A Global Stock Market Protocol - Documentation

Julian Shen and Jiexuan Gao

Supervisor: Yannik Gabelmann

Technical University of Munich (TUM)

Abstract

The study of market anomalies has revealed numerous firm characteristics that systematically predict future stock performances. While many of these anomalies generate significant excess returns in empirical tests, their reproducibility and persistence across markets remain uncertain. Existing frameworks for anomaly validation have largely focused on U.S. equities, overlooking potential differences in international financial environments. To address this limitation, we introduce an open-access platform designed to replicate and evaluate stock market anomalies in a global context. The tool enables users to upload custom predictive signals and automatically conducts a range of established asset pricing tests. The framework is equipped with an integrated web interface that generates complete LaTeX reports as analysis results. Due to its highly modular design, the tool can be easily extended to incorporate additional models, tests, or data sources, making it a flexible foundation for future empirical asset pricing research.

1 Introduction

The study of stock market anomalies has long been an important topic in empirical finance research. Since the early introduction of the Three Factor Model by Fama and French [1] or the work of Jegadeesh and Titman [2] on momentum, it has been well known that asset returns can be systematically linked to firm characteristics and past performance. To date, empirical finance literature has revealed hundreds of cross-sectional firm characteristics that predict future stock returns [3, 4]. Most of these anomalies often generate significant returns when used to construct a portfolio strategy. However, while the initial results of such strategies may appear promising, concerns remain regarding the reproducibility and persistence of these methods, especially under different market conditions. In response, several methodological frameworks have been proposed to improve the testing and validation of anomalies, with the goal of producing more reliable and robust results [5, 6, 7]. Despite these advances, the overwhelming majority of anomaly studies and proposed frameworks have focused almost exclusively on the U.S. equity market [8]. This focus is due to the fact that the U.S. equity market provides extensive and detailed datasets and has historically served as the center of empirical asset pricing research. Platforms such as *Assaying Anomalies* [7] provide valuable resources for anomaly replication and evaluation, but they are confined only to U.S. data. This narrow scope risks overlooking important insights. Financial markets outside the United States are large, diverse, and subject to different institutional frameworks, regulatory environments, and investor behaviors. All of these influences may affect the performance and validity of anomaly-based strategies.

To address this limitation, we propose a new open-access tool designed to replicate and evaluate stock market anomalies in a truly global context. By systematically extending anomaly testing beyond U.S. equities, our platform enables researchers and practitioners to assess the generalizability and robustness of anomalies across international markets. Our framework comes with an integrated web application that allows users to test

a new predictive signal by simply uploading a `.csv` file including three columns: firm identifier, date, and signal. The uploaded signal data is then transmitted to the backend, where a series of anomaly tests are executed automatically. Currently, these include the Fama–French Three Factor Model [1], the Fama–French Five Factor Model [9], and the Q-Factor Model [10] with both value- and equal-weighting schemes, as well as the Fama–MacBeth Regression [11] and a correlation analysis. Furthermore, given the tool's global scope, the analyses are also performed at the level of individual countries or industries. The complete code base is openly available on GitHub[1], ensuring transparency and enabling researchers to build on our work. The framework's architecture and source code are designed in a highly modular way, making it particularly easy to incorporate new portfolio tests or modify the current implementation. Upon completion of the analysis, the application generates a self-contained report summarizing the results and automatically delivers both the corresponding LaTeX source files and the compiled PDF document to the submitter. A key benefit of our tool lies in its usability. It is accessible to all users, requires no coding skills or reliance on a specific platform, and allows tests to be conducted with a single click.

2 Background

A central element of empirical asset pricing research is the evaluation of predictive signals through established factor models and regression frameworks. Factor models explain asset returns as exposures to systematic sources of risk, such as the market risk premium [12] or the size effect [13], while regression techniques are used to estimate these exposures and test whether they account for differences in expected returns across assets. To provide the necessary background for understanding the analyses performed by our tool, this section introduces the factor models, the Fama–MacBeth Regression [11] and the correlation analysis, all of which are implemented in our framework.

2.1 Factor Models

Factor models describe how asset returns can be decomposed into components related to systematic risk factors. One of the most influential among them is the *Fama–French Three Factor Model* [1], which extends the traditional Capital Asset Pricing Model (CAPM) [12] by including the two additional factors size and value to account for return patterns unexplained by market risk alone. Formally, the model specifies the excess return of asset i at time t as:

$$R_{i,t} - R_{f,t} = \alpha_i + \beta_{i,\mathrm{MKT}}(R_{\mathrm{MKT},t} - R_{f,t}) + \beta_{i,\mathrm{SMB}}\,\mathrm{SMB}_t + \beta_{i,\mathrm{HML}}\,\mathrm{HML}_t + \varepsilon_{i,t},$$

where

- $R_{i,t}$ is the return of the asset i for period t,
- $R_{f,t}$ is the risk-free rate for period t,
- $R_{\mathrm{MKT},t}$ is the market return for period t,
- SMB_t is the size factor (Small Minus Big) for period t,
- HML_t is the book-to-market or value factor (High Minus Low) for period t,
- $\beta_{i,k}$ is the sensitivity of asset i to factor k,
- α_i is the intercept (abnormal return not explained by factors),
- and $\varepsilon_{i,t}$ an idiosyncratic error term for period t.

[1] `https://github.com/Proenchen/stock-market-protocol`

Empirical evidence shows that the inclusion of SMB and HML substantially improves explanatory power relative to the CAPM [14, 15], making the Three Factor Model a cornerstone of modern asset pricing.

Building on this foundation, the *Fama–French Five Factor Model* [9] further adds the two dimensions profitability (RMW, Robust Minus Weak) and investment (CMA, Conservative Minus Aggressive) to capture the empirical observation that more profitable and conservatively investing firms earn higher returns. Similarly, the *Q–Factor Model* by Hou, Xue, and Zhang [10] reformulates the factor approach from an investment-based theoretical perspective, focusing on market, size, investment, and profitability factors tied directly to firms' optimal investment decisions.

All three models are available in our framework under both value- and equal-weighting schemes, allowing users to evaluate predictive signals across alternative specifications. For a more detailed explanation of these models, we refer to the original papers [1, 9, 10].

2.2 Fama–MacBeth Regression

The Fama–MacBeth Regression [11] estimates the cross-sectional relationship between firm characteristics and expected returns while allowing these relations to vary over time. Each month, a cross-sectional regression is run, and the resulting coefficients are averaged over time to obtain both the mean effect and its temporal stability. This approach provides interpretable statistics on the magnitude and persistence of a signal's return predictive power.

In our implementation, for each month t we estimate:

$$\begin{aligned} r_{i,t} = \alpha_t &+ \beta_t^{\text{sig}} \,\text{Signal}_{i,t} + \beta_t^{\text{size}} \,\text{Size}_{i,t} + \beta_t^{\text{mom}} \,\text{Mom}_{i,t} \\ &+ \beta_t^{\text{bm}} \,\text{BM}_{i,t} + \beta_t^{\text{ag}} \,\text{AG}_{i,t} + \beta_t^{\text{rd}} \,\text{RD/Sales}_{i,t} + \varepsilon_{i,t}, \end{aligned}$$

where $r_{i,t}$ denotes the monthly USD return of stock i, and the regressors capture standard firm characteristics such as size, momentum, book-to-market ratio, asset growth, and R&D intensity. To ensure comparability across months and mitigate outliers, all characteristics are rank-scaled to the interval $[-1, 1]$ within each month.

2.3 Correlation Analysis

A correlation analysis serves as a first diagnostic step in assessing the economic interpretation and potential redundancy of a predictive signal. The underlying idea is to evaluate whether the signal under investigation is related to known risk factors or captures independent information about expected returns. Formally, the *Pearson correlation* [16] between two variables X and Y is defined as:

$$\rho_{X,Y} = \frac{\text{Cov}(X,Y)}{\sigma_X \, \sigma_Y} = \frac{\sum_{t=1}^{T}(X_t - \bar{X})(Y_t - \bar{Y})}{\sqrt{\sum_{t=1}^{T}(X_t - \bar{X})^2}\sqrt{\sum_{t=1}^{T}(Y_t - \bar{Y})^2}},$$

where $\bar{X}$ and $\bar{Y}$ denote the sample means, and σ_X, σ_Y the standard deviations of X and Y. The resulting coefficient $\rho_{X,Y}$ lies in the interval $[-1, 1]$, with values close to 1 or -1 indicating strong positive or negative linear dependence, respectively, and values near 0 indicating weak or no linear relationship.

In the context of asset pricing, correlations between a new signal and established factor proxies (such as market, size, value, profitability, or investment factors) provide insights into whether the signal represents an already known source of risk, reflected in high correlation with existing factors, or a potentially distinct dimension of return variation, reflected in low or near-zero correlations.

3 Design and Implementation

This chapter describes the design and implementation of our tool. We provide an overview of the frameworks and tools we used, as well as how and why we used them. We also go into detail about our framework's architecture, outline the most important processes and discuss the design patterns that we used to ensure a clean and modular structure. The architecture of our software was designed with a particular focus on maintainability and extensibility, so that users can easily add new or modify existing functionality in the future. Furthermore, we sketch out how the key components and algorithms in our framework are implemented and provide practical examples for the modifiability of our code base.

3.1 Framework and Tools

The project is primarily developed in Python, which was chosen because of its extensive ecosystem for data processing, network communication, and software prototyping. Python's expressive syntax and large library support make it very well suited for developing a system that must combine data ingestion, transformation, and protocol handling with web-based interfaces. In order to expose functionality via an accessible application interface, we employ Flask[2] as a lightweight web framework. This framework provides routing and request handling mechanisms while remaining minimal and flexible, thereby avoiding the overhead of larger frameworks. The web frontend of our application is implemented using standard web technologies, such as HTML and CSS, complemented by the Bootstrap[3] framework. Bootstrap was chosen because it provides a consistent and responsive design system that significantly accelerates frontend development. Its pre-defined components and grid system allow us to create user interfaces that are both visually appealing and device-independent without the need for extensive custom styling. At the same time, Bootstrap integrates seamlessly with our Flask templates, enabling dynamic content rendering while ensuring that the visual layout remains coherent and user-friendly. For the handling and processing of financial data in the backend, the tool makes use of scientific computing libraries including NumPy[4] and Pandas[5]. These libraries allow for efficient time-series operations, numerical computations, and structured data manipulation, all of which are indispensable when dealing with large volumes of market-related data.

3.2 Architecture

The architecture of our framework follows a layered design that separates concerns between data storage, protocol handling, backend logic, and user interaction. The dependencies between the layers are strictly from top to bottom, which is shown in Fig. 1. This modular structure facilitates maintainability and extensibility, enabling developers to introduce new functionality without interfering with existing components. The main layers can be summarized as follows:

- **Data Layer:** Responsible for storing market data. It contains all the data required to perform the portfolio tests, including CRSP data, factor data, and a specific dataset for the Fama–MacBeth analysis. Currently, the data is stored as CSV files, but this part could be replaced by an actual database in the future to improve efficiency and scalability.
- **Protocol Layer:** Encapsulates the core functionality of the stock market protocol. This layer defines the protocols for testing the given anomaly, such as the implementation of factor models or the Fama-Macbeth regression. By strictly separating

[2] `https://flask.palletsprojects.com/en/stable/`
[3] `https://getbootstrap.com/`
[4] `https://numpy.org/`
[5] `https://pandas.pydata.org/`

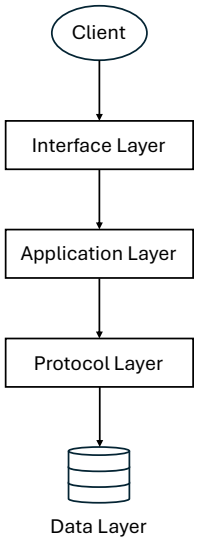

Figure 1: The 4-layered architecture of our tool. The arrows show the direction of dependencies between each layer.

protocol logic from other parts of the system, the framework ensures that it can evolve independently of presentation concerns.

- **Application Layer:** Implements the main logic of the system, including data validation and orchestration of protocol operations. This layer acts as the central mediator between the data and protocol layers, ensuring that input is transformed into meaningful output while maintaining system integrity.

- **Interface Layer:** Provides access points for end-users and external applications. On the server side, a lightweight Flask application exposes REST-like endpoints for interaction with the framework. On the client side, a web frontend built with HTML, CSS, and Bootstrap ensures usability and responsiveness across devices. The interface layer abstracts away the internal complexity of the system, presenting a clean and intuitive surface for users.

Figure 2 illustrates the class diagram of our backend architecture. It highlights the separation of responsibilities between core protocol classes, utility modules, and interface components. At the bottom, the `data` package represents the data layer. It provides all required market data in `.csv` format, which serves as the input for the portfolio tests.

The protocol layer is embodied by the `analyzers` package, which contains the concrete implementations of the anomaly analysis methods. All analyzers extend from the common base class `BaseAnalyzer`, which defines the method `analyze()` for executing the portfolio test algorithm and the method `generate_output()` for invoking `analyze()` and producing the LaTeX or string-based output encapsulated as an `AnalyzerOutput` object. The class constants are defined as follows:

- `ENABLED`: Specifies whether the corresponding portfolio test is included in the complete analysis.

- `ORDER`: Determines the execution order of the individual portfolio tests. The results in the LaTeX output file also adhere to this ordering.

- `TITLE`: Provides the section title in the LaTeX document for the respective analyzer.

The additional attributes are used to store the market data in Pandas dataframes, while `signal_name` refers to the name of the user-defined signal. Moreover, we adopted a *service*

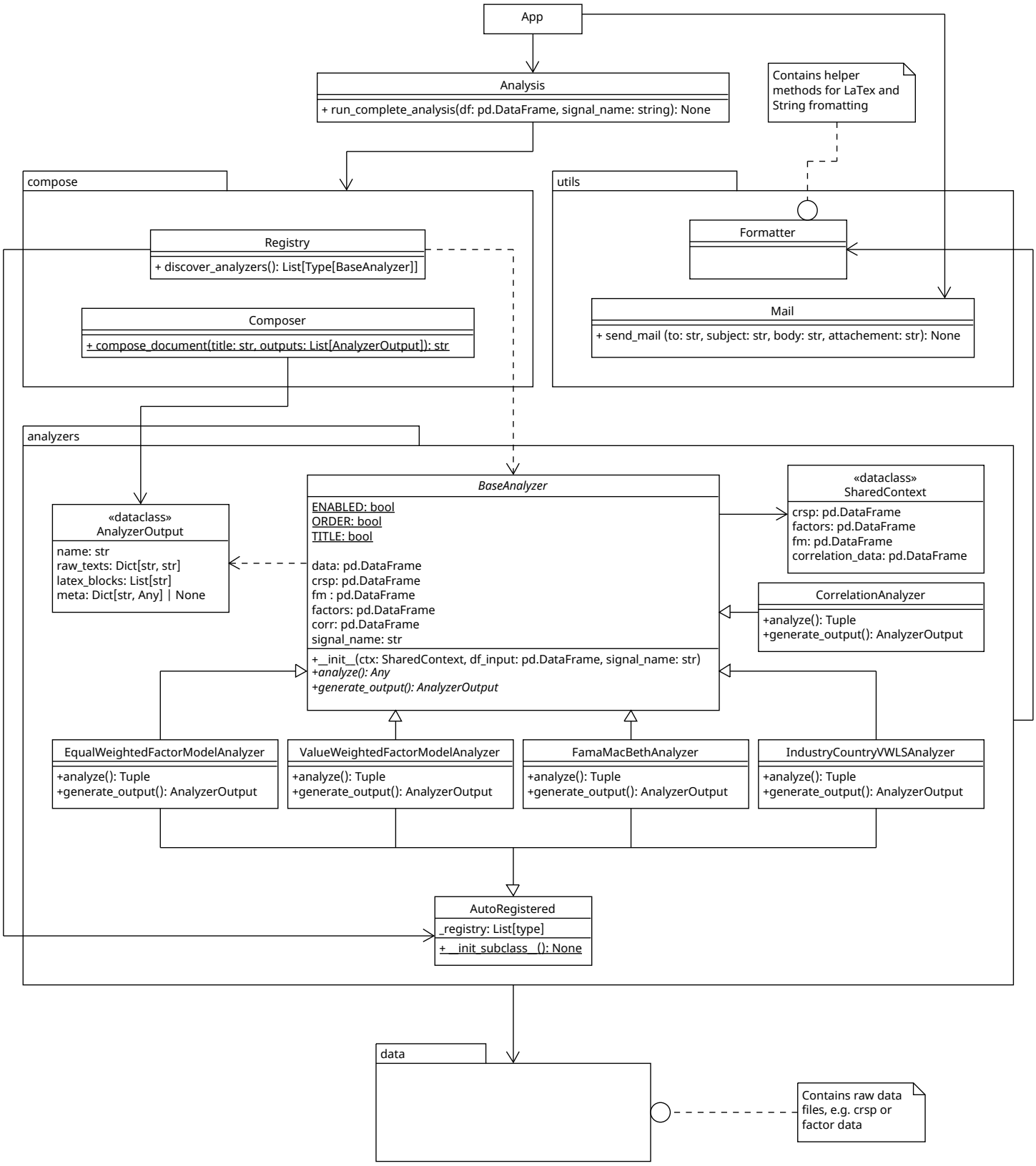

Figure 2: Class diagram of our framework. It contains all key elements of our backend architecture. The layered, hierarchical architecture can clearly be identified.

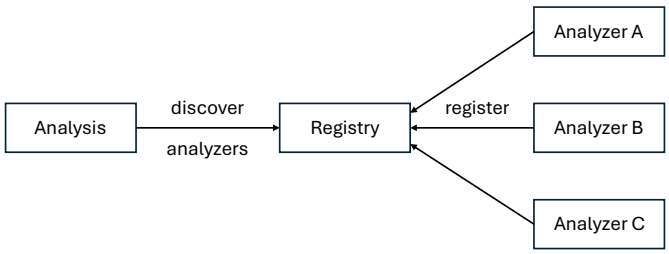

Figure 3: Adaption of the service discovery pattern. The `Analysis` class retrieves newly available analyzers directly from the service registry. All analyzers register themselves in the registry by inheriting the `AutoRegistry` class.

discovery pattern to enable the automatic detection of both existing and newly added analyzers. The service discovery pattern, shown in Fig. 3, is a design pattern that facilitates the dynamic identification of system components without requiring manual configuration. Instead of explicitly registering or hard-coding the location and availability of each service, components are discovered at runtime. Within our framework, this pattern ensures that any newly implemented analyzer is automatically recognized and seamlessly integrated into the analysis pipeline. Since all analyzers inherit from the `BaseAnalyzer` class, they conform to a common interface and can therefore be dynamically loaded and executed without additional configuration. In order for a new analyzer to be discovered through this mechanism, it must also inherit from the `AutoRegistry` class, which automatically registers the class in the service registry by invoking Python's built-in `__init_subclass__()` method. This approach not only minimizes overhead but also greatly enhances extensibility, as new functionality can be introduced simply by defining a new analyzer that adheres to the expected contract.

The `Analysis` class constitutes the application layer of the system. It retrieves the set of enabled analyzers from the registry and executes them sequentially according to their predefined `ORDER` attributes. Upon completion, it collects all resulting `AnalyzerOutput` objects and employs the `Composer` class to generate the final LaTeX document. This document is subsequently passed to the `App` class, which, with the support of the `Mail` utility, delivers the result to the user. In addition, the `App` class is also responsible for receiving the user's input file from the frontend and initiating a new analysis based on it. To further enhance usability, the system supports the simultaneous upload of multiple input files, which are then processed sequentially within the analysis pipeline.

The Interface Layer is realized through a web-based frontend that provides the primary access point for end-users. It is implemented using HTML and CSS to ensure broad compatibility, responsiveness, and usability across different platforms and devices. The frontend communicates with the backend exclusively through the Flask-based interfaces exposed by the `App` class. These interfaces follow a lightweight, REST-like design, enabling the transmission of user input files to the system as well as the retrieval of analysis results once they are generated. By abstracting away the internal complexity of the framework, the interface layer ensures a seamless interaction between users and the underlying system.

In addition to these backend components located in the `logic` package, our project also incorporates the standard Flask packages `static` and `templates`, which are used for the frontend development. The `static` package contains JavaScript code responsible for implementing the responsive behavior of website elements, as well as CSS code that defines the overall visual design and styling of the interface. The `templates` package comprises the HTML code that specifies the structure and content of each individual webpage. Communication between the backend and frontend is facilitated through the use of REST endpoints in Flask, such as `GET` and `POST` requests, which allow dynamic data to be exchanged between client-side components and server-side logic.

The activity diagram in Fig. 4 illustrates the complete analysis workflow.

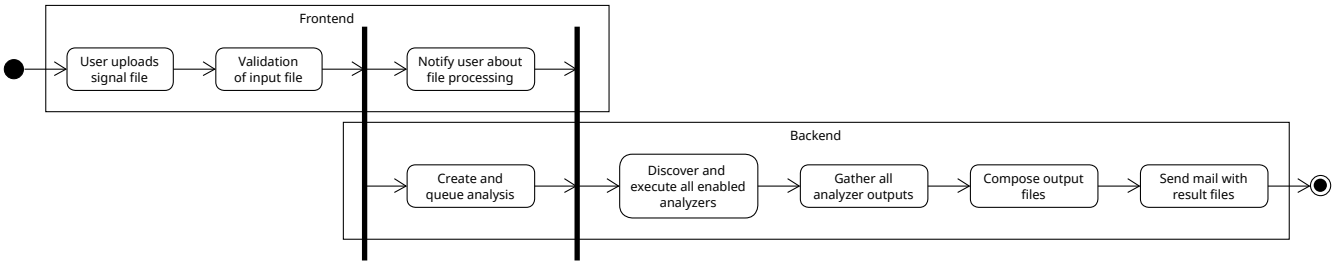

Figure 4: The normal workflow within our tool without the presence of errors.

3.3 Implementation

In this section, we explain key passages in our implementation and cover how the theoretical algorithms are practically implemented. Our implementation is designed to be very robust, so that no errors can arise from undefined values or messy data. In the following, you will see many tests in the source code that check for these edge cases. Most of the operations are implemented using the Pandas library.

First, we explain the implementation of the factor models. The `analyze()` method of the factor model analyzers method begins by standardizing the input data (DSCD, dates, and signal) and aligning them to monthly periods.

```
df = self.data.copy()
df = df.rename(columns={df.columns[0]: "DSCD",
                        df.columns[1]: "dates",
                        df.columns[2]: "signal"})
df["date"] = pd.to_datetime(df["dates"])
df["month"] = df["date"].dt.to_period("M").dt.to_timestamp("M")
```

This is necessary to ensure comparability with the CRSP return database and the factor time series, both of which are aggregated at the monthly level. After that, the CRSP dataset is joined to the signal data and the algorithm assigns the securities into ten portfolios (deciles) based on their signal values.

```
cr = self.crsp.copy()
# ...
cr_small = cr[["DSCD", "month", "RET_USD", "size_lag"]]
df = pd.merge(df, cr_small, on=["DSCD", "month"], how="inner")
df = df.groupby("month").apply(assign_deciles, include_groups=False)
```

This corresponds to the theoretical concept of portfolio sorts, where firms are grouped according to a characteristic of interest. The `EqualWeightedFactorModelAnalyzer` then computes equal-weighted average returns for each decile within each month.

```
def ewret(g: pd.DataFrame) -> float:
    v = g["ret"].dropna()
    return float(v.mean()) if len(v) else np.nan

decile_rets = (
    df.groupby(["month", "decile"]).apply(ewret, include_groups=False)
      .rename("ewret").reset_index()
)
```

This step implements the theoretical assumption of equal-weighted portfolio construction, where each asset contributes identically to the portfolio return regardless of its market capitalization. In contrast, in the `ValueWeightedFactorModelAnalyzer` class, portfolio returns are calculated using market capitalization as weights. Specifically, the lagged firm size (`size_lag`) from the CRSP dataset is used as a proxy for market capitalization and serves as the weight in the averaging procedure.

```python
def vwret(g: pd.DataFrame) -> float:
    w = pd.to_numeric(g["weight"], errors="coerce")
    r = pd.to_numeric(g["ret"], errors="coerce")
    m = w.notna() & r.notna() & (w > 0)

    if not m.any():
        return np.nan

    return float(np.average(r[m], weights=w[m]))
```

The factor dataset is then processed to extract the relevant variables and align them with the monthly decile returns. For each decile and for the long–short portfolio, time-series regressions are estimated against the factor specifications as explained in Sec. 2. The regression procedure uses OLS estimation from the statsmodels (`sm`) package.

```python
model_specs = {
    "FF3": ["MKTRF_usd", "SMB_usd", "HML_usd"],
    "FF5": ["MKTRF_usd", "SMB_usd", "HML_usd", "RMW_A_usd", "CMA_usd"],
    "Q"  : ["EIGA_usd", "ME_usd", "IA_usd", "ROE_usd"],
}


def fit_ols(y: pd.Series, X: pd.DataFrame):
    Xc = sm.add_constant(X)
    mask = y.notna() & Xc.notna().all(axis=1)
    y2, X2 = y[mask], Xc[mask]

    if len(y2) == 0 or X2.shape[0] <= X2.shape[1]:
        return None

    return sm.OLS(y2, X2).fit()


for label, fac_cols in model_specs.items():
    # ...
    for i in dec_nums:
        res_i = fit_ols(ts_dec[i], ts_dec[fac_cols])
        if res_i is not None:
            results_equal[i][label] = res_i
```

Finally, the regression results are passed to the `Formatter` utility in the `generate_output()` method, which generates human-readable summaries as well as LaTeX-formatted tables. Note, that we created the `Formatter` utility for better code readability. If you intend to extend our code base, we recommend to put any code for string and latex formatting into this utility file in order to maintain consistency.

```
results_equal, long_short_results_equal = self.analyze()

ff3_equal, ff5_equal, q_equal = Formatter.results_to_strings(results_equal)
long_short_equal = Formatter.ls_res_to_string(long_short_results_equal)

latex_ff3_equal = Formatter.generate_latex_table(results_equal, "FF3")
latex_ff5_equal = Formatter.generate_latex_table(results_equal, "FF5")
latex_q_equal = Formatter.generate_latex_table(results_equal, "Q")
latex_ls_equal = Formatter.generate_ls_latex_table(long_short_results_equal)

raw = {
    "ff3_equal.txt": ff3_equal,
    "ff5_equal.txt": ff5_equal,
    "q_equal.txt": q_equal,
    "long_short_equal.txt": long_short_equal,
}
blocks = [
    "\\subsection{Fama-French 3-Factor Model}\n" + latex_ff3_equal,
    "\\subsection{Fama-French 5-Factor Model}\n" + latex_ff5_equal,
    "\\subsection{Q-Factor Model}\n" + latex_q_equal,
    "\\subsection{Long-Short Analysis}\n" + latex_ls_equal,
]
return AnalyzerOutput(name=self.TITLE, raw_texts=raw, latex_blocks=blocks)
```

The analyzer class for the Fama–MacBeth regression works in a similar fashion with the major difference that it operates on individual stocks without portfolio formation and employs firm-level controls for the regression. For comparison, the main difference in the regression implementation is:

```
y = pd.to_numeric(g.get("ret"), errors="coerce")
X = g[["signal", "size_lag", "mom_2_12", "bm", "ag", "rd_sale"]]
        .apply(pd.to_numeric, errors="coerce")
X = sm.add_constant(X, has_constant="add")
# ...
res = sm.OLS(y, X).fit()
```

The `CorrelationAnalyzer` computes the correlation for each stock over the complete timeseries and then calculates the mean correlation value over all stocks.

```
# f is a factor proxy and rhos is an empty list
for _, g in df[['DSCD', 'signal', f]].dropna().groupby('DSCD', sort=False):
    # ...
    r = float(g['signal'].corr(g[f]))
        if not np.isnan(r):
            rhos.append(r)

mean_rho = float(np.mean(rhos)) if len(rhos) > 0 else np.nan
```

The factor model analysis by industry and country (`IndustryCountryVWLSAnalyzer`) basically applies the implementation of `ValueWeightedFactorModelAnalyzer` for the long-short analysis with filters on the respective industry or country, and reports the corresponding intercept values. On top of that, it also calculates the aggregated intercept results over all countries and industries based on the market cap value and equal weighting.

```
  def _wavg(g: pd.DataFrame) -> pd.Series:
      w = g["agg_w"].fillna(0.0).values
      # w is the value of mcap for mcap weighting and 1 for equal weighting
      out = {}

      for col in range(1, GROUP_NUM_SLICES+1):
          v = g[col].astype(float)
          m = v.notna() & np.isfinite(w)
          if m.any():
              out[col] = float(np.average(v[m], weights=w[m]))
          else:
              out[col] = np.nan

      return pd.Series(out)
```

In the `Analyzer` class, the enabled analyzers are then automatically discovered by the `discover_analyzers()` method and then executed one by one. Note, that when performing the different analyses, all datasets are only read once and then passed as a `SharedContext` object to the individual analyzers to increase performance:

```
  def run_complete_analysis(df: pd.DataFrame, signal_name: str):
      crsp_full = pd.read_csv("./data/dsws_crsp.csv")
      factors_full = pd.read_csv("./data/Factors.csv")
      fm_full = pd.read_csv("./data/Fama_Macbeth.csv")
      ctx = SharedContext(crsp=crsp_full, factors=factors_full, fm=fm_full)

      outputs = []
      for Plugin in discover_analyzers():
          plugin = Plugin(ctx, df, signal_name)
          out = plugin.generate_output()
          outputs.append(out)
```

The `discover_analyzers()` method finds the enabled analyzers by searching in the registry of `AutoRegistered`.

```
  candidates = []

  for cls in AutoRegistered._registry:
      if hasattr(cls, "ENABLED") and getattr(cls, "ENABLED"):
          candidates.append(cls)

  candidates.sort(key=lambda c: getattr(c, "ORDER", 100))
  return candidates
```

Subclasses of `AutoRegistered` are automatically registered by the `__init_subclass__` method which is executed at the time of the subclass definition.

```
  def __init_subclass__(cls, **kwargs):
      # ...
      AutoRegistered._registry.append(cls)
```

3.4 Usage

In order to setup and use this framework locally, follow the subsequent steps:

1. Open a terminal and clone the repository using:

   ```
   git clone https://github.com/Proenchen/stock-market-protocol.git
   ```

2. Navigate into the cloned `stock-market-protocol` folder.

3. Create a virtual environment:

   ```
   python -m venv venv
   ```

4. Activate the virtual environment:

   ```
   .\venv\Scripts\activate
   ```

5. Install all required dependencies using:

   ```
   pip install -r requirements.txt
   ```

6. Create a `data` folder within the `stock-market-protocol` folder and insert all data files. If necessary, adjust the paths to the data files in `logic/analysis.py` (l. 53-56):

   ```
   crsp_full = pd.read_csv("./data/dsws_crsp.csv")
   factors_full = pd.read_csv("./data/Factors.csv")
   fm_full = pd.read_csv("./data/Fama_Macbeth.csv")
   corr_full = pd.read_parquet("./data/Predictor_different.parquet")
   ```

7. To use the mailing service, create a `.env` file in `stock-market-protocol` folder with the following SMTP and email credentials:

   ```
   SMTP_SERVER=... // e.g. smtp.gmail.com
   SMTP_PORT=... // e.g. 587
   EMAIL_USER=... // e.g. globalstockmarketprotocol@gmail.com
   EMAIL_PASSWORD=... // e.g. abcd efgh ijkl mnop
   ```

   If you use Gmail as mail provider, `EMAIL_PASSWORD` has to be an app password which can be created by following these instructions[6].

8. Now everything is setup. Run the application by executing:

   ```
   python -u "app.py"
   ```

9. Open a browser and enter the URL:

   ```
   http://127.0.0.1:5000
   ```

If you encounter any problems while using this tool, feel free to create an issue in the Github repository[7].

[6] `https://support.google.com/accounts/answer/185833`
[7] `https://github.com/Proenchen/stock-market-protocol/issues`

3.5 Guide for extending our code base

Due to the implementation of the service discovery pattern, it is really easy to integrate new portfolio tests into our framework. In order to add a new analyzer, create a new class which inherits from both `BaseAnalyzer` and `AutoRegistered` and implement the abstract methods `analyze()` and `generate_output()`. Furthermore, provide values for the class constants `ENABLED`, `ORDER` and `TITLE`, e.g.:

```python
from logic.analyzers.base import BaseAnalyzer
from logic.analyzers.base import AutoRegistered, AnalyzerOutput

class NewAnalyzer(BaseAnalyzer, AutoRegistered):
    ENABLED = True
    ORDER = 10
    TITLE = "<Title in result document for this analyzer>"

    def __init__(self, ctx, df_input, signal_name):
        super().__init__(ctx, df_input, signal_name)

    def analyze(self):
        # Your test logic...

    def generate_output(self) -> AnalyzerOutput:
        # Use the result of analyze() to generate
        # the LaTex blocks and/or string output...
```

If you run the analysis, this class will be automatically discovered and the `generate_output()` method will be automatically executed. If you want to extend the datasets used for analysis, simply add the `.csv` files into the `data` package and extend the `SharedContext` class.

```python
@dataclass
class SharedContext:
    crsp: pd.DataFrame
    factors: pd.DataFrame
    fm: pd.DataFrame
    correlation_data: pd.DataFrame
    new_data: pd.DataFrame # NEW
```

Also, add the path to the new dataset in the `Analysis` class and set the values in the constructor of `BaseAnalyzer()` accordingly:

```python
class Analysis:
    @staticmethod
    def run_complete_analysis(df: pd.DataFrame, signal_name: str) -> str:
        crsp_full = pd.read_csv("./data/dsws_crsp.csv")
        factors_full = pd.read_csv("./data/Factors.csv")
        fm_full = pd.read_csv("./data/Fama_Macbeth.csv")
        corr_full = pd.read_parquet("./data/Predictor_different.parquet")
        new_data = pd.read_csv("./data/New_Data.csv")    # NEW
        ctx = SharedContext(
            crsp=crsp_full, factors=factors_full, fm=fm_full,
            correlation_data=corr_full, new_data=new_data)
```

```python
class BaseAnalyzer(ABC):
    ENABLED: bool
    ORDER: int
    TITLE: str

    def __init__(self, ctx, df_input, signal_name) -> None:
        self.data = df_input
        self.crsp = ctx.crsp
        self.factors = ctx.factors
        self.fm = ctx.fm
        self.corr = ctx.correlation_data
        self.new_data = ctx.new_data    # NEW
        self.signal_name = signal_name
```

You can always take the existing analyzer implementations as a reference.

4 Conclusion and Future Work

In this work, we introduced a modular, open-access framework for the replication and evaluation of stock market anomalies in a global context. Unlike many existing platforms that remain confined to U.S. equity markets, our tool enables anomaly testing across diverse countries and industries, thereby broadening the scope of empirical asset pricing research. The framework incorporates widely recognized methodologies, including the Fama–French Three and Five Factor Models, the Q–Factor Model, and the Fama–MacBeth Regression. By integrating these models into an accessible, web-based interface, we aimed to lower the technical entry barrier for researchers and practitioners, while also ensuring transparency and reproducibility through an openly available code base. Our design choices emphasized modularity and extensibility. The adoption of a layered architecture and the service discovery pattern allows for seamless integration of new analyzers, datasets, or protocols without disrupting existing functionality. Furthermore, the automated generation of LaTeX reports and delivery via email provides users with a convenient and professional workflow, bridging the gap between empirical research and practical usability.

Despite these contributions, there remain several possibilities for future development. First, the current implementation relies primarily on CSV-based storage. The migration to a dedicated database system could substantially improve scalability and efficiency. Second, while our framework already supports major factor models, additional portfolio tests could further enhance its analytical capabilities. Third, extending the system with real-time data ingestion and continuous monitoring would increase its practical relevance for portfolio management and trading strategies. Finally, incorporating advanced visualization features and interactive dashboards could improve user experience and support more exploratory modes of analysis.

References

[1] Eugene F. Fama and Kenneth R. French. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1):3–56, 1993.

[2] Sheridan Titman and Narasimhan Jegadeesh. Returns to buying winners and selling losers: Implications for stock market efficiency. *Journal of Finance*, 48:65–91, 02 1993.

[3] Deniz Anginer, Sugata Ray, H. Nejat Seyhun, and Luqi Xu. Expensive anomalies. *Journal of Empirical Finance*, 75:101440, 2024.

[4] John H. Cochrane. Presidential address: Discount rates. *The Journal of Finance*, 66(4):1047–1108, 2011.

[5] Campbell R. Harvey, Yan Liu, and Heqing Zhu. ... and the cross-section of expected returns. *The Review of Financial Studies*, 29(1):5–68, 10 2015.

[6] Kewei Hou, Chen Xue, and Lu Zhang. Replicating anomalies. *The Review of Financial Studies*, 33(5):2019–2133, 12 2018.

[7] Robert Novy-Marx and Mihail Velikov. Assaying anomalies. *SSRN Electronic Journal*, 01 2023.

[8] G. Andrew Karolyi. Home bias, an academic puzzle. *Review of Finance*, 20(6):2049–2078, 03 2016.

[9] Eugene F. Fama and Kenneth R. French. A five-factor asset pricing model. *Journal of Financial Economics*, 116(1):1–22, 2015.

[10] Kewei Hou, Chen Xue, and Lu Zhang. Digesting anomalies: An investment approach. *The Review of Financial Studies*, 28(3):650–705, 09 2014.

[11] Eugene F. Fama and James D. MacBeth. Risk, return, and equilibrium: Empirical tests. *The Journal of political economy*, 81(3):607–636, 1973.

[12] William F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance*, 19(3):425–442, 1964.

[13] Rolf W. Banz. The relationship between return and market value of common stocks. *Journal of Financial Economics*, 9(1):3–18, 1981.

[14] Asmâa Alaoui Taib and Safae Benfeddoul. The empirical explanatory power of capm and the fama and french three-five factor models in the moroccan stock exchange. *International Journal of Financial Studies*, 11(1), 2023.

[15] Kilsgård, David and Wittorf, Filip. The Fama and French Three-Factor Model - Evidence from the Swedish Stock Market, 2010. Student Paper.

[16] Karl Pearson and Olaus Magnus Friedrich Erdmann Henrici. Vii. mathematical contributions to the theory of evolution.& iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 187:253–318, 1896.