16%
Volatility = 64.14%
Sharpe = 0.02
```



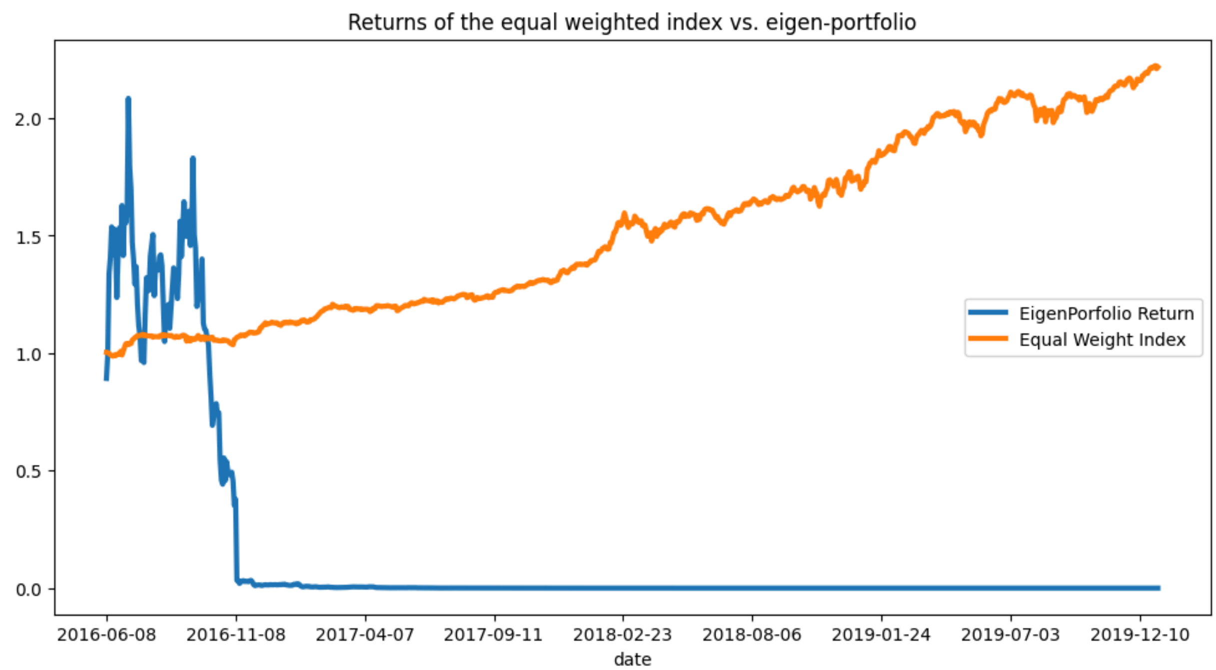

```
Eigen-Portfolio 6
Return = 55.94%
Volatility = 83.31%
Sharpe = 0.67
```



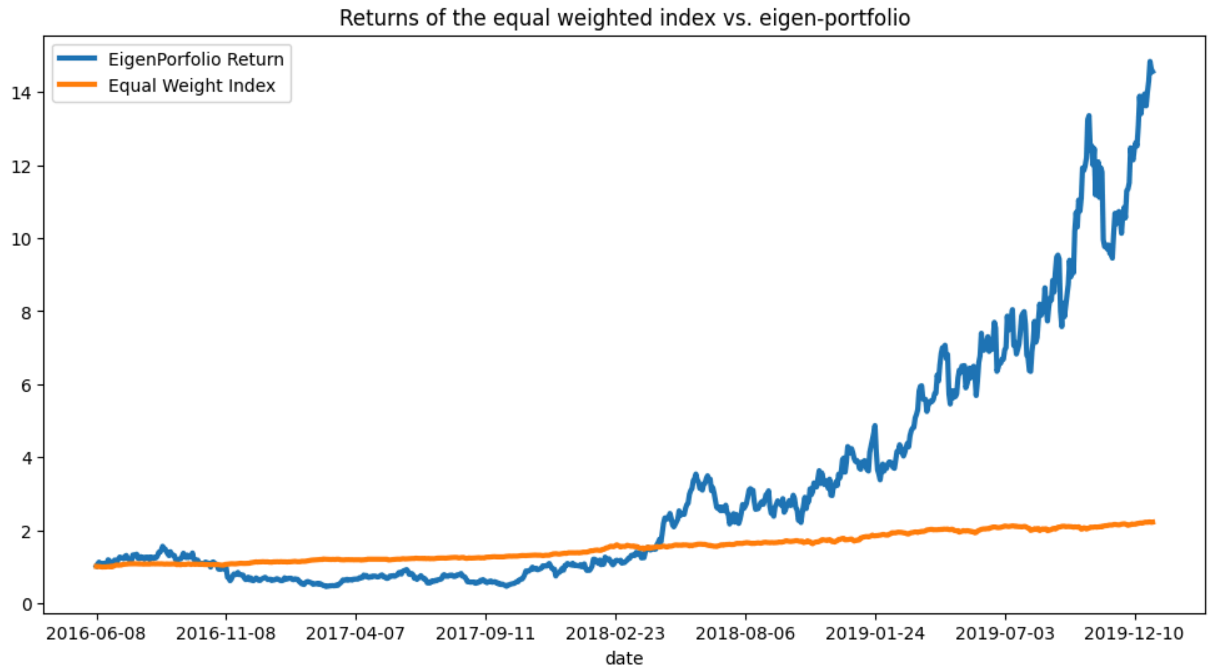

```
Eigen-Portfolio 7
Return = -90.08%
Volatility = 237.98%
Sharpe = -0.38
```



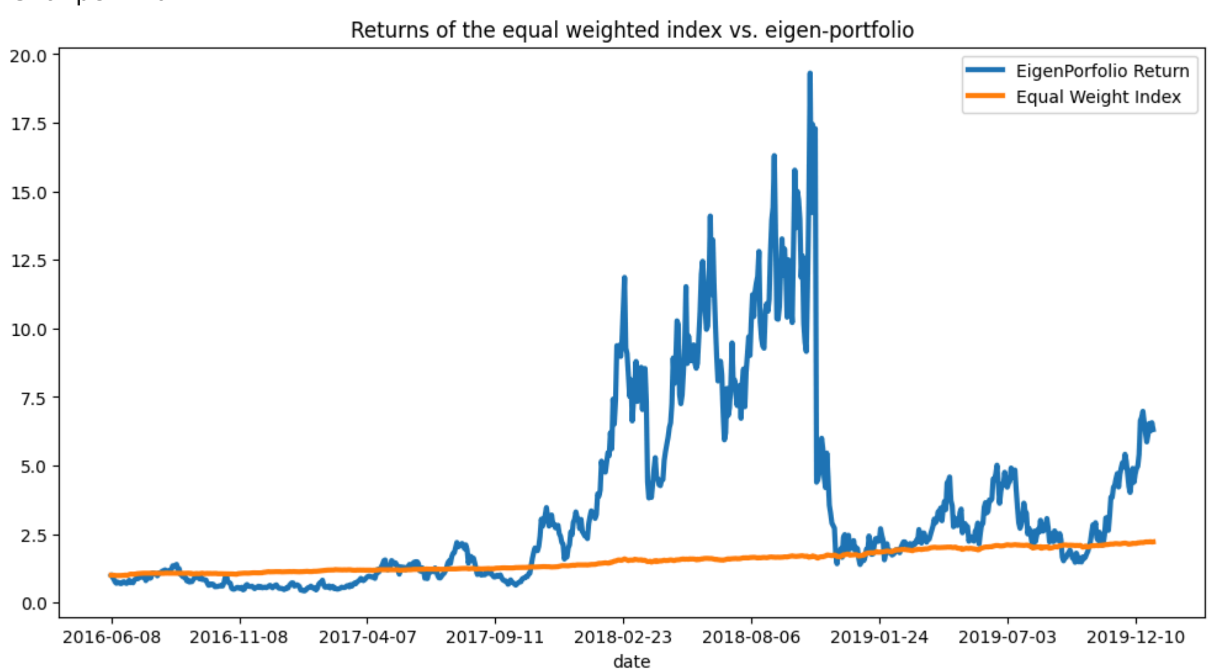

```
Eigen-Portfolio 8
Return = -98.98%
Volatility = 240.56%
Sharpe = -0.41
```



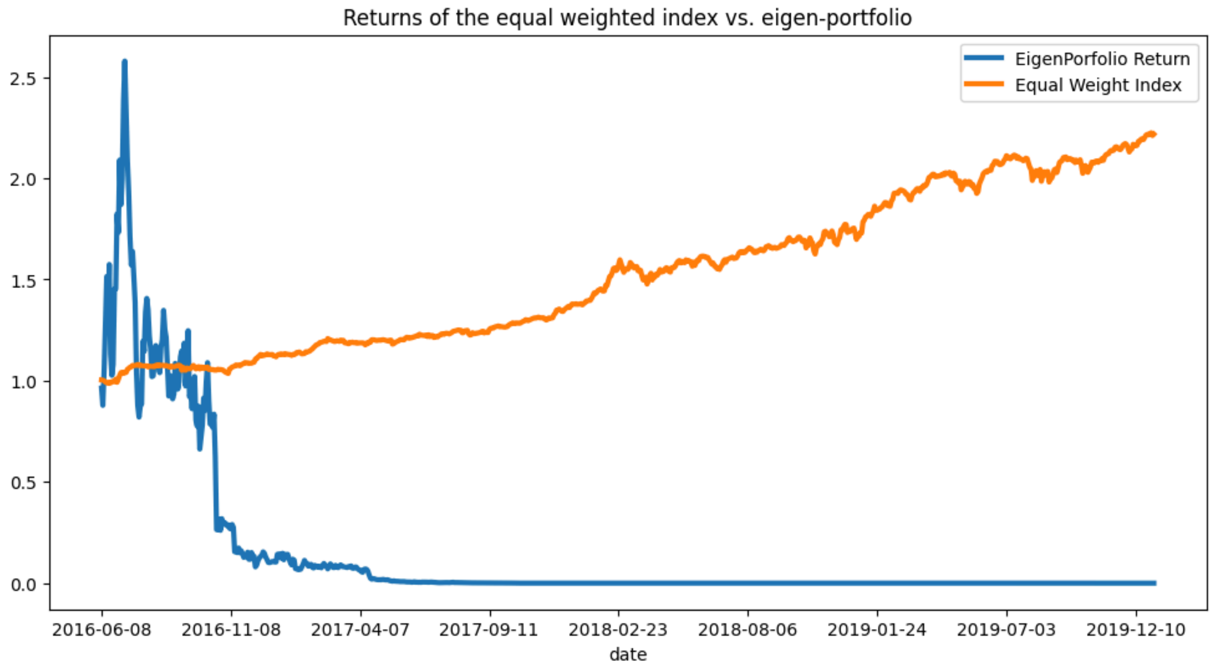

```
Eigen-Portfolio 9
Return = 128.94%
Volatility = 84.51%
Sharpe = 1.53
```



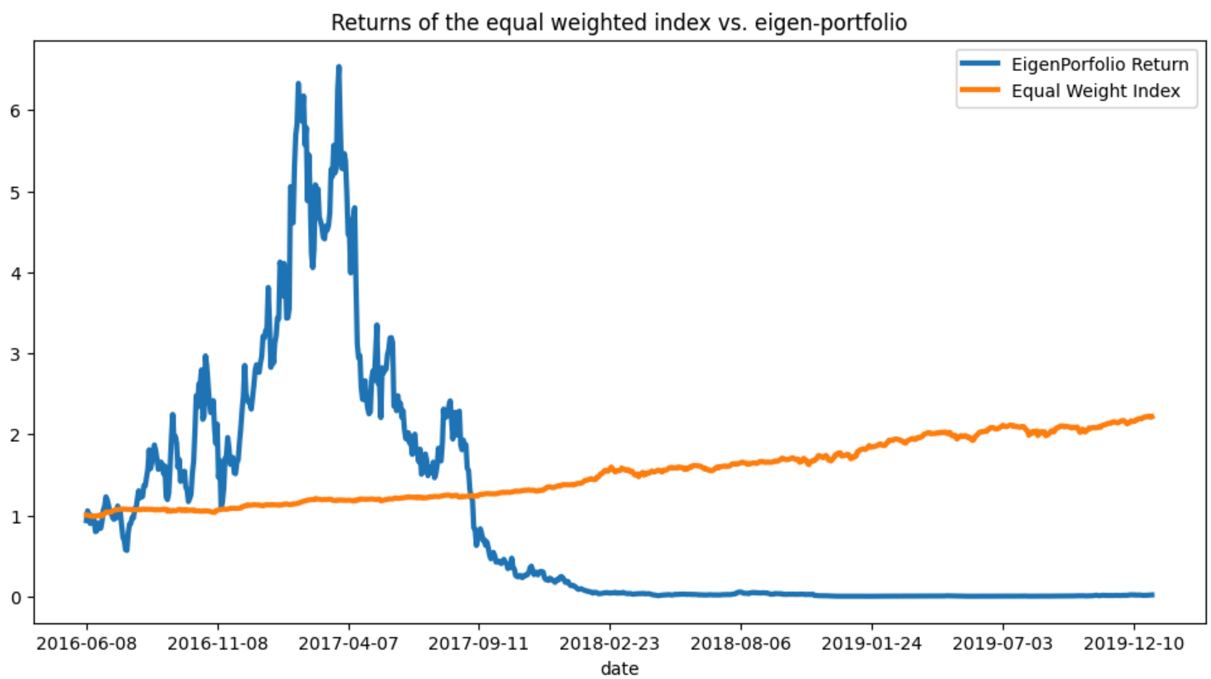

```
Eigen-Portfolio 10
Return = 76.80%
Volatility = 181.32%
Sharpe = 0.42
```



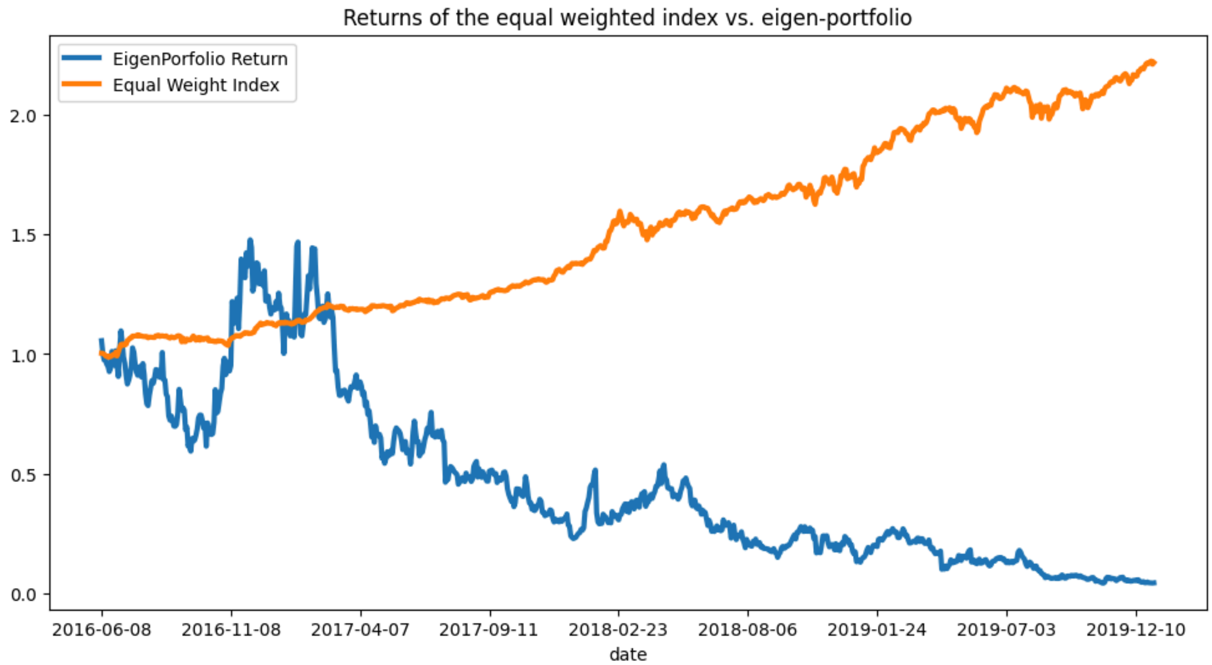

```
Eigen-Portfolio 11
Return = -97.22%
Volatility = 239.24%
Sharpe = -0.41
```



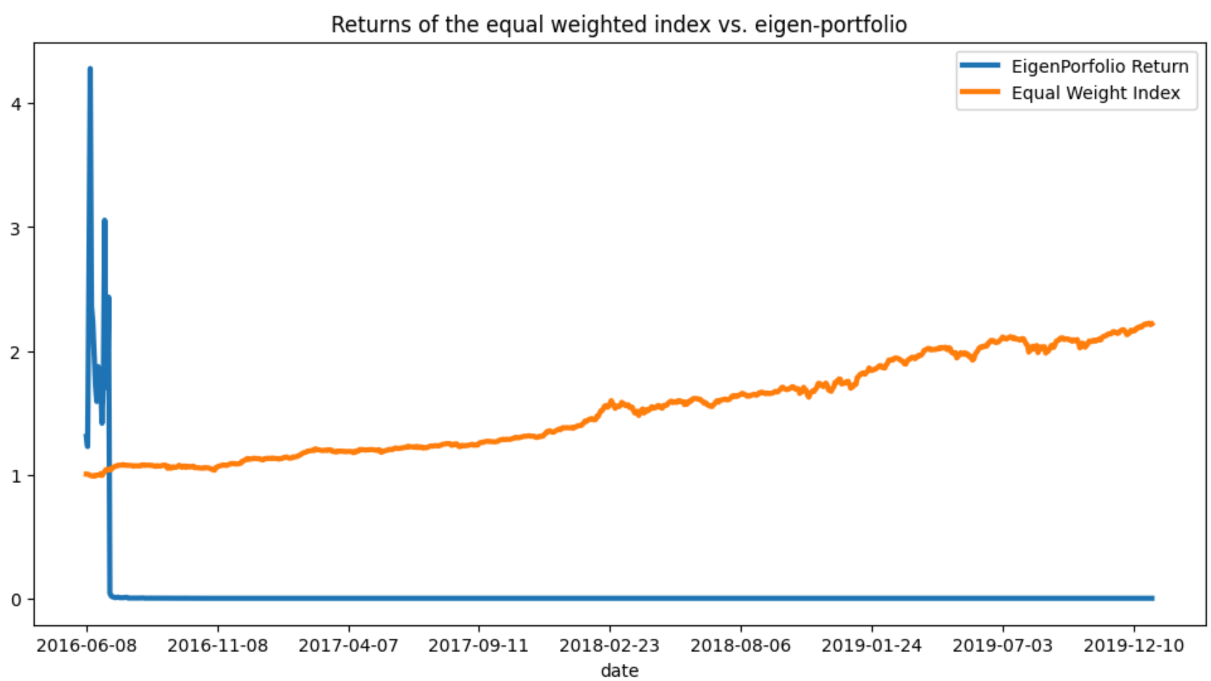

```
Eigen-Portfolio 12
Return = -70.76%
Volatility = 182.26%
Sharpe = -0.39
```



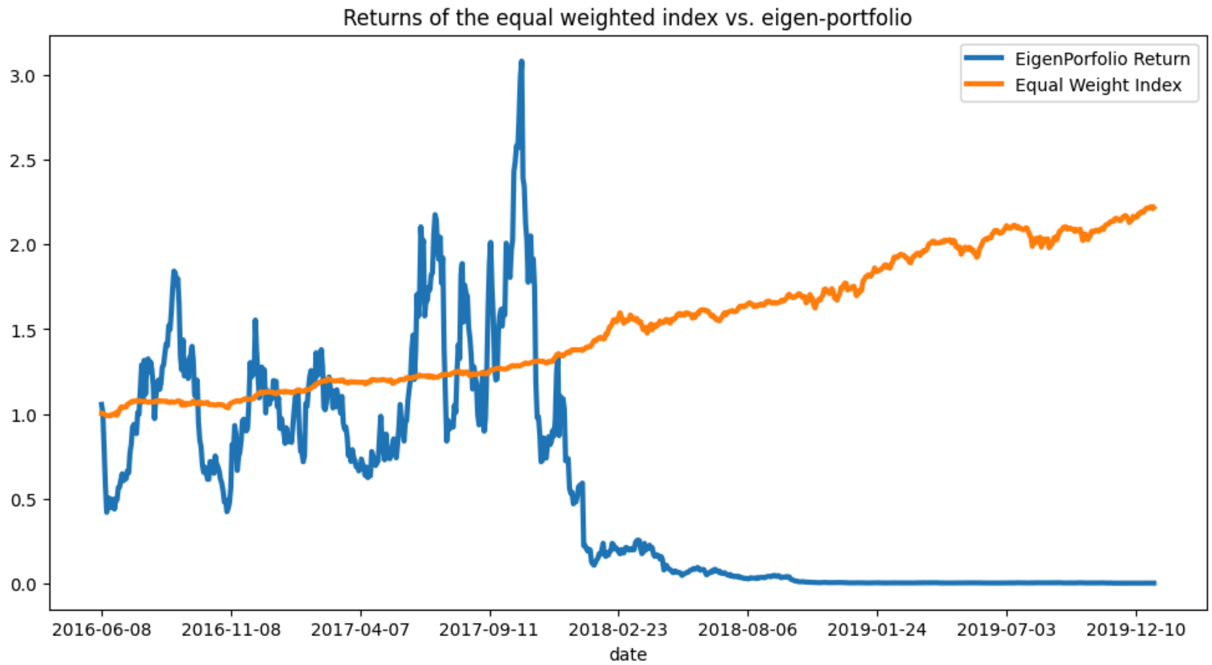

```
Eigen-Portfolio 13
Return = -61.91%
Volatility = 112.17%
Sharpe = -0.55
```



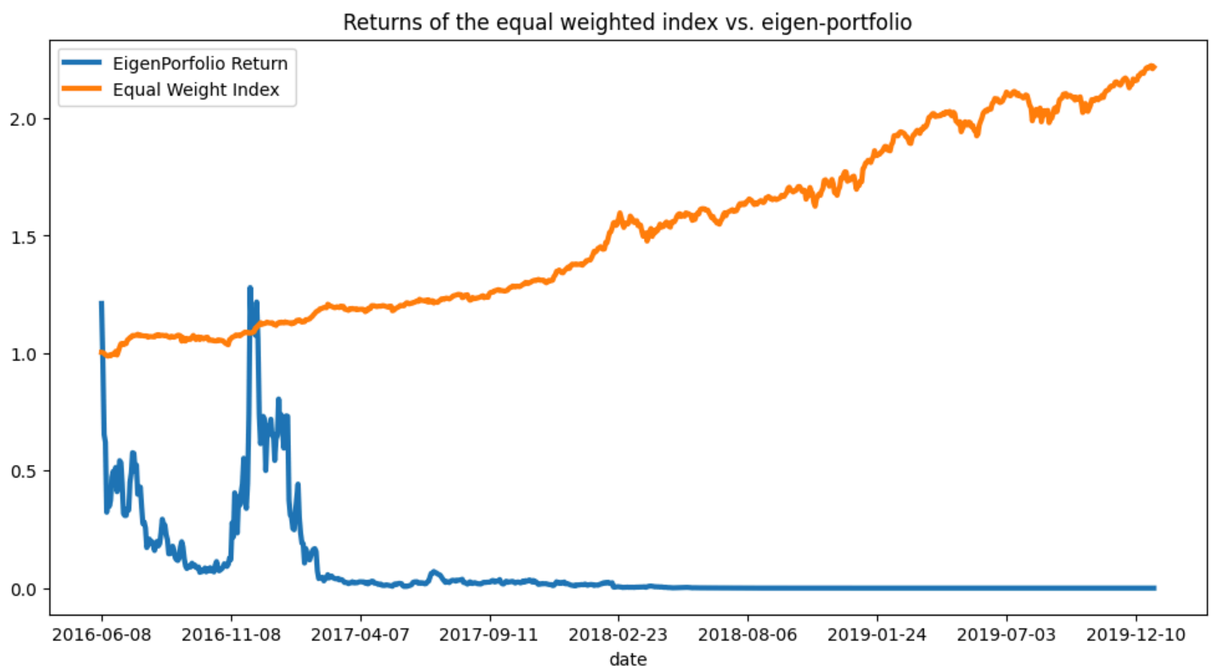

```
Eigen-Portfolio 14
Return = nan%
Volatility = 603.06%
Sharpe = nan
```



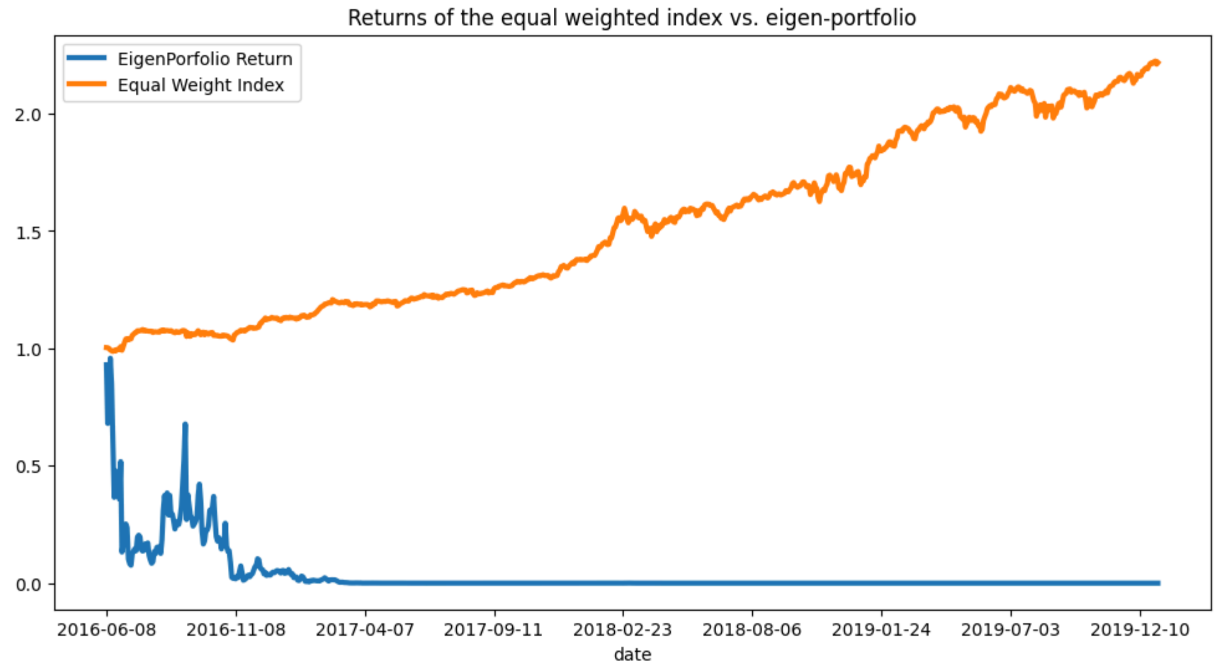

```
Eigen-Portfolio 15
Return = -88.75%
Volatility = 199.73%
Sharpe = -0.44
```



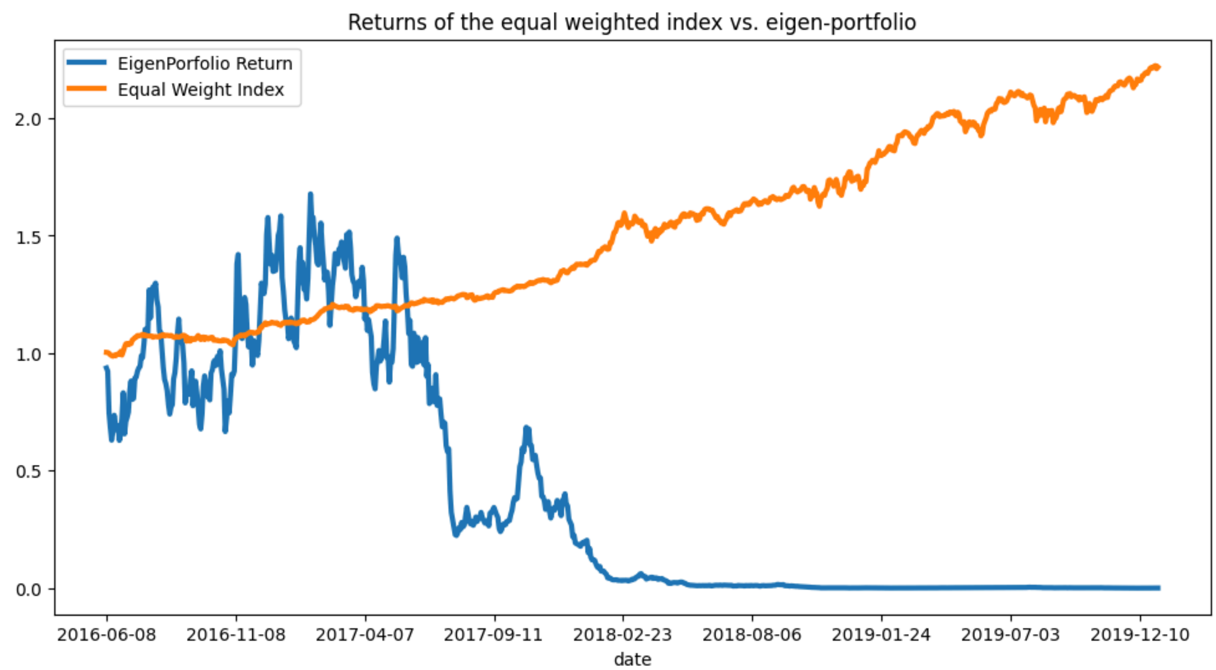

```
Eigen-Portfolio 16
Return = -99.94%
Volatility = 343.76%
Sharpe = -0.29
```



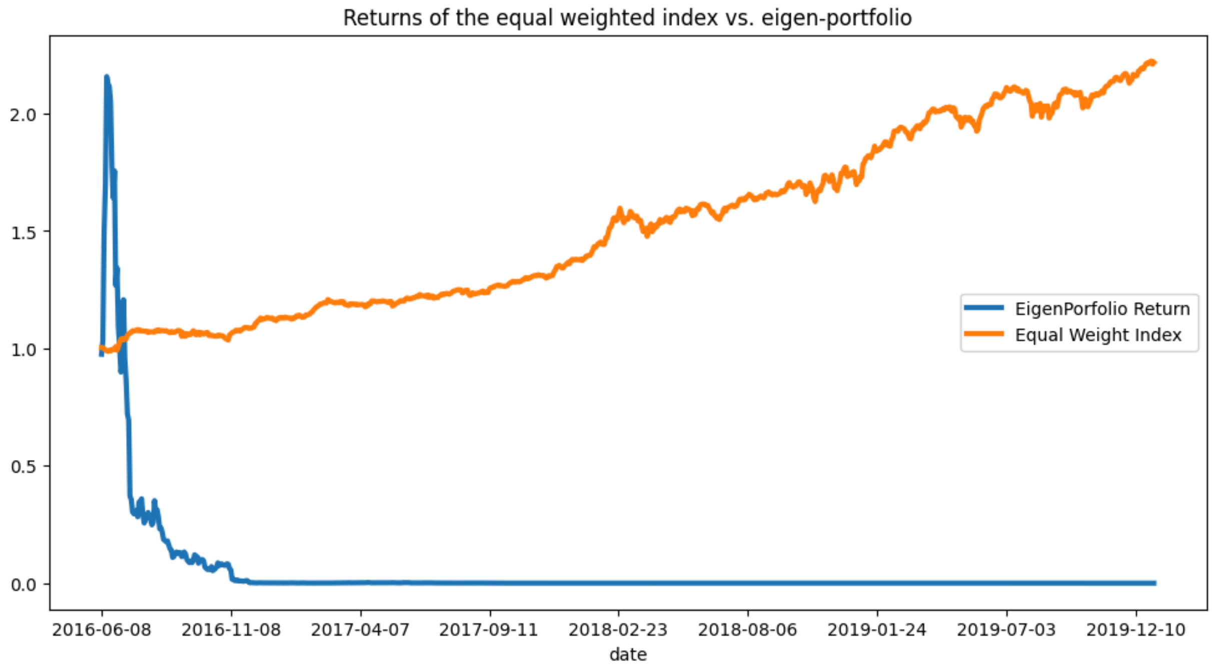

```
Eigen-Portfolio 17
Return = -100.00%
Volatility = 528.60%
Sharpe = -0.19
```



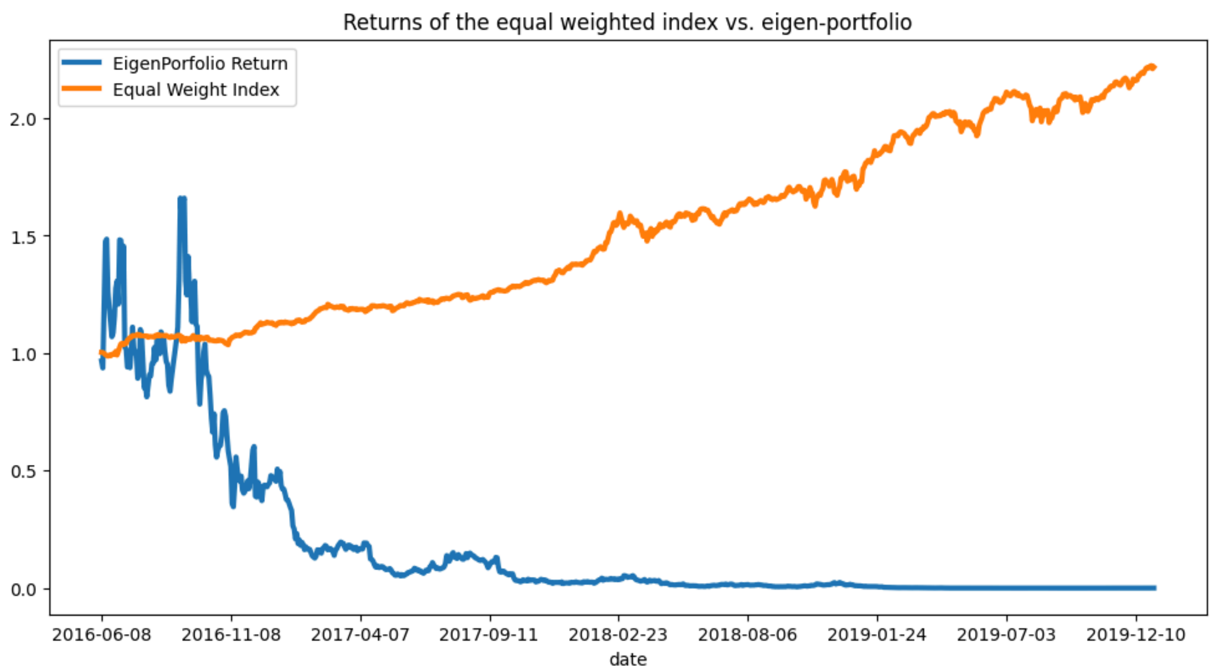

```
Eigen-Portfolio 18
Return = -88.77%
Volatility = 148.48%
Sharpe = -0.60
```



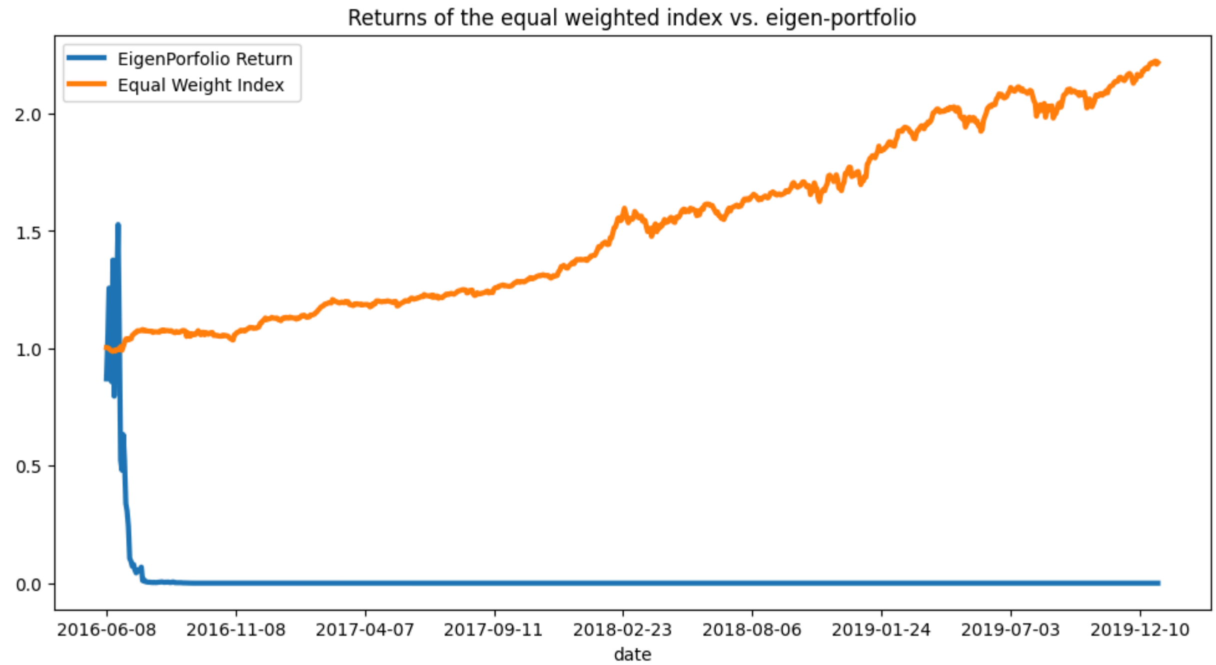

```
Eigen-Portfolio 19
Return = -98.50%
Volatility = 257.46%
Sharpe = -0.38
```



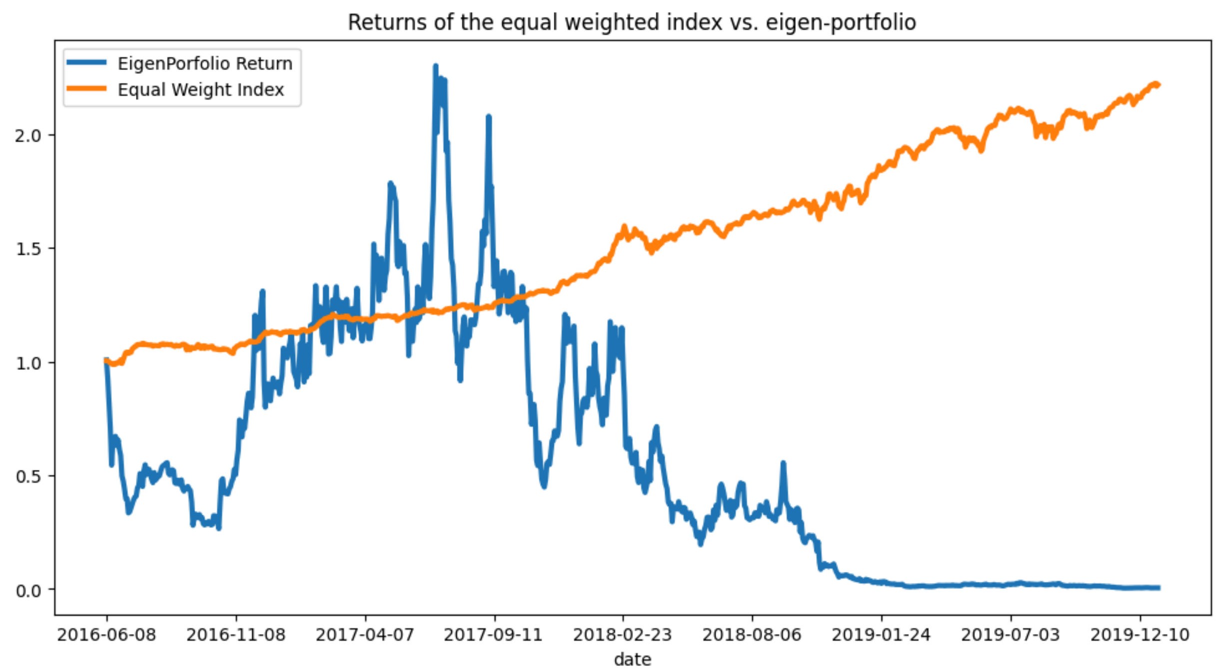

```
Eigen-Portfolio 20
Return = -90.51%
Volatility = 172.10%
Sharpe = -0.53
```



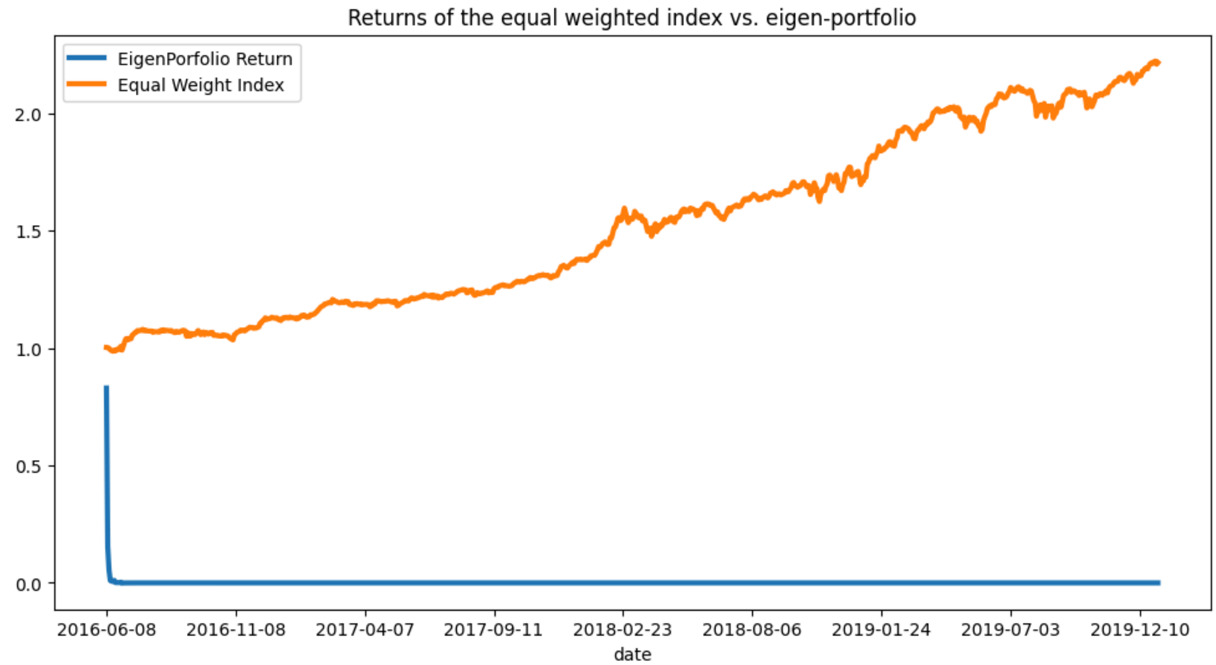

```
Eigen-Portfolio 21
Return = -100.00%
Volatility = 681.05%
Sharpe = -0.15
```



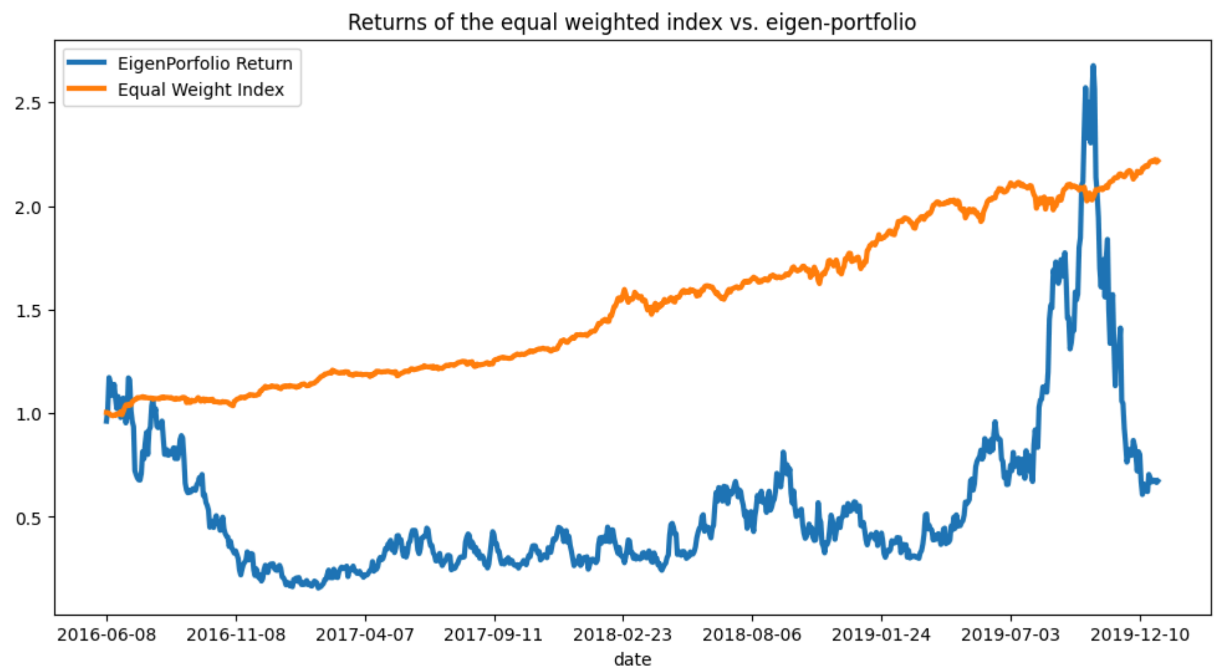

```
Eigen-Portfolio 22
Return = -80.47%
Volatility = 156.03%
Sharpe = -0.52
```



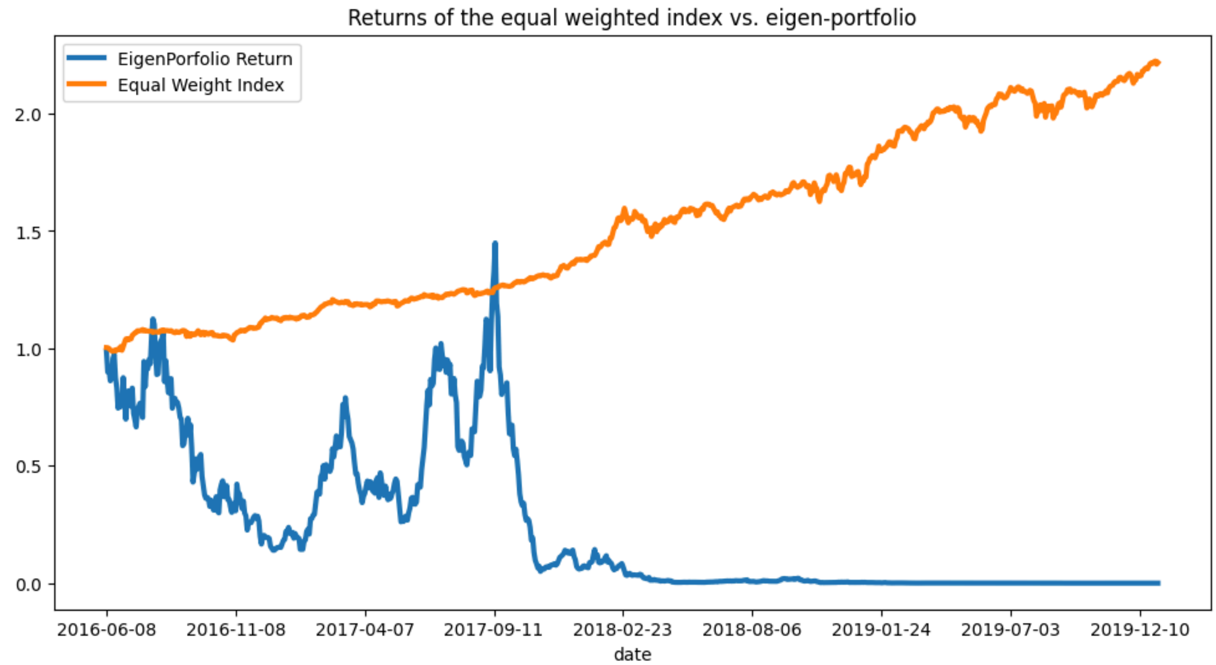

```
Eigen-Portfolio 23
Return = -100.00%
Volatility = 1385.52%
Sharpe = -0.07
```



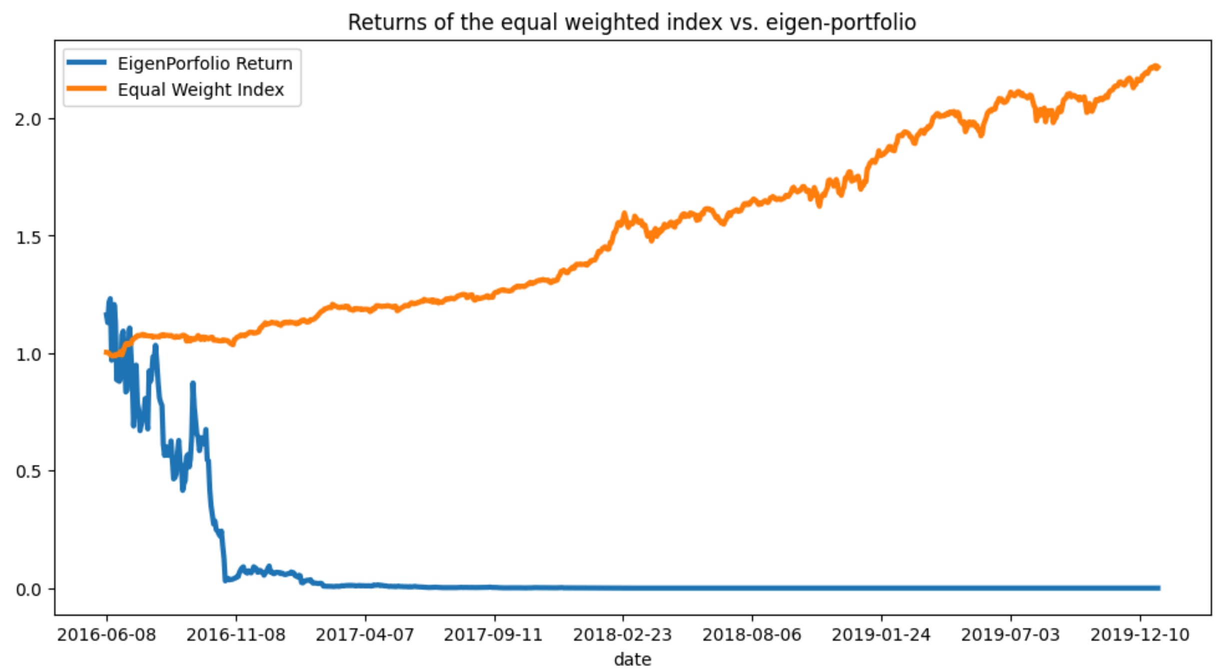

```
Eigen-Portfolio 24
Return = -11.52%
Volatility = 131.33%
Sharpe = -0.09
```




```
Eigen-Portfolio 25
Return = -93.01%
Volatility = 200.14%
Sharpe = -0.46
```




```
Eigen-Portfolio 26
Return = -99.98%
Volatility = 317.60%
Sharpe = -0.31
```



5. Conclusion

Looking at the backtesting result, the portfolio with the best result in the training set leads to the best result in the test set. By using PCA, we get independent eigen portfolios with higher return and sharp ratio as compared to market.

However, while it's valuable, backtesting has significant limitations:

Past Performance is Not Indicative of Future Results: Market conditions change, and a strategy that worked well in the past may not work well in the future.

Overfitting: This occurs when a strategy is excessively tuned to fit the specific nuances of the historical data used for testing. It might look great on past data but fail miserably in live trading because it hasn't captured a robust market edge.

Data Quality Issues Inaccurate, incomplete, or improperly adjusted historical data can lead to misleading backtest results.

Look-Ahead Bias: Accidentally incorporating information into the simulation that would not have been available at the time the trade decision was made.

Ignoring Real-World Factors: Difficulty in perfectly simulating factors like slippage, commission costs, market impact of large orders, and changing liquidity conditions.