PCA Investing

Alan Wang, Derek Yang, Tim Dzhurinskiy, William Yu, and Eugenio Macias

December 2, 2024

1. Using stock returns at the daily (or monthly) level, perform a principal component analysis (PCA) on daily (monthly) stock returns. Feel free to refer to online resources. How many factors are needed to explain a reasonable proportion of the total variance in stock returns? (There is no right answer to this, although there are standard share of variance measures that are some diagnostic tests). (Hint: what preprocessing should you do before performing a PCA, and why?)

   Principal Component Analysis (PCA) is a linear dimensionality reduction method that projects high-dimensional data into a lower-dimensional subspace while preserving as much variance as possible. This derivation follows the foundational principles provided in the APMA 1690 lecture notes and some supplementary material.

   Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ represent a data matrix with n observations and p features, where each row corresponds to a sample and each column corresponds to a variable. Assume that the data is centered such that each feature has mean zero: $\mathbf{X}_c = \mathbf{X} - \mathbf{1}\mu^\top$, where $\mu \in \mathbb{R}^p$ is the mean vector. The covariance matrix of the data is defined as $\mathbf{\Sigma} = \frac{1}{n}\mathbf{X}_c^\top \mathbf{X}_c$. To find the directions (of our PC's) that maximize the variance, we look for a projection vector $\mathbf{w} \in \mathbb{R}^p$ that maximizes the variance of the projected data $\text{Var}(\mathbf{w}^\top \mathbf{X}_c) = \mathbf{w}^\top \mathbf{\Sigma} \mathbf{w}$. To ensure that $\mathbf{w}$ remains a unit vector, we impose the constraint $\|\mathbf{w}\|_2 = 1$ and solve the following optimization problem:

$$\max_{\mathbf{w}} \mathbf{w}^\top \mathbf{\Sigma} \mathbf{w}, \quad \text{subject to } \mathbf{w}^\top \mathbf{w} = 1.$$

   By introducing a Lagrange multiplier λ, our optimization now becomes $\mathcal{L}(\mathbf{w}, \lambda) = \mathbf{w}^\top \mathbf{\Sigma} \mathbf{w} - \lambda(\mathbf{w}^\top \mathbf{w} - 1)$. Taking the derivative with respect to $\mathbf{w}$ and setting it to zero gives $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 2\mathbf{\Sigma}\mathbf{w} - 2\lambda \mathbf{w} = 0$, by simplifying it becomes $\mathbf{\Sigma}\mathbf{w} = \lambda \mathbf{w}$.

   Thus, $\mathbf{w}$ is an eigenvector of $\mathbf{\Sigma}$, and λ is the corresponding eigenvalue. The eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$ of $\mathbf{V}$ represent the variance explained by each principal component. The eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p$ form an orthonormal basis for $\mathbb{R}^p$. The first principal component corresponds to the eigenvector $\mathbf{w}_1$ with the largest eigenvalue λ_1, capturing the maximum variance. To reduce dimensionality, project the data onto the subspace spanned by the top k eigenvectors $\mathbf{W}_k = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k] \in \mathbb{R}^{p \times k}$. The projected data $\mathbf{Z} \in \mathbb{R}^{n \times k}$ is $\mathbf{Z} = \mathbf{X_c} \mathbf{W}_k$. The proportion of variance explained by the first k components is:

$$\text{Explained Variance} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j}.$$

The projections minimize the reconstruction error. The reconstruction in the original space yields $\mathbf{X}_{\text{reconstructed}} = \mathbf{Z}\mathbf{W}_k^\top = \mathbf{X}_c\mathbf{W}_k\mathbf{W}_k^\top$. The residuals are $\mathbf{X}_{\text{residual}} = \mathbf{X}_c - \mathbf{X}_{\text{reconstructed}}$.The residual sum of squares corresponds to the variance not captured by the first k components $\|\mathbf{X}_c - \mathbf{X}_{\text{reconstructed}}\|_F^2 = \sum_{j=k+1}^{p} \lambda_j$.

We implement PCA on the dataset using the scikit-learn Python library. For preprocessing, we use the $StandardScaler$ function, which transforms the dataset into having a mean of zero and standard deviation of 1. The dataset thus has unit variance.

```
import numpy as np
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("data/daily_ret_clean.csv").set_index("date")
stock_names = data.columns

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled = scaler.fit_transform(data)
```

2. Let's focus on a low number of factors: $K = 5$. Compute (based on the total sample for now) the $K = 5$ principal components.

   - Each principal component should be a vector of length N, the number of stocks in your universe. How can one translate a principal component vector v_k into a tradable portfolio? (Hint: what should you do with the vector elements?)

   The principal components are computed by solving the previous eigenvalue problems. The eigenvalue decomposition of the covariance matrix $\mathbf{\Sigma}$ yields:

$$\mathbf{\Sigma} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^\top,$$

   where $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$, and $\mathbf{W} \in \mathbb{R}^{p\times p}$ is the matrix of the calculated eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p$. The $K = 5$ principal components are defined by the 5 largest eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_5$, that capture the directions of maximum variance in the data.

   To translate a principal component $\mathbf{w}_k$ into a tradable portfolio, we must normalize the eigenvector to yield portfolio weights $\mathbf{w}'_k$ that satisfy a specific investment criterion. The weights can be normalized to $w'_k[i] = \frac{w_k[i]}{\|\mathbf{w}_k\|_1}$, to ensure that the portfolio represents a fully invested position. In our case, we normalize by risk to adjust the volatility of individual stocks, using weights: $w'_k[i] = \frac{w_k[i]}{\sigma_i}$, where σ_i is the standard deviation of stock i's daily returns. By projecting the original data onto the subspace spanned by $\mathbf{W}_K = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_5]$, we get a lower dimensional representation that keeps the most significant features of the data's variance structure.

```
pca = PCA(n_components=5)
pca.fit_transform(scaled)

eigenvalues = pca.explained_variance_
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratio)
```

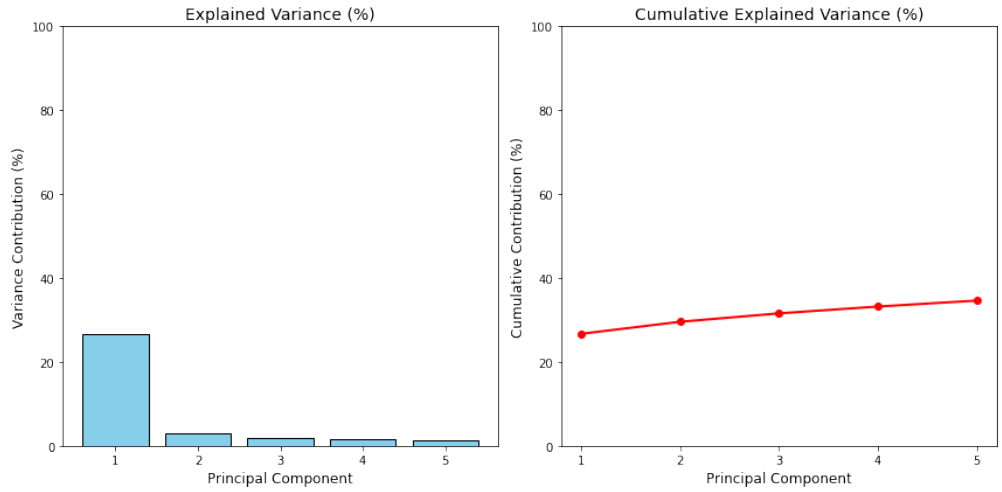


Figure 1: Explained Variance and Cumulative Explained Variance from the First 5 Principle Components.

3. For each principal component vector v_k, compute the factor returns for the eigenportfolio:

$$f_k^t = \sum_i \frac{v_k[i]}{\bar{\sigma}_i} \cdot r_{i,t}$$

where $\bar{\sigma}_i$ is the standard deviation for stock i returns. Why do we want to normalize the v_k by the standard deviation?

The principal component $\mathbf{v}_k$ is

$$\mathbf{\Sigma}\mathbf{v}_k = \lambda_k \mathbf{v}_k,$$

However, eigenvectors do not account for the differing risk profiles of individual stocks. By dividing each element $v_k[i]$ by $\bar{\sigma}_i$, we are normalizing the weights with respect to the standard deviation of returns, ensuring that the contribution of each stock to the factor return is proportional to its risk adjusted exposure. The normalized weight is given by: $w'_k[i] = \frac{v_k[i]}{\bar{\sigma}_i}$. Thus, the factor return for the k-th eigenportfolio can be rewriten as $f_k^t = \sum_{i=1}^{p} w'_k[i] \cdot r_{i,t}$.

This normalization makes sure the portfolio weights account for the relative risk of each stock. Stocks with larger volatility $\bar{\sigma}_i$ contribute less to the eigenportfolio, reducing the portfolio's overall risk exposure. The normalized weights $w'_k[i]$ become dimensionless, making them comparable to stocks with different units measurement or levels of volatility. This also ensures that the portfolio stays diversified.

```
weights = components/data.std(axis=0)
factor_returns = weights.dot(data.T) / len(stock_names)
```

4. Focus on the first factor return: what's the correlation of that with respect to market returns?

   The first factor return $f_1^t = \sum_{i=1}^{p} \frac{v_1[i]}{\bar{\sigma}_i} \cdot r_{i,t}$, represents the portfolio weighted return with the direction of the first principal component, which captures the largest variance of the dataset.

   To measure the correlation of f_1^t with the market return $r_{\text{mkt},t}$, we compute :

$$\text{Corr}(f_1, r_{\text{mkt}}) = \frac{\text{Cov}(f_1, r_{\text{mkt}})}{\sqrt{\text{Var}(f_1) \cdot \text{Var}(r_{\text{mkt}})}},$$

   Since the first principal component captures the largest variance in the stock returns, it is often highly correlated with the market return, which is also the case in our dataset.

```
market_returns = np.mean(data, axis=1)
factor_returns.iloc[0].corr(market_returns)
```

5. For each stock i, run the regression:

$$r_{i,t} = \alpha_i + \sum_k \beta_{k,i} f_k^t + \epsilon_{i,t}$$

   What is the interpretation of α_i?

   For each stock i, the regression models the return of stock i at time t as a linear combination of the factor returns f_k^t, an intercept term α_i, and an error term $\epsilon_{i,t}$. The intercept term, α, represents the average return of stock i that cannot be explained by the factors f_k^t, i.e. the stock's factor independent return.

$$\alpha_i = \mathbb{E}[r_{i,t}] - \sum_{k=1}^{K} \beta_{k,i} \mathbb{E}[f_k^t],$$

   $\beta_{k,i}$, is the factor loading for the k-th factor, measures the sensitivity of stock i's returns to the factor f_k^t. The residual term, $\epsilon_{i,t}$, captures the portion of $r_{i,t}$ that cannot be explained by our regression model.

```
from sklearn.linear_model import LinearRegression

X = factor_returns.T
Y = data

model = LinearRegression()
model.fit(X, Y)

coefs = pd.DataFrame(model.coef_, index=data.columns, columns=X.
    columns)
coefs["alpha"] = model.intercept_
```

6. How does this approach differ from standard factors (i.e., using market, value, momentum)?

   In contrast to other standard factors such as market, value, and momentum, PCA generates factors directly from the covariance structure of the data, whereas standard factors are defined from economic principles. Our factors are solely data-driven and maximize variance capture. Standard factors are predefined e.g. The *market factor* is observed as the return of a broad market index $r_{\text{mkt},t}$; The *value* is constructed as the return difference in high and low book to market stocks, i.e. $r_{\text{high},t} - r_{\text{low},t}$; The *momentum factor* captures the return difference in past winners and past losers, i.e. $r_{\text{winners},t} - r_{\text{losers},t}$. By construction, our principal components are orthogonal by construction, i.e $\mathbf{v}_k^\top \mathbf{v}_j = 0 \quad \text{for } k \neq j$ ensuring $\text{Cov}(f_k^t, f_j^t) = 0 \quad \text{for } k \neq j$. Standard factors are not orthogonal and may show significant correlations. For instance, the market factor $r_{\text{mkt},t}$ normally correlates with value or momentum factors which can complicate the interpretation of factor contributions to stock daily returns. In contrast, our eigenvectors $\mathbf{v}_k$ and their factors f_k^t lack direct economic interpretation as they represent abstract mathematical directions that just capture the largest variance in the dataset.

7. Construct a trading strategy based on your estimate of α_i. If you are totally out of ideas, one thing would be to sort the stocks based on your estimates of α_i, go long stocks in the top quintile of alpha, go short stocks in the bottom quintile of alpha. A trading strategy should consist of how much of each stock i to hold (can be negative) at any point in time t. Note that if you put weights $w_{i,t}$ on stock i at time t, your returns $r_{\text{port},t+1}$ are given by:

$$\sum w_{i,t} \cdot r_{i,t+1}.$$

   - Feel free to be more entrepreneurial/creative in this section.
   - Make sure your trading strategy is not based on future returns, e.g., you go long stocks that will move up tomorrow.

   Let α_i be the idiosyncratic return of stock i independent of factor returns f_k^t. Stocks associated to higher α_i are statistically expected to do better, making them candidates for long positions, while stocks with lower α_i for short positions. Let the stocks be ranked based on their α_i. Without loss of generality, assume $\alpha_1 \geq \alpha_2 \geq ... \geq \alpha_n$. Our trading strategy is market neutral long/short based on the entire set of stocks. We assign a weight to each stock based on its alpha in proportion to the entire alpha of the stocks in the dataset. The weights are then normalized to achieve market neutral conditions, wherein the weights of the stocks gone long equals the weights of the stocks gone short. Thus, the strategy is isolated from general market movements, or beta. Rather, its movement lies solely in its alpha. More detailed code is within our attached Jupyter Notebook, but the general notion is as follows:

```
weights = coefs["alpha"] / coefs["alpha"].abs().sum()

total_long = weights[weights > 0].sum()
total_short = weights[weights < 0].sum()
weights[weights > 0] /= total_long
weights[weights < 0] /= -total_short

normalized_portfolio_returns = (weights * data).sum(axis=1)
```

8. Visualize / report the cumulative returns of the trading strategy. Let $r_{\text{port},t}$ be the returns of your trading strategy for time t. The cumulative returns of the trading strategy measure the aggregated performance over time. Let $r_{\text{port},t}$ be the portfolio return at time t. The cumulative return up to time T is defined as: $R_{\text{cum},T} = \prod_{t=1}^{T}(1 + r_{\text{port},t}) - 1$. Alternatively, if $r_{\text{port},t}$ is small, the cumulative return can be approximated by: $R_{\text{cum},T} \approx \sum_{t=1}^{T} r_{\text{port},t}$. The portfolio return $r_{\text{port},t}$ at time t is computed as: $r_{\text{port},t} = \sum_{i=1}^{n} w_{i,t} r_{i,t}$ The weights $w_{i,t}$ are defined according to the chosen trading strategy, such as the by quintils allocation. Given the time series of portfolio returns $\{r_{\text{port},1}, r_{\text{port},2}, \ldots, r_{\text{port},T}\}$, the cumulative returns can be computed through iteration as $R_{\text{cum},t} = R_{\text{cum},t-1}(1 + r_{\text{port},t}), \quad$ setting $R_{\text{cum},0} = 0$. To visualize the cumulative returns, we can plot $R_{\text{cum},t}$ against time t to have a visual understanding of how our strategy performs over the investment time. An overall increasing $R_{\text{cum},t}$ indicates profitability, while an abrupt draw downs would be a period of under performance.

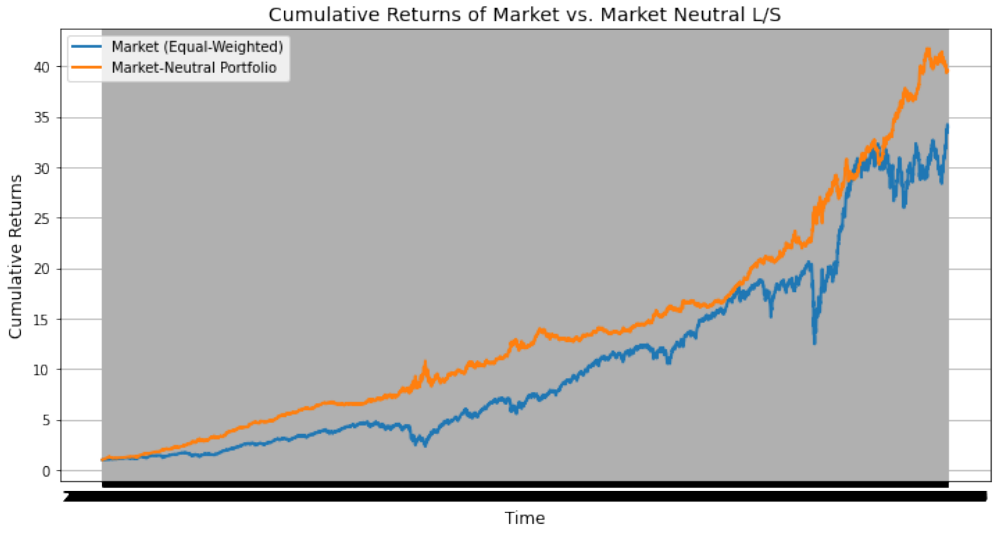


Figure 2: Cumulative Returns of the Market compared to Our Market-Neutral Long/Short Strategy.

9. Compute the Sharpe ratio of your trading strategy.

   The Sharpe ratio of our porfolio is defined as $S = \frac{\mathbb{E}[r_{\text{port}} - r_f]}{\sigma_{\text{port}}}$, where r_f is the risk-free rate of returns. For every time period, we compute the excess return of the portfolio: $r_{\text{excess},t} = r_{\text{port},t} - r_f$, the expected excess returns $\mathbb{E}[r_{\text{port}} - r_f] = \frac{1}{T}\sum_{t=1}^{T} r_{\text{excess},t}$, and the portfolio risk (i.e. its standard deviation) $\sigma_{\text{port}} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(r_{\text{port},t} - \mathbb{E}[r_{\text{port}}])^2}$. The Sharpe ratio penalizes a trading strategy with high volatility relative to their returns.

```
def sharpe(self, rets, *, annual=False):
        """
        Calculate Sharpe ratio from simple returns
        """
        returns = np.log(1+rets)
        if annual:
            returns = returns.resample("YE").sum()
```

```
8
9           return (np.e ** returns-1).mean() / (np.e ** returns-1).std
                ()
```

10. Compute the information ratio of your trading strategy:

    - Formally, run the regression: $r_{\text{port},t} = \alpha + \beta r_{\text{mkt},t}$. Your reference return (for the information ratio) should be $\beta \cdot r_{\text{mkt},t}$.

      The information ratio (IR) of a strategy measures the risk-adjusted return of the portfolio relative to the benchmark (the market in this case):

    - Is your strategy market neutral?

      For the strategy to be market neutral in this case, the estimated β from the regression would be close to zero. This indicates that the portfolio returns from the strategy are uncorrelated with market returns.

```
1  def information(self, rets, market_rets, *, annual=False):
2          """
3          Calculate Information ratio based on simple strategy and
               market returns
4          """
5          returns = np.log(1+rets)
6          market_returns = np.log(1+market_rets)
7
8          if annual:
9              returns = returns.resample("YE").sum()
10             market_returns = market_returns.resample("YE").sum()
11
12         return (np.e ** returns - np.e ** market_returns).mean() /
               (np.e ** returns).std()
```

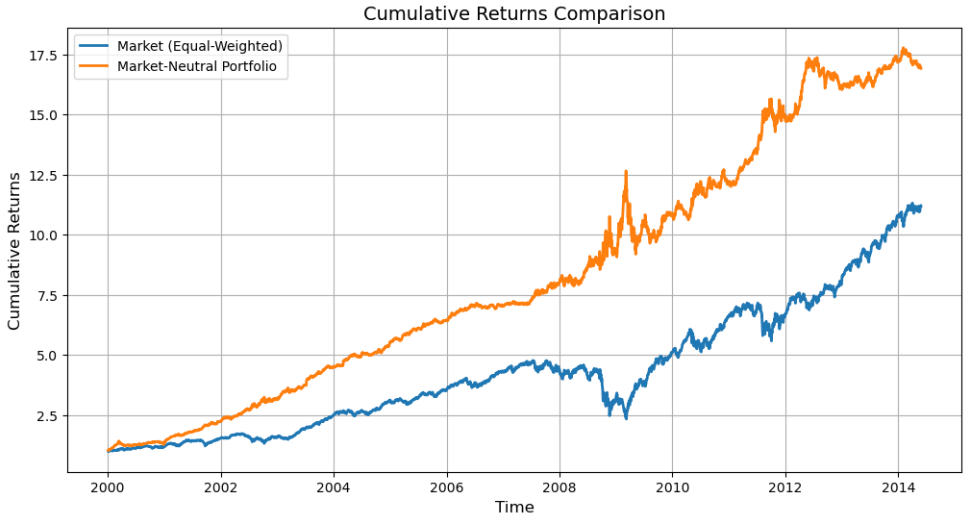
11. Note that you have estimated the PCA using the entire sample. Thus, your estimates of your trading performance should be somewhat positively biased. Now, repeat your exercise, where you simulate a trading strategy across time, where you:

    (a) Estimate your factors only based on data in the past (so at 2015 January, you construct your PCAs using data up to 2015 January).

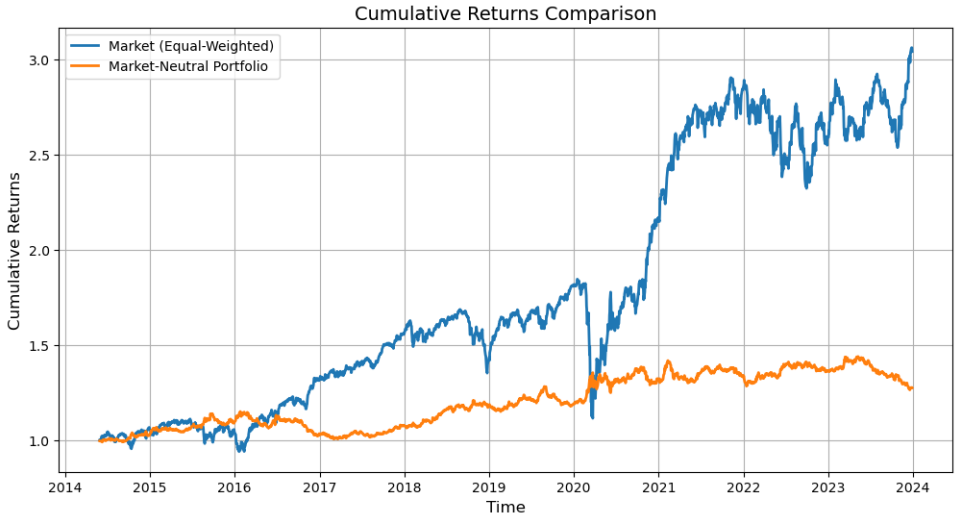
    (b) Construct a trading strategy, and see how this strategy performs.

    A simple starting point is to estimate α_i's using data prior to 2015, and assume α_i's stay fixed afterwards. Of course, you can also update your α_i estimates every day/month when you roll forward. Be very clear how you avoided the look-ahead bias.

    (a) We estimate the alphas using the first 60% of the data. This is up to around May 2014. The performance of the strategy on the training set is shown below:



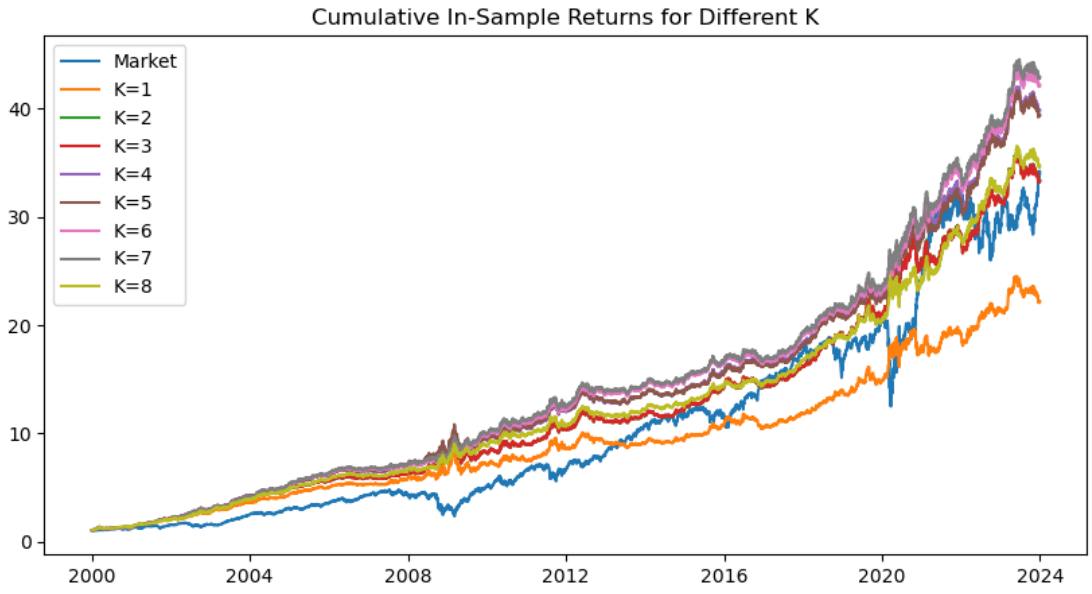
(b) Using the strategy (portfolio weights) from before, we find that we do very poorly on the test set. We avoid look-ahead bias by splitting the date at which we train and test the strategy. Although we are market neutral, the alphas from the previous 14 years don't extrapolate, and our returns are very low.



12. Why stop at $K = 5$? Across multiple values of K, re-do all of the above exercises.

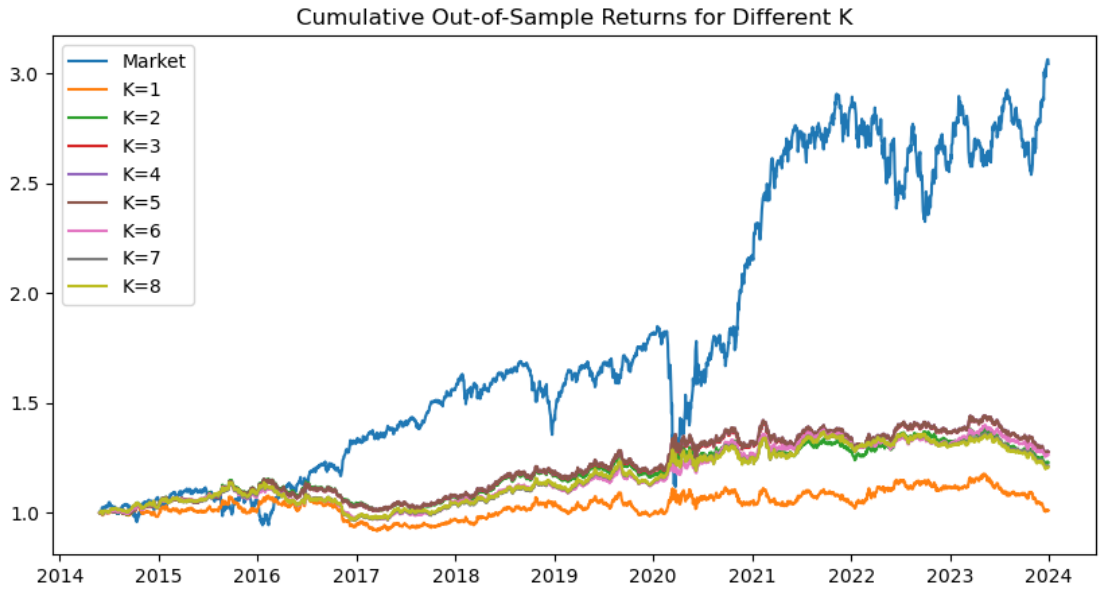
    - Which K seems to work best?
    - Is there a significant difference between in-sample assessments (one where you estimate the factors based on all the data) and out-sample assessments (one where you estimate the factors only based on data available at the time and roll forward)? Discuss.

We do the same as above and plot the in-sample and out-of-sample results for different K:



K	Annualized Sharpe	Annualized Information
1	1.115904	-0.235948
2	1.186715	-0.074593
3	1.184741	-0.074922
4	1.208120	-0.009341
5	1.215491	-0.014099
6	1.228088	0.011294
7	1.226720	0.017726
8	1.294702	-0.072495

In the above graph, we train on the entire set and test and on the entire set. In the graph below, we train up to May 2014 and test on the rest.



K	Annualized Sharpe	Annualized Information
1	0.042127	-2.276885
2	0.418279	-1.998014
3	0.466256	-1.815535
4	0.466945	-1.798729
5	0.464532	-1.792968
6	0.424462	-1.714822
7	0.328391	-1.619763
8	0.315743	-1.601695

In both in-sample and out-of-sample testing, it seems that principal component 2 is very important to include. The performance of the strategy is significantly higher when including both PC 1 and 2 versus just including PC 1.

K=4 delivers the highest Sharpe Ratio. It is clear that testing in-sample is very heavily positive-biased. Testing out-of-sample shows that our strategy does not extrapolate well to future data. One way we would mitigate this is by creating a rolling strategy that exponentially weights previously trained alphas from the past as you move forward in time. This continual adjustment of weights would potentially mitigate the risks of bias in our strategy.

References

- Gundersen, Gregory. 2022. ”Returns and Log Returns.” February 6. Available at: https://gregorygundersen.com/blog/2022/02/06/log-returns/.
- Meng, Kun. 2023. *Lecture Notes of APMA 1690: Probability and Statistics in Engineering*. Brown University, pp. 156–163.
- ”Chapter 18: Principal Components Analysis.” *Lecture Notes on PCA*. No author, no date. Provided document. Available at: https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch18.pdf.
- Bach, Stephen. 2024. *CS 1420: Principal Component Analysis*. Machine Learning course slides, Brown University.
- Scikit-learn Documentation. 2024. ”Preprocessing.” Available at: https://scikit-learn.org/1.5/modules/preprocessing.html.