Minerva Technologies Task

Edoardo Bertoli
bertoliedoardo99@gmail.com

September 3, 2024

Contents

1 Introduction

This is the report for the task assigned to me by the team at Minerva Technologies. The purpose of this task is to develop a long-only trading model of five stock indices resulting in a portfolio weight matrix. The input data is comprised of daily historical logarithmic returns from the 30th September 2003 to the 31st December 2019 for the five stock indices, as well as a series of other factors: rates, commodities, Forex currency crossings. Macroeconomic indices of the stock indices' countries and fundamentals of the stock were also given.

Given the amount of data and task's objective I decided to use a simple but effective model: the portfolio will be optimized using a risk-parity approach were the allocation of assets is determined by imposing that each asset's contribution to the portfolio total variance is equal. In order to find the covariance matrix of expected returns based on the historical data, necessary to optimize the portfolio, I decided to use a linear multi-factor model combined with Principal Component Analysis on the factors in order to reduce their dimensionality but also eliminate linear correlation, grouping them by how much they contribute to the overall variation of the factors data.

The model will be tested by implementing a rolling window approach with a weekly rebalancing frequency. For each week the model will be retrained and the weights recalculated in order to calculate the portfolio weight matrix and test the portfolio return and volatility. A K-Fold Cross Validation is used on the training data in order to choose the best linear regressor based on the their performance scores.

Even though normality is required only for the residuals of the linear regression, having data that is approximately normally distributed makes calculation easier since most of the information is captured by the first two moments: mean and variance. For this reason Section 3 is dedicated to testing data for its normality as well as for stationarity and cross-asset relationships to better understand the nature of the data and better interpret the result of the model.

2 Data Cleaning

The input data given for this task is comprised of:

1. Five historical series of daily logarithmic returns pertaining to stock indices from five different geographic regions. (Identified throughout the report as 'Stock returns')
2. Six historical series of Fundamental Indicators for each geographic area. ('Fundamentals')
3. Two historical series of daily log returns for Rates for each geographic area. ('Rates returns')
4. Two historical series of Macroeconomic indicators for each geographic area. ('Macro indices')
5. Three historical series of daily log returns related to currency crossings. ('Forex returns')
6. Three historical series of log returns for three different commodities. ('Commodities returns')

The historical series span from 30th September of 2003 to 31st of December of 2019, business days only. Before running any type of cleaning process I applied a log transformation to the historical series of Macroeconomic indices and Fundamentals Indicators.

From a first look at the data it is obvious that some cleaning is required. There are invalid numeric values like zeroes and NaN (Not-a-Number). I decided to use a careful and simple approach to avoid introducing too much noise in the data. First I replaced every type of invalid value into a NaN, then I forward and backward filled the data to make sure each value in each Dataframe is a valid number (float64).

At this point a pretty straightforward problem arises: forward and backward filling just copies data, either from after or before, leaving us with a lot of repeated values. To smooth out this values a rolling average of five days is applied on the previously invalid values only.

This is the most simple approach in order to do the minimum required to clean the input data. Another way could be to just throw away every value that is not valid, but it is better to try and keep as much data as possible for consistency, especially since part of the data will be reserved for testing the model's performance.

More invasive approach can be used, like Winsorization or LOWESS (Locally Weighted Scatterplot Regression), but the risk of changing the data distribution too much without really moving towards a normal distribution is high.

During the data cleaning process the historical data was sampled with the original timeframes. The resampling into daily data has been done right after the data cleaning process.

The module **clean_data.py** is dedicated to cleaning the data. It will read raw data from a file called 'data.xlsx' and will output an xlsx file called 'formatted-data.xlsx' with the cleaned data.

3 Historical Data Analysis

This stage is dedicated to extracting as much information as possible from the cleaned data. This step is necessary to have an idea of what kind of data we have in order to be able to correctly inteprept the results of the chosen model and understand its limitations. The data used in this test is from the 'formatted-data.xlsx' file. It has only been cleaned and scaled to zero mean and unit variance when necessary. It has not yet been divided into training and testing. The custom python module **eda.py** is dedicated to this section.

3.1 Normality

The assumption that stock returns follow a univariate normal distribution not always holds for financial data. When it does, at least approximately, calculation and interpretation of calculations tend to be easier because most of the data information can be captured by the first two moments: mean and variance. For this reason the stock returns data is checked against a standardized gaussian distribution to understand each stock's behaviour and understand how heavy the assumption of data distributed normally is.

In Table 1 the results of fitting each stock returns against a normalized Gaussian distribution are shown. The data has been standardized to unit variance and zero mean. For each asset the first four moments are provided, as well as the p-value for the Anderson-Darling test. The ratio for p-value acceptance is 0.05%. For this calculation I used the function *scipy.stats.goodness_of_fit* from the python library *Scipy*. The reference values for a normalized Gaussian distribution are:

- Mean: 0
- Standard Deviation: 1
- Skewness: 0
- Excess Kurtosis (Kurtosis - 3): 0

As we can see both from the plot in Figure 1 and Table 1 the problem is clear: the first two moments are extremely close to being those of a standardized gaussian, in fact the shape of the histogram is not extremely different from that of gaussian distributed data. The skewness is relatively small, in fact we cannot see any extreme rupture of symmetry to the right or left. The problem is the excess kurtosis which is way too big because of the huge amount of data clustering at the center, which would make the gaussian distribution way too peaked. This also explains

Stock return	Mean	Standard deviation	Skewness	Excess Kurtosis	Anderson-Darling p-value
Indice Azionario Paese 1	-6.701652777742044e-18	1.0001179175760244	-0.26527445129737953	6.252408544066505	0.0001
Indice Azionario Paese 2	1.0052479166613066e-17	1.0001179175760244	-0.06867531417815065	6.560510364685937	0.0001
Indice Azionario Paese 3	-3.350826388871022e-18	1.0001179175760244	-0.06963505684987985	7.389173595223623	0.0001
Indice Azionario Paese 4	4.607386284697655e-18	1.0001179175760244	-0.06963505684987985	9.171379931995125	0.0001
Indice Azionario Paese 5	-7.539359374959799e-18	1.0001179175760244	-0.15364747544085475	8.853385699414373	0.0001

Table 1: Descriptive statics and results of fit of stock returns against a standardized Gaussian distribution.

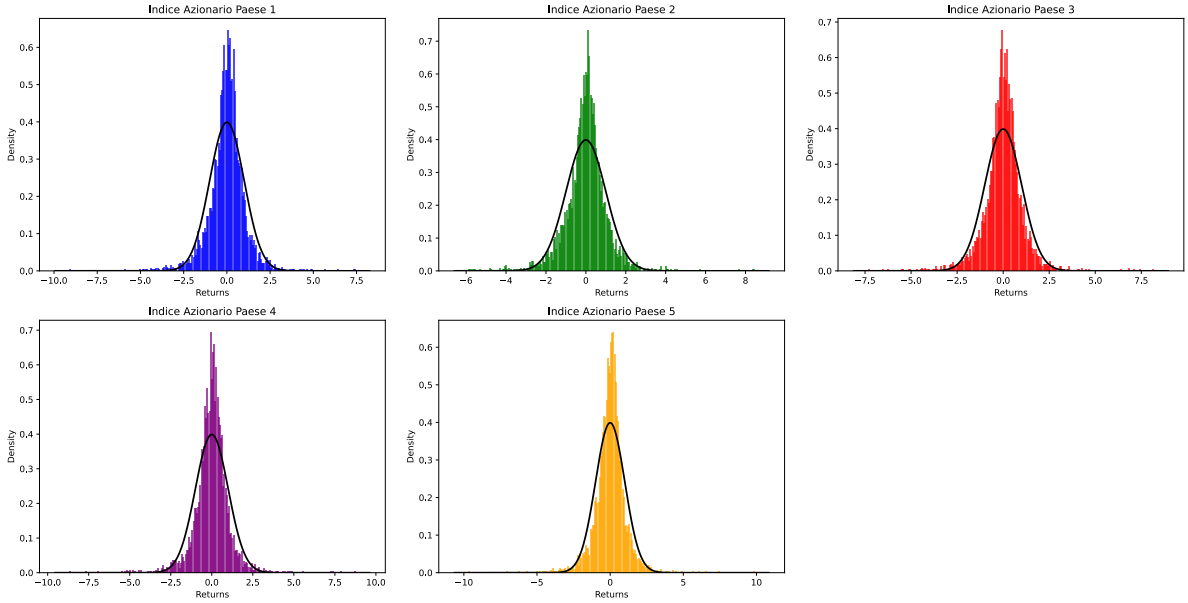

Figure 1: Comparison of standardized stock log returns fits against a standardized Gaussian distribution

the low pvalue of the Anderson-Darling test which by taking into account the skewdness of the data has to conclude that the data is not normal. There have been proposed ways of modeling the stock returns as more complex distributions like elliptical distributions, as shown in [1].

3.2 Time Dependence

Stationarity is a very important characteristic of financial data. It has to be taken into consideration especially when using Multi-Factor models that do not inherently consider the time relations between data. Also the assumption of stock returns following a normal distribution requires a non time-varying mean and standard deviation. For this reason in Figure 2 are shown the time plots of the weekly standard deviation of the stock returns.

As we can see the variance is mostly oscillating between 0 and 1, apart from huge spikes of volatility in the stock returns that coincide with periods of extreme economic uncertainty (like the 2008-2010). This needs to be taken into account when interpreting the portfolio results, especially to understand how much the model can adapt to sudden changes in the market.

3.3 Correlation Analysis

Linear correlation between factors and stock is extremely important to study in order to efficiently implement Principal Component Analysis on the factors and understand the correlation between factors and stock returns. The correlation between factors will be eliminated choosing the direction of greater variance using PCA. As we can see in Figure 3 there are two main areas of relative higher correlation: between the stock returns (top-left) and between macroeconomic indices and fundamentals of the stock returns (triangle below the diagonal). For this reason I

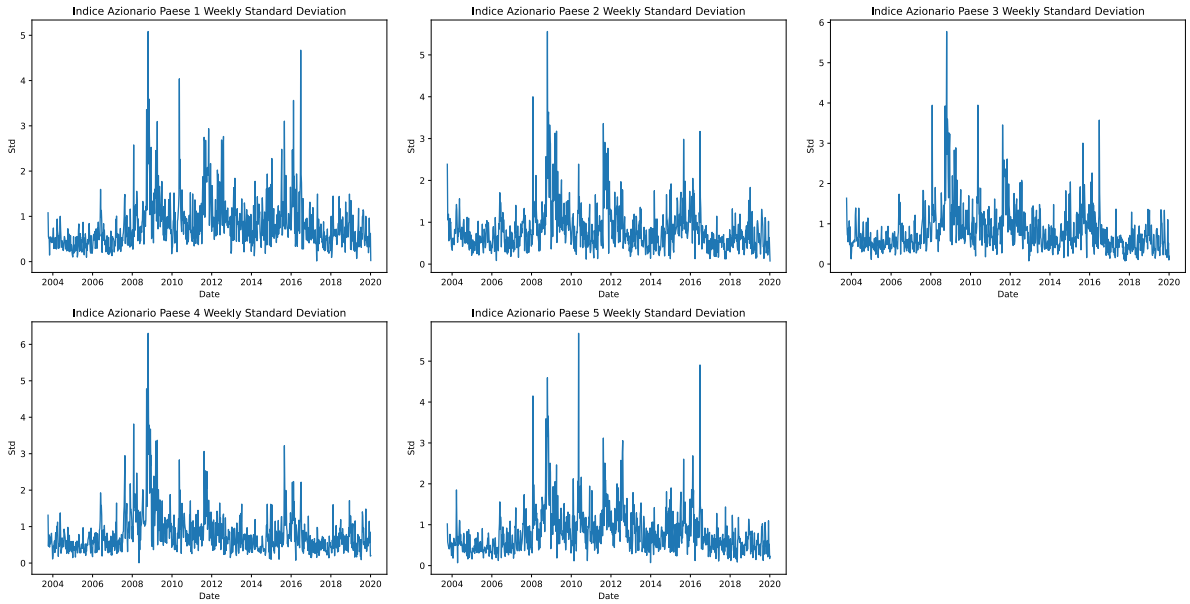

Figure 2: Comparison of stock returns fits against a standardized Gaussian distribution

will first apply PCA between Macro indices and fundamentals. Then I will apply again PCA between the results of the previous steps and the rest of the data, apart from the stock returns, on which no PCA will be applied.

3.4 Historical data visualization

Another very important piece of information can be extracted from the historical behaviour of the stock returns throughout the whole dataset. In Figure 4 the stock returns are plotted against time, from the first to the last date available in the dataset. As we can see the stock index for Country 1, 2, 3 and 4 close the timeframe with a positive log return, while the index stock of Country 5 closed with a negative log return. Also we can see that the stock index of the first country has the greatest return over this time-period, while Country 2, 3 and 4 were pretty close during the entire time-frame. These information must be kept in mind when evaluating the allocation of these assets in the portfolio.

4 Linear Multi-Factor Model with PCA

This section describes the process of implementing and training a multi-factor model in conjunction with Principal Component Analysis on the factors for forecasting expected returns and volatility to later use for the portfolio optimization. The model has been implemented in the custom python module **module.py**.

First it's important to establish the goal of this model: finding a relationship between stock returns and all the other historical daily data in the dataset, which we will call factors (rates, currency crossings, commodities, macro indices, fundamentals of stock), such that it's possible to use this relationship to forecast future expected stock returns knowing the values of the factor at the current time. Three main questions immediately come to mind: **what distribution should one use to model stock returns as random variables ? What kind of mathematical relationship is there between the factors and the stock returns ? How should one choose which factors to use ?**

The first question requires an assumption, usually it is required that either the actual returns or the log returns follow a normal distribution. As shown in the previous section, the stock

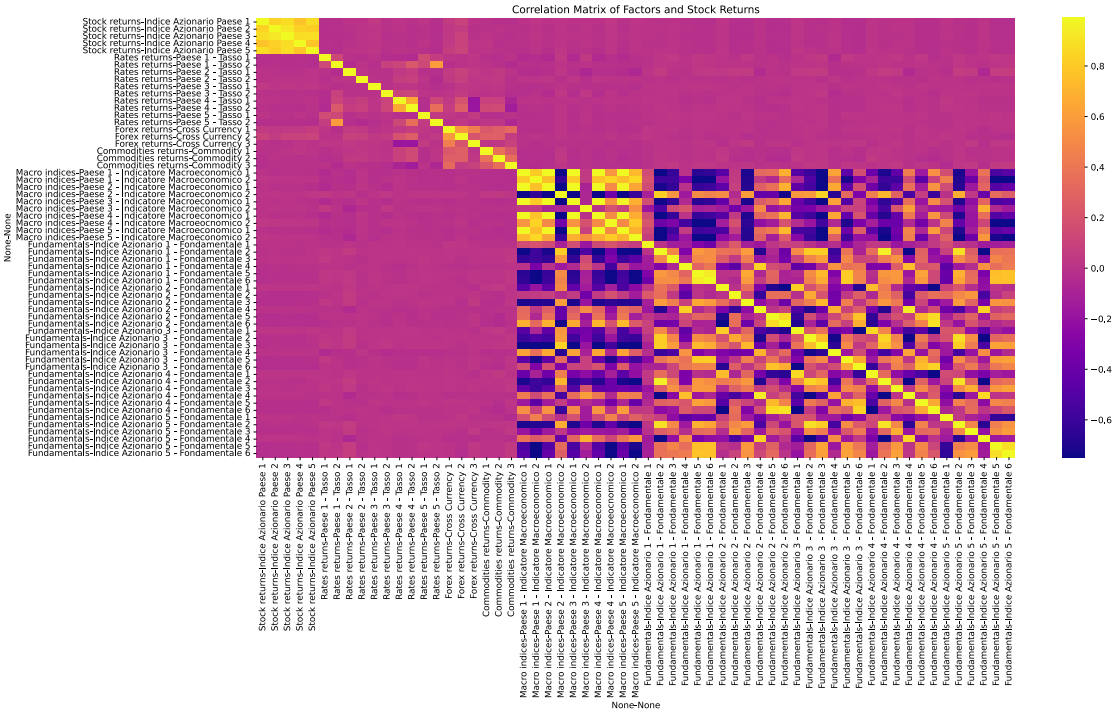

Figure 3: Correlation matrix heatmap of the whole dataset given as input for this task

indices' log returns are 'decently' close to being described by a normal distribution if not for the presence of clustering near the center and not normal tails. For this task we will proceed as if the stock returns (as logarithmic daily data) follow a univariate gaussian distribution so that the vector of stocks follows a multivariate normal distribution:

$$\vec{r} = N(\vec{\mu}, \vec{\sigma}) \tag{1}$$

where: $\vec{r} = (r_1, r_2, r_3, r_4, r_5)$ is the stock returns vector, $\vec{\sigma} = (\sigma_1, \ldots, \sigma_5)$ is the vector of stock returns' standard deviations.

The portfolio return will follow this normal distribution:

$$r_p = N(\vec{w} \cdot \vec{\mu}, \vec{w}^T \mathbf{\Sigma} \vec{w}) \tag{2}$$

where: $\vec{w}$ is the vector of weights which will define the portfolio allocation. The weights are the result of the optimization process. $\Sigma = (\sigma_{i,j})$ $(i, j = 1, ..., 5)$ is the covariance matrix of expected returns. Normality is not really required for the multi-factor model but it is required when optimizing the portfolio using a risk-parity approach where all the information present in the objective function solely comes from the covariance matrix of the forecasted stock returns. This needs to be taken into account when interpreting both the model and the portfolio optimization results. For more information see [2].

Now for the second question we need another assumption: the relationship between the stock returns and the chosen factors is **linear**, so:

$$r_i = a_i + B_{1,i} F_{1,i} + ... + \epsilon_i \tag{3}$$

where ϵ is the random error from the regression model used to determine the parameters in the above equation. The only thing an ordinary least squares linear regression model requires is the normality of ϵ. For this reason we will show the performance of different linear regressor that are more robust and do not need the residuals to be normally distributed.

Finally to tackle the problem of choosing the factors and at the same time dealing with correlation between factors I decided to use Principal Component Analysis on the factors in

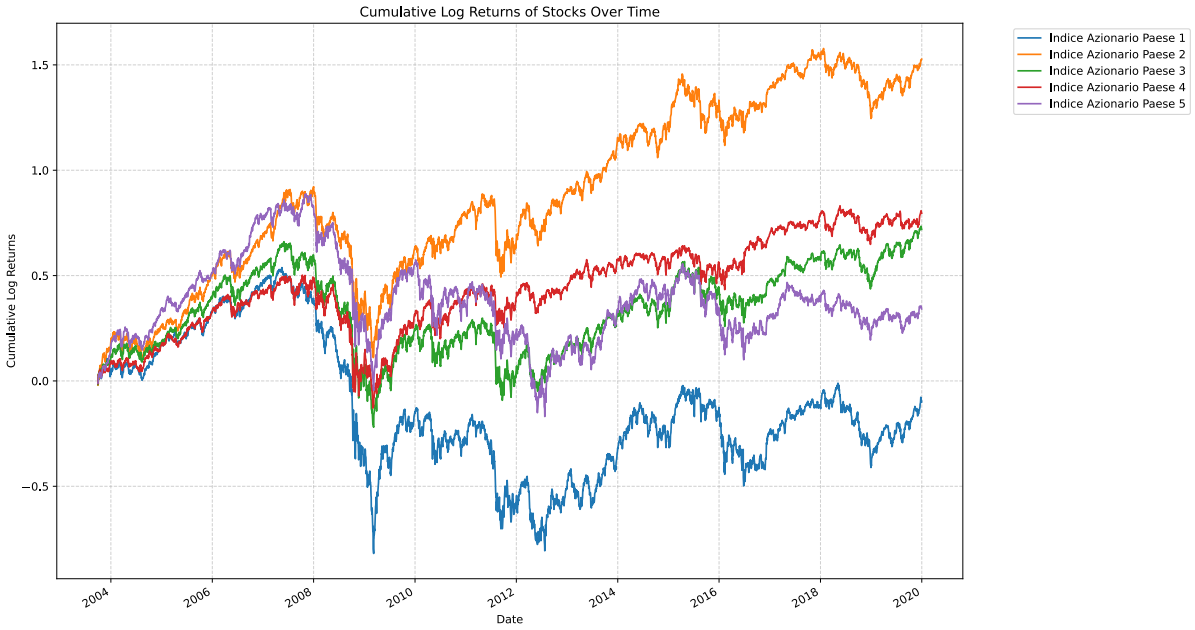

Figure 4: Cumulative log returns for each stock index over the entire dataset time period

order to reduce their dimensionality by choosing only the first 5 principal components which explain around 75-85% of the total variance.

Equation (3) can be written in vector-matrix notation:

$$\vec{r}_{t+1} = \vec{b_0} + \mathbf{B}\vec{F}_t + \vec{\epsilon} \tag{4}$$

Where: $\mathbf{B} = (b_{i,j})$ is the matrix of factors exposures, i runs over the number of stock returns (N) while j runs over the number of factors (K) chosen for the regression model, thus it's a (N,K) matrix. The matrix of the factors exposures is the objective of the regression model. The vector $\vec{F}_t$ is a (1,K) vector which represents the factor at time t. The vector $\vec{r}_{t+1}$ represents the expected returns at time t+1. The vector $\vec{\epsilon}$ represents the vector of errors of the regression model.

Given this model, it can be shown that the covariance matrix of expected returns can be written as:

$$\mathbf{\Sigma} = \mathbf{B}\mathbf{\Sigma}_F\mathbf{B}^T + \mathbf{S} \tag{5}$$

Where: $\mathbf{S}$ is a diagonal matrix consisting of each stocks' error ϵ's variance. $\mathbf{\Sigma}_F$ is the covariance matrix of the factors used in the regression model. Since we are using PCA, this matrix will always be diagonal. It is very important to notice that the only place where time dependence of data is considered is in the regression model where the stock returns of time t+1 are calculated based on the factors at time t. There are more sophisticated methods to model the time-dependence of data, for example using multiple lagged features, not only the day before, or using more advanced time series models like ARIMA or GARCH for capturing complex temporal patterns.

There exists more complex models which incorporate both time-volatility and non-linear relations between stock returns and factors. For example in [3] it is proposed the use of a Long Short-Term Memory Recurrent Deep Neural Network, combined with Layer-wise Relevant Propagation (LRP) in order to interpret the results. As they show in the table comparing different methods the LSTM + LRP model definitely outperforms the linear multi-factor model.

The idea is to use a linear regression model to calculate both $\mathbf{B}$ and $\mathbf{S}$. Then it is possible to obtain the covariance matrix of expected returns and use it to optimize the portfolio as shown in Section 4.3.

4.1 Principal Component Analysis on Factors

As we have seen in Figure 3 the highest correlation is between Macro returns and Fundamentals. For this reason, and also to simplify the interpretation of the remaining principal components, I decided to use standard PCA between Macro returns and Fundamentals. With these principal components selected I will run standard PCA again against all the other remaining factors and the 'Macro-Fundamental' principal components I found before. The standard Principal Components Analysis basically diagonalizes the covariance matrix and finds its eigenvectors. Using these eigenvectors as the transformation matrix it is possible to write every element of the initial base into the new base where the covariance matrix is diagonal (here the absence of covariance). One can then choose how many principal components to keep, thus how many of the eigenvectors with the biggest eigenvalues to keep. These principal components are directions in the new base that explain the most variance of the analyzed data. For this task I decided to choose the first five principal components, after some trial and error I found it was the most optimal choice, as it's shown in the following table and graphs. In reality once should carefully decide how to choose principal components, maybe even forcing a certain total explained variance, leaving the number of components chosen as variable.

The following results have been calculated using the whole dataset apart from the last year which was kept for testing purposes. In Figure 5 we can see the contribution of each Macroeconomic factor and Fundamentals factor to the first five principal components chosen. There are some peaks, suggesting that there are certain factors that prevail on the others but overall the distribution is pretty equal. In Figure 6 the principal component of the final PCA run are shown. Here the situation is quite different: we can see that the first three final principal components are basically the same as the first three Macro and Fundamentals principal components; the last two final principal components are a combination of rates, commodities and forex. These final five principal components will be used in the next section as the input data to train the linear model. Table 2 shows the percentage of the total variance of the factors explained by each principal component chosen, as well as the total explained variance of the first and second PCA run.

	PC0	**PC1**	**PC2**	**PC3**	**PC4**	**Total explained variance (%)**
Macro-Fundamental PCA (%)	42.31	22.66	14.04	4.66	3.74	87.40
Final PCA (%)	33.23	17.79	11.02	5.58	4.53	72.15

Table 2: Percentages of explained variance for each principal component and the total variance for the first and second PCA run

4.2 K-Fold Cross Validation of Linear Regressors

The principal components of the factors are the independent variables in the linear multi-factor model in eq.(4). The next step is to decide which linear regression model and which kind of loss function to use. First I decided to use a simple Ordinary Least Squares model (scikit-learn LinearRegression() model) in order to see the distribution of the residuals. I kept the last year of data out of the training sample for testing purposes.

The linear model has been implemented in the custom python module **model.py**.

In Figure 7 each stock index residual distribution's histogram is shown. We can immediately see the same problem we encountered in Figure 1, a clear grouping around the center of the distribution. We can confidently conclude that these residuals do not strictly follow a Gaussian distribution.

For this reason I decided to perform K-Fold Cross Validation on different kinds of more robust linear regression models and loss functions. This cross validation has been performed on the whole dataset apart from the last year that was kept for later testing. The splitting strategy

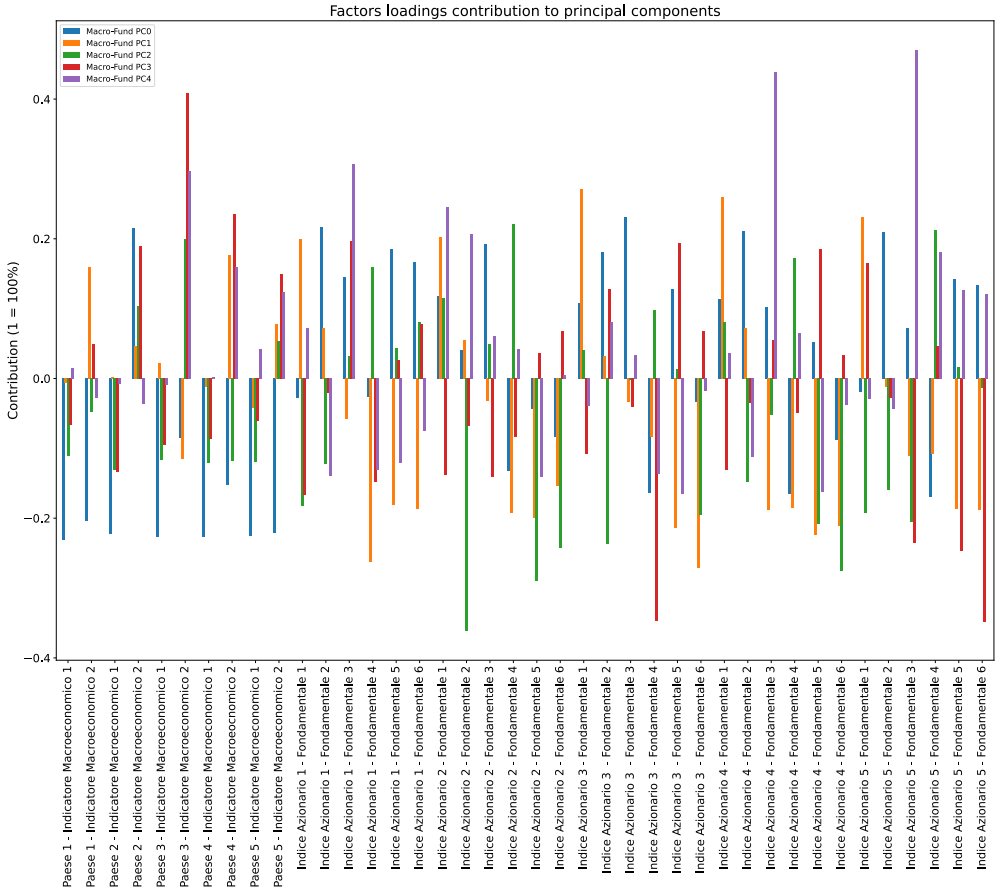

Figure 5: Macroeconomic and Fundamentals PCA factors loadings

used respects the time nature of the data because successive training sets are supersets of those that came before them. The library uses the appropriate scoring method for each model and it normalizes different scores' results such that the highest number always represents the best score. For more information see [4].

Here is the list of linear regression models from the python library *scikit-learn* that I used to perform this cross validation:

- LinearRegression()
- PassiveAggressiveRegressor()
- ElasticNet()
- HuberRegressor()
- TweedieRegressor()
- SGDRegressor(loss='huber', shuffle=False)
- SGDRegressor(loss='epsilon_insensitive', shuffle=False)
- SGDRegressor(loss='squared_epsilon_insensitive', shuffle=False, penalty='elasticnet', epsilon = 0.01 * Y.std().mean())

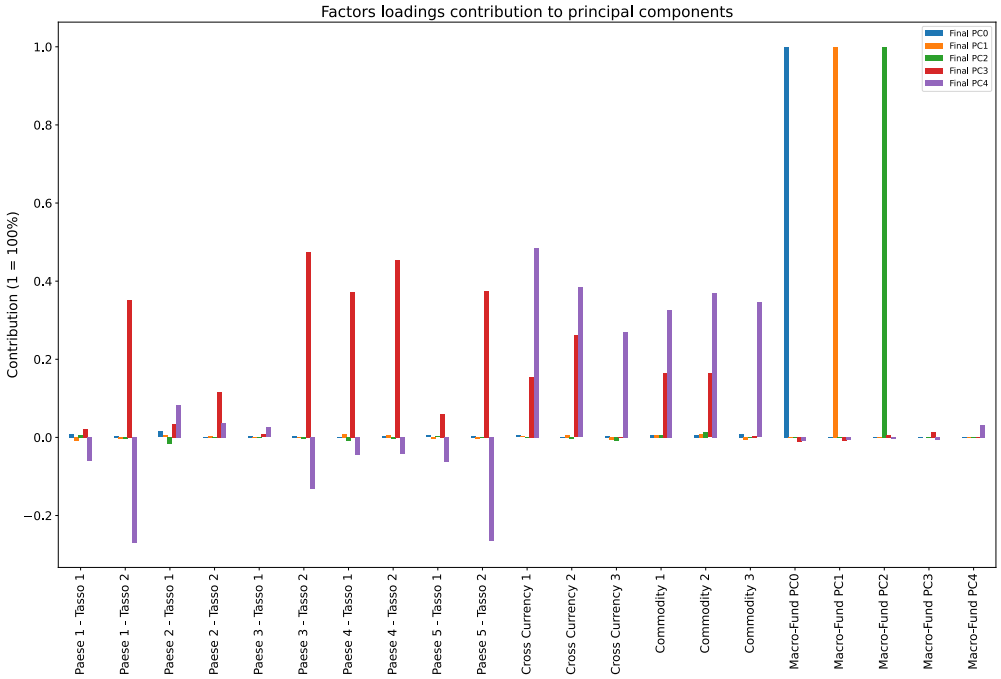

Figure 6: Composition of final principal components

Regressors	**Mean Score**
LinearRegression (OLS)	-18.29
Passive Aggressive Regressor	-14.75
OLS + ElasticNet	-18.28
Huber Regressor	-19.40
Tweedie Regressor	-18.28
SGD + Huber	-12.26
SGD + Epsilon Insensitive	-13.70
SGD + Squred Epsilon Ins. + Elastic Net	-5.06

Table 3: K-Fold timeseries cross validation results.

Note: "()" means the regression model was called with the library default parameters.

In Table 3 the cross validation results are shown. It's clear that the Stochastic Gradient Descent Regressor with a squared epsilon insensitive loss function, L1 and L2 penalties (Elastic Net penalty) and epsilon chosen as 1% of the mean of the volatility of each stock index return is the best performing regressor.

Let's consider a set of training data $(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)$ where $\vec{x_i} \in \mathcal{R}^m$ (our principal components) and $y_i \in \mathcal{R}$ (our stock indices returns). The goal is to learn a linear scoring function $f(x) = \vec{w}^T \cdot \vec{x} + b$ with model parameters $\vec{w} \in \mathcal{R}^m$ (our factor exposures) and intercept $b \in \mathcal{R}$. To find the model parameters, the following function is minimized using the Stochastic Gradient Descent method as explained in [5]:

$$E(\vec{w}, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\vec{x_i}, \vec{w}, b)) + \alpha R(\vec{w}) \tag{6}$$

This function is called the regularized training error where L is the Squared Epsilon Insensitive loss function and R is the Elastic Net regularization term. α is a non negative hyperparameter

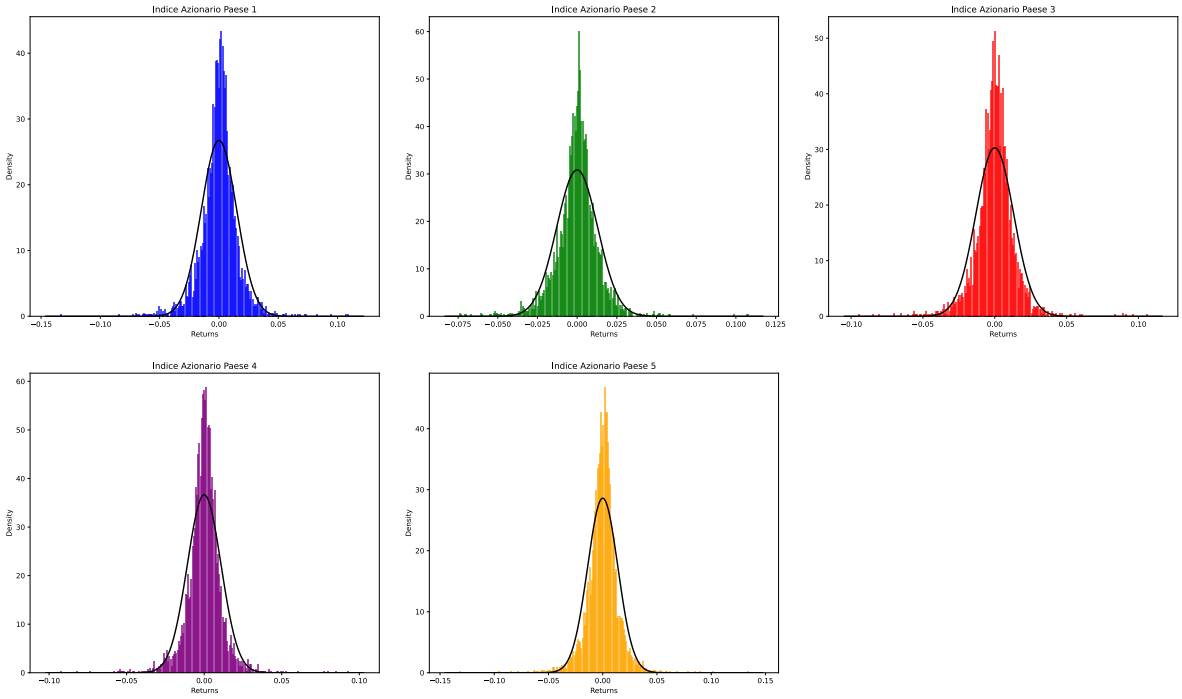

Figure 7: Gaussian fit for the residuals of each stock using Ordinary Least Squares

that controls the regularization strength. The Epsilon Squared Insensitive loss function is the following:

$$L(y_i, f(\vec{x_i})) = max(0, (y_i - f(\vec{x_i}))) - \epsilon)^2 \tag{7}$$

Thus when the error is greater than ϵ the loss function is quadratic, otherwise it's zero, ignoring that error. The above mentioned choice for epsilon now makes sense, we are telling the model to ignore errors that are less than 1% of the average returns standard deviation throughout the whole training period.

The Elastic Net regularization function is:

$$R(\vec{w}) = \frac{\rho}{2}||\vec{w}||_2^2 + (1-\rho)\sum_{j=1}^{m}|w_j| \tag{8}$$

with ρ set at default library value (0.15). The L2 norm term makes the loss function strongly convex, and therefore it has a unique minimum. It also handles highly correlated variables better than OLS. The L1 norm (second term) handles possible sparsity of coefficients.

Using this method on the training data we can calculate the matrix of factor exposures **B** and the matrix **S** in eq. (4) and use eq.(5) to calculate the covariance matrix of expected returns. With this matrix we can implement an optimization algorithm to find the best asset allocation for our portfolio. All subsequent results have been calculated using the model with the highest score (the last one in the list).

4.3 Portfolio Optimization - Risk-Parity Approach

Since Markov first economic theories there have been developed many ways to optimize the allocation of different assets in a portfolio. Should one consider the historical average of the returns ? Should one try to find the portfolio with the highest return ? Or with the lowest standard deviation ? Or maybe one should allocate assets in a portfolio in order to reach the minimum possible variance. For this task I decided to allocate the risk instead of allocating

the capital. I used a risk-parity approach that can be summed up like this: the portfolio is optimized such that each asset's contribution to the total portfolio volatility is equal. I followed the procedure in the third section of [1].

Before writing the objective function of the optimization problem, two quantities need to be introduced: the marginal risk contribution (MRC) and the risk contribution (RC). The MRC is calculated as follows:

$$M\vec{R}C = \mathbf{\Sigma}\vec{w} \tag{9}$$

where Σ is the covariance matrix of expected returns. The components of MRC represent the contribution of each asset to the portfolio volatility based on the asset's weight and the asset's volatility.

The Risk Contribution then is:

$$\vec{RC} = \frac{\vec{w} * (M\vec{R}C)^T}{\vec{w}^T \mathbf{\Sigma} \vec{w}} \tag{10}$$

where $*$ represents element-wise multiplication and the denominator is the total portfolio volatility as in eq.(2). Now the objective function can be constructed easily, we impose that each element of the Risk Contribution vector is equal, thus:

$$f(\vec{x}) = \sum_{\substack{i=1 \\ j=i+1}}^{N} (\vec{RC}_i(\vec{x}) - \vec{RC}_j(\vec{x}))^2 \tag{11}$$

So the minimization problem, adding the constraints for a long-only trading model and complete asset allocation is:

$$\begin{cases} \text{minimize } f(\vec{x}) \\ \text{subject to:} \\ \sum_{i=1}^{N} x_i = 1 \\ \vec{x} \in \mathcal{R}_+^N \end{cases}$$

where $\vec{x}$ is the portfolio weights vector and N is number of assets in the portfolio. I used the function *scipy.optimize.minimize* with the Sequential Least Squares Programming method.

The solutions to this minimization problem are the optimal, risk-parity adjusted, weights with which to build the portfolio.

The optimization procedure has been implemented in the custom python module **optimize.py**.

4.4 Model Implementation

In order to better represent an actual real-world implementation of this model I decided to use a Rolling Window approach. I start with a certain division at a certain date between training and testing data with the following algorithm:

Algorithm 1 Rolling Window Model Deployment

```
TEMPDATE = initial training/testing division date
while TEMPDATE < FINALDATE do
    Update training and testing datasets with new dates from TEMPDATE
    Normalize the factors to unit variance and zero mean
    Apply PCA on factors as described in Section 4.1
    Lag factors so that the factors of the day before are used with the next day stock returns
    Run SGD regressor as described in Section 4.2 and get factor exposures
    Calculate covariance matrix of expected returns with eq. (5)
    Find optimized portfolio weights as described in Section 4.3
    Apply optimized weights to testing data
    Store portfolio daily returns
    go to next period: TEMPDATE += chosen OFFSET (day, week, month, year)
end while
return Daily portfolio returns over testing period from initial value of TEMPDATE to FINALDATE
```

For example given an offset of one month and the initial training comprised of the first $(N-1)$ years of historical data, the algorithm would run the model using the first month of the last year as testing data during the first run, using the first year as training data. In the second run the previous month is added to the training data and the next month is selected for testing. This repeats until we reach the end of the historical data.

5 Model's Performance Analysis

This last section is dedicated to showing the model performance by applying the rolling window approach defined in the previous section to different situations.

Table 4 shows the portfolio total return, total volatility and Sharpe ratio starting the rolling window on the given periods of historical data using a weekly (5 business days) rolling frequency.

For each of these periods the portfolio matrix is provided as an Excel file in the *portfolio-matrices/* folder.

Rolling Window Period	Total Return	Total Volatility	Sharpe Ratio	Max single return	Min single return	Max single volatility	Min single volatility
2003-12-31 to 2019-11-29	89.37%	77.60%	1.15	287.79%	-12.12%	212.65%	155.45%
2018-12-31 to 2019-12-30	21.78%	11.76%	1.85	34.13%	13.35%	33.73%	26.47%

Table 4: Portfolio returns using the trading model with a rolling window approach and rebalancing frequency of one business week, thus five business days. Max/min single represent the max/min single index volatility/return between the assets of the portfolio.

As we can see from Figures 8 and 10 the portfolio is risk-parity optimized.

In all three periods the portfolio has the lowest variance but still manages to compete fairly in terms of returns with all the other stocks.

In Figure 9 we can see the behavior of the model around 2008-2010, the weights become extremely volatile but the risk-parity approach is still able to mitigate the risk. It could be interesting to further study the correlation between a crossing of weights over time as a signal of a period of extreme market volatility.

As shown in [3] an LSTM Neural Network is way more efficient at describing financial assets, this is especially true thanks to the concept of long and short term memory, exactly what this linear model needs. The idea is that it would be reasonable to expect the model to care more about data that is temporally closer instead of data from years prior. The fine tuning is in how to construct long and short term memory, so that the model can adapt quickly to changes in market volatility while also keeping the information of older but nonetheless important market conditions.

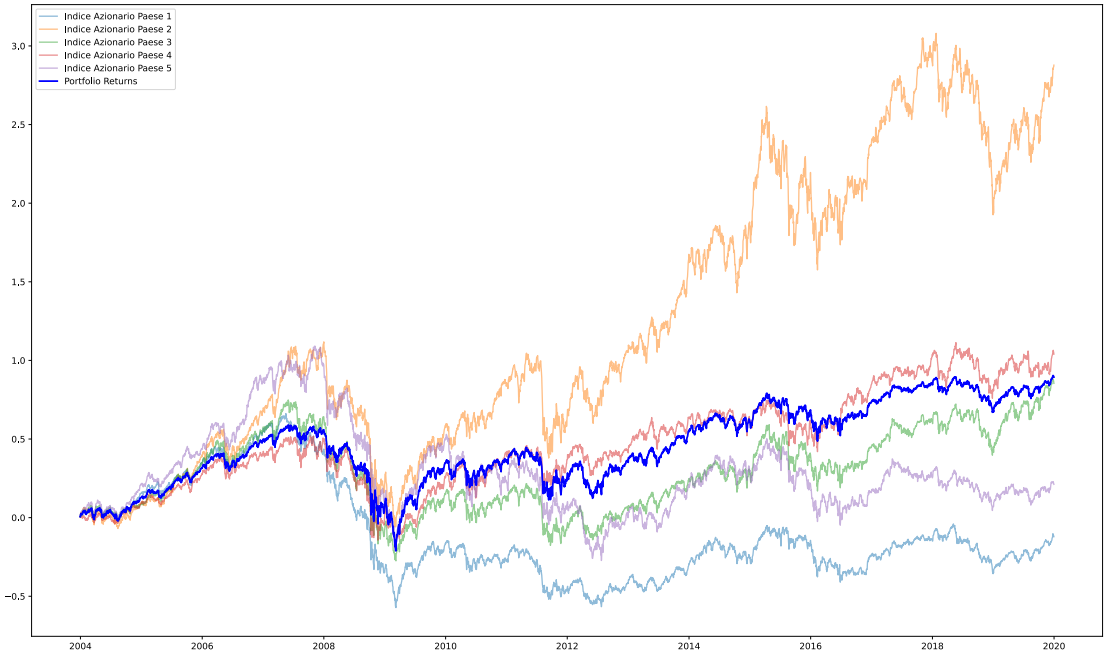

Figure 8: Portfolio and single stock indices cumulative returns for first rolling window period in table 4

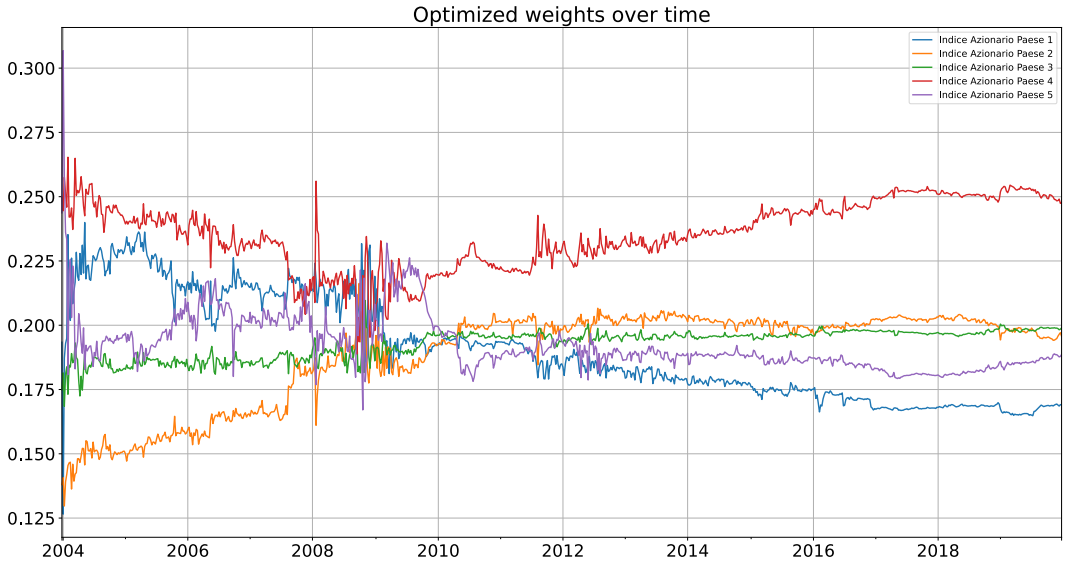

Figure 9: Optimized weights over time for first rolling window period in table 4

Figure 10: Portfolio and single stock indices cumulative returns for the second rolling window period in table 4

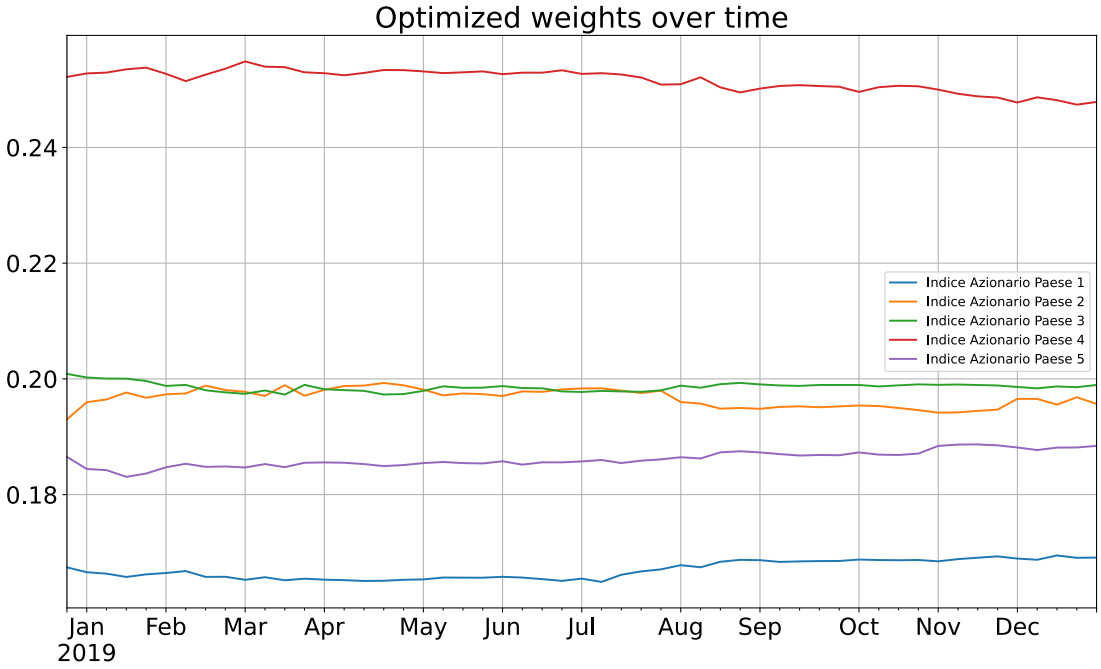

Figure 11: Optimized weights over time for the second rolling window period in table 4

6 Conclusion

In conclusion, this study implemented a long-only trading model for five stock indices using a linear multi-factor approach combined with Principal Component Analysis on the factors. The risk-parity portfolio optimization strategy demonstrated robust performance across different time periods, effectively balancing risk among the assets. Especially we can see the ability of the model to adapt to the period of high market volatility in the 2008-2010 period.

Future work could explore the integration of more advanced time series models, such as LSTM neural networks as in [3], to better capture long and short-term memory effects in financial markets. Additionally, investigating the relationship between weight crossings and periods of extreme market volatility could yield valuable insights for risk management strategies.

Overall, this model provides a solid foundation for portfolio optimization in a multi-asset context, with potential for further refinement and application in real-world trading scenarios.

List of Figures

References

[1] Marc S. Paolella, Pawel Polak, and Patrick S. Walker. "Risk Parity Portfolio Optimization under Heavy-Tailed Returns and Time-Varying Volatility". In: *SSRN* (Dec. 2023). URL: http://dx.doi.org/10.2139/ssrn.4652551.

[2] Edward E. Qian, Ronald H. Hua, and Eric H. Sorensen. *Quantitative Equity Portfolio Management: Modern Techniques and Applications*. I. Taylor and Francis Group, 2007. ISBN: 1-58488-558-0.

[3] Kei Nakagawa et al. "Deep Recurrent Factor Model: Interpretable Non-Linear and Time-Varying Multi-Factor Model". In: (2019).

[4] scikit-learn developers. *sklearn.model_ selection.TimeSeriesSplit*. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html.

[5] Tong Zhang. "Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms". In: *Proceedings of the 21st International Conference on Machine Learning*. Banff, Canada: ACM, 2004, pp. 116–123. DOI: 10.1145/1015330.1015332.