UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ADE Facultad de Administración y Dirección de Empresas /UPV

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Evaluating Score-Investing Methodologies: A Systematic Review of Tweenvest's Algorithm for Long-Term Stock Investing Using Descriptive Analytics and Predictive Modeling.

Trabajo Fin de Grado Integrado
Grado en Administración y Dirección de Empresas
Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Author: Carlos Eduardo Domínguez Martínez

Tutor: Joaquín Martínez Minaya (GADE)

Tutor: Alberto Albiol Colmener (GITST)

Extern Co-Tutor: José Tatay Sangüesa (Tweenvest)

Course: 2024-2025

Valencia, June 2025

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO.ES-CV-000030

Abstract

This study investigates the effectiveness of a factor-investing methodology developed by Tweenvest, leveraging a proprietary algorithm grounded in fundamental financial analysis. The algorithm scores companies across four key factors: Quality, Growth, Value, and Dividends.

The research aims to evaluate the profitability of investment strategies based on these scores over multiple profitability horizons from 1 month to five years. A comprehensive dataset was constructed, integrating factor scores with additional variables such as sector and geographic region, standardized for currency and timeframes.

Statistical analyses will explore relationships between these factors and returns, identifying optimal investment periods. Subsequently, predictive modeling—including econometric regressions, time series models, and neural networks—will be applied to assess the translation of these factors into market performance.

The study incorporates an interdisciplinary approach, combining financial theory and econometrics with advanced programming, data engineering, and machine learning. This integration bridges Business Management and Telecommunications Engineering, offering insights into the practical application of Tweenvest's scoring algorithm and contributing to the advancement of financial technology analytics.

To my parents.

Contents

List of Figures

List of Tables

Acronyms

ARIMA AutoRegressive Integrated Moving Average.

CAGR Compound Annual Growth Rate ($\left(\frac{\text{Ending Value}}{\text{Beginning Value}}\right)^{\frac{1}{n}} - 1$).

CAPEX Capital Expenditures.

Cash Ratio $\frac{\text{Cash and Equivalents}}{\text{Current Liabilities}}$.

CI/CD Continuous Integration/Continuous Deployment.

CROCE Cash Return on Capital Employed.

CROIC Cash Return on Invested Capital.

Cron Cron is a time-based job scheduler in Unix-like computer operating systems. It is used to schedule jobs (commands or shell scripts) to run at specified time intervals. Cron is typically used to automate tasks that need to be performed on a regular basis, such as backups, system maintenance, or data processing..

Current Ratio $\frac{\text{Current Assets}}{\text{Current Liabilities}}$.

DB Database.

Decorator Decorator is a function that takes another function and extends the behavior of the original function without explicitly modifying it. Decorators are typically used to add functionality to functions, such as logging, caching, or authentication..

Dividend Yield $\frac{\text{Dividend per Share}}{\text{Price per Share}} \times 100\%$.

DPS Dividends per Share.

EBIT Earnings Before Interest and Taxes.

EBITDA Earnings Before Interest, Taxes, Depreciation, and Amortization.

EPS Earnings per Share.

EV/EBIT $\frac{\text{Enterprise Value}}{\text{TTM EBIT}}$.

EV/EBITDA $\frac{\text{Enterprise Value}}{\text{TTM EBITDA}}$.

EV/FCF $\frac{\text{Enterprise Value}}{\text{TTM Free Cash Flow}}$.

EV/Sales $\frac{\text{Enterprise Value}}{\text{TTM Revenue}}$.

FCF Free Cash Flow.

Financial Leverage $\frac{\text{Total Assets}}{\text{Total Equity}}$.

GAM Generalized Additive Model.

ICD Island Cases Detection.

IF Isolation Forest.

IQR Interquartile Range.

Job Code script that is executed by the system to perform a task, this could be a daily update of the financial information of the stocks, or a necessary action such as sending emails to get authentication pins..

LOF Local Outlier Factor.

MAE Mean Absolute Error.

MCOD Multi-Criteria Outlier Detection.

NN Neural Network.

OCF Ratio $\frac{\text{Operating Cash Flow}}{\text{Current Liabilities}}$.

OCR Optical Character Recognition.

P/B $\frac{\text{Market Cap}}{\text{Tangible Equity}}$.

P/CF $\frac{\text{Market Cap}}{\text{TTM Operating Cash Flow}}$.

P/E $\frac{\text{Market Cap}}{\text{Adjusted TTM Earnings}}$.

P/S $\frac{\text{Market Cap}}{\text{TTM Revenue}}$.

Pair Plot Pairwise relationship plot.

Payout Ratio EPS $\frac{\text{DPS}}{\text{EPS}}$.

Payout Ratio FCF $\frac{\text{DPS}}{\text{FCF}}$.

Payout Ratio Owner Earnings $\frac{\text{DPS}}{\text{Owner}}$.

Quick Ratio $\frac{\text{Current Assets} - \text{Inventory}}{\text{Current Liabilities}}$.

REDIS REDIS is an open-source, in-memory data structure store, used as a database, cache, and message broker. It supports data structures such as strings, hashes, lists, sets, and sorted sets with range queries, transactions, and different levels of on-disk persistence..

RMSE Root Mean Square Error.

ROA Return on Assets.

ROCE Return on Capital Employed.

ROE Return on Equity.

ROIC Return on Invested Capital.

SHAP SHapley Additive exPlanations.

SVM Single Vector Machine.

Unit Test Unit test is a type of software testing where individual units or components of a software are tested. The purpose of unit testing is to validate that each unit of the software performs as expected. Unit tests are typically written by the developers and run automatically as part of the development process..

Part I

Main Report

Chapter 1

Introduction and Objectives

1.1 Fundamental Analysis Investing

Fundamental Analysis is a methodology used to evaluate the intrinsic value of a company, asset, or market by analyzing various economic, financial, qualitative, and quantitative factors. Unlike technical analysis, which focuses on price movements and chart patterns, fundamental analysis seeks to determine an asset's *"true value"* to identify investment opportunities that may be undervalued or overvalued in the market.

This intrinsic value is defined by numerous experts and renowned investors—such as Benjamin Graham, Warren Buffett, and Pat Dorsey [11]—as a company's ability to adapt to an ever-changing environment, create value in the market, and establish barriers to entry for competitors–commonly referred to as *economic moats*.

Measuring these moats is challenging due to the difficulty of quantifying certain variables, such as brand strength and market influence. However, over the long term most literature show that these **intangible factors translate into tangible financial data**, which are reflected in a company's reports: balance sheet, income statement, and cash flow statement. When companies are exposed to the public market share, all of this information transforms into the needs of buying or selling the stocks, altering the companies stocks' profitability. So from now on, these **economic moats will be called *alpha***, Sharpe [20].

This complexity is what made José Tatay, Tweenvest's founder, develop a series of algorithms to calculate certain scores, based on the fundamental analysis principles, to help users identify possible *alpha* in a company; which is the main objective for all of the long-term investors. This methodology will be explained further more in the next chapter.

1.2 Possible Problems With the Scores' Algorithm

In the context of this thesis, we have to explain the need to make a systematic review of the scores to **check wether the simplifications and hypotheses assumed** by the financial consensus, and the way that they are implemented in Tweenvest's scores, **truly show a company's ability to create *alpha*** over time or not.

1.2.1 Unquantified Variables

Since the current algorithm only includes variables found in a company's financial statements, there is a significant amount of relevant information being left out, for example:

- The **perceived differentiation** of a product is one such element that may not be properly captured by traditional financial analysis. A trusted brand or strong reputation can allow a company to charge premium prices and build customer loyalty, generating extraordinary long-term profits. For instance, brands like Tiffany or Rolex have high perceived value that justifies elevated prices—something that cannot be easily quantified using standard financial metrics such as profit margins or return on capital.
- Additionally, strategies such as **cost leadership** and offering products at lower prices can provide a significant competitive advantage that is not always directly reflected in financial data.
- Companies may also establish **barriers to entry** and **high switching costs** for customers, making it more difficult for them to move to competitors. These strategies may involve investments in technology, patents, or simply building long-term customer and employees relationships.

To properly measure these variables, a more exhaustive analysis of each company would be required, pulling from multiple secondary information sources such as news articles, conference transcripts, customer blogs, competitive product reviews, and more; which could also itself affect the market. This task is commonly known as *social listening* [22]; and is far more difficult to automate via code, as it would require multi-modal AI techniques –thus, it will fall outside the scope of this work.

1.2.2 Emergent Effects in Complex Systems

In complex systems like financial markets, network effects emerge, adding noise to actual reliable data behind it. For example:

- **Herd behavior** is a network effect where investors tend to follow the actions of others rather than rely on their own analysis. This can amplify market movements, both upward and downward. Herd behavior can lead to speculative bubbles and abrupt corrections when collective expectations shift.
- **Feedback loops** are another effect where market participants' actions reinforce existing market behavior. For example, rising asset prices may attract more investors, which in turn drives prices even higher. This type of positive feedback can cause price escalation that becomes detached from underlying fundamentals. Conversely, negative feedback can occur during a sell-off, where falling prices trigger more selling, further accelerating the decline.
- **Macro-level influences** are another crucial factor where broader structural changes—such as political decisions, economic policies, technological shifts, or industry-wide trends—can significantly impact market behavior. These larger forces often operate beyond the scope of individual company analysis and can create ripple effects throughout the entire market ecosystem.

1.2.3 Linearity and Stationarity

Non-linearity

The relationships between financial variables are often non-linear, meaning that changes in one variable may not result in proportional changes in another.

- **Economies of scale** can create non-linear relationships between production volume and costs. As a company grows, it may experience decreasing marginal costs due to better resource utilization, bulk purchasing discounts, or spreading fixed costs over larger output.
- **Market saturation** can lead to diminishing returns on marketing spend or R&D investments. Initial investments might yield significant returns, but as the market becomes saturated, additional spending may produce smaller incremental benefits.
- **Competitive dynamics** can create threshold effects where small changes in market share or pricing can trigger significant shifts in competitive position or profitability.

Non-stationarity

Traditionally, standard growth ratios have been used, based on the assumption that in competitive markets, when a new business model or opportunity emerges with above-average margins, entrepreneurs quickly move in to capitalize on it; eventually saturating the opportunity and driving margins back down to average levels over time. But what happens when there is a significant shift in trend?

Markets are "fluctuating entities", so **static metrics can become problematic when underlying trends change**. A clear example is the rise of artificial intelligence and the surge in stock prices of companies involved in the production and development of the necessary technologies.

1.3 Objectives

Given the limitations identified mentioned for the current algorithm, the main aim of this work is to **evaluate if the scores represent *alpha* for different time frames**, **guide users how to interpret them based on different investment strategies**, and to **propose possible changes to the scores calculation** in order to present the results with more accuracy. To achieve this, we followed these objectives:

1. **System Architecture Enhancement**: Modify Tweenvest's database architecture to enable historical score storage and retrieval and create the datasets for the analysis.
2. **Data Curation**: Clean and preprocess the collected data to ensure consistency and reliability, handling missing values and outliers appropriately.
3. **Exploratory Analysis**: Process and analyze the data to check the distributions and correlations between variables, looking for early on patterns to later compare and use in the modeling.
4. **Predictive Modeling**: Develop multiple predictive models to better understand the scores and their relations with other variables and time frames.
5. **Validation & Benchmarking**: Back-test the algorithms to check their performance and consistency to see if the scores generate positive *alpha* over time.

1.4 Variables for the Analysis

For doing so, we are going to look for correlations between the profits of a company with different scores and other categorical variables.

1.4.1 Response Variables

To help users understand how to use the scores depending on their investment horizon, we used different profits that from now on will be represented as P_i for each time horizon: $i \in \{1 \text{ month}; 3 \text{ months}; 6 \text{ months}; 1 \text{ year}; 2 \text{ years}; 5 \text{ years}\}$.

The profits are defined as:

$$R_i^n = \frac{(P_{i+n} + D_{acc}) - P_i}{P_i} \tag{1.1}$$

Here, R_i^n represents the total return achieved over a period of n months starting from time i. This return accounts for both the price appreciation and the dividends received during the specified time frame. Specifically, P_i denotes the price at time i, while D_{acc} is the sum of all dividends accumulated from time i to $i + n$.

1.4.2 Explanatory Variables

To relate the profits with the scores, we need to use different variables:

- **Main variables**, these are numerical variables from **Tweenvest's Scores**, that are in the range of 0 to 100: S_j where $j \in \{$ Quality; Value; Dividend; Growth$\}$
- **Dummy variables**, these are boolean variables that allow us to tag the companies by:
  - **Industry**: I_k where $k \in \{$Financials; Healthcare; etc.$\}$
  - **Geographical Region**: G_l where $l \in \{$Europe; North America; etc.$\}$
- **Market Variables**, these are numeric variables that represent the companies *"size"* using the: **Market Cap** and **Volume**.

Chapter 2

Methodology & Theoretical Framework

2.1 Tweenvest's Scores

The **long-term advantage** explained earlier in the finance is used in many indexes such as MSCI [17], IBEX-35, etc to achieve higher returns than average. So following this approach, Tweenvest developed a series of scores to **help users identify possible *alpha* in a company**. Upon these scores, we can highlight four –Quality, Growth, Value, and Dividend– that are strictly related to key focus areas that fundamental-based investors consider before making any decision, these areas are:

- Profitability
- Financial Health
- Predictability
- Consistent Growth
- Entrance Moment
- Dividends Payed

2.1.1 Quality

To understand what the Quality score aims to capture, it is useful to consider the three key categories commonly used by successful investors when analyzing financial statements: **profitability**, **financial health**, and **predictability**. Each category encompasses a range of financial ratios that reflect different aspects of a company's performance. By combining these dimensions, we can construct a more comprehensive view of a company's overall financial "quality".

To better understand the score, we need to look at each category separately. Starting with **profitability** we can separate:

Profitability Margins

These are essential for understanding how a company is managing its costs and generating profits from its revenues.

- **Net margin**: Represents net profit as a percentage of total sales, indicates a company's efficiency in generating profits after accounting for all expenses, taxes, and costs.
- **Operating margin**: Measures operating profit (EBIT) as a percentage of total sales, providing a clear view of the profitability of a company's core operations, excluding interest and taxes. This metric is fundamental for evaluating a company's operational efficiency.
- **EBITDA margin**: Removes the effects of capital structure and accounting policies, offering a clear view of the company's pure operational profitability.
- **Gross margin**: Focuses on revenues after deducting the cost of goods sold, is a key measure of production efficiency and a company's ability to manage its direct costs.

Performance Ratios

These are used to measure the overall performance of the company.

- **ROA**: Measures how efficiently a company converts its assets into profits. This is especially important in capital-intensive sectors, where efficient asset management can make a significant difference in profitability.
- **ROE**: Focuses on the profits generated per dollar of equity invested by shareholders. This ratio is crucial for evaluating a company's overall profitability from the shareholders' perspective. It is an especially valuable metric for investors seeking to maximize their returns on equity investment.
- **ROIC**: Focuses on the return generated by all the funds invested in the company, including both shareholders' equity and debt. It is a comprehensive measure of a company's ability to generate value from all its sources of financing.
- **ROCE**: Measures the funds used to finance operations, regardless of the source. This ratio is useful for comparing the efficiency of companies with different capital structures, as it focuses on total capital employed rather than just equity.

Also, to not only look at actives, Tweenvest uses the cash generated to calculate: **CROIC**, **CROCE**, **OCF/Sales**, and **FCF/Sales**. And lastly it takes account also the Owner's income to calculate: **Owner's Income/Sales**, **Owner's CROIC** and **Owner's CROCE**.

Continuing with the **financial health**, we need to analyze the company's debt in different aspects:

Leverage Ratios

These ratios assess how much a company relies on debt to finance its assets and operations, and are essential for evaluating financial risk and long-term solvency.

- **Financial Leverage**: Measures the proportion of a company's assets that are financed by shareholder equity. A higher ratio suggests the company is using more debt relative to equity, indicating greater financial risk but also potential return amplification through leverage.
- **Total Debt/Assets**: Indicates what portion of the company's assets is financed through debt. A lower ratio implies a more conservative capital structure, while a higher one may indicate increased risk if the company becomes over-leveraged.

- **Total Debt/Capital**: Measures the share of total capital (debt + equity) that comes from debt. This ratio is useful for understanding how dependent the company is on borrowed funds compared to its overall capital base.
- **Total Debt/Equity**: Compares the company's total debt to its shareholder equity. It provides insight into the balance between debt and equity financing. A high ratio may signal financial risk, but also the potential for higher returns if debt is managed well.

Debt Coverage Ratios

These metrics evaluate a company's ability to cover its debt using its earnings or cash flow, reflecting the sustainability of a company's debt in relation to its operational performance.

- **Net Debt/EBIT**: Shows how many years it would take for a company to repay its net debt using EBIT.
- **Net Debt/EBITDA**: Similar to the above, but adds back depreciation and amortization. This gives a more cash-focused view of a company's ability to handle its debt load, and is especially useful for comparing companies in capital-intensive industries.
- **Net Debt/FCF**: Evaluates how many years of free cash flow would be needed to pay off net debt. Since FCF includes investment needs, this ratio gives a more conservative view of debt sustainability.
- **Net Debt/Owner's Income**: Compares net debt to the income available to equity holders, after all operating and investing costs.

Interest Coverage Ratios

These ratios measure how easily a company can meet its interest payments on outstanding debt — critical for assessing short-term debt service capability.

- **EBIT/Interest**: Indicates how many times a company can cover its interest expenses with its operating income.
- **EBITDA/Interest**: Similar to the above, but adds back depreciation and amortization. This gives a clearer picture of available cash earnings before fixed financial obligations, ideal for heavily asset-based businesses.
- **FCF/Interest**: Since FCF considers investment needs, this is a stringent test of how much real, discretionary cash is available for debt servicing.
- **Owner Earnings/Interest**: Evaluates a company's ability to meet interest payments based on the earnings effectively attributable to shareholders. It accounts for operational cash flow minus necessary capital expenditures.

Liquidity Ratios

These ratios measure a company's ability to meet short-term obligations with its short-term assets. They are essential for evaluating near-term financial health and risk of insolvency.

- **Current Ratio**: Shows whether a company has enough assets to cover its short-term liabilities. A value above 1 is generally considered healthy, though excessively high values may imply inefficiency.
- **Quick Ratio**: A more stringent version of the current ratio that excludes inventory, which may not be easily liquidated. It's useful in assessing true short-term liquidity.
- **Cash Ratio**: The most conservative liquidity metric, focusing only on cash and equivalents. It shows the immediate solvency of a company in a worst-case scenario.
- **OCF Ratio**: Assesses how well the company's operational cash flows can cover its current obligations. This offers a realistic view of liquidity since it's based on actual cash generation rather than accounting figures.

Predictability

And finally, for the quality score we need to look at the company's predictability. This is achieved by trying to fit values related to the company's success —such as Sales— to a exponential curve, using the ordinary least square method, which is supported by large financial literature [17].

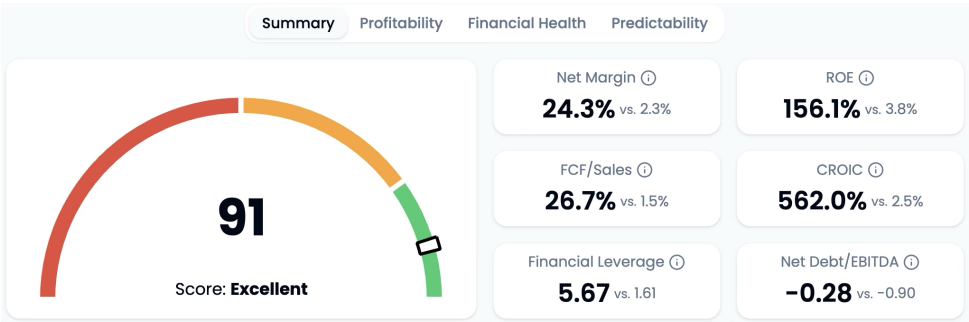


Figure 2.1: Tweenvest's Quality Score

After calculating all of this ratios, Tweenvest compares them to the sectors' median and interpolates each of them to create a single score for the ratio, then it aggregates them all using personalized weights to create the final Quality Score, which is then showed to the clients as shown in Figure 2.1.

2.1.2 Growth

The Growth Score evaluates a company's historical growth across multiple key metrics, and compares them to industry standards. This comprehensive approach ensures a balanced assessment of growth across different aspects of the business:

Revenue & Profitability Growth

- **Sales**: Measures growth in core revenue streams
- **EBITDA**: Captures growth in operational cash-generating ability before non-cash and financing impacts
- **Operating Income**: Reflects growth in profit from core business operations
- **Net Income**: Includes all income sources, showing overall profitability growth

Cash Flow Growth

- **Operating Cash Flow**: Measures growth in cash generation from operations
- **Simple FCF**: A straightforward proxy for available cash after essential investments
- **Levered/Unlevered FCF**: Provide detailed views of free cash flow with and without debt impact
- **Owner Earnings**: Useful for volatile capex cases, emphasizing cash available to shareholders

Capital Base Expansion

- **Total Assets**: Indicates expansion in overall asset base
- **Equity**: Reflects growth in shareholders' claim on the business
- **Tangible Book Value**: Highlights growth in physical net assets, excluding intangibles
- **Invested Capital**: Captures total capital being put to productive use
- **Capital Employed**: A broader measure of capital supporting business operations

Per-Share Value Growth

- **Diluted EPS**: Tracks per-share earnings growth, accounting for dilution effects
- **Diluted Shares**: Included to track share count changes, ensuring EPS growth is not artificially inflated by buybacks or dilution
- **Ordinary DPS**: Tracks the growth of shareholder payouts, a proxy for confidence in future earnings

To compute the Growth Score, Tweenvest calculates 10-year, 5-year, and 3-year averages and then interpolates the growth rate to industry standards. This approach reinforces the long-term investment philosophy, giving lasting growing companies a better score.

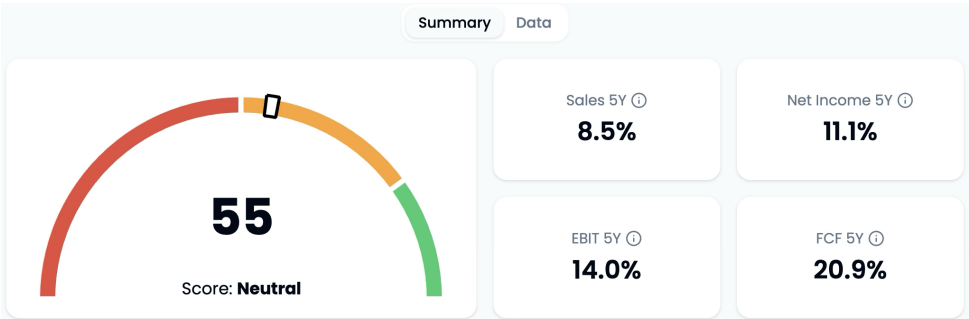


Figure 2.2: Tweenvest's Growth Score

2.1.3 Value

The Value Score measures how attractively a company is priced relative to fundamental multiples such as earnings, cash flow, sales, and dividends. This is critical for investors following value investing principles, where the goal is to buy quality companies for less than their intrinsic worth. The algorithm evaluates a series of valuation multiples—both price-based and enterprise value-based—and dividend yield.

Price-Based Multiples

- **P/E**: Measures how much investors are willing to pay per dollar of earnings
- **P/S**: Useful when earnings are volatile; shows valuation relative to sales
- **P/CF**: Reflects valuation relative to cash-generating ability
- **P/B**: Especially relevant for asset-heavy sectors like banks or industrials

Enterprise Value-Based Multiples

- **EV/Sales**: Indicates how much investors are willing to pay for each unit of revenue, providing a sense of overall company valuation relative to sales.
- **EV/EBITDA**: Shows the value the market assigns to a company compared to its operating profitability before non-cash and financing items, useful for comparing companies with different capital structures.
- **EV/EBIT**: Reflects the market value relative to operating earnings after depreciation and amortization, offering insight into core business profitability.
- **EV/FCF**: Assesses how the market values a company in relation to the actual cash it generates, highlighting the ability to produce free cash flow for investors.

Yield-Based Valuation

- **Dividend Yield**: Measures the annual return on investment from dividends, expressed as a percentage of the current share price.

Each of these ratios is compared to multiple historical statistics and sector benchmarks to create individual scores and then average them.

Note:

- *Market Cap = Share Price × Total Outstanding Shares*
- *Enterprise Value = Market Capitalization + Total Debt - Cash + Marketable Securities*

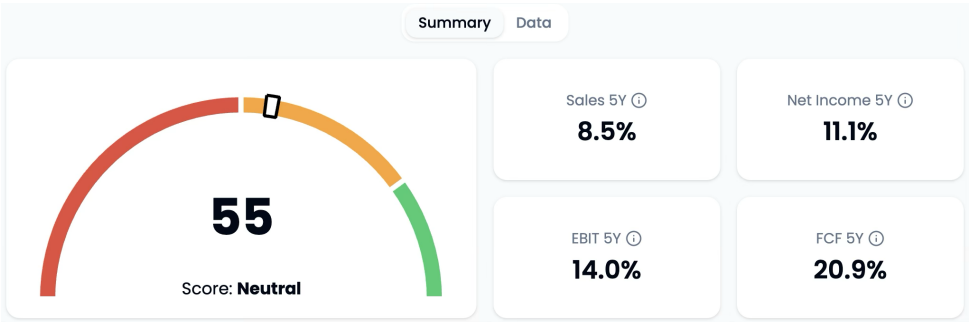


Figure 2.3: Tweenvest's Value Score

2.1.4 Dividend

When analyzing Tweenvest most common investors profile, we see a high tendency to income-focused and long-term investors using mainly dividends as their principal concern. Thats why the platform dedicated a full section to this topic.

The Dividend Score measures the attractiveness, reliability, and growth potential of a company's dividend payments. It helps investors assess whether the dividend is both rewarding today and sustainable for tomorrow. To do so, the score was built from three primary components:

Safety

This part of the score is used to assess the sustainability of the dividend.

- **Payout Ratio EPS**: Shows if dividends are covered by accounting earnings
- **Payout Ratio FCF**: Shows if dividends are funded by real cash generation
- **Payout Ratio Owner Earnings**: A conservative test of sustainability, excluding CAPEX.

Growth

Measures how consistently and strongly the dividend has grown over time.

- **Ordinary DPS CAGR**: 3-Year, 5-Year, 10-Year growth rates

Yield

This sub-score evaluates the attractiveness of the dividend today relative to the company's historical averages and sector benchmarks on Dividend Yield.

These components are individually scored weighted and interpolated against industry benchmarks to form a composite score.

Figure 2.4: Tweenvest's Dividend Score

Finally, as seen in Figure 2.4, the algorithm adjusts this score based on how many years the dividend has been maintained or increased, **rewarding consistency**.

2.2 Code practices

Since the code that needs to be changed is used in a production environment by Tweenvest, it is important to follow some *good practices* to ensure the code is easy to understand, maintain, and to avoid introducing new bugs.

- **Understanding and Documenting**: Before starting to work on the code, it is important to understand the codebase and the purpose of the code. For this, it is recommended to use some tools like flux diagrams, code comments, and documentation. As it will be shown later on.
- **Testing**: To ensure proper functionality, every function and class should have unit tests that cover all possible scenarios. These tests are run automatically by the CI/CD pipeline.
- **Pair code review**: After completing the previous steps, the code undergoes a pair code review process, where at least one other team member reviews and approves the changes in *Github* before merging into the main branch.
- **Logging**: After each feature is implemented, it is important to analyze the actual performance in the production environment by looking at its logs to check if the feature is working as expected, and to look for any possible optimizations to be made.

2.3 Preprocessing

Once we have created our raw dataset, we need to inspect it and make sure it is consistent and ready to be used for the modeling. This part is really important to avoid any bias in the modeling process and to only use the data that is actually relevant for the analysis.

2.3.1 Data Consistency

Since all of the data is coming from a data provider that uses OCR as one of their tools whenever they do not have the data in a structured format, we need to be very restrictive with the data used. Also, as we mentioned before, some of the algorithms for calculating the scores use long-term data such as 10 year growths rates.

This is one of the biggest **limitations to the study**: we will be missing the data of companies that stopped existing or that were acquired by other companies, if they did not exist for 10 or more years.

For inspecting the data we will be using Python as the main language, enhanced with: *pandas* [21], *matplotlib* [13], and *seaborn* [23], which are powerful tools for data manipulation and analysis. The focus will be on the following aspects:

- **Missing Values**: We need to check if there are any missing values in the data.
- **Data Distributions**: We need to make sure that the data is distributed in a way that is suitable for the modeling.
- **Variable Correlations**: We also have to see if there are any correlations between the variables. For this, we will use the correlation matrix defined as:

$$\rho_{X,Y} = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y} \tag{2.1}$$

  where $\text{Cov}(X,Y)$ is the covariance between the variables X and Y, σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y. This matrix will be created with a custom heatmap function (see Appendix 6.4).
- **Data Patterns**: We need to check if there are any visible data patterns between the variables. To address this, we will use a custom *Pair Plot* function from *seaborn*, which is a powerful tool for data visualization and exploration depending on the type of variable (see Appendix 6.4).

Additionally, we decided to assign a value of 0 to all null Dividend scores, as a null value indicates that the company does not pay dividends. This choice was made to ensure that the Dividend score remains part of the analysis when evaluating its relevance to long-term profitability, rather than being excluded due to missing values.

2.3.2 Data Transformation

To ensure a robust and well-performing model, it is often necessary to transform the data into a format more suitable for learning algorithms, with the obligatory condition of being able to revert it back to its original scale when needed.

This step is particularly important because many components of machine learning models—such as the RBF kernel in Support Vector Machines or the regularization terms (L1 and L2) in linear models—implicitly assume that features are centered around zero and have comparable variances. Without this adjustment, features with significantly larger variances could dominate the optimization process, leading the model to underweight or ignore more informative but lower-variance features.

Standard Scaler

One of the predictive models we will use are neural networks, and for improving the training process of this type of models it is really important to standardize the data. We can easily use

the *StandardScaler* from the *scikit-learn* library that transforms a feature x_i with the following formula:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{2.2}$$

where μ is the mean of the feature and σ is its standard deviation. **This transformation is a very common practice**, and it results in a distribution with a mean of 0 and a standard deviation of 1.

Gaussian Transformation

In numerous modeling applications, having normally distributed features is advantageous. **Power transformations** represent a set of parametric, monotonic functions that are an extension of the Box-Cox transformation. They are designed to convert data from various distributions into approximately Gaussian distributions, thereby reducing variance fluctuations and decreasing distribution asymmetry.

We decided to use the **Yeo-Johnson transformation** for the profits response variables, because it allows for negative values and it is reversible.

$$x_i^{(\lambda)} = \begin{cases} \frac{[(x_i+1)^\lambda - 1]}{\lambda}, & \text{if } x_i \geq 0, \lambda \neq 0 \\ \ln(x_i + 1), & \text{if } x_i \geq 0, \lambda = 0 \\ -\frac{[(-x_i+1)^{2-\lambda} - 1]}{2-\lambda}, & \text{if } x_i < 0, \lambda \neq 2 \\ -\ln(-x_i + 1), & \text{if } x_i < 0, \lambda = 2 \end{cases} \tag{2.3}$$

Where λ is a power parameter that helps minimize the skewness of the data. And since we have already deleted extreme outliers, the final distribution should not be too distorted.

In Figure 2.5, we depict some examples of the transformation for different distributions:

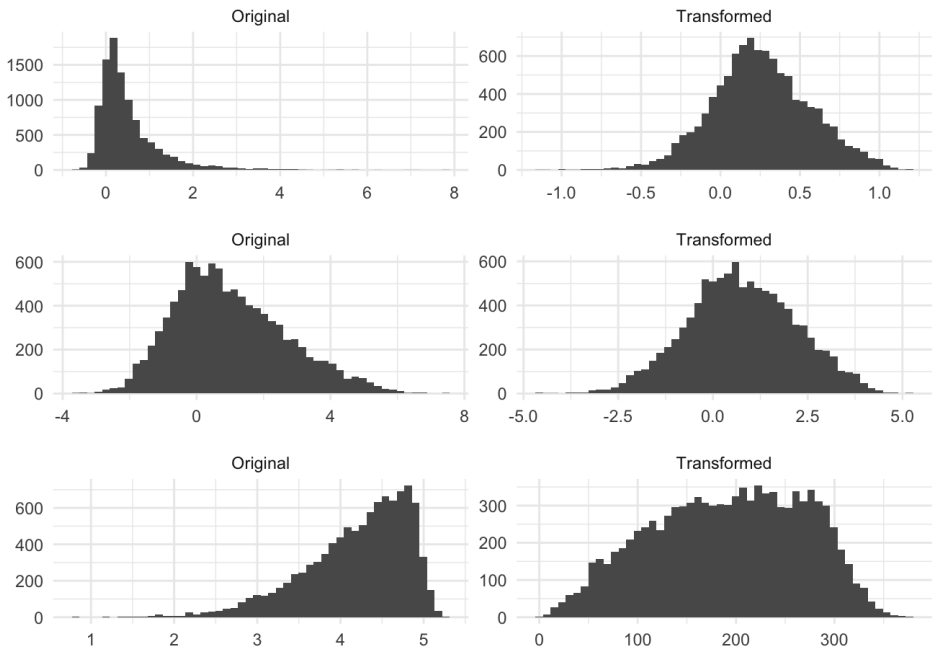


Figure 2.5: Yeo-Johnson Transformations examples

2.4 Outlier Detection

Anomalies are data patterns that have different data characteristics from normal instances. The ability to detect anomalies has significant relevance since often provide critical and actionable information in many different contexts.

For example, anomalies in credit card transactions could signify fraudulent use of credit cards, or an unusual computer network traffic pattern could stand for an unauthorized access.

2.4.1 Inter Quartile Range

Given a univariate dataset, the Interquartile Rule identifies outliers based on the IQR. The steps are as follows:

1. Compute the first quartile (Q_1), which is the 25th percentile of the data.
2. Compute the third quartile (Q_3), which is the 75th percentile of the data.
3. Calculate the interquartile range:
$$\text{IQR} = Q_3 - Q_1$$
4. Define the lower and upper bounds for non-outlier values following the 1.5 rule:
$$\text{Lower bound} = Q_1 - 1.5 \times \text{IQR}$$
$$\text{Upper bound} = Q_3 + 1.5 \times \text{IQR}$$
5. Any data point x is considered an outlier if:
$$x < \text{Lower bound} \quad \text{or} \quad x > \text{Upper bound}$$

As we can see this is a very simple and intuitive method, but it has some limitations:

Pros	Cons
Simple to compute and interpret.	Not suitable for multivariate data with correlated features.
Non-parametric.	May misclassify values in skewed distributions as outliers.
Robust to extreme values, since it relies on percentiles rather than the mean.	Fixed multiplier is arbitrary and may need tuning for each dataset.

Table 2.1: Pros and Cons of the Interquartile Rule

Even though this method is simple to compute and interpret, it is not suitable for multivariate data with correlated features, and that is where the rest of the methods come into play.

2.4.2 Single Vector Machine

The Single Vector Machine (SVM) represent a robust family of unsupervised learning algorithms that can be effectively adapted for anomaly detection tasks. This algorithm establishes itself on the premise that the majority of real-world data is inherently normal. Where the goal is to define a boundary encapsulating the normal instances in the feature space, thereby creating a region of familiarity.

To capture the bases of the *scikit-learn* implementation [7] used in this work, we can say that the main idea is to find a minimum volume sphere that contain all the training samples. This sphere, described by its center c and its radius r, is obtained by solving the constrained optimization problem: [24]

$$\min_{r,c} r^2 \quad \text{subject to} \quad \|\Phi(x_i) - c\|^2 \leq r^2 \quad \text{for} \quad i = 1, 2, \ldots, n \tag{2.4}$$

This boundary is strategically positioned to maximize the margin around the normal data points, allowing for a clear delineation between what is considered ordinary and what may be deemed unusual. This emphasis on margin maximization is akin to creating a safety buffer around the normal instances, fortifying the model against the influence of potential outliers or anomalies.

In summary, this method has the following characteristics:

Pros	Cons
Effective for high-dimensional data and complex boundaries.	Sensitive to the choice of kernel and hyperparameters (e.g., ν, γ).
Works well when the training set contains mostly or only normal instances.	Computationally intensive, especially on large datasets.
Can capture nonlinear patterns using kernels (e.g., RBF kernel).	Difficult to interpret or explain the decision function.
No need for labeled data from the outlier class.	–
Solid theoretical foundation and widely supported in machine learning libraries.	–

Table 2.2: Pros and Cons of One-Class SVM for Outlier Detection

2.4.3 Isolation Forest

As we are seeing in this small fraction of outlier detection methodologies, most of the existing anomaly detection approaches are based on the premise of normal distributions, then identify anomalies as those that do not conform to the normal profile.

But to solve this issue we can use the Isolation Forest algorithm, which is explained in the original paper Liu, Ting, and Zhou [14]. The algorithm constructs multiple isolation binary-trees (iTrees) from the dataset, where anomalies are identified as data points that exhibit shorter average path lengths across these trees.

Lets consider a dataset $X = \{x_1, \ldots, x_n\}$ containing n points in d-dimensional space, and a subset $X' \subset X$. An Isolation Tree (iTree) is constructed as follows:

- Each node T in the tree is either:
  - An external node (leaf) with no children, or
  - An internal node with exactly two children (T_l and T_r) and a test condition
- The test condition at each internal node consists of:
  - A randomly selected attribute q
  - A split value p
  - A test $q < p$ that determines whether a point goes to T_l or T_r

The tree construction process recursively partitions X' by randomly selecting attributes and split values until either:

- The node contains only one instance, or
- All instances at the node have identical values

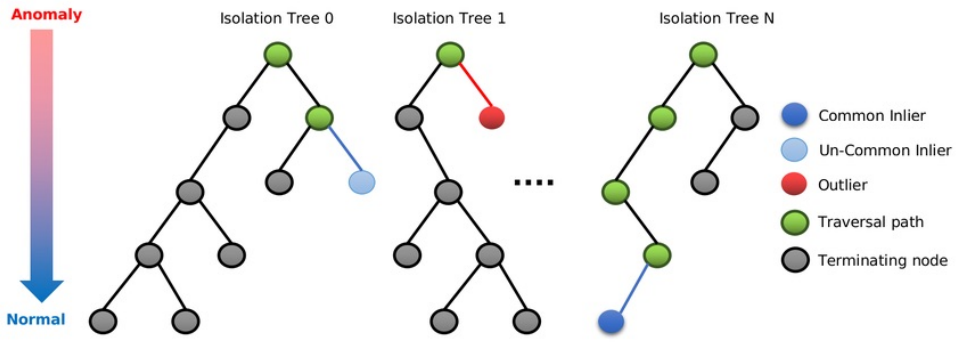


Figure 2.6: Visualization of how Isolation Forest works.

As we can see in Figure 2.6, once the iTree is fully constructed, each point $x_i \in X$ is isolated at a leaf node. The path length $h(x_i)$ of a point x_i is defined as the number of edges traversed from the root to its leaf node. Points with shorter path lengths are considered more likely to be anomalies, as they require fewer splits to be isolated from the rest of the data.

This method works very good in most cases, but it has also its own limitations:

Pros	Cons
Efficient and scalable to large datasets due to its tree-based structure.	Less effective for detecting local outliers in densely clustered regions.
Non-parametric: does not assume any data distribution.	Performance can vary with the choice of parameters like number of estimators and subsample size.
Naturally handles high-dimensional data.	Not suitable for detecting subtle anomalies that closely resemble normal data.
Requires little preprocessing (no need for feature scaling).	–
Robust to irrelevant features due to random sub-sampling.	–

Table 2.3: Pros and Cons of the Isolation Forest Method for Outlier Detection

2.4.4 Local Outlier Factor

Since we are looking at outliers from the perspective of correlations between the different response variables to see which companies do not behave naturally. We also checked on the Local Outlier Factor (LOF) for being based on hyper-plane densities of data.

The algorithm measures the local deviation of the density of a given sample with respect to its neighbors. It is local in that the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood. More precisely, locality is given by k-nearest neighbors, whose distance is used to estimate the local density. By comparing the local density of a sample to the local densities of its neighbors, one can identify samples that have a substantially lower density than their neighbors. These are considered outliers. Breunig et al. [3]

Pros	Cons
Can detect outliers relative to their local neighborhood, allowing it to identify context-specific anomalies.	LOF scores are relative and lack a universal interpretation threshold for outliers.
Effective in datasets with varying densities — unlike global methods, it can recognize outliers near dense clusters.	Sensitivity to parameter selection (e.g., number of neighbors) can affect performance and consistency.
Applicable in any domain where a dissimilarity measure is defined, not limited to vector spaces.	Can detect outliers patterns that could create a bias for models, because they do not belong to the normal distribution.
Works well across domains, such as network intrusion detection or classification tasks.	–
Easily generalizable and adaptable for use in spatial data, temporal data, and network structures.	–

Table 2.4: Pros and Cons of the LOF Method

2.4.5 Multi-Criteria Outlier Detection

There is a big controversy in the field about the best method to detect the outliers, since the results can vary a lot depending on the algorithm used.

So to attempt to fix this issue, after reading some aggregative methods such as Abro, Taşci, and Uğur [1], we created a multi-criteria method that combines the results of the different methods to get a more robust result.

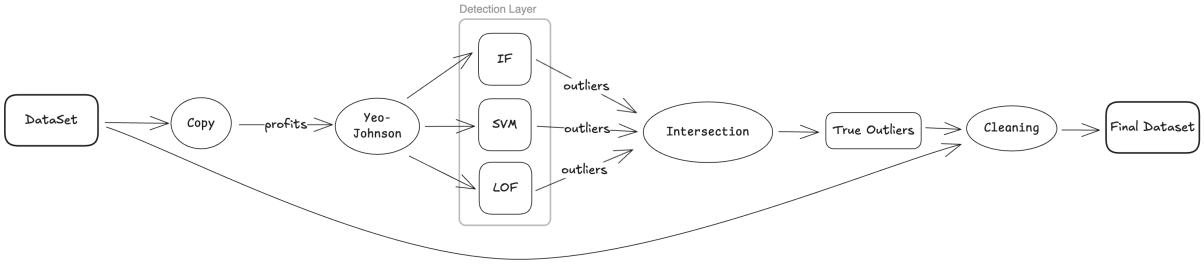


Figure 2.7: MCOD Schema

As shown in the Figure 2.7, this starts by homogenizing the variables distribution with a common data transformation so they are more Normal, but to make sure that the final data is not being transformed we copy the original dataset earlier. After this, we use multiple methods to detect the different outliers and then we make the intersection between them to get the common outliers. By the end of this process we have tagged the original dataset rows, assuring that the data that will be excluded is only formed by *"true outliers"*.

Island Cases detector

When looking at the 2.7, we can expect it to save different data points that are not common between all of the methods due to its intersection of *true outliers*. This might be good for many classification tasks but when it comes to regression it can cause bias since there could be *islands* –clusters– of data that do not follow a normal distribution. Thats why we created a variation of the MCOD method that should only keep the normal data but also saving those islands of data for further future analysis. Since those points do not have to be errors in the data, they could show **extreme but consistent behavior** that should be modeled separately.

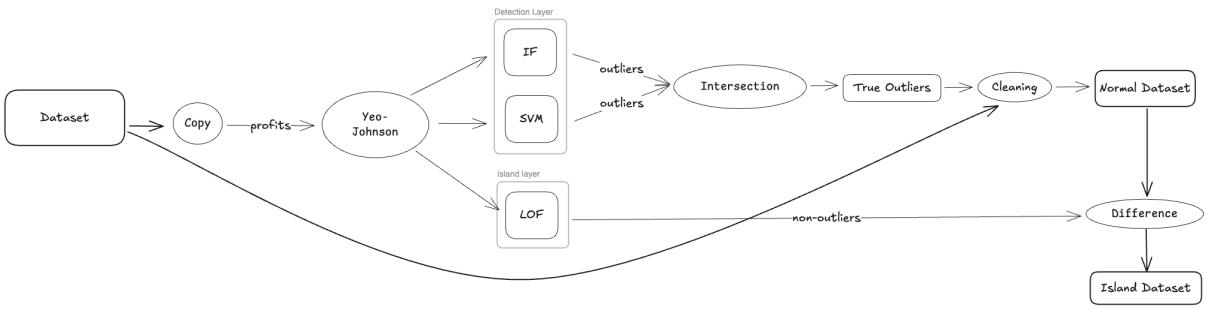


Figure 2.8: ICD Schema

2.5 Predictive Models

The main objective of this thesis is to check if the scores have any relation with the future profits of the companies. For doing this in the most generalized way, instead of creating a portfolio following some subjective criteria on the scores, markets and other variables, we used predictive modeling to check for the actual relationships between the response variables and the explanatory ones.

Since the main objective is to check the relations between the variables, we will use regression models to understand it those relationships. Starting with simple linear regression and then generalizing to non-linear models. And finally to test if we can also predict future profits we will try to use neural networks.

2.5.1 Regression Models

To create the econometrical models, we wanted to be able to understand the actual relationship between the response variables and the explanatory ones. For this we started with the creation of multiple regression models.

Linear Regression

As for the first models, we created a family of linear regression models ($i \times j$) for each factor i-return and j-score. These models include the interactions between the dummy variables and the j-score used in each model.

$$\mu_{i,j} = \beta_0 + \beta_1 \cdot S_j + \sum_{n=2}^{N} \beta_n \cdot D_n + S_j \cdot \sum_{m=N+1}^{M} \beta_m \cdot D_m \tag{2.5}$$

where:

- $R_{i,j} \sim \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$ is the return for each time horizon i and factor j. So $R_{i,j} = \mu_{i,j} + \varepsilon_{i,j}$ where $\varepsilon_{i,j}$ is the error and $\mu_{i,j}$ is the expected return.
- S_j is the score for each factor j
- $D_{n,m}$ are the dummy variables for the industry and region, where $D_{n,m} := I_k \cup G_l$
- β are the coefficients that weight each variable.

For fitting the models, we also applied a **backward selection** to remove the variables that are not statistically significant, using the *p-value* criterion **with a 20% tolerance**. This threshold will be used for most of the analysis because in financial data it is harder to get high precisions.

Generalized Additive Models

Since the relationships between the variables are most likely non-linear, we also used the Generalized Additive Models (GAM) for the mentioned ($i \times j$) family to check if they are able to improve the performance, and also see if there are consistent patterns across the different time horizons.

With this analysis we could then let know Tweenvest what are the *"best"* different scales for each score to better guide its users. Since right now –as it was shown in Figures 2.1 to 2.4– they are scaled from [0,50] as a bad score, (50, 80] as a good score, and (80, 100] as a great score. And this has not been tested yet.

According to Servén and Brummitt [19], GAMs are smooth semi-parametric models that can capture non-linear relationships between variables. They take the form:

$$g(\mathbb{E}[y|X]) = \beta_0 + f_1(X_1) + f_2(X_2, X_3) + \cdots + f_M(X_N) \tag{2.6}$$

where:

- $X^T = [X_1, X_2, \ldots, X_N]$ are the explanatory variables.
- y is the dependent variable, in our case the returns.
- $g()$ is the link function that relates the predictor variables to the expected value of the dependent variable.
- $f_m()$ are feature functions built using penalized B-splines, which automatically model non-linear relationships without requiring manual transformation of variables.

In our case, we used pyGAMs LinearGAM since the model gives a Normal error distribution, and an identity link. Using this library also gives us the possibility to differentiate between the different variable types:

- **s(i) - Smooth Splines:** Standard smooth splines (cubic by default) for continuous variables like scores: $f_m(X_m) = \sum_{j=1}^{k} \beta_{mj} B_j(X_m)$ where B_j are B-spline basis functions
- **f(i) - Factor Splines:** For categorical variables (sectors and regions): $f_m(X_m) = \sum_{j=1}^{k} \beta_{mj} I_j(X_m)$ where I_j are indicator functions
- **te(i,j) - Tensor Splines:** For modeling non-linear interactions between variables: $f_m(X_i, X_j) = \sum_{k=1}^{K_i} \sum_{l=1}^{K_j} \beta_{kl} B_k(X_i) B_l(X_j)$

In Figure 2.9 we can see an example of how GAMs can show the non-linear relationships between variables.

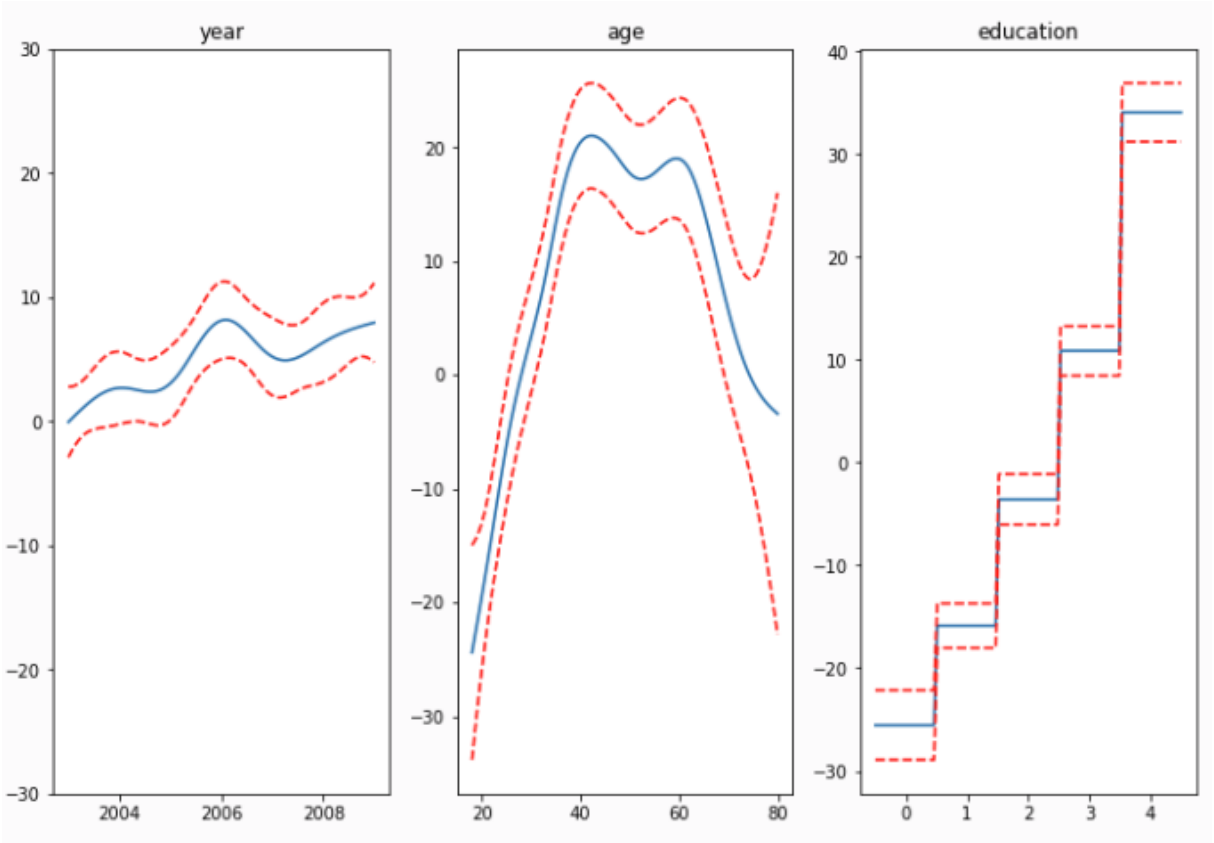


Figure 2.9: pyGAM - Example of GAMs regression splines

2.5.2 Time Series

In the stock market, the prices of the stocks are not independent from each other so as we will see with the descriptive analysis, they are correlated in the time series and they also have memory on past behavior of the company. This is what is known as **Momentum**, so companies that have a good past performance are more likely to have a good future performance, and vice versa.

For this reason, we contemplated two possibilities:

ARIMA Models

To take account the profit tendency, we implemented ARIMA models to improve the regression models performance by using the residues of the already fitted model:

$$P_i = \hat{P}_i + \varepsilon_i \quad \longleftrightarrow \quad \varepsilon_i \sim ARIMA_i(p, d, q) \tag{2.7}$$

The ARIMA model is defined as:

$$\phi_i(B_i)(1 - B_i)^{d_i}\varepsilon_i^t = \theta_i(B_i)\eta_i^t \tag{2.8}$$

where for each i profit:

- ε_i^t is the residual at time t.

- η_i^t is a white noise error term (innovation).
- B_i is the backshift operator, such that $B\varepsilon_i^t = \varepsilon_i^{t-1}$.
- $\phi_i(B) = 1 - \phi_{i1}B - \phi_{i2}B^2 - \cdots - \phi_{ip}B^p$ is the autoregressive (AR) polynomial.
- $\theta_i(B) = 1 + \theta_{i1}B + \theta_{i2}B^2 + \cdots + \theta_{iq}B^q$ is the moving average (MA) polynomial.
- d_i is the order of differencing.

But this approach supposed a *big issue* that we will explain in the development section.

Windowed Models

So for trying to not fall into the same mistake and for capturing possible memory of the companies behavior in different aspects, we also implemented windowed models that used the last N scores statistical properties to predict future profits.

So for each score j, we calculated the **average of the scores** over the last N periods. Let S_j^t be the score at time t, then for a window of size N:

$$\bar{S}_j^N = \frac{1}{N} \sum_{i=t-N+1}^{t} S_j^i \tag{2.9}$$

where:

- $\bar{S}_j^N$ is the average score over window N for score j
- N can be 3 months, 6 months, 1 year, or 2 years.
- t is the current time point

For adding also the deviation from the average, we also calculated the **standard deviation of the scores** over the same window:

$$\sigma_j^N = \sqrt{\frac{1}{N} \sum_{i=t-N+1}^{t} (S_j^i - \bar{S}_j^N)^2} \tag{2.10}$$

These statistical measures are used as additional features in our predictive models to capture the temporal behavior of each score.

2.5.3 Neural Networks

NN are a type of machine learning model that is inspired by the structure and function of the human brain. They are composed of layers of interconnected nodes (neurons) that process information through a process of iterative optimization, as it is shown in Figure 2.10.

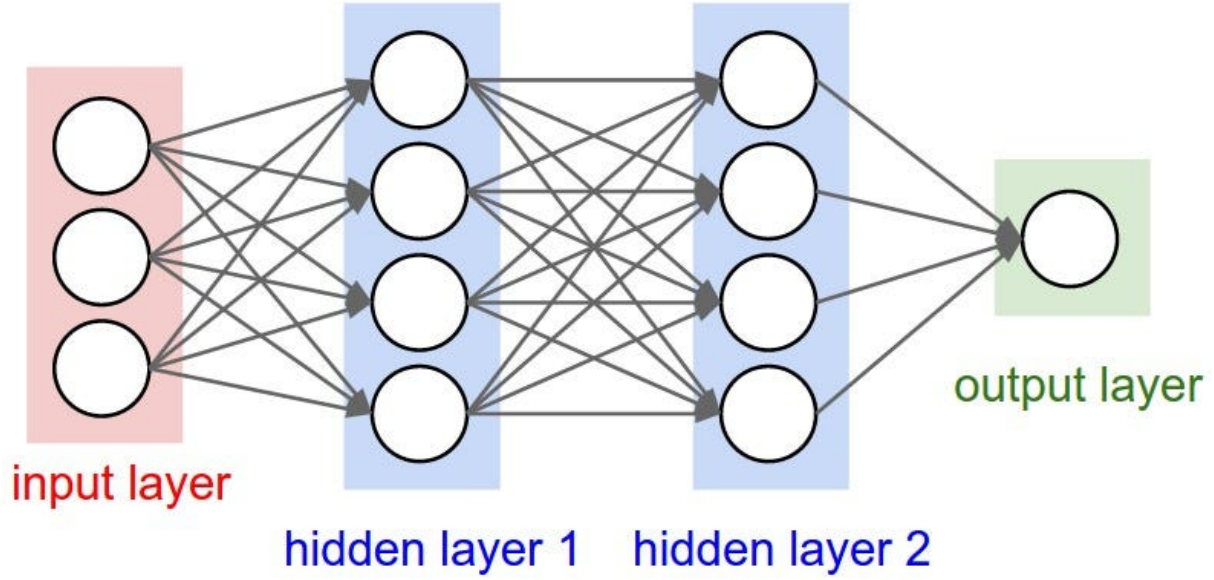


Figure 2.10: Simple Architecture of a NN

Finally, to test if we have enough information to model future profits, we will use two different approaches of different types of NN: **regressors** and **classifiers**.

Regressor Neural Networks

A **regressor neural network** is a type of artificial neural network specifically designed to predict continuous numerical values, rather than discrete categories. In the context of our analysis, these networks are trained to estimate future profit values based on a set of input features, such as historical scores and other relevant company data.

For our implementation, we utilize the **MLPRegressor** from the *scikit-learn* library [6], which provides a flexible and efficient framework for building and training these models.

Classifier Neural Networks

To simplify the implementation, we are going to use two different quantification of the profits:

- **Binary Classification:** In this approach, we partition the profit variable into two classes using the following rule:
$$y = \begin{cases} 0 & \text{if } P \leq 0 \\ 1 & \text{if } P > 0 \end{cases} \tag{2.11}$$
where P denotes the profit for a given time horizon, and y is the resulting binary class label.
- **Multi-class Classification:** Here we created different intervals of profits, using the average

inflation rate and the S&P 500 index as references– [2, 18].

$$y = \begin{cases} -3 & \text{if } P_a \leq -0.1 \\ -2 & \text{if } -0.1 < P_a \leq -0.04 \\ -1 & \text{if } -0.04 < P_a \leq 0 \\ 1 & \text{if } 0 < P_a \leq 0.04 \\ 2 & \text{if } 0.04 < P_a \leq 0.1 \\ 3 & \text{if } 0.1 < P_a \end{cases} \tag{2.12}$$

Where P_a is the **annualized profit**, calculated as:

$$P_a = (1 + P)^{\frac{12}{n}} - 1 \tag{2.13}$$

Here, P is the total profit over a period of n months, and P_a is the equivalent annualized return.

And for building these NN we will use the **MLPClassifier** from the *scikit-learn* library [5], which provides a flexible and efficient framework for building and training these models.

Architecture Exploration

Before building the models, we need to explore different architectures to find the best hyper-parameters for the models. In our case we are only going to explore the different hidden layers and the number of neurons in each layer, since the rest of the hyper-parameters are already being optimized by the library. To systematically explore the effect of the architectures, we will use the following configurations:

- **Single hidden layer:** Networks with one hidden layer and varying width:
  - (16), (32), (64), (128), (256), (512)
- **Two hidden layers:** Networks with two hidden layers, both with decreasing and uniform widths:
  - (64, 32), (128, 64), (256, 128), (512, 256), (128, 32), (256, 64)
  - (32, 32), (64, 64), (128, 128), (256, 256)
- **Three hidden layers:** Networks with three hidden layers, including both bottleneck and uniform structures:
  - (128, 64, 32), (256, 128, 64), (512, 256, 128), (64, 32, 16), (256, 64, 16), (128, 64, 8)
  - (128, 128, 128), (256, 256, 256)
- **Four hidden layers:** Deeper networks with four hidden layers, both with gradually decreasing and large bottleneck structures:
  - (512, 256, 128, 64), (256, 128, 64, 32), (128, 64, 32, 16), (64, 32, 16, 8), (1024, 512, 256, 128)
- **Hard Bottlenecks:** Networks where the number of neurons sharply decreases in each subsequent layer:
  - (256, 64, 16), (128, 32, 8), (512, 128, 32), (1024, 256, 64)

This comprehensive set of architectures allows us to compare the impact of network depth, width and *bottlenecking* on predictive performance, and to identify the optimal configuration for each problem. So we should end up with a set of best architectures for each score and profits, depending on the NN type.

2.6 Back-testing

In order to assess and compare the modeling techniques employed in this thesis, it is important to have a consistent criteria to evaluate them. Firstly we have to make clear that the main intent of this work is not to find the model that best predicts future profits, but to check weather the scores are correlated with the profits, and if so, how.

2.6.1 Training and Testing

Knowing this, as we already explained, we will use the linear models and the GAM to check the correlations and the statistical significance of each variable. Then we will use the NN to actually test the true predictability of the models.

- **Training:** We will use 80% of the stocks of the dataset to train the models, excluding the last 4 months of each stock.
- **Testing:** When it comes to testing, we want to check the NN-models in two different situations:
  - **Unknown stocks:** Here we are using 20% of the stocks left of the dataset, to see if just with the explanatory variables we can predict the future profits of those stocks.
  - **Known stocks:** For these tests we will use 80% of the stocks, that we used to train the models, but we will use the last 4 months of each stock to test if the models are able to predict when they already know past behavior of the stock.

2.6.2 Performance

The first thing we need to see is if the models are able to detect any relationships between the co-variables and the response variable, and how accurate they are:

- **Prediction capability:** we want to see if the models are able to predict the future profits, so we need to quantify the proportion of variance explained by the model. For this we will use:
  - **The coefficient of determination**, defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{2.14}$$

    where y_i are the observed values, $\hat{y}_i$ are the predicted values, and $\bar{y}$ is the mean of the observed values.
  - **The adjusted coefficient of determination**, to take into account the number of predictors in the model:

$$R^2_{adjusted} = 1 - \left(1 - R^2\right) \frac{n-1}{n-p-1} \tag{2.15}$$

    where n is the number of observations and p is the number of predictors.

    Since we are also using a NN Classifier, we cannot use the same statistics as for regression.

  - **F1 Score:** So for prediction capability in classification, we will use the *F1 Score (macro)* from *sklearn*, which is defined as the harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.16}$$

    where Precision $= \frac{TP}{TP+FP}$ and Recall $= \frac{TP}{TP+FN}$, with TP being true positives, FP false positives, and FN false negatives.

    This interprets as how many correct positive predictions the model makes relative to all positive predictions and actual positives, so a higher F1 score indicates better classification performance.

- **Accuracy:** We also need to check on how much error the model is making when adjusted. For this we used:
  - **The Mean Absolute Error**, defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.17}$$

  - **The Root Mean Square Error**, for penalizing the larger errors or possible excess of outliers:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.18}$$

    And again, for classification we decided to use:
  - **Accuracy Score:** From *sklearn* we used the accuracy score, which is defined as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{2.19}$$

  - **Log Loss**: For mimicking the penalization for large errors we are using the log loss from *sklearn*:

$$\text{Log Loss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j}\log(\hat{y}_{i,j}) \tag{2.20}$$

    where in this context, $y_{i,j}$ represents the true class label, and $\hat{y}_{i,j}$ represents the model's predicted probability for that class.

2.6.3 Consistency and Overfitting

But a good performance does not mean that the models are explaining the behavior of the variables, they could be overfitting the data. To detect this, we need to check if the models reflect stable, interpretable patterns in the relationships between scores and the *alpha*. For this, we will use the following criteria:

- **Sign of coefficients:** We will check if the sign of the scores coefficients remains stable across different time horizons. This would tell us that the score behaves as expected.
- **Shape of functions:** We will check if the shape of the spline functions of the GAMs remains stable across different time horizons.
- **Number of significant variables:** We will check if the number of significant variables increases with the performance of the models, using the *p-value* criterion.

And for the NN, since we cannot access the internal structure of the models, we are going to use the **SHapley Additive exPlanations (SHAP)** to check the importance of the variables in the predictions, and then use the same criteria for looking at the consistency of the variables in the models– Lundberg and Lee [16].

2.6.4 Robustness of Results

Even when the models perform well and have a good consistency, we need to check if the the residuals of the models resemble **white noise**, indicating the absence of non-modeled structure and valid statistical inference Enders [12]. We will do so by using both, visual and statistical tests.

- **Mean Value Test:** The mean of the residuals should be close to 0.
  - **Distribution Plots:** Visualize the distribution of residuals to assess if the mean is centered at zero.
  - **T-test:** Checks if the mean of the residuals is close to 0.
- **Heteroskedasticity Tests:** The variance of the residuals should be constant across all time horizons.
  - **Residuals vs Fitted Plots:** In a model with homoscedasticity (constant variance), the residuals should be randomly scattered around zero with a roughly constant spread.
  - **White Test:** Checks for general forms of heteroskedasticity.
  - **Breusch-Pagan Test:** Tests whether the variance of the errors depends on the values of the independent variables.
- **Normality Tests:** The distribution of residuals should follow a Normal distribution.
  - **Q-Q Plots:** Visual tool to compare the quantiles of the residuals to a normal distribution. If the data points in the Q-Q plot fall close to a straight diagonal line, it suggests that the data's distribution is similar to the theoretical distribution
  - **Shapiro-Wilk Test:** Statistical test for normality.
  - **Jarque-Bera Test:** Tests whether sample data have the skewness and kurtosis matching a normal distribution.
- **Autocorrelation Test:** The errors should not be correlated with each other.
  - **ACF and PACF Plots:** Visualize the autocorrelation and partial autocorrelation of residuals.
  - **Durbin-Watson Test:** Checks for the presence of autocorrelation in the residuals.

Chapter 3

Development

3.1 Architecture Enhancement

At the current moment of beginning this work, Tweenvest kept a large DB of most of the needed information for creating the dataset. This included all of the price histories, historical currency multipliers to dollar, dividends payed per stock... but when it came to the scores we had a traceability problem.

For optimizing the costs and structure of the DB, Tweenvest chose to only save the latest score for each company and overwriting it each day when calculating the newest one, which led to the **first main task**.

3.1.1 Updating Tweenvest Code

After locating where the score calculators were called, we had to understand the code's architecture and the relationships between the different database models to see exactly what needs to be changed. Tweenvest uses *Django* as the main backend's technology to handle a relational database with *PostgreSQL*, this DB was chosen due to the nature of the financial data where information is related to another.

For handling all of the different technologies and environments, Tweenvest uses **Docker [8]** to create a virtual machine that contains all of the needed dependencies to run the platform. This way, they have a consistent environment for developing, testing and deploying the code separately. So as we can see in Figure 3.1, each of the components is running in a different container.

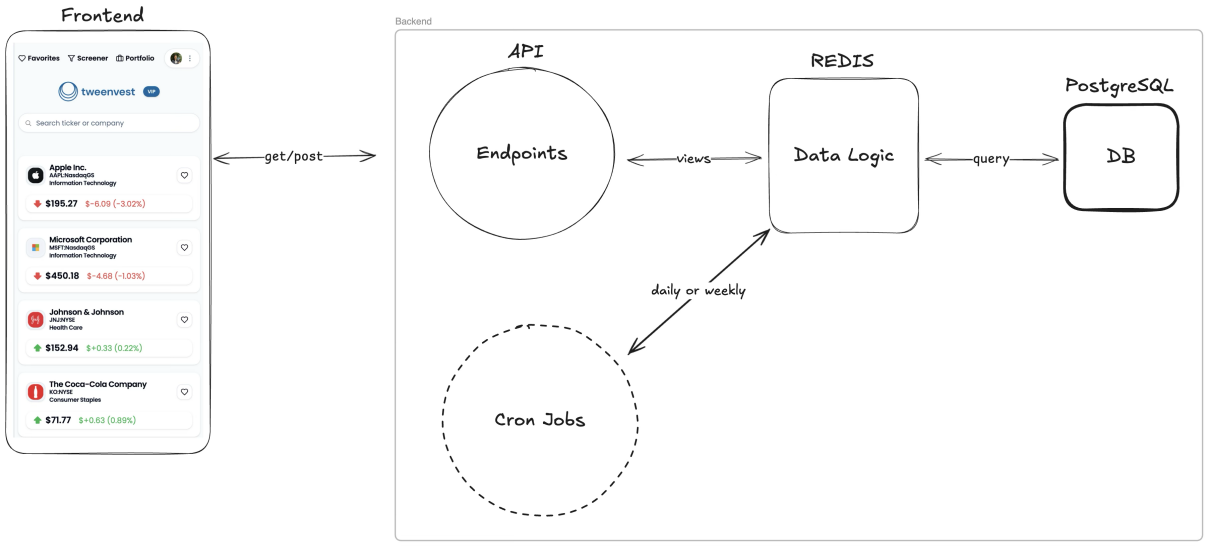


Figure 3.1: General Tweenvest's Architecture

Once we had a clear understanding of the architecture, we could look at the two different actions occurring at the same time:

1. Requests originating from user interactions.
2. Jobs that must be executed daily, weekly or at the moment, such as:
   - Updating all financial information of the stocks.
   - Performing necessary actions like sending emails to deliver authentication pins.

Since we were changing the data logic for calculating and storing historical data of the factor scores, we needed to assure that the systems capabilities do not exceed the platforms needs, and that the changes do not affect the normal calculation of the factor scores, so we had to look at the corresponding code of the factor scores.

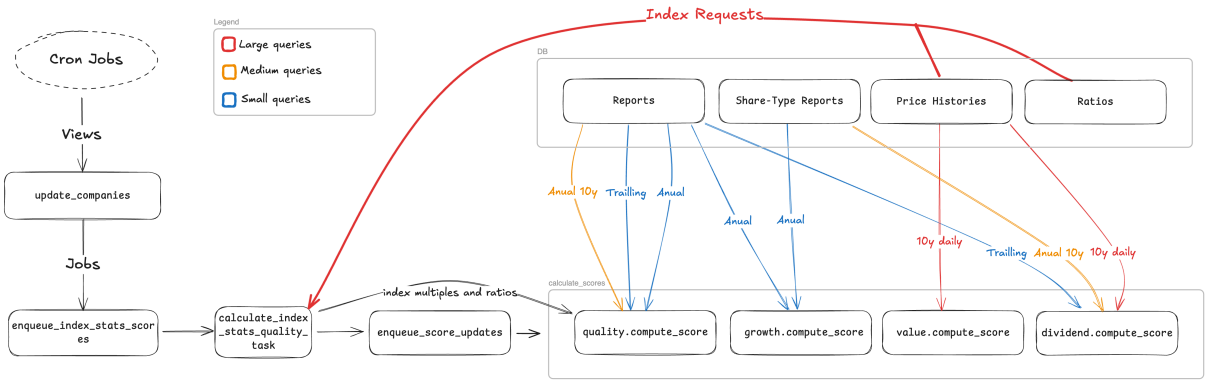


Figure 3.2: Scores Calculations Code Schema

The code integrates several methods and data-flows, so we created the schema shown in Figure 3.2 to simplify the logic behind several functions related to the calculations of each factor score. Seeing the chronology of the events, and the different queries to the DB, we quickly realized that we had to modify the queries to include an extra filter– *calculation_date*– to get the models related to the score at any time, this way we could set it as "today" whenever we

are calculating the present scores, not modifying the normal behavior of the system; while also having the ability to propagate the calculation to a specific past date.

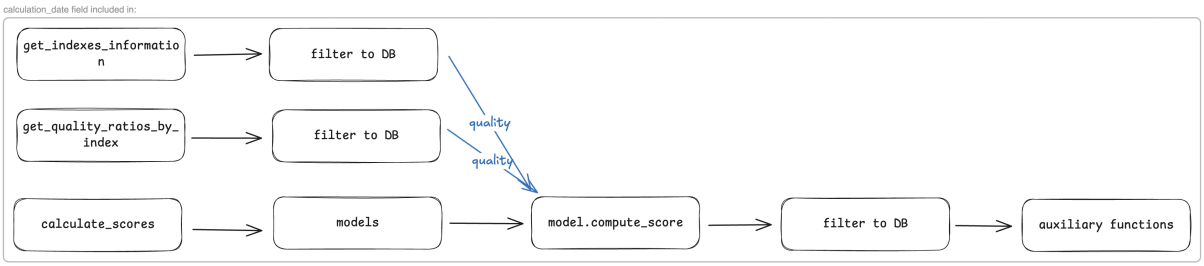


Figure 3.3: *calculation_date* Field Propagation Schema

For not adding redundant code, we had to inspect the propagation schema shown in Figure 3.3 to determine the root changes from the main functions: *calculate_scores*, *get_indexes_information* and *get_quality_ratios_by_index*. These are the ones used by the production *Cron* to calculate the factor scores daily, so for a normal function call we set the *calculation_date* to None and whenever we want to calculate the scores for a specific date we set it to the desired date.

This helped us create very readable logic to adapt the filters to both scenarios, for example in the value score shown in Listing 3.1:

```
if calculation_date is None:
    calculation_date = date.today()
    filters = {
        "stock": stock,
        "date__gte": calculation_date - relativedelta(years=10),
    }
else:
    filters = {
        "stock": stock,
        "date__gte": calculation_date - relativedelta(years=10),
        "date__lte": calculation_date,
    }
phs = (
    PriceHistory.objects.filter(**filters)
)
```

Listing 3.1: Value Score Filtering

So as we can observe, the main change for this part of the code was to add a date filter to get only data "lower than or equal" to the *calculation_date*. This may seem a bad approximation, but it has to be done this way due to the nature of the data: most of the financial data come from annual, semestral, and quarterly reports, except to the price history that is updated daily.

Along the way of analyzing the code's structure we made a small refactoring to simplify the legibility:

- **Renaming functions** whenever they are internal methods or generic auxiliary methods called by different functions.
- **Externalizing mocking functions** that help create fake models and data for *Unit Tests*.

- **Homogenize the code's structure**, field types, to help with readability and keeping a standard order along the backend and frontend.

Another very important part of these updating procedure was to **adapt the Unit Tests** to make sure that the original code still behaved as expected, **and creating new ones** to check that the changes calculated the factor scores properly.

3.1.2 Designing new Jobs

Once all tests have passed and the team had approved the changes, the next step was to create new Jobs to populate the current DB with the historic factor scores while maintaining the servers performance in normal levels. For this, we had to update a basic method, *bulk_create_or_update*, used many times through the whole codebase and add a new conditioning so it can handle conflicts depending if the models are empty or not. After reading the official Django Documentation [9], we saw that in the newest version available they had included the needed fields to its original method *bulk_create*, so **we had to update the *Django* version** in our *Docker* machine, making sure that it did not introduce any unwanted issues.

After this was implemented, we decided to create three different jobs for enqueueing the tasks and being able to separate the calculations. This way we could restart them in case something broke during the process (see Appendix 6.1). Also, we had to manage the server resources, so for assuring the platform's performance, we used the following hierarchy (see Listing 6.4):

1. Seven workers for primary daily jobs from different API data integration
2. One dedicated worker for guaranteeing email sending, and 6 prioritized shared workers
3. Four shared workers with lower priority, except for 1 which is set higher than email.

A crucial part of this code was replicability, equal distribution through different sectors, and mostly to assure that the companies existed during enough time for the factor scores to calculate the 10 years averages.

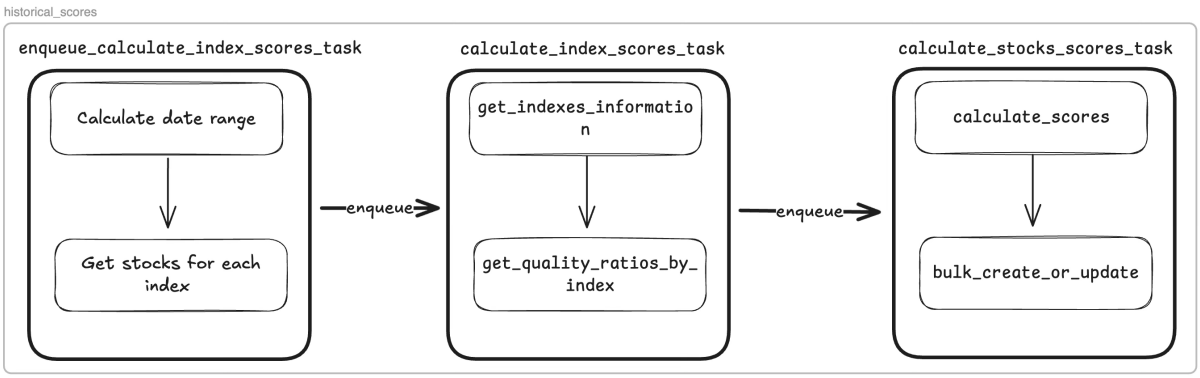


Figure 3.4: Historical Jobs First Iteration Schema

We created the first jobs with the schema shown in Figure 3.4, and during the first small test it worked in local, but when tested with the production DB we started to see important issues with how the functions where designed due to the large amount of data being processed:

- Passing incorrect arguments between jobs—such as entire lists of stock objects—led to the REDIS DB reaching its maximum storage capacity.
- Jobs ran out of time, due to the large amount of data being processed.

We then changed the day range logic to enqueue a *calculate_index_scores_task* for each day, so the job did not have to calculate all of the index data for the dates all at once, as shown in Figure 3.5. Also, we modified the function *args* to only uses models identifiers "id" to reduce the REDIS memory used. The complete implementation of this job can be found in Appendix 6.1.

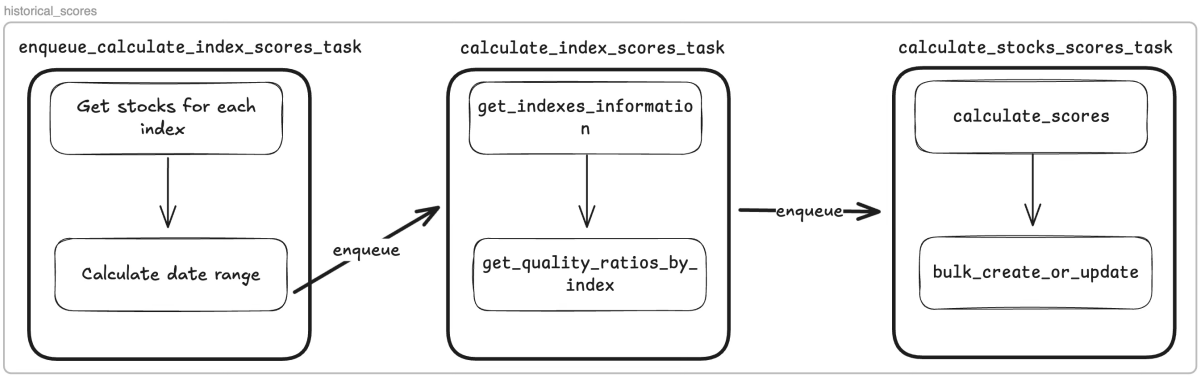


Figure 3.5: Historical Jobs Final Version Schema

3.1.3 Telemetry Tracing

Finally we had the functional jobs for calculating the scores, and ran the first calculation for a dataset of 1,500 stocks. After the first jobs finished we quickly realized that they were taking too much time to process, and saw a disproportionate amount of queries to the DB so we started a process to check on where was the bottleneck using telemetry tracing tools.

To do so, we had to inspect each method and function that was being used through the calculations of the factor scores. Luckily we had just recently implemented *DataDog* which is a software that provides monitoring and analytics for applications and infrastructure, offering real-time metrics, event monitoring, and end-to-end tracing.

To apply the tracer, we simply had to add a specific *Decorator* before each method and set up the environment for being able to trace not only production launched jobs, but also development tests.

Here's an example of how the tracer was implemented in the quality score calculation:

```
from opentelemetry import trace

tracer = trace.get_tracer(__name__)

@tracer.start_as_current_span("compute_quality_score")
def compute_quality_score(
    stock: "Stock",
    index_quality_ratios: Dict[str, IndexRatio],
    calculation_date: date | None,
) -> tuple[dict, dict]:
```

```
    ...
    return final_score
```

Listing 3.2: Telemetry Tracing Implementation

We then looked at the *DataDog* dashboard and clearly saw an excess of queries to the DB when calculating each score, and tried to reduce them by looking for a simple fault in the logic of the queries. Similar issues were resolved before by adding a *select_related* in the queries. To clarify what this does, here is the definition with a simple use-case:

"Returns a QuerySet that will follow foreign-key relationships, selecting additional related-object data when it executes its query. This is a performance booster which results in a single more complex query but means later use of foreign-key relationships will not require database queries." – Documentation [10].

In Django, *select_related()* and *prefetch_related()* are designed to stop the deluge of database queries that are caused by accessing related objects. *select_related()* "follows" foreign-key relationships, selecting additional related-object data when it executes its query. *prefetch_related()* does a separate lookup for each relationship and does the "joining" in Python. So one uses *select_related* when the object that you're going to be selecting is a single object, so *OneToOneField* or a *ForeignKey*.

After numerous attempts to improve performance, we were forced to reconsider our approach and began analyzing the problem from a different perspective. It quickly became evident that we were operating on a small virtual machine (Table 3.1) shared with the regular platform workload. Tweenvest's servers were not designed for high computational demand; rather, they were optimized for routine operations, where standard score calculations typically take between 3 to 4 hours. This setup works well under normal conditions, as calculations are performed after market hours and do not impact the user experience. However, it proved inadequate for our needs—specifically, processing data over five or more years within a reasonable timeframe, even when computing just one score per month.

VCPUS	RAM
2	4 GB

Table 3.1: Tweenvest Server Specifications

3.1.4 Dedicated Server Deployment

We made the decision to establish a dedicated server for this thesis, equipped with enhanced computational power to facilitate the calculations and data fitting.After evaluating several providers, we opted for Hetzner servers due to their exceptional quality-price ratio. The final setup consists of a dedicated server with the following specifications:

VCPUS	RAM	SSD Storage	Extra Volume
16	32 GB	320 GB	480 GB

Table 3.2: Hetzner Server Specifications

Deploying the project in a scalable and reproducible environment involved several structured

steps, encompassing server provisioning, secure remote access configuration using SSH, and initializing the containerized application deployment via Docker, with the application codebase managed through GitHub.

After all of the setup was done, we saw an amazing **x100 performance improvement**, going from 15-minute calculation times for index statistics to 8 seconds. Finally, we could launch the factor scores historical calculations, so we created two subsets of 1,500 random companies each with the factor scores calculated for the 1st day of the month for the following periods: 2015-2020 and 2020-2025, which took a total of 3 hours.

Now that we had filled the DB with the necessary factor scores for the study, we ran a big calculation to get all of the possible factor scores for the +100,000 companies, which took several days.

3.2 Dataset Creation

3.2.1 Aggregating the data

For being able to continue, we needed to create the final dataset with all the necessary data for later on doing the analysis and developing the predictive models, so we created a simple algorithm with this logic:

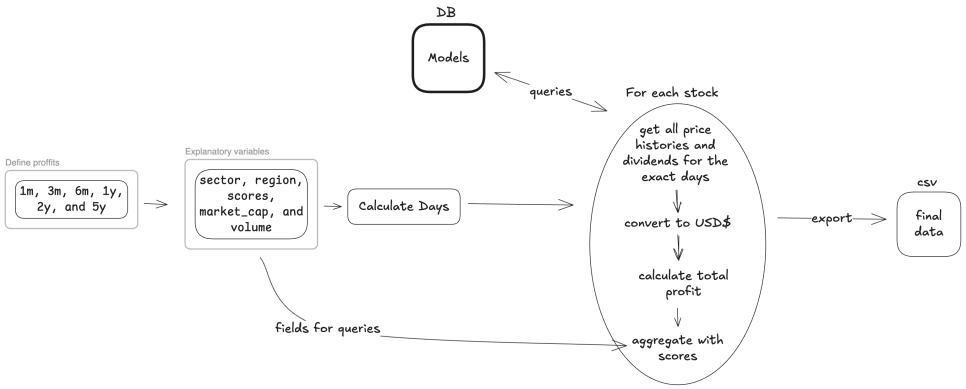


Figure 3.6: First Data Aggregation Schema

Once we had the final csv file we noticed that we had too many empty values for profits, the problem was that some of the days calculated were empty for a specific company, so there was no price history for them. Since companies belong to multiple countries, different holidays could be causing another major loss in the information. To fix this in a general way **we implemented an internal method** for estimating price histories when they did not exist for the wanted date. The complete implementation of this data export and price estimation process can be found in Appendix 6.3.

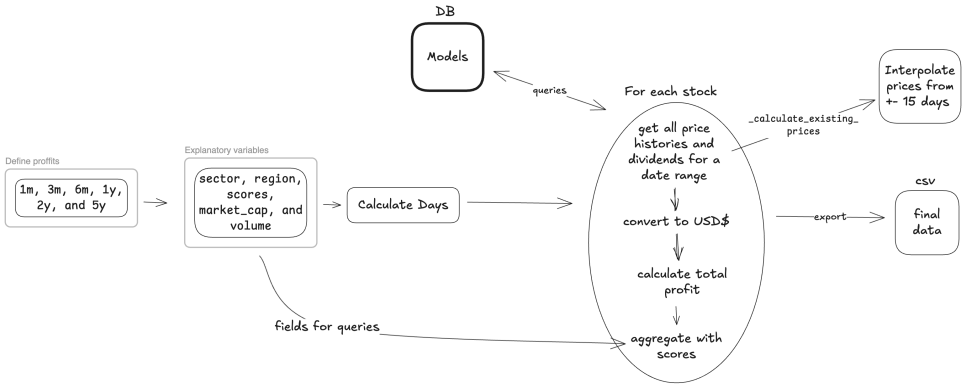


Figure 3.7: Final Data Aggregation Schema

By looking at the Figure 3.7 we can see that we are interpolating those price histories. Since we are only looking for long term profitabilities, we can **approximate the price** on a day x by interpolating the price on $x - y$ and $x + z$, weighted by how close they are to the original day x, with a maximum distance from x of 15 days. This significantly reduced the amount of missing values.

3.2.2 Datasets Construction

For this study, we have attacked the objectives from multiple perspectives. First we are making a descriptive analysis to understand the data, and then we are proposing different models to see if there are any relationships between the scores factors and long term profitabilities. Thats why when creating the datasets, we made two different approaches to have more information for the analysis:

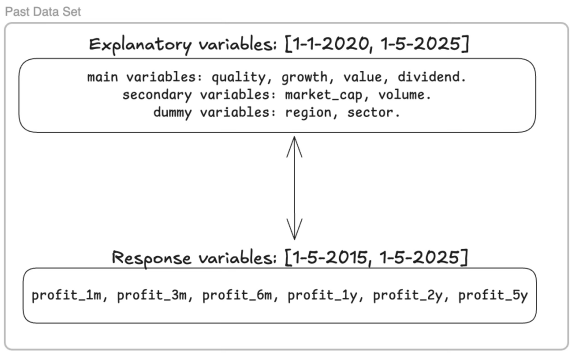


Figure 3.8: Past Dataset

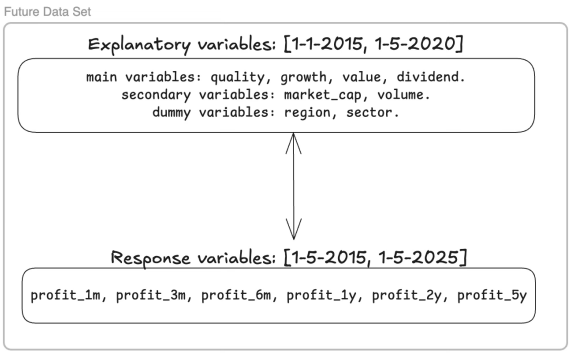


Figure 3.9: Future Dataset

For the past dataset, we calculated the scores for the range 2020-2025 and then related them to the profits obtained until the specified date. For example, the date 1-1-2021 would be linked to the 2 years' profitability obtained until that date:

$$Profit_{\text{1-1-2021}}(\%) = \frac{P_{\text{1-1-2021}} - P_{\text{1-1-2019}} + D_{\text{acc}}}{P_{\text{1-1-2019}}} \tag{3.1}$$

Which interprets to **how earlier profits have influenced on the explanatory variables**. This will give us valuable information in the descriptive analysis.

For the model fitting, **we want to know if the current scores are good predictors of future**

profits. So we calculated the scores for the range 2015-2020 and related them to the profits obtained from that date to the future. For example, the date 1-1-2021 would be linked to the 2 years' profitability that will be obtained in the future:

$$Profit_{\text{1-1-2021}}(\%) = \frac{P_{\text{1-1-2023}} - P_{\text{1-1-2021}} + D_{\text{acc}}}{P_{\text{1-1-2021}}} \tag{3.2}$$

As this approach is what we actually want to use for the predictive models, we will be using the future dataset structure for most of the analysis.

3.2.3 Preprocessing

Even with all of the protocols created to have the most reliable and filled datasets, we still had to follow the methodologies explained in the previous chapter to analyze the data and see if there were any problems not solved.

Cleaning the Data

To begin, when looking at the data structure we saw that **about 50% of growth scores were empty**. This turned on many alerts from problems with the scores calculation algorithm, because compared to the rest of the variables there was a significant difference, since the **normal absence was only around 8%**.

quality	growth	value	profit_1m	profit_3m	profit_6m	profit_1y	profit_2y	profit_5y
13.4	54.0	7.4	2.6	3.1	3.2	3.2	3.3	6.0

Table 3.3: Missing values (%) per column for Small Data Future.

But after digging into the data we saw that it was due to the fact that many companies stopped sending reports or selling but kept existing, so we decided to deleted them from our dataset because they were not behaving as a "normal company", and this could create a bias in the models. Additionally, as noted earlier, we replaced all NaN values in the dividend scores with 0, since Tweenvest's algorithm omits a score when a company does not pay dividends. That's why we do not see any missing values in the dividend scores.

3.2.4 Outlier Detection

In this section we will present the performance of the different outlier detection methods. But the main analysis of the final datasets will be done in the results chapter.

Inter Quartile Range

This method is useful for big datasets that need to be filtered quickly and where the meaning of the outliers is not multivariate. But as it will be explained in the results chapter, we saw correlations between the different profits, so we needed to use multivariate outlier detection methods, otherwise we could be creating a bias in the data for the model fitting.

Isolation Forest

We started of using this technique due to its speed and accuracy for non-linear multivariate clusters.

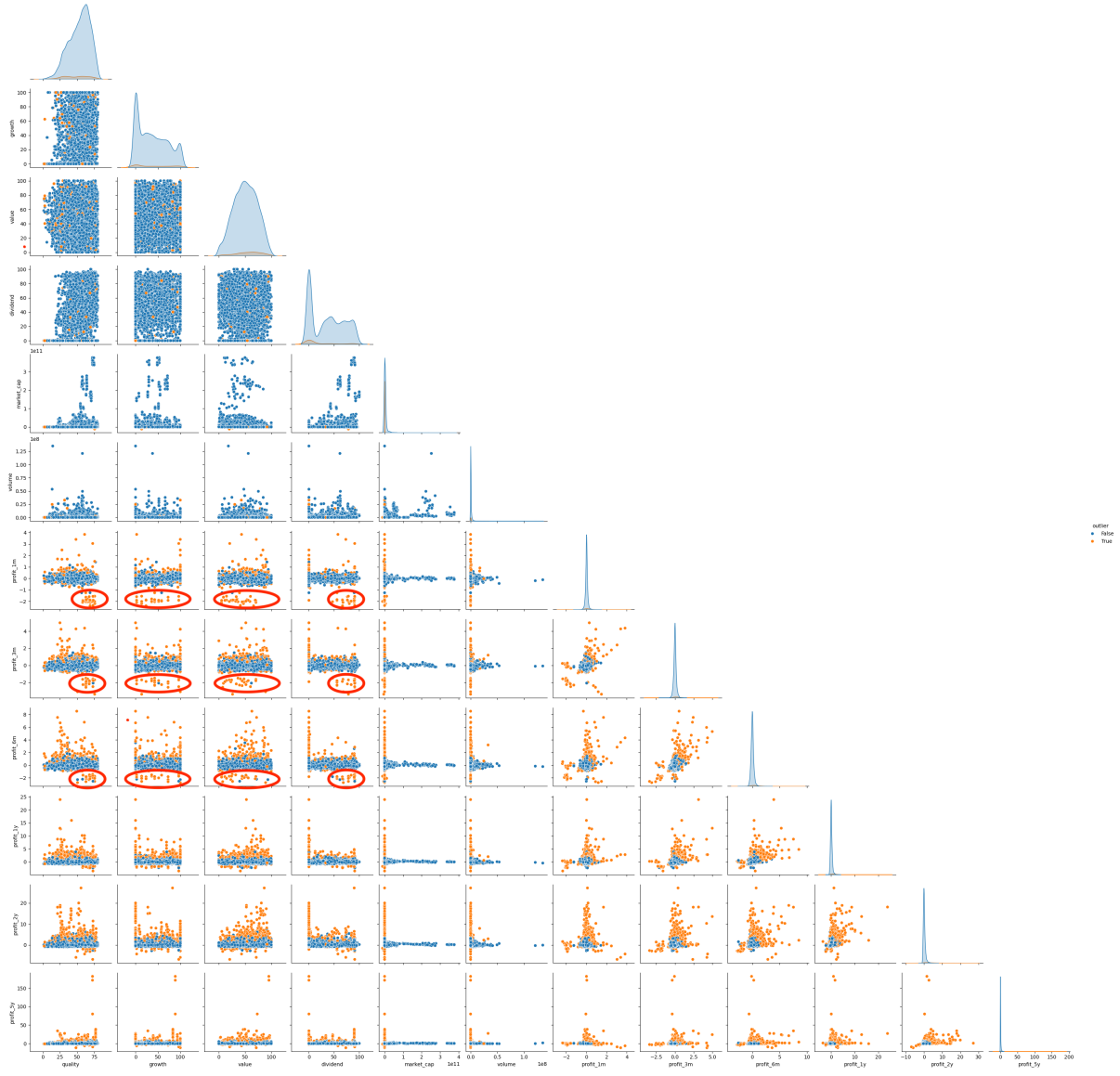


Figure 3.10: Small Data Future - IF (5% tolerance)

This Pair Plot shows how the IF method is detecting outliers in the profits data. In this case it is detecting as outliers much of the data that seems to form a structure in the profits. Also, it is deleting some interesting clusters when comparing the profits with the scores, and here we might lose some important information– this is highlighted with the red circles.

Single Vector Machine

When using the SVM we saw a similar detection compared to the IF method, which is very interesting since their approach is completely different.

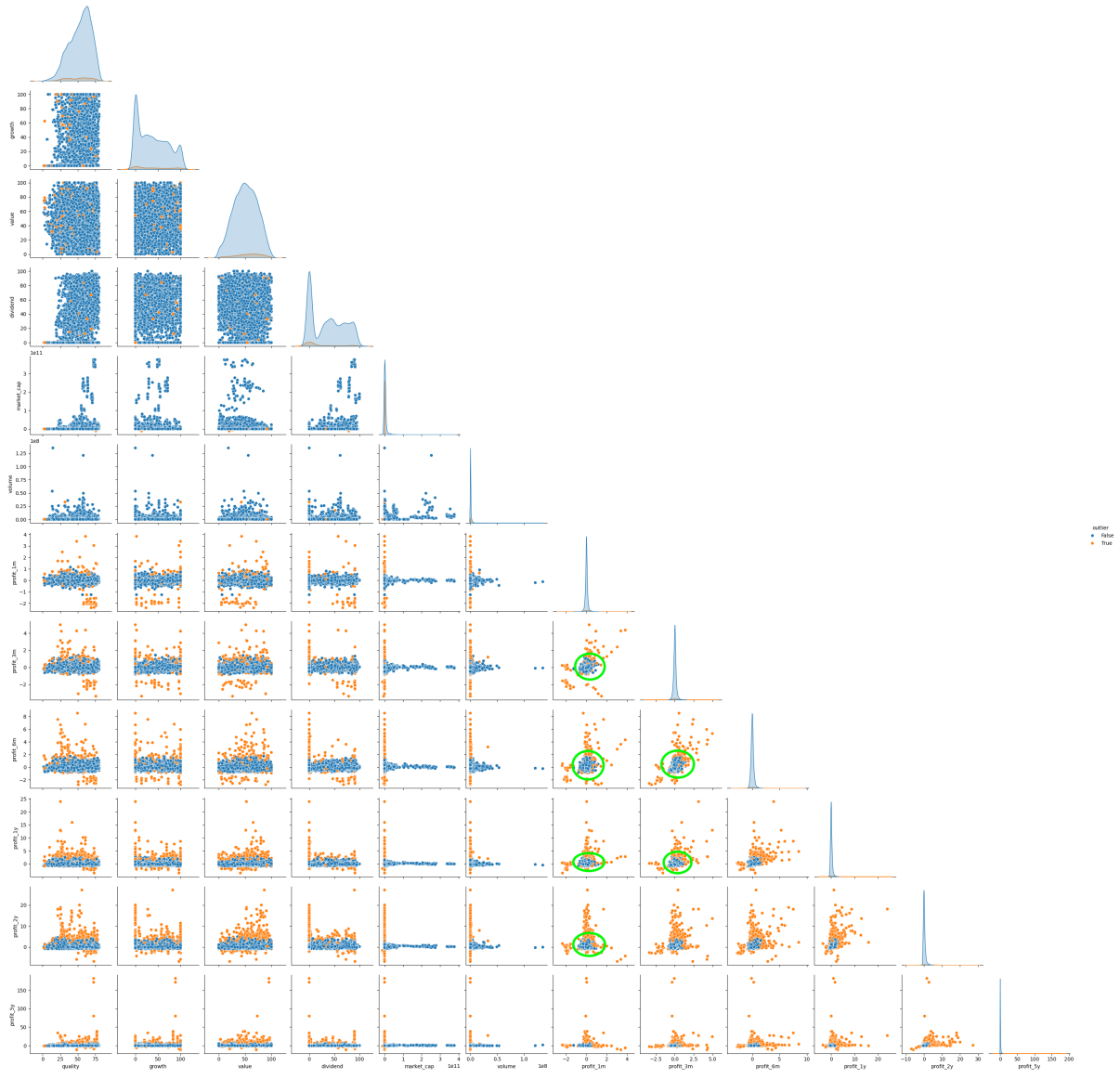


Figure 3.11: Small Data Future - SVM (5% tolerance)

But in this case we also see a slight improvement with recuperating the structures in some of the profits– this is circled with the green color.

Local Outlier Factor

Surprisingly, with the LOF method we see a good improvement for both the profits and the scores clusters– which is highlighted with the green circles.

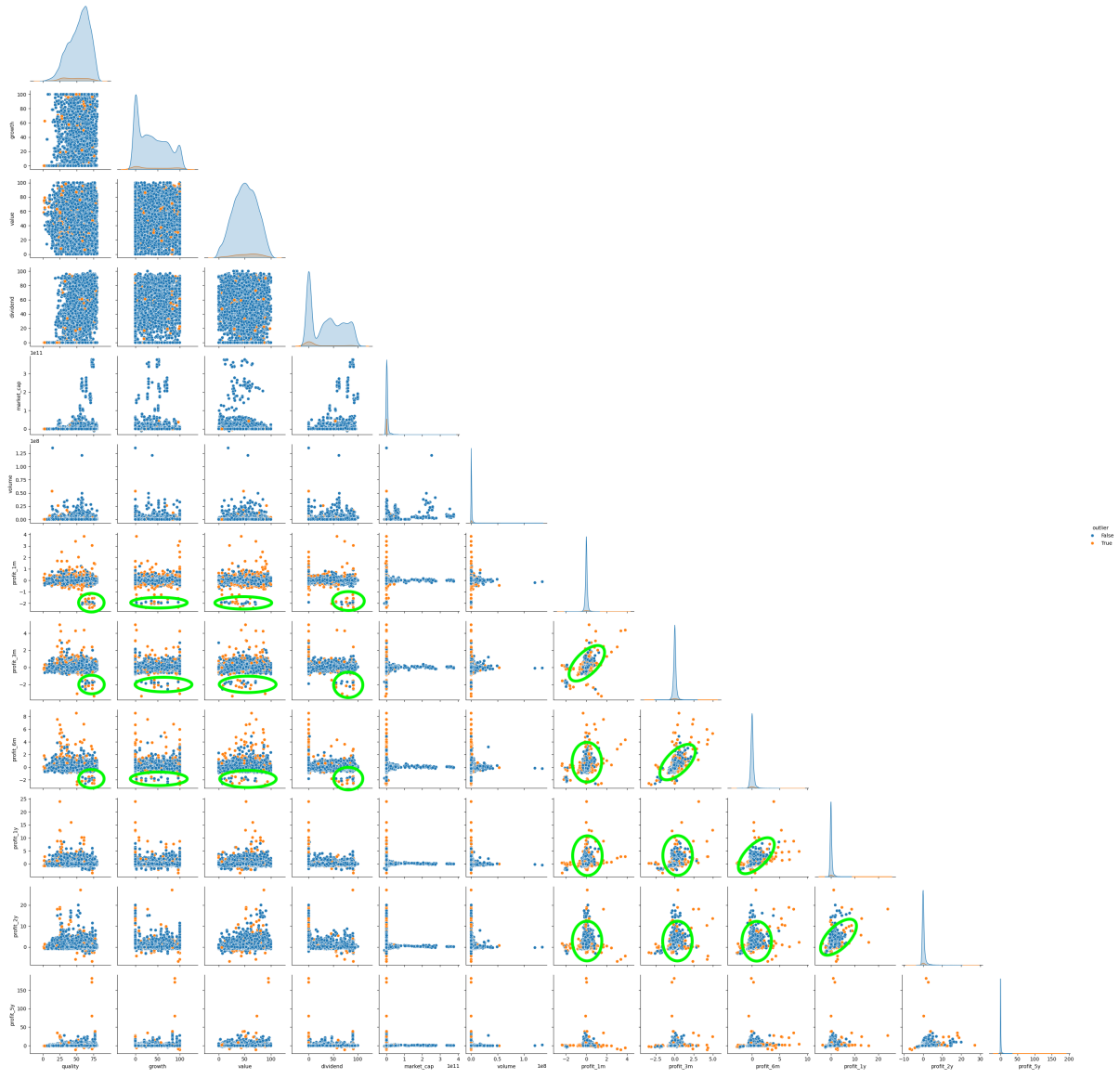


Figure 3.12: Small Data Future - LOF (5% tolerance)

We knew we were going in the right direction, but with this method we were missing some outliers that were detected before in the score-profits clusters.

Multi-Criteria Outlier Detection

That's when we realized that if each method was good at detecting some outliers but bad at others, so it made a lot of sense to use them together. Following this approach and after multiple tests, we found that the bests results came when increasing the overall tolerance and then selecting the intersection of outliers between them, so with a 20% tolerance we only excluded around 8% of the data.

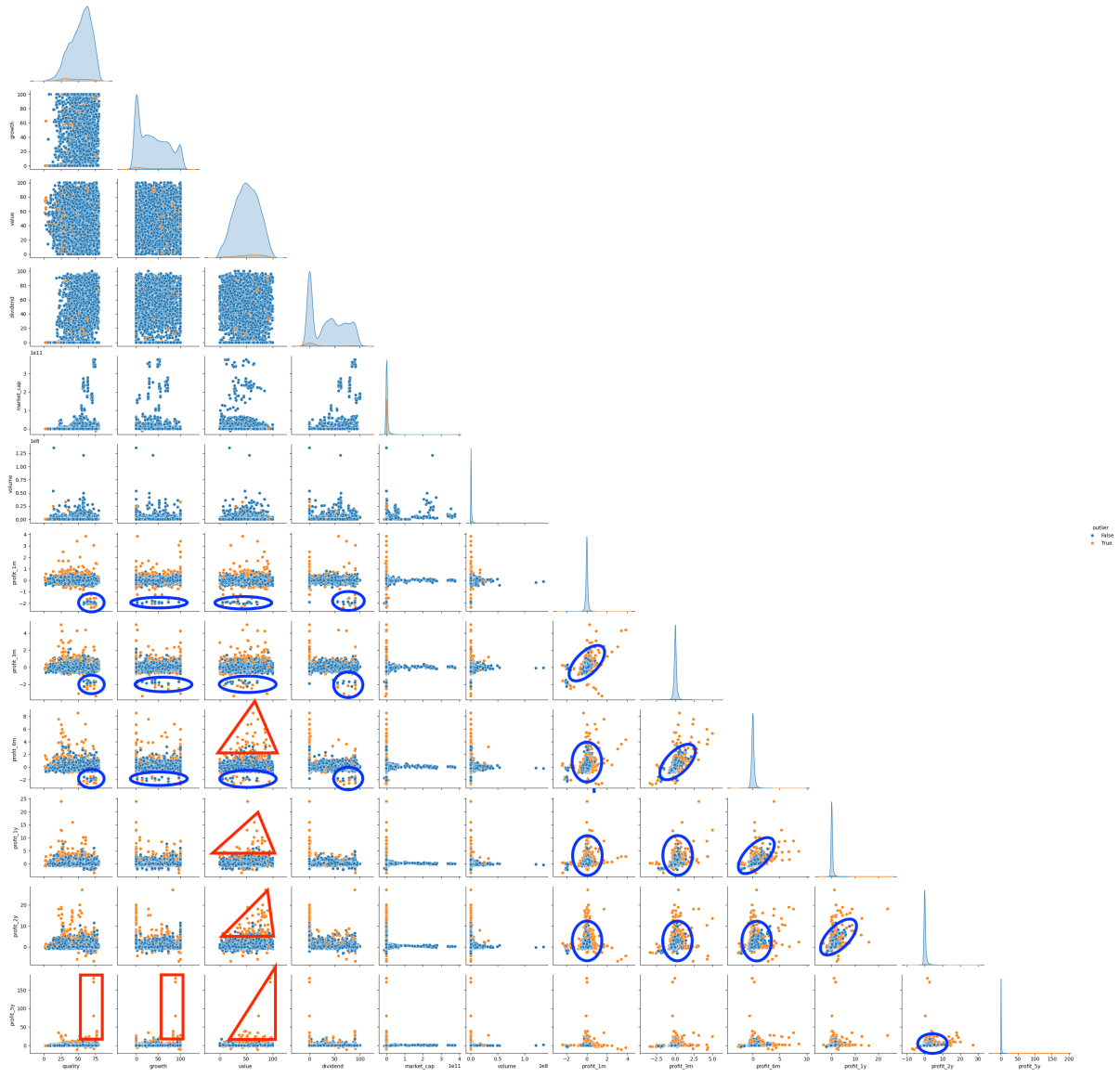
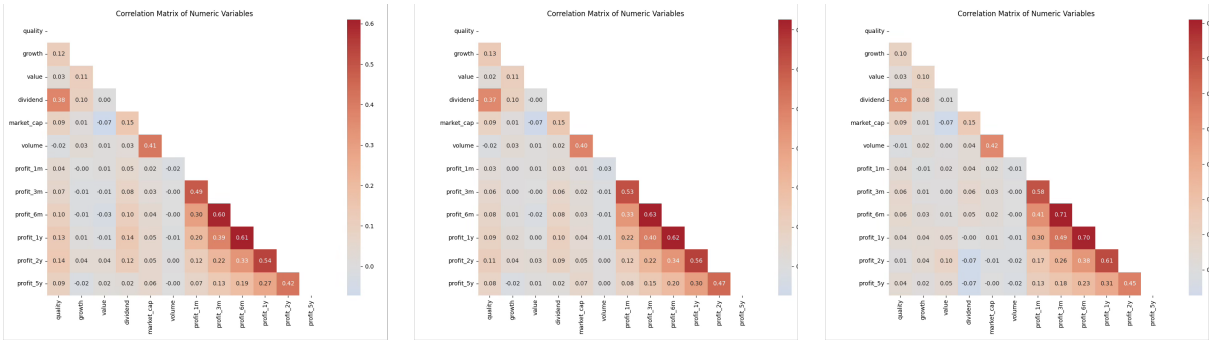


Figure 3.13: Small Data Future - MCOD (8% tolerance)

As shown in the Pair Plot, this method is able to maintain most of the structures of the profits and also keep the small clusters for the scores, which is highlighted with the blue circles. But of course, it is not perfect and could be missing important information for extreme cases –highlighted with the red shapes.

Islands of Outliers

After the previous results we started working with the models, but we quickly noticed that those clusters that were not being deleted, on purpose, were affecting the results. So we decided to take a big look at the data again to see what was happening:



Isolation forest (15%) SVM (15%) LOF (15%)

Figure 3.14: Correlations between variables for different outlier detection methods

As we can see, the IF and SVM methods are showing some correlations between the profits and the scores, but the LOF is not showing any correlations, even negative ones. And that is what made us think about the clusters of anomalies that were not being deleted by the LOF method, so we built the ICD, mentioned in the previous section.

3.2.5 Summary

To summarize how we have treated the data, we ended up with different datasets groups, each with 17 columns for the explained variables, the date and also the company's name and stock ticker (see Appendix Figure 6.1).

Simple Datasets

For the first part of the analysis that consisted of the descriptive analysis, we started by using the Big Datasets, that contain all of the Tweenvest's data for the past 10 years. This way we could see the real distributions and correlations, as we will explain in the next chapter. Also we created the 2 subsets of Small Data with only 1,500 stocks for quicker calculations, making sure that the sectors were balanced, and using a 4% tolerance for the MCOD method.

Dataset	Raw	Cleaned	No Outliers
Big Data Past†	16,119,852	9,991,895	9,492,300
Big Data Future†	31,933,137	14,518,624	13,792,692
Small Data Past	83,243	48,338	46,415
Small Data Future	65,166	26,214	25,220

Table 3.4: Simple Datasets Total Rows

† In the *Big Data Past* and *Big Data Future* datasets, we removed the outliers using the IF method for saving computational time.

Geographical Datasets

For the second part of the analysis, that consists of exploring the regression models, while advancing with it we had to inspect how the geographical regions were affecting the results, so we created one dataset per region applying the same sector balancing. Here we kept using the 4% tolerance with the MCOD.

Dataset	Raw	Cleaned	No Outliers
Small Data Future USA	60,503	21,240	20,463
Small Data Future EU	62,725	25,217	24,245
Small Data Future AS	65,772	28,085	27,156
Small Data Future LAT	48,412	20,610	19,851
Small Data Future AF	58,341	21,182	20,335

Table 3.5: Geographical Datasets Total Rows

Restrictive Datasets

For the third part of the analysis, once we had seen the nature of the data and the simple models results, we realized how much the outliers and countries were affecting the results, thats why we chose the GAM non-linear models. So for improving the consistency we used the ICD and the IF method with higher tolerances.

Dataset	Raw	Cleaned	ICD 8%	IF 15%
Small Data Future	65,166	26,214	22,214	22,282

Table 3.6: Restrictive Datasets Total Rows

Validation Dataset

In the final stage of our analysis, our goal was to obtain the most realistic results possible, which required retaining as much data as we could. To achieve this, we revisited the Big Data Future dataset and applied the IF method with a very high tolerance of 35%. Additionally, we balanced the data by regions and sectors. These steps, however, led to a substantial reduction in the number of rows.

Dataset	Raw	IF 35%	Balanced
Medium Data Future	14,518,624	9,437,105	110,203

Table 3.7: Validation Datasets Summary

These restrictions resulted on a surprisingly higher simple correlations of the profits with the scores, showing that we were going in the right path. But this will be explained in the following section.

Conclusion

Finally, we have to clarify that outliers are anomalies in the *normal* behavior of the data, so even thought we are excluding them for obtaining a general set of models, we could be missing valuable information such as the possible structures shown with the red structures in the

Figure 3.13. That is why we propose for future work a separate analysis for the outliers, to see if really good profits (or bad ones) have any characteristics, such as average high or low scores.

Chapter 4

Results

4.1 Descriptive Analysis

Before jumping into the model fitting, it is necessary to understand the data that we are working with. For doing so, we will be looking at the variables distributions, correlations and possible structures.

4.1.1 Variables Distributions

Scores

When looking at the scores in the Figure 4.1, we see that there are several scores that saturate. So for a better visualization we created a temporal dataset where the dividend and growth scores saturating at 0 were deleted, this is shown in the Figure 4.2.

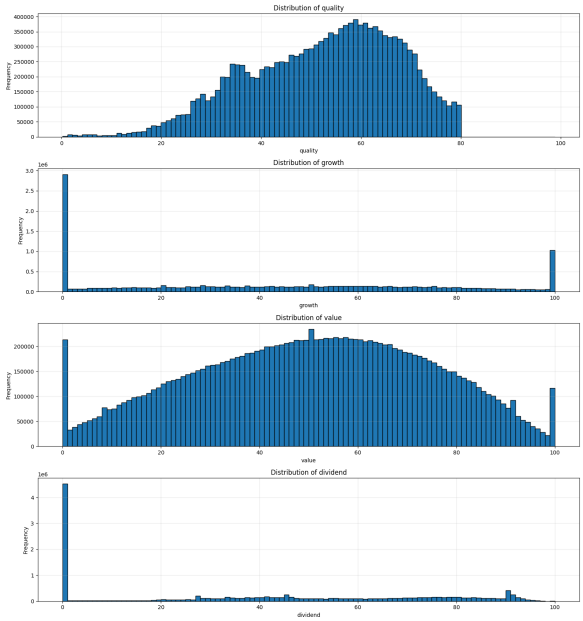


Figure 4.1: Big Data Future - Scores

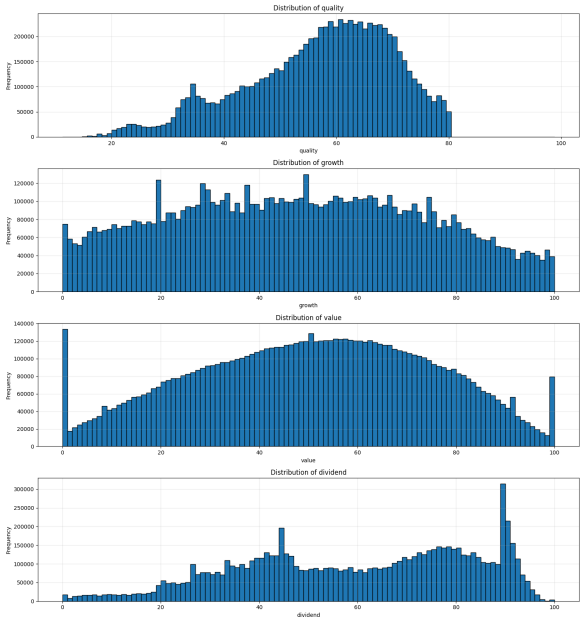


Figure 4.2: Big Data Future - Scores Deep

Starting with the **quality score**, we can see that its distribution shows a **clear negative skewness**, meaning that there are more companies closer to having a good quality score. This could be

due to the fact that we had deleted companies that did not behave as a “normal company”, so we will later analyze the distributions of the profits to see if this was the case. Also, there is a **secondary peak around the 30-40 scores**, which could be related to the time-frame of the data.

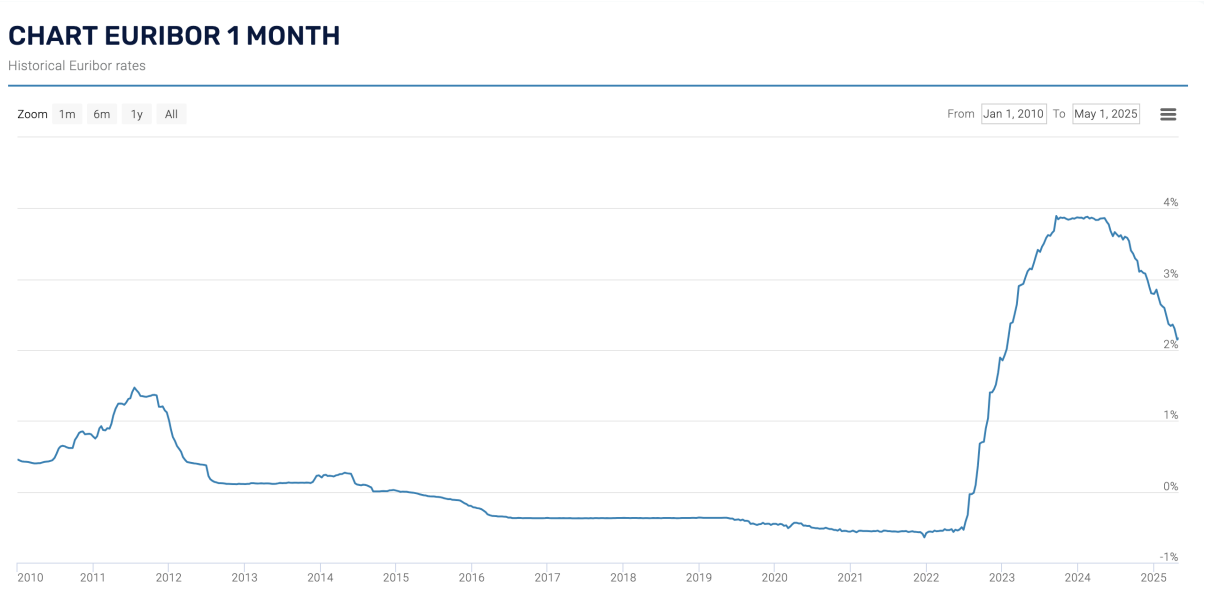


Figure 4.3: EMMI - Euribor Interest Rates

Around 2015–2020 interest rates were at a historical low, so there could be more companies with more debt than normal. Some central banks, particularly in Europe, experimented with negative interest rate policies on deposit facilities during this period (Figure 4.3). This possibility should be studied in more detail, but knowing that this does not happen for the dataset between 2020–2025 where interest rates are higher, we have a clue that this could be whats causing it (see Appendix Figure 6.3).

Finishing with quality, there is a “defect” in the dataset. We do not have 10 year averages for the companies that are in the 2015–2020 period –Tweenvest only has data after 2012– so the algorithm gives 0 to some of its criteria making the maximum quality be 80. But since this is happening for all of the companies, **it is not a problem for the model fitting**, and we could easily rescale to 100.

Continuing with **growth score**, we notice that there are many companies saturating the score at 0 and 100, and the same happens with the Dividend. But looking at the deeper plots, we can see that it is closer to a uniform distribution with a descend slope after 65, which means that most companies have a similar growth rate, but fewer have higher ones.

When looking at the **value score**, we can see that the distribution almost has an average of 50, and there is a continuum of values trough the whole range decreasing as the score diverges from that average, so it seems that it is a good scale for distributing the companies.

Lastly for the **dividend score**, as mentioned before, there are many companies that do not pay dividends. But aside from that, we have an irregular distribution for the score, with a peak around 85–92 meaning that there are several companies that care about their dividend policy.

Dummies

When it comes to the dummies we see a uniform distribution with the sectors because this condition was forced in the data creation process. But with the regions we have a different result, seeing a clear **predomination of the Asiatic region**, which happens in all of the datasets. Which could create issues for the model fitting, since it is known that in countries like China have more volatility and lower returns Chen et al. [4].

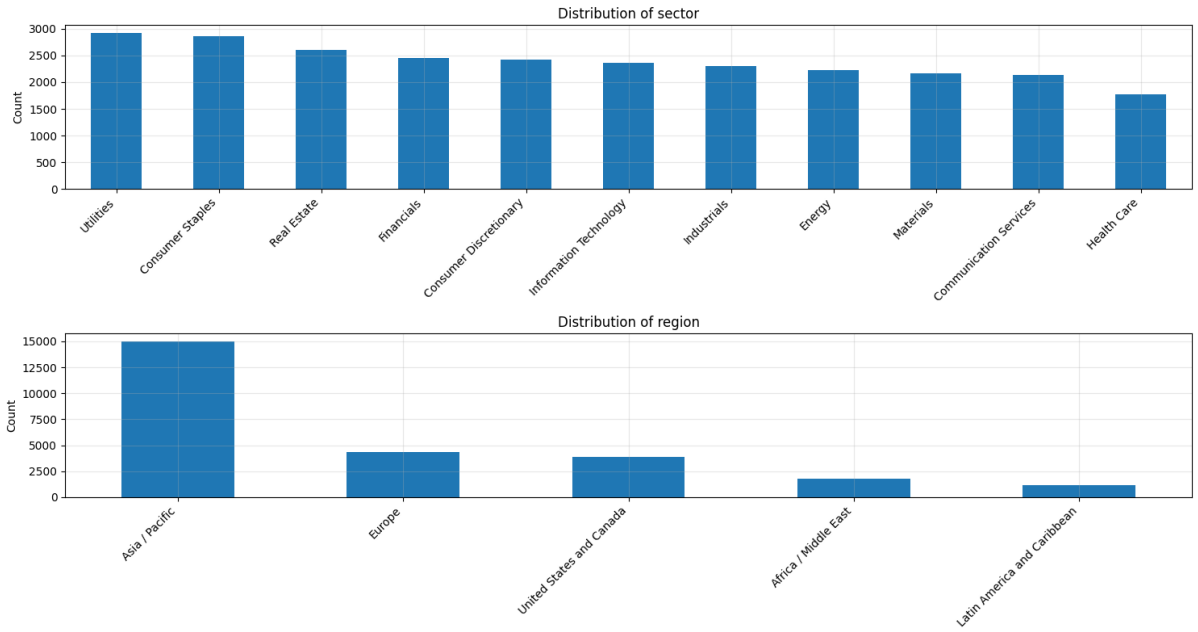


Figure 4.4: Small Data Future - Dummies

Also, their auditors often lack true independence, especially when auditing state-owned enterprises (SOEs) or firms with strong political ties. The Chinese government tightly controls domestic accounting firms and imposes restrictions on foreign auditors. Even international firms like *PwC-China* operate under local joint ventures, often with limited autonomy, as reviewed in Liu and Tian [15]. So we will have to check if the dummies variables are enough to take this effect into account, or if we need a different approach for modeling the geographical effects.

It is worth noting that the distribution of companies across sectors varies significantly over time, as shown in Appendix Figures 6.6 and 6.7. This temporal variation in sector representation presents an interesting opportunity for future research.

Profits

When analyzing the profit distributions across different time horizons, we observe that the distributions contain a numerous amount of outliers that do not let us see the distributions clearly (Appendix Figures 6.8 and 6.9). So just for this descriptive analysis purposes, **we used the IQR method** mentioned in previous chapters to remove the outliers.

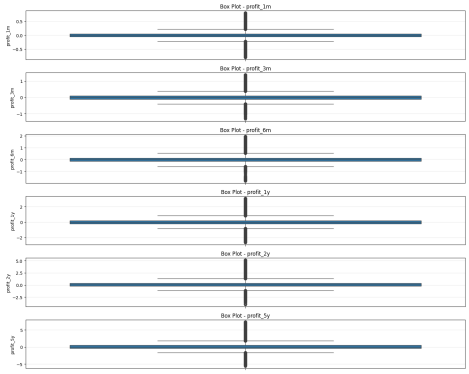


Figure 4.5: Big Data Future IQR - Profits Boxplot

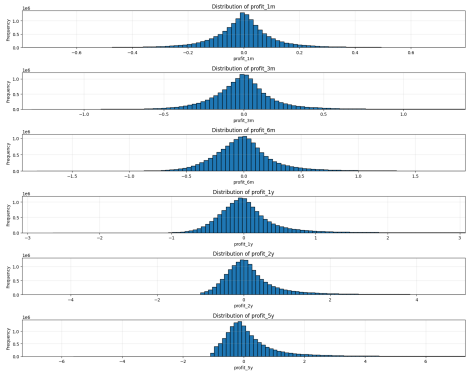


Figure 4.6: Big Data Future IQR - Profits

Interestingly, we notice that the **short-term profit** distributions (1-month and 3-month) are more symmetric and **closer to normal distributions**, which is consistent with the efficient market hypothesis suggesting that short-term price movements are largely random.

However, as we extend to **longer time horizons**, the **distributions become more positively skewed**, which aligns with the general expectation that most companies do not have good returns, except of the market winners that tend to generate extraordinary positive returns due to the mentioned **economic moats**. This is particularly evident in the longer-term profit distributions (2-year and 5-year), where we see a **longer tail extending toward higher positive returns**.

There is also a tendency to having **less negative returns** compared to the positive ones the longer the time-frame. Logically, companies with bad returns do not last long.

4.1.2 Correlations

For analizing the correlations between the variables, we will start firstly with just the cleaned big datasets, and then we will use also the datasets without the outliers, since these are the ones that will be used for the model fitting in most of the analysis.

Here is the only section that we will be using the past Datasets, to see if past profits are being reflected in current scores.

Clean Datasets

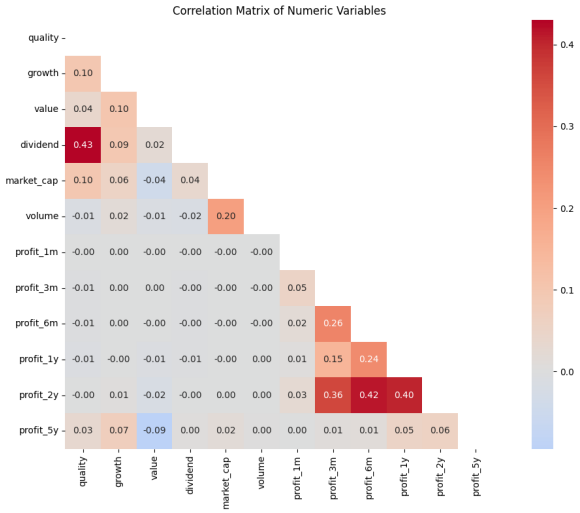


Figure 4.7: Big Data Past - Correlations

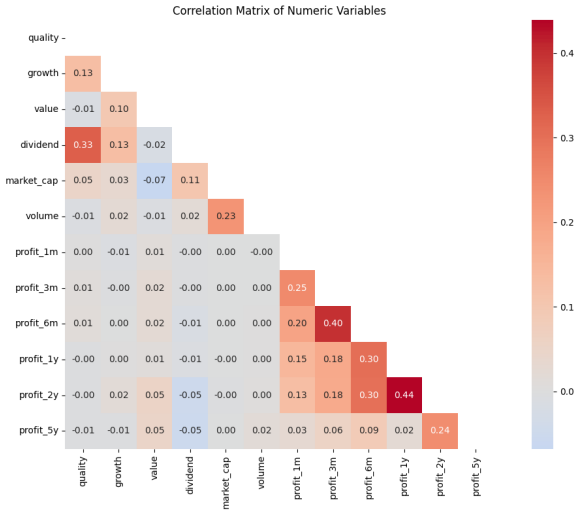


Figure 4.8: Big Data Future - Correlations

We can appretiate three main expected correlations:

- **Companies with good quality** –that is so, they have consisten growth, good liquidity and healthy debt– **are more likely to give dividends**.
- There is a strong relationship betwen the market cap of a company and the volume of the shares traded.
- **Companies have inertia**, or how it is called *Momentum*, meaning that the profit of a company is positively correlated with the profit of the previous period.

Outliers Datasets

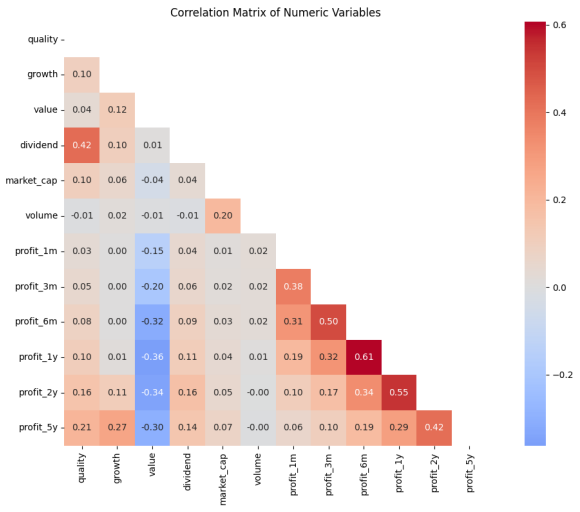


Figure 4.9: Big Data Past IF - Correlations

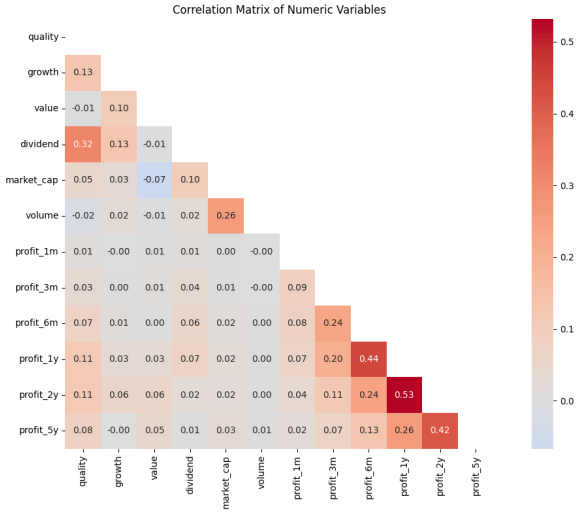


Figure 4.10: Big Data Future IF - Correlations

As expected, the correlations already seen stay similar with a slight increase in the *Momentum*, showing more importance of it than other variables. But actually, there are **new correlations**

seen for both datasets:

- **Accumulated profits affect the scores**. In the past dataset we can see that long term past profits translate into higher quality and dividend scores– as expected by their correlation. But also, they make the value score to decrease. This could be explained by two effects:
  - **Bull markets**. As explained in the second chapter, a company can experience good returns in the stock market that are not followed by good financial growth. This makes their value multiples to raise, making their value score to decrease.
  - **The company is not able to maintain its profitability**. If a company has had good past profits, but is not able to maintain them, the market will eventually realize that and the company will have a lower value score.
- **Long term stock returns affect the company's growth**. This is a very interesting result, since it explains the reason why companies try to go to the market to raise capital and help the company grow.
- **Quality is correlated to future profits**. This could be explained by the fact that a company with good quality is more likely to have good future profits, but also by the fact that the market is efficient and the price of a company will reflect all of the information available.
- There is a **mild correlation between most scores**, which could be explain by the fact that a company that does well in one aspect, is more likely to do well in the others. This correlation will not affect the models because they are separated by score.

Althought we have to make clear that these correlations do not stand along all of the datasets separated by region (see Appendix Figures from 6.10 to 6.19), so we are not sure yet if these relationships will be properly explained with linear models.

4.1.3 Data clusters

Finally, we are going to try to compare these results with the Pair Plot of the datasets, to look for clusters of data when comparing many variables at once.

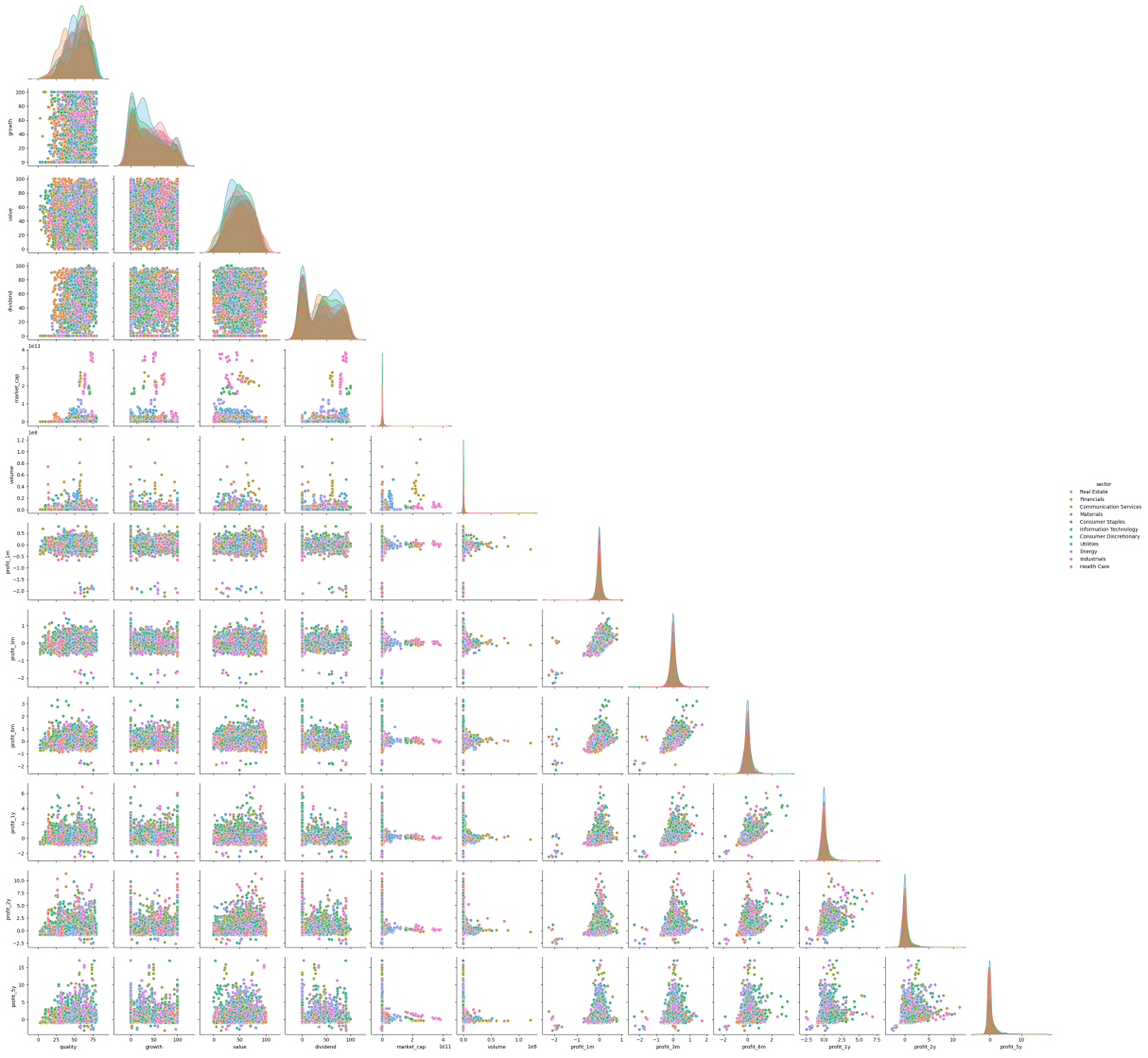


Figure 4.11: Pairplot of Small Data Future - MCOD

Here it is harder to eye-see the data structure that could form clusters since there are many datapoints, but there are some exceptions:

- **Profits**. We can see linear correlations between some of the different profits, but for longer horizons the relationships are no longer linear. Which is expected since the further the horizon, more variables take part in the calculation of the profit.
- **Market Cap**. Companies with bigger market caps tend to have higher quality and dividend scores. Bigger companies should have a more stable financial situaition, otherwise they would not be as big in the long term. Also there are some sectors that tend to have higher market caps than others, like in Health Care or Consumer Staples.
- **Sector**. It seems that there could be a pattern seen in the sectors distribution for each scores, but it is not clear.

Althought, these results are not as clear and they also variate depending on the dataset. Showing early results of non-linear relationships with the dummy variables.

4.2 Predictive Models

Now that we understand the data, and the different datasets created. We are going to explain how was the process of refining the methodologies and improving the results.

4.2.1 Linear Regression

First of all, we started with a simple linear regression model with very few variables and not excluding the outliers. Which of course resulted in a very bad model that could not predict anything. But after the first outliers removal used for the first series of datasets. We started to see some results:

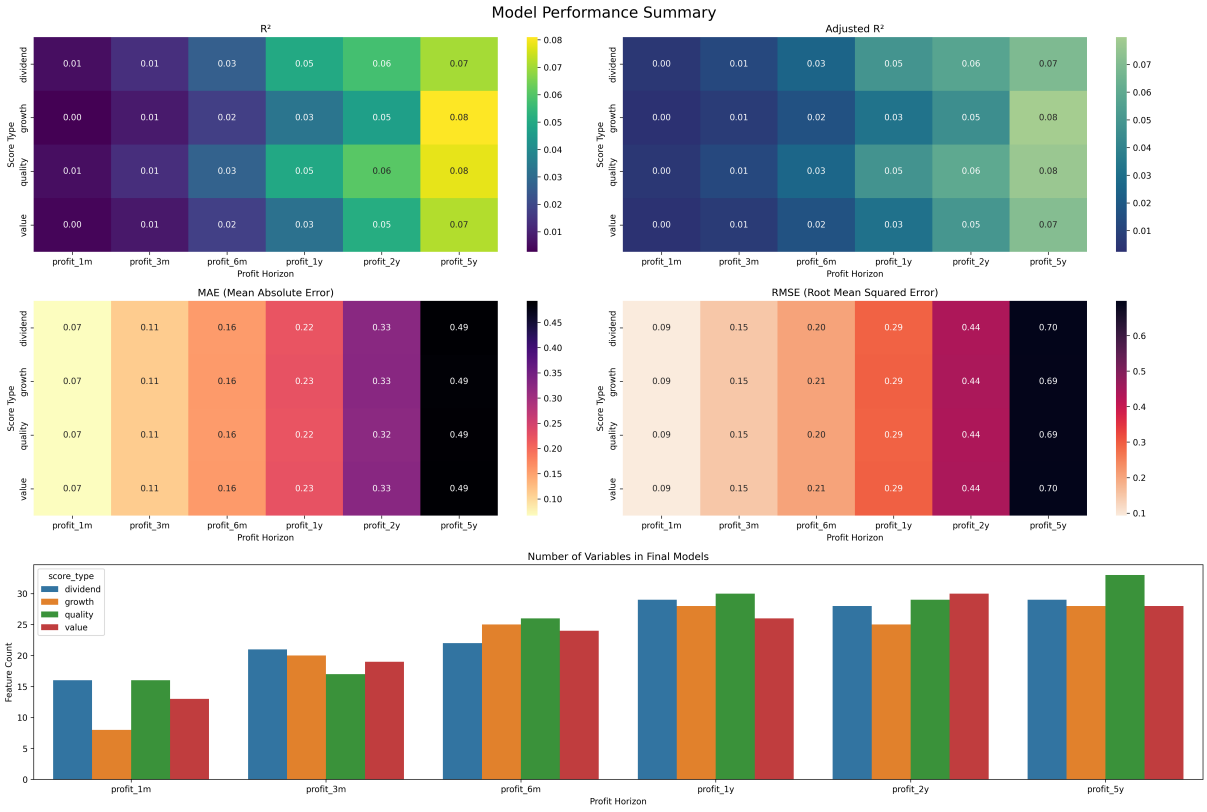


Figure 4.12: Small Data Future - MCOD 5%

Here we can see a pattern that is going to be constant in all of the models, the longer we go in the future the more the model is able to predict the profits. But at the same time, the higher the error is going to be. This is very normal, since on the long term variances are higher but also if a company is stable it will have a more predictable behavior. Also we can see that the RMSE is almost the double from the MAE, which is a good sign that there could be a lot of outliers in the data.

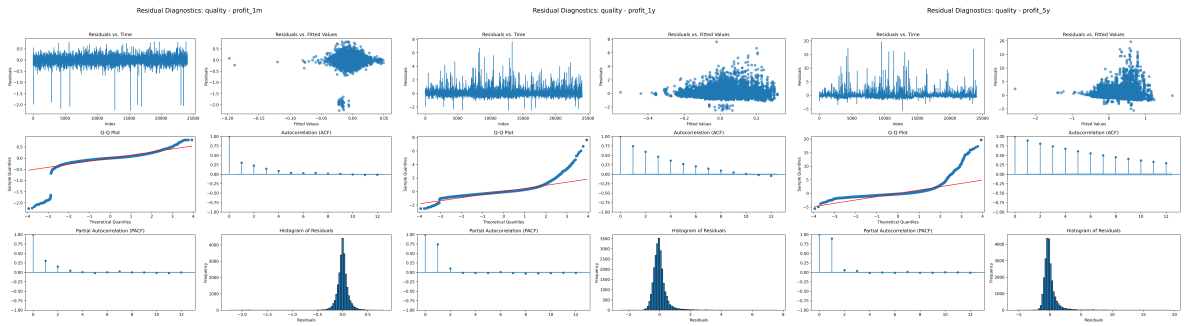


Figure 4.13: Small Data Future - MCOD 5% - Residuals

Also we went to see the residues and saw that they were multiple problems with them:

- **Normality**. When looking at the Q-Q plot, we can see that the residuals are not normally distributed, but there appears to be a section where normality is achieved.
- **Heteroskedasticity**. When looking at the residuals vs fitted values plot, we can see that the variance is not constant across all time horizons.
- **Autocorrelation**. When looking at the residuals vs fitted values plot, we can see that the residuals are not independent of each other.

Finally we didn't even see any consistency with the scores since its effect (sign) kept changing and the coefficients themselves didn't have statistical significance.

Table 4.1: Regression Results by Score and Time Horizon

Score	Metric	1M	3M	6M	1Y	2Y	5Y
Dividend	Coef.	-0.0013	–	-0.0004	-0.0026	–	0.0003
	Sign	–	0	–	–	0	+
	t-value	-6.38	–	-3.22	-6.92	–	6.44
Growth	Coef.	0.0018	–	0.0007	0.0047	0.0001	–
	Sign	+	0	+	+	+	0
	t-value	11.27	–	5.75	13.86	1.62	–
Quality	Coef.	-0.0038	-0.0059	-0.0010	-0.0076	-0.0003	0.0002
	Sign	–	–	–	–	–	+
	t-value	-6.93	-4.54	-4.29	-8.61	-1.87	2.47
Value	Coef.	0.0015	-0.0054	–	–	–	-0.0002
	Sign	+	–	0	0	0	–
	t-value	4.64	-4.36	–	–	–	-1.81

TODO: CONTINUE

The problem with ARIMA

The first thing we tried seeing the clear autocorrelation problems with the residues was to use the ARIMA models to improve the performance of the linear regression models and also fix that autocorrelation. This worked like a charm, and we were able to get a better model:

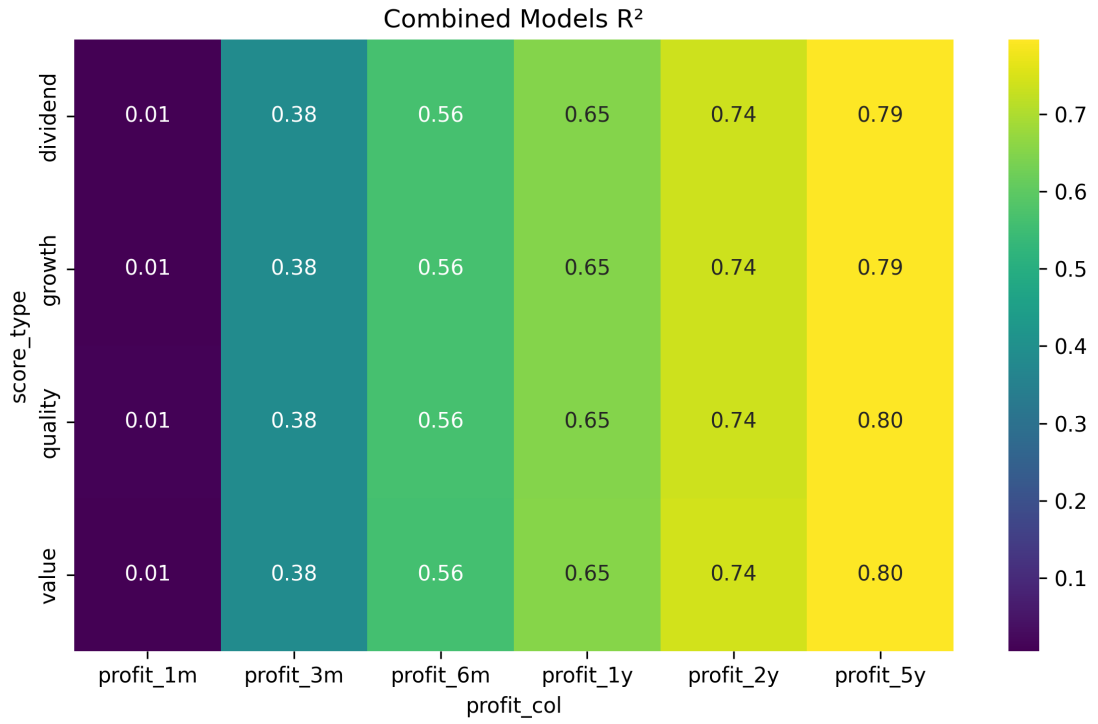


Figure 4.14: Small Data Future - MCOD 5% - ARIMA

TODO: CONTINUE

Geographical Effects

As we have already mentioned, the countries and sectors have both a big effect on the profits. Tweenvest's algorithm already takes into account the sector effect, but we thought that the regions could be having a big effect on the models, so we separated the data by regions and proceed to analyze the models' results:

The importance of a clean dataset

We saw that there were countries with far more outliers than others, so we decided to create a new dataset with less outliers in total and balancing the regions and sectors. Doing this improved a lot the models' performance, and residues.

TODO: CONTINUE

4.2.2 GAM

But of course, these results did not complete all of the objectives of this work, such as seeing the non-linear relationships between the variables and also help Tweenvest set its scores for a better understanding of how they work.

Also as expected, we improved the models' performance and residues:

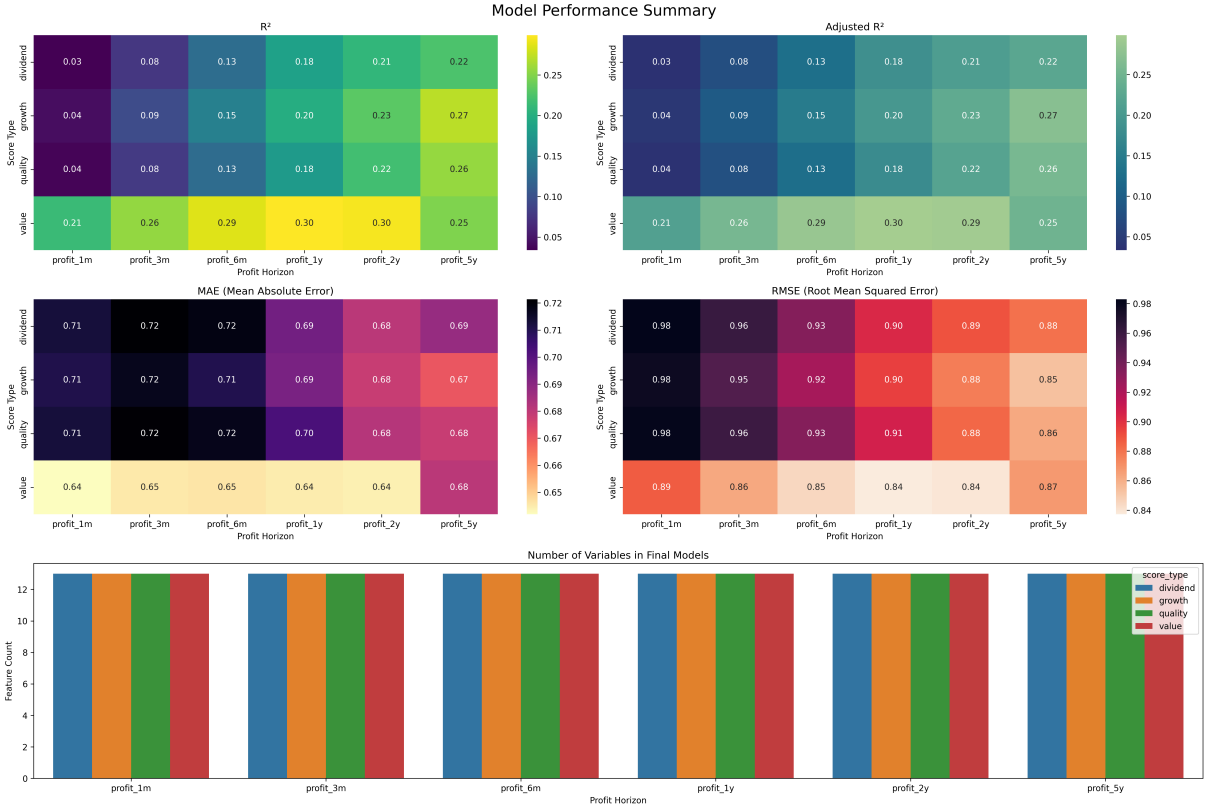


Figure 4.15: Small Data Future - MCOD 5% - GAM

Non-linear relationships

We also wanted to see if there were any non-linear relationships between the variables, so we tried to use the GAM model. This model is able to capture non-linear relationships between the variables, and it is also able to capture the geographical effects.

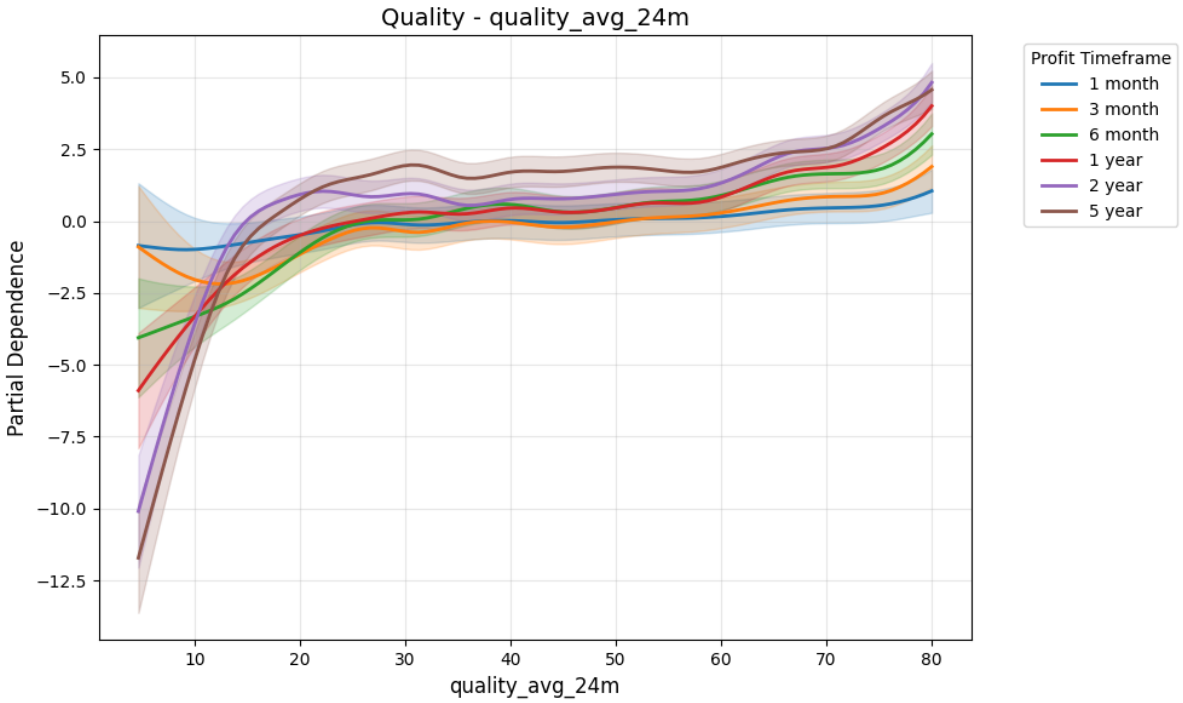


Figure 4.16: Small Data Future - MCOD 5% - GAM - Avg 24M Quality

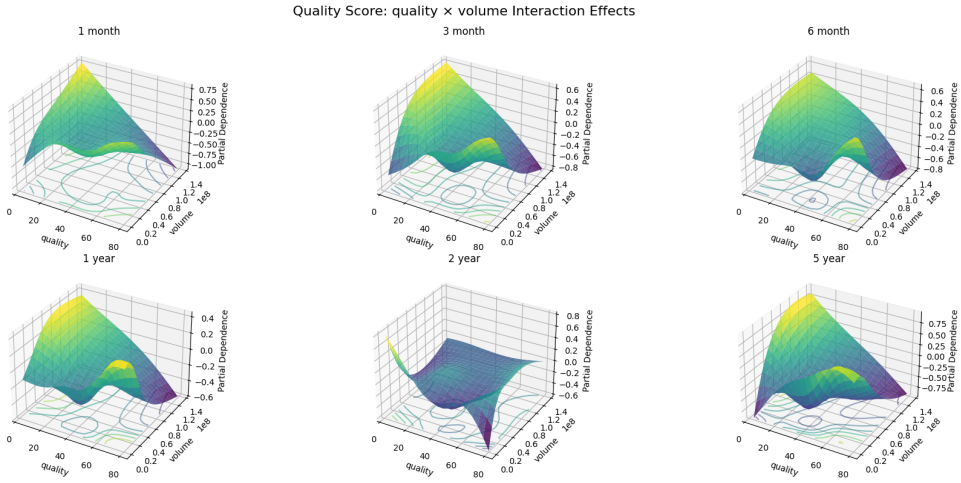


Figure 4.17: Small Data Future - MCOD 5% - GAM - Quality x Volume

4.2.3 Neural Network

Regressor

Binary Classifier

Multi-class Classifier

Chapter 5

Conclusion and Discussion

5.1 Conclusions

5.1.1 Goals achieved

5.1.2 Goals not achieved

5.1.3 Future work

5.2 Discussion

5.2.1 Limitations of the study

5.2.2 Recommendations to Tweenvest

Bibliography

[1] A. A. Abro, E. Taşci, and A. Uğur. "A Stacking-based Ensemble Learning Method for Outlier Detection". In: *Balkan Journal of Electrical & Computer Engineering* 8.2 (Apr. 2020). URL: https://dergipark.org.tr/en/download/article-file/1106223.

[2] World Bank. *Global Inflation Rates*. URL: https://www.worldbank.org/en/research/brief/inflation-database. 2025.

[3] Markus M. Breunig et al. "LOF: identifying density-based local outliers". In: *ACM sigmod record*. 2000. URL: https://dl.acm.org/doi/pdf/10.1145/335191.335388.

[4] X. Chen et al. "Economic policy uncertainty and the predictability of stock returns: impact of China in Asia". In: *China Accounting and Finance Review* 26.5 (2024), pp. 645–679. DOI: 10.1108/CAFR-11-2023-0144. URL: https://doi.org/10.1108/CAFR-11-2023-0144.

[5] scikit-learn developers. *MLPClassifier*. scikit-learn Documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. 2025.

[6] scikit-learn developers. *MLPRegressor*. scikit-learn Documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor. 2025.

[7] scikit-learn developers. *OneClassSVM*. scikit-learn Documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html. 2025.

[8] Inc. Docker. *Docker Compose Documentation*. Online documentation. URL: https://docs.docker.com/compose/. 2024.

[9] Official Documentation. *bulk_create Method*. Django Documentation. URL: https://docs.djangoproject.com/en/5.2/ref/models/querysets/#bulk-create. 2025.

[10] Official Documentation. *select_related Method*. Django Documentation. URL: https://docs.djangoproject.com/en/5.2/ref/models/querysets/#django.db.models.query.QuerySet.select_related. 2025.

[11] P. Dorsey and J. Mansueto. *The Five Rules for Successful Stock Investing*. 2011.

[12] Walter Enders. *Applied Econometric Time Series*. URL: http://sggscclibrary.saraswatilib.com/UploadImg/637617646888135985.pdf. 1948, p. 293.

[13] John D. Hunter et al. *Matplotlib: Python plotting package*. Online documentation. URL: https://matplotlib.org/stable/. 2025.

[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation-based anomaly detection". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.1 (2012), p. 3. URL: https://www.lamda.nju.edu.cn/publication/tkdd11.pdf.

[15] Qigui Liu and Gary Tian. "Controlling shareholder, expropriations and firm's leverage decision: Evidence from Chinese Non-tradable share reform". In: *Journal of Corporate Finance* 18.4 (2012). Special Section: Contemporary corporate finance research on South America, pp. 782–803. ISSN: 0929-1199. URL: https://www.sciencedirect.com/science/article/pii/S0929119912000533.

[16] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *arXiv preprint arXiv:1705.07874* (2017). URL: https://arxiv.org/abs/1705.07874.

[17] MSCI. "Fundamental Data Methodology". In: (June 2025). URL: https://www-cdn.msci.com/documents/10199/0b41b3a3-9da8-46f6-fce7-1dc1177e4a23.

[18] S&P Dow Jones Indices. *S&P 500*. URL: https://www.spglobal.com/spdji/en/indices/equity/sp-500/#overview. 2024.

[19] Daniel Servén and Charlie Brummitt. *Generalized Additive Models*. pyGAM Documentation. URL: https://pygam.readthedocs.io/en/latest/notebooks/tour_of_pygam.html. 2018.

[20] William F. Sharpe. *Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk*. Vol. 19. 3. Sept. 1964, pp. 425–442.

[21] The pandas development team. *pandas Documentation*. pandas Documentation. URL: https://pandas.pydata.org/. 2024.

[22] Xinjie Wang et al. "The causal relationship between social media sentiment and stock return: Experimental evidence from an online message forum". In: *Economics Letters* 216 (2022), p. 110598. ISSN: 0165-1765. DOI: https://doi.org/10.1016/j.econlet.2022.110598. URL: https://www.sciencedirect.com/science/article/pii/S0165176522001793.

[23] Michael L. Waskom. *seaborn: statistical data visualization*. Seaborn Documentation. URL: https://seaborn.pydata.org/index.html. 2024.

[24] Noumir Zineb, Paul Honeine, and Cedue Richard. "On simple one-class classification methods". In: *IEEE International Symposium on Information Theory Proceedings* (2012). URL: https://www.researchgate.net/publication/261248728_On_simple_one-class_classification_methods.

Part II

Appendix

Chapter 6

Code Listings

6.1 Historical Scores Job Implementation

The following code snippets shows the complete implementation of the historical scores job that was used to populate the database with historical factor scores data:

6.1.1 Enqueueing Historical Scores Job

```
@job("historical_scores", timeout="3h")
def enqueue_calculate_index_scores_task(
    start_date: datetime.date,
    end_date: datetime.date,
    index_names: list[str] | None = None,
    region_names: list[str] | None = None,
    total_stocks: int | None = None,
    seed: float | None = None,
    daily: bool = False,
):
    logger.info(
        "[H1] Starting enqueue_year_tasks from %s to %s",
        start_date,
        end_date,
    )

    days = get_range_from_dates(
        start_date,
        end_date,
        delta=relativedelta(days=1) if daily else relativedelta(
            months=1),
    )
    days.reverse()

    if not index_names:
        index_names = list(
            Index.objects.exclude(name__in=[Index.INDEX_ALL, "Other"
                ])
            .only("name")
            .values_list("name", flat=True)
        )
```

```
30      logger.info("[H1] Enqueuing tasks for %s indexes", len(index_
            names))
31
32      stocks_per_index = (
33          int(total_stocks / len(index_names)) if total_stocks else
                None
34      )
35
36      stock_count = 0
37      total_stocks_dict = {}
38      for index_name in index_names:
39          stock_ids = get_stock_ids_for_index(
40              index_name,
41              stocks_per_index,
42              start_date,
43              end_date,
44              seed,
45              region_names,
46          )
47          if not stock_ids:
48              raise ValueError(
49                  f"[H1] No stocks found for index '{index_name}' after
                        {start_date}"
50              )
51          total_stocks_dict[index_name] = stock_ids
52          for day in days:
53              calculate_index_scores_task.delay(
54                  index_name=index_name,
55                  day=day,
56                  stock_ids=stock_ids,
57              )
58
59          stock_count += len(stock_ids)
60
61      logger.info(
62          "[H1] Enqueued days %s for stocks:%s .", len(days), stock_
                count
63      )
64
65      return {
66          "stocks_processed": stock_count,
67          "days": len(days),
68          "total_stocks": total_stocks_dict,
69      }
```

Listing 6.1: Enqueueing Historical Scores Job

This job function is responsible for:

- Accepting date ranges and configuration parameters for historical score calculation
- Generating a list of dates to process (either daily or monthly intervals)
- Retrieving stock IDs for each index based on the specified criteria
- Enqueuing individual calculation tasks for each day and index combination
- Providing detailed logging and return statistics about the processing

6.1.2 Calculate Index Scores Data

```python

@job("historical_scores", timeout="45m")
def calculate_index_scores_task(
    index_name: str,
    day: datetime.date,
    stock_ids: list[int],
):
    start_time = time.perf_counter()
    logger.info(
        "[H2] Calculating index",
        extra={
            "index": index_name,
            "day": day,
            "stocks": len(stock_ids),
        },
    )

    # ---------------- Index calculation ----------------- #
    index_stats = get_indexes_information(
        index_names=[index_name], calculation_date=day
    )
    index_quality = get_quality_ratios_by_index(
        index_names=[index_name], calculation_date=day
    )
    stats = index_stats.get(index_name)
    quality = index_quality.get(index_name)

    # ------------------------------------------------------ #
    if (
        not stats
        or not quality
        or all(value is None for value in quality.values())
        or all(
            value is np.nan
            for stat in stats.values()
            for value in stat.values()
        )
    ):
        raise RuntimeError(
            f"[H2] Missing index data for {index_name} on {day},
               stats: {stats}, quality: {quality}"
        )

    calculate_stocks_scores_task.delay(
        day=day,
        stats=stats,
        quality=quality,
        stock_ids=stock_ids,
    )
    return {
        "index_calculated": index_name,
        "total_time": time.perf_counter() - start_time,
    }
```

Listing 6.2: Calculate Index Scores Data

This job function is responsible for:

- Calculating all the necessary index stats data
- Enqueuing the calculation of the stocks scores for the given index and date
- Providing detailed logging and return statistics about the processing

6.1.3 Calculate Stocks Scores Task

```python
@tracer.start_as_current_span("calculate_stocks_scores_task")
@job("historical_scores", timeout="4h")
def calculate_stocks_scores_task(
    day: datetime.date,
    stats: dict,
    quality: dict,
    stock_ids: list[int],
):
    """
    Job 3: For each stock, calculate scores and bulk save.
    """
    start_time = time.perf_counter()
    logger.info(
        "[H3] Calculating scores",
        extra={"day": day, "stocks": len(stock_ids)},
    )

    if not stock_ids:
        return {
            "stocks_processed": 0,
        }
    # Select related for trying to avoid multiple queries to the DB
    stocks = Stock.objects.filter(id__in=stock_ids).select_related(
        "share_type",
        "share_type__company",
        "share_type__company__industry",
        "share_type__company__industry__industry_group",
        "share_type__company__industry__industry_group__sector",
    )
    factor_scores = []
    for stock in stocks:
        score = calculate_scores(
            stock,
            stats,
            quality,
            calculation_date=day,
            bulk_create_or_update=True,
        )
        if not score:
            logger.warning(
                "[H3] No score for calculated",
                extra={"stock_id": stock.pk, "day": day, "score":
                    score},
            )
            continue  # Not as important as the others, need to check
                errors in DataDog
        factor_scores.append(score)
```

```
46
47      bulk_create_or_update(
48          model=FactorScore,
49          object_list=factor_scores,
50          unique_fields=["stock", "date"],
51          update_fields=["quality", "growth", "value", "dividend"],
52      )
53
54      return {
55          "stocks_processed": len(stock_ids),
56          "total_time": time.perf_counter() - start_time,
57          "time_per_stock": (time.perf_counter() - start_time) / len(
                stock_ids),
58      }
```

Listing 6.3: Calculate Stocks Scores Task

This job function is responsible for:

- Calculating the factor scores for each stock in the given index and date
- Bulk saving the factor scores for the given index and date
- Providing detailed logging and return statistics about the processing

6.2 Workers Management

In this section we have 2 different configurations for the workers management, since we had to switch servers.

6.2.1 Tweenvest's Production Server

```
1 [program:worker1]
2 command=python manage.py rqworker internal_tasks ciq_priority fmp_
      priority ciq indices_partial indices_full historical_scores fmp
      emails --with-scheduler
3 directory=/app
4 autostart=true
5 autorestart=true
6 redirect_stderr=true
7 stdout_logfile=/dev/fd/1
8 stdout_logfile_maxbytes=0
9 environment = SERVICE_NAME="tweenvest-worker"
10
11 [program:worker2]
12 command=python manage.py rqworker internal_tasks ciq_priority fmp_
      priority ciq indices_partial indices_full historical_scores fmp
      emails --with-scheduler
13 autostart=true
14 directory=/app
15 autorestart=true
16 redirect_stderr=true
17 stdout_logfile=/dev/fd/1
18 stdout_logfile_maxbytes=0
19 environment = SERVICE_NAME="tweenvest-worker"
```

```
20
21 [program:worker3]
22 command=python manage.py rqworker internal_tasks ciq_priority fmp_
       priority ciq indices_partial fmp emails historical_scores --with-
       scheduler
23 autostart=true
24 directory=/app
25 autorestart=true
26 redirect_stderr=true
27 stdout_logfile=/dev/fd/1
28 stdout_logfile_maxbytes=0
29 environment = SERVICE_NAME="tweenvest-worker"
30
31 [program:worker4]
32 command=python manage.py rqworker ciq_priority fmp_priority ciq
       indices_partial fmp emails historical_scores --with-scheduler
33 autostart=true
34 directory=/app
35 autorestart=true
36 redirect_stderr=true
37 stdout_logfile=/dev/fd/1
38 stdout_logfile_maxbytes=0
39 environment = SERVICE_NAME="tweenvest-worker"
40
41 [program:worker5]
42 command=python manage.py rqworker fmp_priority ciq_priority ciq fmp
       emails --with-scheduler
43 autostart=true
44 directory=/app
45 autorestart=true
46 redirect_stderr=true
47 stdout_logfile=/dev/fd/1
48 stdout_logfile_maxbytes=0
49 environment = SERVICE_NAME="tweenvest-worker"
50
51 [program:worker6]
52 command=python manage.py rqworker emails --with-scheduler
53 autostart=true
54 directory=/app
55 autorestart=true
56 redirect_stderr=true
57 stdout_logfile=/dev/fd/1
58 stdout_logfile_maxbytes=0
59 environment = SERVICE_NAME="tweenvest-worker"
60
61 [program:worker7]
62 command=python manage.py rqworker ciq_priority ciq emails summaries
       --with-scheduler
63 directory=/app
64 autostart=true
65 autorestart=true
66 redirect_stderr=true
67 stdout_logfile=/dev/fd/1
68 stdout_logfile_maxbytes=0
69 environment = SERVICE_NAME="tweenvest-worker"
70
71 [program:worker8]
```

```
72 command=python manage.py rqworker ciq_priority ciq emails summaries
       --with-scheduler
73 directory=/app
74 autostart=true
75 autorestart=true
76 redirect_stderr=true
77 stdout_logfile=/dev/fd/1
78 stdout_logfile_maxbytes=0
79 environment = SERVICE_NAME="tweenvest-worker"
80
81 [program:worker9]
82 command=python manage.py rqworker historical_scores ciq_priority
       summaries ciq emails --with-scheduler
83 directory=/app
84 autostart=true
85 autorestart=true
86 redirect_stderr=true
87 stdout_logfile=/dev/fd/1
88 stdout_logfile_maxbytes=0
89 environment = SERVICE_NAME="tweenvest-worker"
90
91 [program:worker10]
92 command=python manage.py rqworker summaries ciq_priority fmp_priority
       ciq fmp emails --with-scheduler
93 directory=/app
94 autostart=true
95 autorestart=true
96 redirect_stderr=true
97 stdout_logfile=/dev/fd/1
98 stdout_logfile_maxbytes=0
99 environment = SERVICE_NAME="tweenvest-worker"
```

Listing 6.4: Production Workers Management

6.2.2 Personal Development Server

```
1 [program:worker1]
2 command=python manage.py rqworker historical_scores
3 process_name=%(program_name)s_%(process_num)02d
4 numprocs=16
5 directory=/app
6 autostart=true
7 autorestart=true
8 redirect_stderr=true
9 stdout_logfile=/dev/fd/1
10 stdout_logfile_maxbytes=0
11 environment = SERVICE_NAME="tweenvest-worker"
```

Listing 6.5: Hetzner Workers Management

6.3 Data Export Job Implementation

The following code shows the complete implementation of the data export job that was used to create the final dataset, including the price estimation for missing values due to weekends and

holidays:

```python
def export_factors_and_pricehistory_task(
    stocks_id: list[int],
    start_date: datetime.date,
    end_date: datetime.date,
    export_name: str,
    daily: bool = False,
):
    """
    Export FactorScore, PriceHistory y rentabilidades a CSV sin cargar todo en memoria.
    """
    start_time = time.perf_counter()
    print("Starting CSV export...")

    periods = {
        "profit_1m": relativedelta(months=1),
        "profit_3m": relativedelta(months=3),
        "profit_6m": relativedelta(months=6),
        "profit_1y": relativedelta(years=1),
        "profit_2y": relativedelta(years=2),
        "profit_5y": relativedelta(years=5),
    }

    factor_fields = [
        "stock__ticker",
        "stock__share_type__company__name",
        "stock__share_type__company__industry__industry_group__sector__name",
        "stock__share_type__company__country__region",
        "date",
        "quality",
        "growth",
        "value",
        "dividend",
    ]
    price_fields = ["market_cap_usd", "volume"]

    days = get_range_from_dates(
        start_date,
        end_date,
        delta=relativedelta(days=1) if daily else relativedelta(months=1),
    )
    days.reverse()

    profitability_fields = list(periods.keys())
    export_fields = factor_fields + price_fields + profitability_fields
    missing_prices = 0
    missing_factors = 0
    fs_path = f"api/factors/data_exports/{export_name}.csv"
    with open(fs_path, mode="w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=export_fields)
        writer.writeheader()

        for stock_id in tqdm(stocks_id, desc="Stocks"):
```

```
53
54              price_qs = PriceHistory.objects.filter(
55                  stock__id=stock_id,
56                  date__range=(
57                      start_date,
58                      end_date + relativedelta(years=5),
59                  ),
60              ).values("date", "close_price", "market_cap", "volume", "
                    fx_mult")
61              price_lookup_stock = {
62                  (stock_id, ph["date"]): ph
63                  for ph in price_qs
64                  if ph["fx_mult"] is not None
65              }
66
67              dividend_qs_stock = DividendHistory.objects.filter(
68                  stock__id=stock_id,
69                  ex_dividend_date__range=(start_date, end_date),
70              ).values("ex_dividend_date", "adjusted_dividend", "fx_
                    mult")
71              dividend_lookup = {
72                  (stock_id, dh["ex_dividend_date"]): (
73                      float(dh["adjusted_dividend"]) * float(dh["fx_
                            mult"])
74                  )
75                  for dh in dividend_qs_stock
76                  if dh["fx_mult"] is not None
77              }
78              fs_qs_stock = (
79                  FactorScore.objects.filter(stock_id=stock_id, date__
                        in=days)
80                  .select_related(
81                      "stock__share_type__company__industry__industry_
                            group__sector",
82                      "stock__share_type__company__country",
83                  )
84                  .values(*factor_fields)
85              )
86
87              fs_lookup = {(stock_id, fs["date"]): fs for fs in fs_qs_
                    stock}
88
89              for day in days:
90                  factor = fs_lookup.get((stock_id, day))
91                  if not factor:
92                      missing_factors += 1
93                      continue
94                  day = factor["date"]
95                  price_day = day
96                  while price_day.weekday() >= 5:
97                      price_day -= datetime.timedelta(days=1)
98
99                  price = price_lookup_stock.get((stock_id, price_day))
100                 if not price or price["close_price"] is None:
101                     price = _calculate_existing_prices(
102                         price_day, price_lookup_stock, stock_id
103                     )
104                     if not price:
```

```
                        missing_prices += 1
                        continue

                try:
                    market_cap_usd = (
                        float(price.get("market_cap", 0) or 0)
                        * price["fx_mult"]
                    )
                except Exception:
                    market_cap_usd = None
                try:
                    volume = (
                        float(price.get("volume", 0) or 0) * price["
                            fx_mult"]
                    )
                except Exception:
                    volume = None

                if price["close_price"] is None or price["fx_mult"]
                    is None:
                    continue
                close_now_usd = float(price["close_price"]) * price["
                    fx_mult"]

                profitabilities = {}
                for field, delta in periods.items():
                    future_date = day + delta
                    while future_date.weekday() >= 5:
                        future_date -= datetime.timedelta(days=1)

                    future_price = price_lookup_stock.get(
                        (stock_id, future_date)
                    )
                    if not future_price or future_price["close_price"
                        ] is None:
                        future_price = _calculate_existing_prices(
                            future_date, price_lookup_stock, stock_id
                        )
                        if not future_price:
                            profitabilities[field] = None
                            missing_prices += 1
                            continue

                    close_future_usd = float(
                        future_price["close_price"]
                    ) * float(future_price["fx_mult"])
                    if close_future_usd == 0:
                        profitabilities[field] = None
                        missing_prices += 1
                        continue
                    # Dividend sum
                    dividend_sum = sum(
                        v
                        for (s_id, ex_div_date), v in dividend_lookup
                            .items()
                        if s_id == stock_id
                        and day <= ex_div_date < future_date
                    )
```

```
                    profitabilities[field] = (
                        close_future_usd + dividend_sum - close_now_
                            usd
                    ) / close_now_usd

                row = {**factor}
                row["market_cap_usd"] = market_cap_usd
                row["volume"] = volume
                row.update(profitabilities)
                writer.writerow(row)

    elapsed = time.perf_counter() - start_time
    print(f"Exportado a {fs_path} en {elapsed:.2f}s")
    print("MISSING PRICES:", missing_prices)
    print("MISSING FACTOR SCORES:", missing_factors)
```

Listing 6.6: Data Export Job Implementation

This job function is responsible for:

- Exporting factor scores, price history and profitability data to CSV without loading everything into memory
- Handling missing price data due to weekends and holidays by estimating prices from nearby dates
- Converting all monetary values to USD using appropriate exchange rates
- Calculating profitability for different time periods (1m, 3m, 6m, 1y, 2y, 5y)
- Including dividend payments in profitability calculations
- Tracking and reporting missing data points for both prices and factor scores

6.4 Custom Python Functions for Data Analysis

```
def correlation_matrix(
    df: pd.DataFrame, columns: list[str], file_path: str | None =
        None
) -> None:
    """
    Plot a correlation matrix for the DataFrame showing only the
        lower triangle.
    """
    df_columns = df[columns]
    corr_matrix = df_columns.corr()

    # Create a mask for the lower triangle
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

    # Plot using seaborn for better visualization
    plt.figure(figsize=(12, 8))
    sns.heatmap(
        corr_matrix,
        mask=mask,
        annot=True,
```

```
        cmap="coolwarm",
        center=0,
        fmt=".2f",
        square=True,
    )
    plt.title("Correlation Matrix of Numeric Variables")
    plt.tight_layout()

    if file_path:
        plt.savefig(file_path)
        plt.show()
    else:
        plt.show()
```

Listing 6.7: Correlation Matrix Plot Function

```
def pairplot(
    df: pd.DataFrame,
    file_path: str | None = None,
    kde: bool = False,
    hue: str = "region",
    hue_order: list[str] | None = None,
    palette: dict[str, str] | None = None,
) -> None:

    if len(df) > 10000:
        df = df.sample(10000, random_state=42)

    g = sns.pairplot(
        df,
        hue=hue,
        diag_kind="kde",
        hue_order=hue_order,
        palette=palette,
        corner=True,
    )
    if kde:
        g.map_lower(sns.kdeplot, levels=4, color=".2")

    if file_path:
        plt.savefig(file_path)
        plt.close()
    else:
        plt.show()
```

Listing 6.8: Custom Pairplot Function

6.5 Data Preprocessing

6.5.1 Data Cleaning

```
    import sys
    from pathlib import Path

    # Add the parent directory (code/) to sys.path
```

```
5     sys.path.append(str(Path().resolve().parent))
6     from utils import load_data, plot_numeric_distributions
7     import pandas as pd
8
9     # %% [markdown]
10    # # Loading the data
11
12    # %%
13    file_name = "Small Data future AF"
14    data_path = f"../data/raw/{file_name}.csv"
15
16    # Read the CSV file
17    df = load_data(data_path)
18
19    # %% [markdown]
20    # ## Cleaning and renaming
21
22    # %%
23    df = df.rename(
24        columns={
25            "stock__share_type__company__industry__industry_group__
                sector__name": (
26                "sector"
27            ),
28            "stock__share_type__company__country__region": "region",
29            "market_cap_usd": "market_cap",
30            "stock__share_type__company__name": "company",
31            "stock__ticker": "ticker",
32        }
33    )
34
35    df["dividend"] = df["dividend"].fillna(0.0)
36    df["date"] = pd.to_datetime(df["date"])
37    df_no_nan = df.dropna()
38
39
40    # %% [markdown]
41    # # Analyzing
42
43    # %%
44    export_path = f"../data/cleaned/basic/plots/{file_name}.png"
45    plot_numeric_distributions(df, file_path=export_path)
46
47    # %% [markdown]
48    # # Export cleaned data
49
50    # %%
51    # Export the cleaned dataset to CSV
52    cleaned_data_path = f"../data/cleaned/basic/{file_name}.csv"
53    df.to_csv(cleaned_data_path, index=False)
54
55    print(f"Cleaned data exported to: {cleaned_data_path}")
```

Listing 6.9: Basic Data Cleaning

6.5.2 Outlier Detection

Listing 6.10: Outlier Detection

6.6 Extra Images

6.6.1 Data

Columns and variables

Response Variables	Explanatory Variables				Time	Extra
Profit 1 month	**Scores**	**Dummies**		**Size**	Date	Company Name
Profit 3 months	Quality	***Region***	***Sector***	Market cap		Stock Ticker
Profit 6 months	Growth	USA and Canada	Communication Services	Volume		
Profit 1 year	Value	Europe	Consumer Discretionary			
Profit 2 years	Dividend	Latin America	Consumer Staples			
Profit 5 years		Asia and Oceania	Energy			
		Africa	Financials			
			Health Care			
			Industrials			
			Information Techonology			
			Materials			
			Real State			
			Utilities			
6	4	1	1	2	1	2

Figure 6.1: Number of columns per dataset

6.6.2 Distributions

Here we have all of the not shown images for further analysis, if needed.

Scores

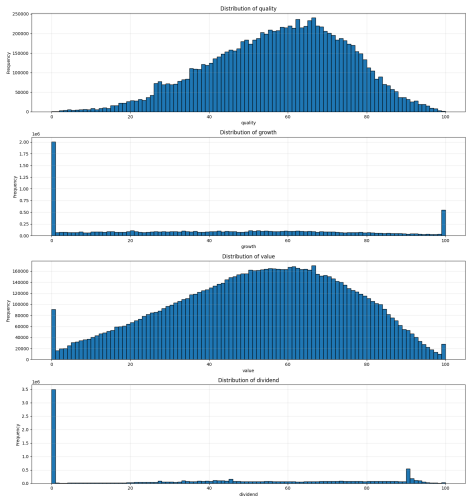


Figure 6.2: Big Data Past - Scores

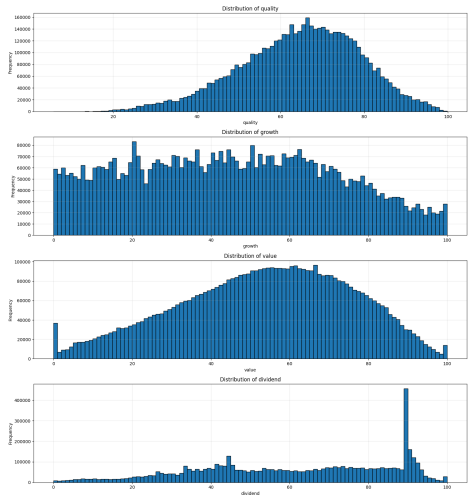


Figure 6.3: Big Data Past - Scores Deep

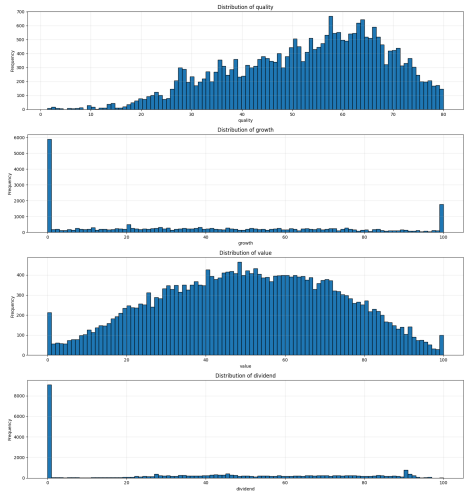


Figure 6.4: Small Data Future - Scores

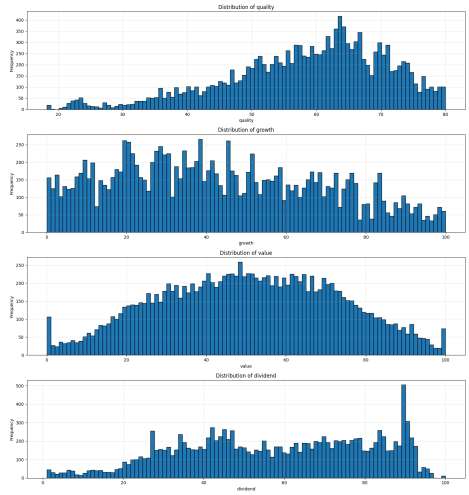


Figure 6.5: Small Data Future - Scores Deep

Dummies

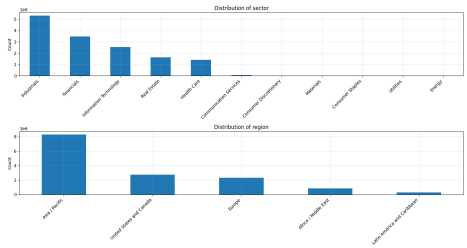


Figure 6.6: Big Data Future - Dummies

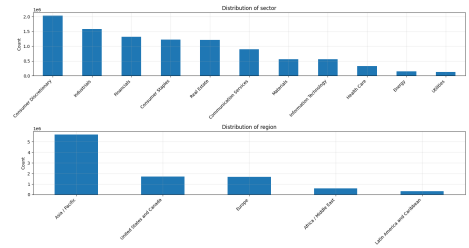


Figure 6.7: Big Data Past - Dummies

Profits

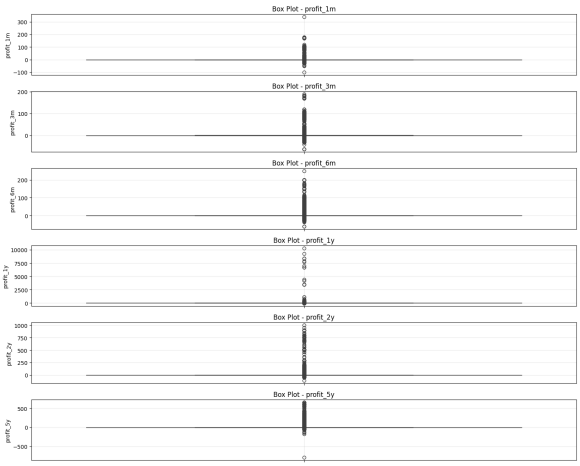


Figure 6.8: Big Data Future - Profits Boxplot

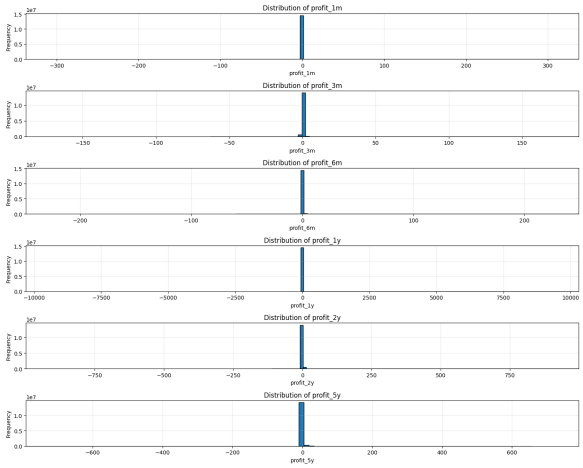


Figure 6.9: Big Data Future - Profits

6.6.3 Correlations

USA and Canada

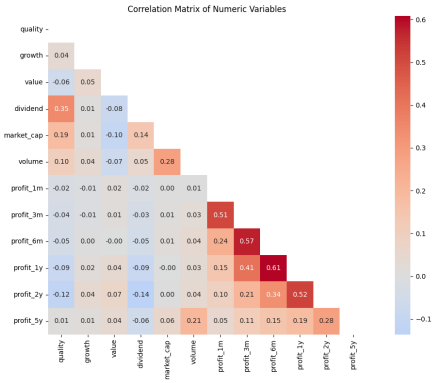


Figure 6.10: Small Data Future USA

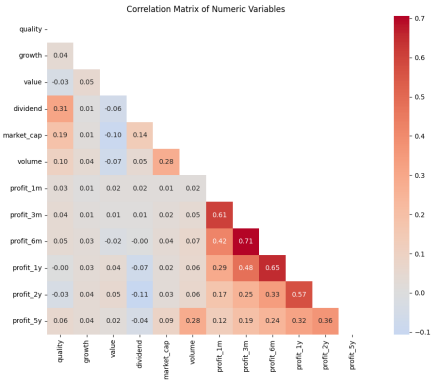


Figure 6.11: Small Data Future USA - MCOD

Europe

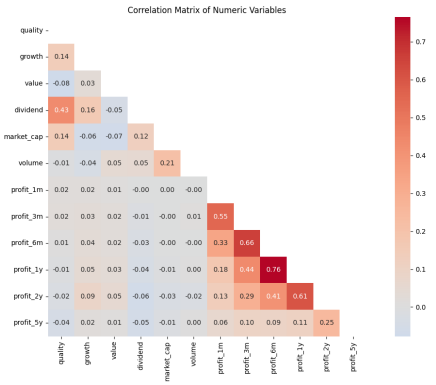


Figure 6.12: Small Data Future EU

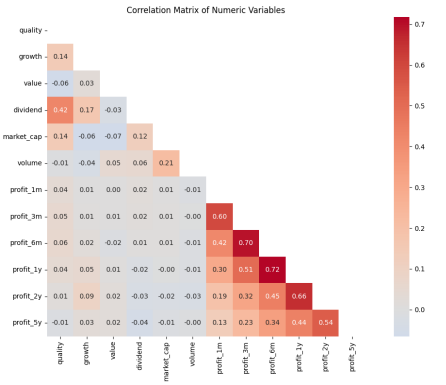


Figure 6.13: Small Data Future EU MCOD

Latin America

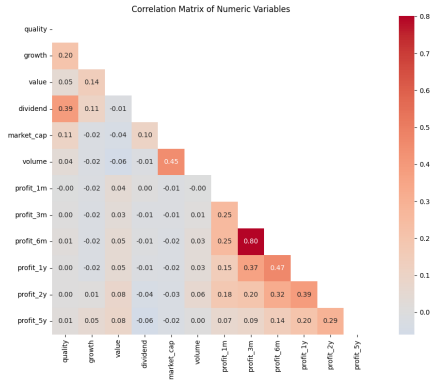


Figure 6.14: Small Data Future LAT

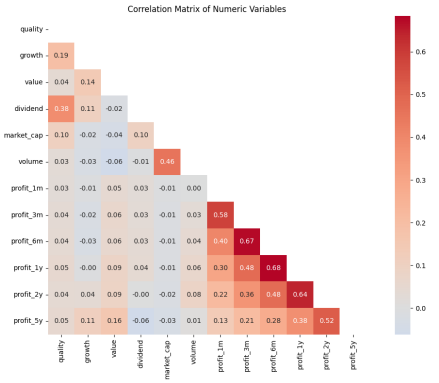


Figure 6.15: Small Data Future LAT - MCOD

Africa

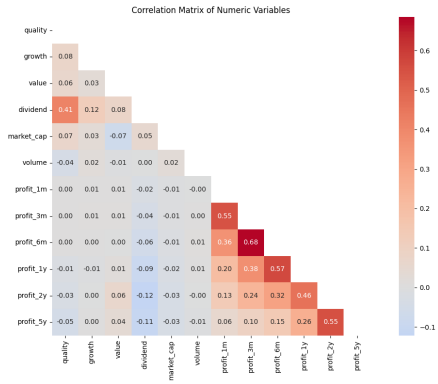


Figure 6.16: Small Data Future AF

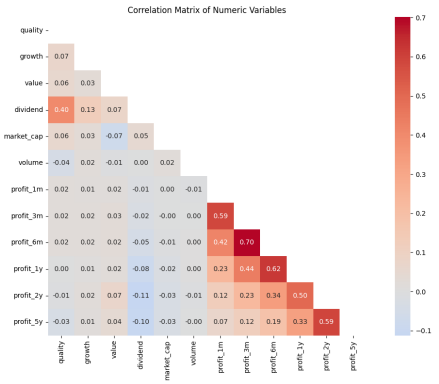


Figure 6.17: Small Data Future AF - MCOD

Asia and Oceania

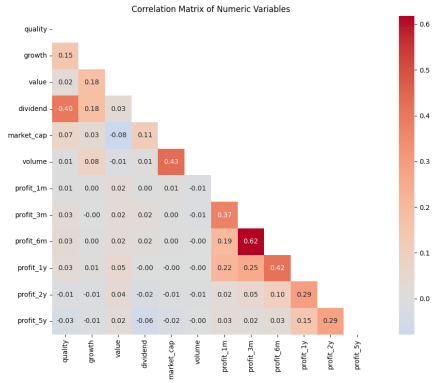


Figure 6.18: Small Data Future AS

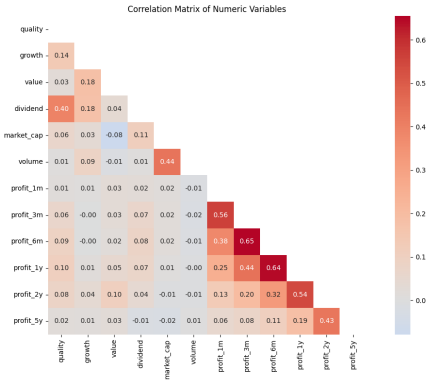


Figure 6.19: Small Data Future AS - MCOD