Kyiv School of Economics

ZEvA

Computer Science Department

Bachelor's Thesis

SpectraCast Suite

Software toolkit for automating the imputation, analysis, and visualization of time series in economic analytics

Matvii Talalaievskyi

Supervisor
KSE, Vadym Yaremenko, Lead Software Engineer, vyaremenko@kse.org.ua

Expert
KSE, Dmytro Krukovets, PhD, dkrukovets@kse.org.ua

Submission date
15 June 2026

Academic Integrity Statement

I, undersigned, hereby declare that this capstone project is the result of my own work.

- All ideas, data, figures and text from other authors have been clearly cited and listed in the bibliography.
- No part of this project has been submitted previously for academic credit in this or any other institution.
- All code, diagrams, and third-party materials are either my original work or are used with permission and properly referenced.
- I have not engaged in plagiarism or any form of academic dishonesty.
- Any assistance received (e.g. from peers, tutors, or online forums) is acknowledged in the acknowledgements section.

I understand that failure to comply with these declarations constitutes academic misconduct and may lead to disciplinary action.

Place, date	Kyiv, 15.06.2026
Signature	_____

Contents

Acknowledgements

I spent a long time thinking about how to begin this text. There are so many words I want to say and emotions I want to express, yet they still don't seem to form a complete picture. But now I realize that this uncertainty perfectly describes my journey at KSE. It was sometimes strange, sometimes exciting, sometimes exhausting, but always interesting.

To be honest, many of my relatives and close friends were against me enrolling at KSE. There were plenty of reasons: it is a private university, the degree wasn't accredited back then, and the bachelor's program had only recently launched. However, I stood by my choice. That was the first lesson this university taught me—to take responsibility for your decisions and shape your own destiny.

Over these four years, I have met many people at the university. Looking back, I want to thank every single one of them. No matter who you are or where we stand now, each of you has made me a better person, as every interaction carries lasting value. Of course, my deepest gratitude goes to my family and friends. Mom, Dad, and my entire family—I don't think they will read this, but this recognition belongs here because they will always worry about me and celebrate my every achievement. To my friends—thank you for putting up with my antics and supporting me through the toughest moments (and a special shoutout to my university friends for being the best teammates in group projects).

And, of course, thank you KSE for introducing me to the Institute. It will always hold a special place in my heart, as it became the wonderful setting for my first real job. And. most important for me, it was the first place where I found true friends.

Finally, I want to express my gratitude to my thesis supervisors—Dmytro Krukovets and Vadym Yaremenko. This project would not exist without them. Thank you for every edit and every comment you made; this work is truly a result of your guidance.

It has been a fascinating yet incredibly challenging journey. Part of it is something I might still be processing, as it coincided with the toughest times for both myself and our country. There is much I will reflect on for a long time to come. But I feel happy. In the end, I think that is what matters most.

Abstract

While the era of big data presents many opportunities for economic analytics, professionals face a major roadblock. Much of their time goes to routine dataset preparation instead of generating insights. This project documents the development of SpectraCast Suite, a web application that fills this gap by automating the imputation, analysis, and visualization of macroeconomic time series.

Given the lack of a unified standard for detecting anomalies and the trade-off between computing power and data privacy, an opportunity arose to create a secure, centralized analytical ecosystem. This project aimed to develop a fully functional toolkit with: (1) an automated Data Quality module for strong outlier detection and missing value imputation, (2) a Leading Indicators module that uses LLMs and search trends to generate predictors, (3) a Visual Standardizer for automated chart generation and code sanitization, and (4) a hybrid computing architecture that runs Python scripts locally in the browser to ensure the confidentiality of sensitive economic data.

To reach these goals, we conducted a thorough analysis of existing analytical tools to find market gaps and set system requirements. The development followed an Agile methodology with three iterative sprints. We used a modular monolith architecture with FastAPI and Python for the backend, integrating SQLAlchemy, Pydantic, and specialized statistical libraries. The frontend was built with a React and TypeScript application, incorporating a Pyodide Web Worker environment for local browser computations. The system was deployed on AWS cloud infrastructure, using external integrations like the Gemini API model and SerpAPI, along with Bring Your Own Key (BYOK) technology.

As a result, the project delivered a production-ready Minimum Viable Product (MVP) that automates the most time-consuming stages of economic data preparation. Validation procedures, including RMSE evaluation of mathematical algorithms and Locust load testing, showed high predictive accuracy and strong backend performance, with 95% of requests processed in under 200 ms. Additionally, testing confirmed the absolute privacy of the local execution mode, ensuring no data transmission to the server during isolated analysis. Ultimately, SpectraCast Suite illustrates the effective use of software engineering in economic research. It creates a secure and efficient alternative to disorganized manual workflows while laying the groundwork for improved analytical tools.

Keywords:

KSE, *Software Engineering*, *SpectraCast Suite*, *Economic Analytics*, *Time Series Processing*, *Data Imputation*, *Hybrid Architecture*, *Local Computing*, *Python Web Application*, *Automated Diagnostics*

1 Introduction

1.1 Initial Situation Presentation of the Topic

In the era of big data, economic analysts face a paradoxical bottleneck: the significant share of their time is spent not on generating insights, but on the technical preparation of datasets. As data volumes scale up, the absolute number of structural inconsistencies and complex edge cases inevitably grows, requiring advanced automated preprocessing before any reliable analysis can be conducted. Modern research suffers from an excessive amount of time spent on routine operations. Industry surveys and academic studies consistently highlight that analysts spend up to 80% of their workflow on technical dataset preparation—such as cleaning time series, diagnosing outliers, and handling missing values—rather than on generating insights. This process is further complicated by limited or restricted access to government and commercial databases, forcing professionals to expend resources on the manual aggregation of disparate information. Despite the availability of complex statistical models, the foundational stages of data cleaning and the subsequent configuration of visualizations remain fragmented and are predominantly performed manually. This leads to the low efficiency of routine processes and a high risk of human error.

1.2 Relevance of the topic & motivation

Solving this problem has high scientific and practical significance. Currently, there is no unified standard in the analytics field for diagnosing and eliminating anomalies in time series. Due to the lack of a comprehensive and well-grounded strategy, specialists are often forced to rely solely on their own intuition. This approach carries the risk of critically distorting analytical conclusions during hypothesis testing. Furthermore, the manual search and selection of leading indicators for predictive models is an extremely complex and time-consuming process, which significantly reduces the efficiency of feature engineering. Automating these processes is a fundamental condition for ensuring the reliability, accuracy, and reproducibility of economic analysis.

The choice of this topic is also closely related to my personal motivation and practical experience. During my regular work with macroeconomic datasets, I consistently encountered the problem of anomalies and a significant number of missing values. Their manual detection and cleaning consumed a disproportionately large amount of time that could have been spent directly on generating insights. Moreover, the frequent limitations of accessible open data required a constant, exhausting search for additional information across disparate sources. It was precisely this practical encounter with the “bottlenecks” of manual analysis and the desire to optimize my own work that became the main driving force behind developing a system capable of automating these routine operations.

1.3 Objectives

The main objective of this bachelor’s thesis is the design and development of SpectraCast Suite, a comprehensive web application aimed at enhancing the efficiency of economic

analysts. The primary task is to minimize human error and accelerate the analytical cycle through automation. The developed system will consist of three integrated modules: an automated data quality controller (Data Quality), a search system for leading indicators (Leading Indicators) based on Google Trends and LLMs, and a graphical reporting standardization module (Visual Standardizer), and designing a secure hybrid computing architecture that guarantees the privacy of sensitive economic data through zero-payload local processing.

1.4 Development cycle approach

The development approach for SpectraCast Suite was based on a structured methodology combining comprehensive domain research with Agile development practices:

1. **Discovery Phase:** An in-depth analysis of the specifics of working with macroeconomic data was conducted, existing analytical tools on the market were evaluated, shortcomings of manual data cleaning processes were identified, and key functional and architectural requirements for the system were defined.
2. **Iterative Development:** The system implementation process was divided into three consecutive sprints, each with defined goals, architectural tasks, and deliverables:
   - **Sprint 1: Prototyping analytical modules and CLI development.** At the initial stage, the system's foundation was laid, and the first functional prototypes of two key application components were created: Data Quality and Visual Standardizer. A CLI was developed for the isolated testing of mathematical algorithms, econometric methods for time series cleaning, and verification of anomaly detection logic.
   - **Sprint 2: Integration with external APIs and methodological testing.** The second development cycle focused on expanding module functionality and establishing external connections. Integration with the Google Gemini API was implemented, and the PyTrends library was used for the automated collection of statistical data from the Google Trends platform. Simultaneously, rigorous stress testing of the algorithms embedded within the analytical modules was conducted: verifying the accuracy of structural break diagnostics, the correctness of missing value imputation, and the stability of outlier processing on real datasets.
   - **Sprint 3: Infrastructure deployment, local compute engine creation, and final release.** The final stage was dedicated to infrastructure tasks and ensuring a hybrid architecture. A complete deployment of the cloud environment on AWS was executed: configuring an EC2 server instance with an Nginx + Uvicorn + FastAPI stack, an RDS PostgreSQL database, and Amazon S3 for artifact storage. The key engineering task of this sprint was implementing an autonomous local computing mode in the browser by integrating Web Workers and the Pyodide environment, enabling the execution of analytical Python code directly on the client side. Furthermore, the PyTrends library was replaced with SerpAPI to improve the stability of the Leading Indicators module.
3. **Technology Selection:** The technology stack was selected based on practical feasibility, existing development experience, and the optimal compatibility of the chosen tools for implementing the project's architecture. React with Vite is used to create an interactive client interface, while the Pyodide environment is used to execute Python scripts

in the browser. The server side is built on FastAPI (Python). The cloud infrastructure is deployed on the AWS platform (Amplify, EC2, RDS, S3), ensuring scalability and cost-efficiency, while the Google Gemini API and SerpAPI are integrated for advanced analytics.

1.5 Structure of the Bachelor's thesis

The thesis sequentially outlines the theoretical foundations, design, and practical implementation of SpectraCast Suite. The first chapter analyzes the current state of economic data processing and examines existing market solutions, establishing the problem statement. The second chapter details the architectural design and methodological approach to developing the three key modules: Data Quality, Leading Indicators, and Visual Standardizer. The third chapter is dedicated to the technical implementation, describing the integration of the frontend with the backend and the deployment infrastructure. The fourth chapter contains the validation of the system's performance on real macroeconomic datasets, followed by general conclusions and prospects for the further development of the project.

2 Research

2.1 Research Questions and Functional Requirements

2.1.1 Research Questions

To design SpectraCast Suite, the following research questions must be answered:

- What tools are currently used by economic analysts for data preparation and enrichment?
- How can the time spent on the routine cleaning of macroeconomic time series from anomalies and the imputation of missing values be effectively reduced?
- What architectural approaches exist to ensure confidentiality when processing sensitive statistical information without compromising the convenience of web access?
- What strategies do economic analysts employ to formulate and test hypotheses under conditions of limited access to statistical data?
- How can the proposed software solution integrate the disparate stages of the analytical cycle and outperform existing market alternatives in terms of efficiency?

2.1.2 Functional Requirements

- Automated diagnostics, cleaning, and enrichment of time series.
- Interactive management of the analytical process (the user's ability to confirm or modify strategies for resolving anomalies and imputing missing values).
- Predictor generation using LLMs and search trend analysis.
- Hybrid computing architecture with the capability of local script execution to guarantee confidentiality.
- Standardized and rapid data visualization with support for various style configurations.
- Seamless import and export of results (uploading input datasets, downloading cleaned arrays, generated code, and finished charts).

2.2 Overview of existing solutions and their classification

The analytical software market offers a range of solutions that partially meet the needs of economists.

Tool	Main Focus & Strengths	Weaknesses in Economic Data Analysis	Cost & Accessibility
1. Data Preparation (Data Quality)			
Tableau Prep / Power Query (PowerBI, Excel)	Powerful built-in tools for basic cleaning, dataset transformation, and table merging	Require manual configuration of cleaning rules. Lack automated mathematical algorithms for de-	Excel: Part of MS Office (from $6/month). Power BI: Desktop version is free, Pro license from $10/month.

Tool	Main Focus & Strengths	Weaknesses in Economic Data Analysis	Cost & Accessibility
	via a graphical interface.	tecting econometric anomalies in time series. No recommendations for the optimal strategy.	Tableau: Commercial product ($15 to $75/month), has a limited public version.
Classic econometric packages (EViews, Stata)	Industry standard for time series analysis. Contain all necessary built-in econometric tests and models, accessible via a graphical interface without writing code.	Outdated desktop architecture. Lack of web access.	Very high cost for commercial and academic licenses, although limited student versions exist.
Macroeconomic terminals (Macrobond / Bloomberg Terminal)	Industry standard. Provide access to millions of global time series, feature built-in forecasting and correlation search functions.	Closed ecosystems. Do not allow easy integration of custom Python scripts.	High. Accessible only to large corporations.
ChatGPT Advanced Data Analysis	Powerful LLM assistant. Capable of independently writing and executing Python code for cleaning, analyzing, and visualizing any uploaded datasets via a conversational interface.	Requires full dataset upload to OpenAI servers. Prone to "hallucinations" when selecting econometric methods. Has limits on uploading large files.	Freemium. Basic functionality is available for free, but the model may be downgraded to a "weaker" one. Continuous operation requires a commercial subscription (from $7/month).
YData Profiling	Automatic generation of data health reports	Functions solely as a static report generator. Lacks error	Open-source.

Tool	Main Focus & Strengths	Weaknesses in Economic Data Analysis	Cost & Accessibility
	(Data Health). Implemented as a specialized package for the Python environment (open-source solution).	correction scenarios and an interactive UI for resolving them.	
Zoho DataPrep	Powerful data cleaning using a pipeline approach and automatic suggestions.	Focuses on business data (CRM, sales). Lacks specialized econometric methods.	Commercial tool (from $15/month, limited free tier available).
OpenRefine	Local data cleaning tool with powerful features for handling textual errors and categories.	High barrier to entry. Complicated to use due to the lack of an intuitive interface.	Open-source.
Mito	Interactive tool (integrated directly into Jupyter Notebook as a Python library) that combines Excel functionality with basic AI code generation.	Lacks specific econometric diagnostic methods (e.g., for structural breaks).	Freemium. Free for basic local use; paid Pro/Enterprise licenses for advanced features.
Basic Python and R libraries (Pandas, SciPy, statsmodels, tidyverse)	Maximum analysis flexibility. Contain a full range of mathematical and econometric methods for processing data of any complexity.	High labor intensity: writing and maintaining code for every routine task takes significant time, distracting from direct analysis. Lack of a graphical interface for rapid visual validation	Open-source.

Tool	Main Focus & Strengths	Weaknesses in Economic Data Analysis	Cost & Accessibility
		of implemented changes.	
2. Factor Search (Leading Indicators)			
SparkBeyond / Zoho Zia	Enterprise solutions. Utilize predictive AI to automatically search for external factors and generate hypotheses.	Completely closed commercial products. Inaccessible to a broad range of researchers, require data transmission to the cloud.	SparkBeyond: Extremely high cost (Enterprise licenses, thousands of dollars). Zoho Zia: Available only within paid Zoho plans ($12–$35/month).
Julius AI	Uses an LLM (chatbot format) for data interpretation and insight discovery.	Focuses exclusively on analyzing user-uploaded data, rather than automatically generating external macroeconomic factors from an API.	Freemium. Limited free monthly queries; premium subscription starts at $20/month.
Google Trends	Basic tool for analyzing the dynamics of search trends over time.	Provides only "raw" data. Requires manual query entry and does not verify statistical correlation with the target variable.	Open-source. Closed API access (unless using specialized paid solutions).
3. Visualization Standardization (Visual Standardizer)			
Tableau / PowerBI / Excel	Market leaders for dashboard creation. Flexibility in building charts for any data.	Require building charts "from scratch". Cannot function as a linter/standardizer for existing Python	Excel: Part of MS Office (from $6/month). Power BI: Desktop version is free, Pro license from $10/month. Tableau: Commer-

Tool	Main Focus & Strengths	Weaknesses in Economic Data Analysis	Cost & Accessibility
		code (e.g., matplotlib).	cial product ($15 to $75/month), has a limited public version.
Datawrapper	Focuses on clean design, font readability, and avoiding visual noise.	Builds visualizations from scratch, completely lacks integration with the analyst's code. Cannot function as a linter/standardizer for existing Python code.	Freemium. Extensive free functionality for standard charts; corporate plans with brand customization are very expensive (from $599/month).

Table 1: Comparative Analysis of Existing Analytical Tools

2.3 Comparative analysis of architectural approaches

Modern data analysis tools use several main architectural approaches, each with important trade-offs between computational power, privacy, and usability.

- Client-server cloud architecture involves performing all resource-intensive computations and data storage on remote servers. This provides compatibility across different platforms and removes the need for user hardware. However, this method is not suitable for handling confidential data because of the risks of information leakage during transmission and storage on third-party servers.
- Local desktop architecture processes datasets entirely on the user's computer. This ensures privacy but creates the issue of an isolated environment. For example, using basic Python libraries requires the researcher to manually set up dependencies, interpreters, and packages. Additionally, desktop solutions do not offer the flexibility of web technologies for quickly integrating external data enrichment tools, such as third-party APIs, which need more setup.

2.4 Identified gaps and rationale for development

The analysis and comparison of existing tools revealed gaps that justify the creation of SpectraCast Suite:

- **Closed ecosystems and high financial barriers:** Leading analytical platforms (such as SparkBeyond or specialized econometric software packages like EViews) have proprietary, closed architectures and extremely high licensing costs. This makes them financially inaccessible to many independent researchers and think tanks.

- **The dilemma between labor-intensive coding and the limitations of widespread products:** Analysts face a choice: either expend significant resources writing and maintaining code from scratch (using Pandas or SciPy), or use tools that lack built-in econometric algorithms for specific time series cleaning (e.g., resolving structural breaks or complex outlier filtering). To perform such tasks in these systems, an analyst still has to write complex custom functions or integrate external Python/R scripts, bringing them back to the problem of routine programming.
- **Unjustified choice of data cleaning strategies:** Due to the absence of automated recommendation algorithms, the selection of a data cleaning method is often methodologically unfounded. Decisions are made based on the analyst's subjective assumptions rather than the mathematical and contextual characteristics of the specific dataset.
- **Fragmentation of the analytical pipeline:** There is no single comprehensive solution on the market that seamlessly integrates econometric data preparation, predictor generation based on external trends, and data visualization. Researchers are forced to constantly switch between different environments.
- **A strict trade-off between data privacy and the infrastructure barrier:** Powerful commercial platforms for automated factor discovery or cloud-based AI tools require data to be transmitted to the cloud, posing a threat to confidential information. Conversely, local tools require the analyst to have programming skills and manual environment configuration.

3 Design

3.1 Architecture Overview and Requirements Alignment

The architecture of the SpectraCast Suite is designed as a hybrid due to two key requirements: data confidentiality and full automation of analytical procedures. Economic time series data are often sensitive; therefore, critical processing stages can be executed locally in the browser via Pyodide, without transmitting data to the server. Automating diagnostics, cleaning, and enrichment requires server-side modules that utilize standardized pipelines, results storage, and access to external APIs.

Functional requirements are aligned with the architecture as follows:

1. **Automated diagnostics, cleaning, and enrichment** of time series are implemented through the server-side Data Quality and Leading Indicators modules.
2. **Interactive management of the analysis process** is ensured by mechanisms for selecting imputation/anomaly processing strategies and rollback capabilities.
3. **Predictor generation via LLMs and trend search** are implemented in the Leading Indicators module using a BYOK (Bring Your Own Key) model (or without it, using a pre-installed key).
4. **Hybrid computing** provides a local script execution mode to guarantee confidentiality.
5. **Visualization** is handled by the Visual Standardizer using various styles.
6. **Import/export of results** is organized through managed uploads and the retrieval of CSVs/charts.

3.2 Technology Stack Selection and Justification

Technology	Purpose	Rationale
Frontend		
React + TypeScript + Vite	Building the SPA interface	Mature ecosystem for complex UIs, TypeScript adds static typing to reduce errors, Vite ensures fast development and lightweight bundling.
Zustand	Local state management	Lightweight, simple API, saves development time.
Tailwind CSS	Unified UI styling	Utility-first framework for fast and consistent styling, eases maintenance of themes/style configurations.
Pyodide (Web Worker)	Local Python computations in the browser	Enables executing pandas/numpy locally, ensuring raw data never leaves the client environment—key for absolute privacy requirements.
Backend		

Technology	Purpose	Rationale
FastAPI	Server API (DQ, LI, VS, LLM Proxy, Upload)	High performance, simple route definition, and built-in integration with Pydantic for validation; rapid prototyping with minimal overhead.
Pydantic	Validation and serialization schemas	Strict validation of incoming requests/responses, enhancing the reliability of server pipelines and enabling the formation of correct API contracts.
SQLAlchemy	Relational DB access	ORM for RDS — allows working with transactions and domain data models, convenient for developing the metadata model (users, files).
pandas / numpy / scipy / scikit-learn / statsmodels	Scientific and statistical computations	Responsible for diagnostics, imputation, correlation analysis, and modeling; standardized libraries with rich functionality.
boto3	S3 integration	Reliable interaction with AWS object storage, necessary for saving large CSVs/charts without overloading the database.
SerpAPI	Google Trends time series retrieval	Provides programmatic access to trend data used for predictor generation and correlation analysis.
HTTPX	Asynchronous HTTP requests	Used for LLM proxy and external calls; asynchronous nature integrates with FastAPI for non-blocking operations.
Infrastructure		
AWS Amplify	Static frontend hosting	Rapid SPA deployment with CDN, simple HTTPS configuration, and automated releases — well-suited for an MVP frontend.
EC2 (without containerization)	Hosting the server-side FastAPI	Deployment simplicity and direct access to instance resources for heavy computations; minimizes operational complexity at the MVP stage.
RDS (Postgres)	Metadata and user storage	Managed relational DB for secure storage of accounts and file links.
S3	Storing input and result files	Scalable storage for CSV/PNG/artifacts, decoupled from the transactional DB.

Table 2: Technology Stack Selection and Justification

3.3 Architectural Decisions, System Architecture and Major Decisions

SpectraCast Suite is implemented as a modular monolith, where all business modules are hosted within a single FastAPI backend application. This approach maintains a cohesive data model, simplifies request tracing, and reduces operational overhead during the MVP stage.

The architecture has a hybrid computational nature: some tasks (at the user's discretion) can be executed locally in the browser, while the server side provides centralized pipelines, artifact storage, integration with external APIs, and user data management.

3.3.1 Core Connections and External Dependencies

- **Backend API RDS (PostgreSQL):** The database is used to store users and dataset metadata. Access is implemented via SQLAlchemy. This ensures transactional consistency and data access control.
- **Backend API S3:** Object storage is used to save input CSV data, cleaned datasets, analysis results, and visualizations. Decoupling this from the database increases reliability and scales file operations.
- **Backend API External APIs (LLM, SerpAPI):** The system integrates with LLM providers for predictor generation and with SerpAPI for retrieving trends. Access to the LLM is implemented through a proxy layer using a BYOK (Bring Your Own Key) model, which minimizes the risk of storing keys on the server. This allows the user to choose the most suitable option: using their own key or using the application's preselected LLM.
- **Frontend SPA Backend API:** Communication occurs via REST and SSE (Server-Sent Events) for long-running operations.

3.3.2 Key Architectural Decisions and Rationale

Decision	Rationale
Modular monolith FastAPI on EC2 (MVP)	Minimization of operational complexity, rapid deployment, centralized tracing, and state management without the overhead of microservices.
Pyodide in the browser as a local computational layer	Ensuring absolute confidentiality: raw data can be processed locally without transmission to the server.
BYOK model for LLM (keys in headers)	Reduction of API costs and elimination of the need to store keys on the server, enhancing security.
S3 for object storage + RDS for metadata	Decoupling of file artifacts and transactional data, increasing reliability, scalability, and manageability.

Table 3: Key Architectural Decisions and Rationale

3.4 Data Flow

- **Scenario A — Local Mode:** CSV → Browser → Pyodide Web Worker → UI. Data is read in the browser, converted into pandas objects locally, and the results are returned to the interface without being transmitted to the server.
- **Scenario B — Cloud Mode:** CSV → Backend API → S3/Memory → Output. The CSV is uploaded via the API, stored in S3, processed by server modules, and the results are saved as CSV/PNG files and returned to the client.

All requests and responses undergo schema validation via Pydantic, ensuring formal data correctness and type control.

3.5 Deployment Topology

The deployment of SpectraCast Suite is implemented in the AWS cloud environment with a clear separation of the static and server tiers. The frontend is published via AWS Amplify as a static SPA. The Backend API is deployed on an EC2 instance as a system process (Nginx + Uvicorn + FastAPI) within a single Virtual Private Cloud. Since the MVP version does not anticipate extreme loads, the system operates without an Elastic Load Balancer, and all requests are routed directly to the Nginx proxy. To ensure fault tolerance and data preservation, AWS managed services are utilized: daily automated database snapshots with a 7-day retention period are configured for Amazon RDS, and object versioning can be enabled for Amazon S3 to prevent the accidental deletion of client datasets.

3.6 C4 Architecture Diagrams

3.6.1 C4: System Context Diagram

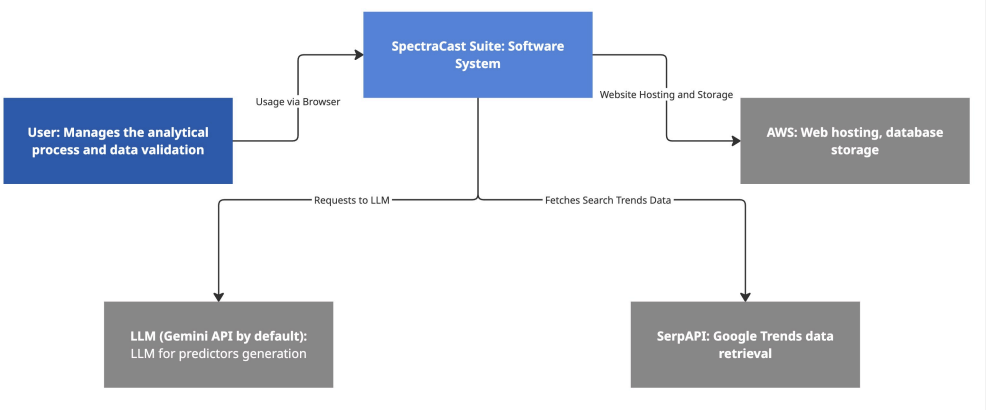

Figure 1: C4: System Context Diagram

The context diagram illustrates the SpectraCast Suite at the highest level of abstraction, showing its interaction with the user and external providers (AWS, LLM (Gemini), SerpAPI).

3.6.2 C4: Container Diagram

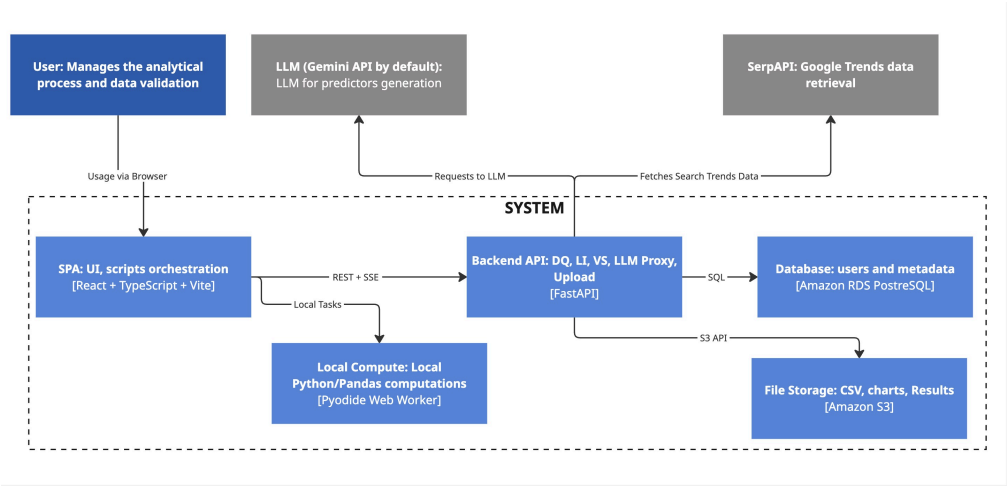

Figure 2: C4: Container Diagram

The system's container diagram is decomposed into independently deployable components: the client SPA, the local Pyodide environment, the cloud Backend API, and the data stores, indicating the communication protocols.

3.6.3 C4: Component Diagram

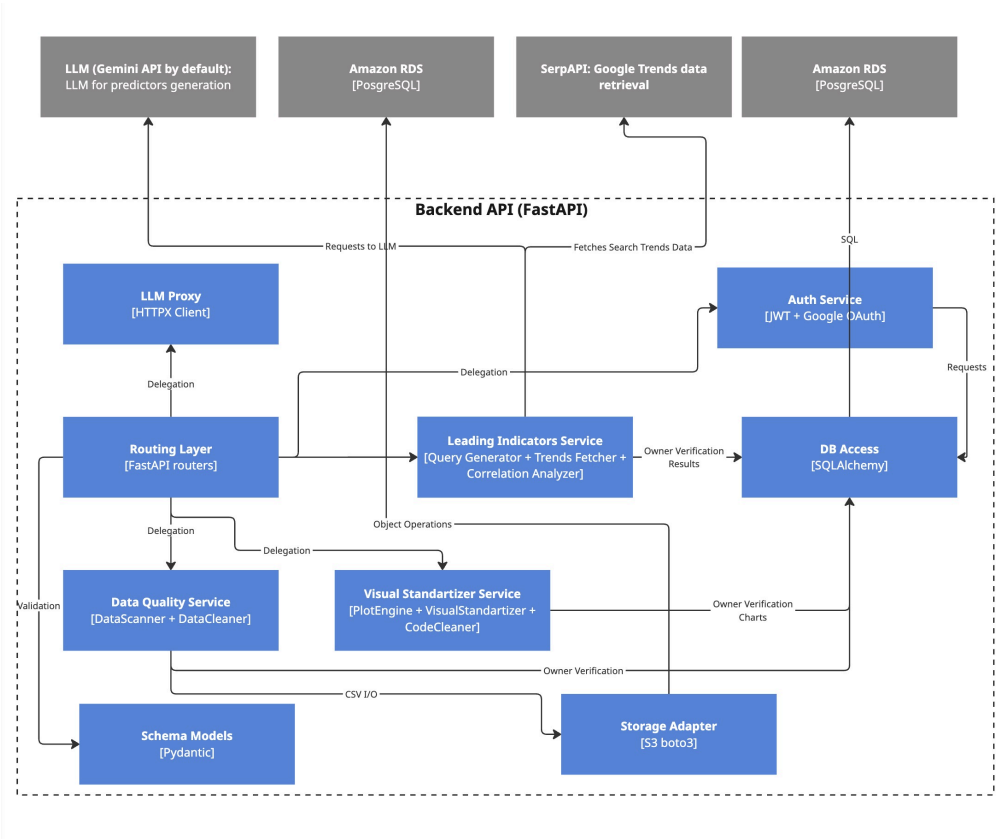

Figure 3: C4: Component Diagram

The component diagram details the internal structure of the Backend API, illustrating the delegation of tasks from the routing layer to specific services (Data Quality, Leading Indicators) and data access modules.

3.7 Cross-cutting Concerns

Security: All client-server communications occur exclusively via the encrypted HTTPS protocol. A JWT token mechanism combined with Google OAuth is used for user authentication and access to cloud functions. Protection against the leakage of external LLM keys is guaranteed by the BYOK model, where the API key is transmitted in the HTTP header and stored only in the server's RAM for the duration of the request execution.

Logging & Monitoring: At the MVP stage, infrastructure monitoring is conducted using basic AWS CloudWatch metrics (CPU load, EC2 instance network traffic, and RDS metrics). Application-level logging is implemented using the built-in capabilities of FastAPI and Uvicorn, where all critical errors and system events are recorded in standard OS system logs (systemd/journalctl).

Scalability: Since the current architecture is a monolith deployed on a single EC2 instance (Bare Metal), the primary strategy is vertical scaling—increasing the virtual machine's computational resources. Storing files on S3 and metadata on RDS relieves the load on the server's file system, establishing the foundation for future horizontal scaling (such as adding a Load Balancer).

4 Implementation

This chapter is devoted to the practical implementation of the developed architecture and the translation of theoretical algorithms into working code for the SpectraCast Suite system. Given the platform's complex hybrid nature, this chapter focuses not on trivial routing operations, but on the implementation of the system's most critical and technically challenging components.

4.1 Data Quality Module Operation

DQ handles two scenarios:

- Data quality audit: detecting missing values, time gaps, outliers, and generating recommendations.
- Cleaning/normalization: applying selected imputation and outlier processing strategies.

This provides a structured data health report, automatically selects correct strategies based on statistical metrics, and applies them within a single module, ensuring consistent results across the UI, API, and batch processing.

4.1.1 Step-by-Step Algorithm

The Data Quality module performs automated analysis and cleaning of time series in several sequential stages:

1. **Time frequency identification:** The module detects the presence of a valid time scale or date column, parses it, and creates a DatetimeIndex.
2. **Frequency determination and gap detection:** The dominant time step between dates is identified, an ideal time scale is generated, and missing intervals are calculated.
3. **Missing values diagnostics:** For each numeric column, the number of missing values is calculated, and an imputation strategy is recommended.
4. **Anomaly (outlier) diagnostics:** A robust analysis of extreme values is conducted, emphasizing the IQR method.
5. **Recommendation generation:** Seasonality, autocorrelation, volatility, and cross-series correlations are factored in.
6. **Strategy application:** The selected or recommended strategies are applied within a single module, ensuring consistent results.

4.1.2 Determining Time Frequency and Gaps

```
sorted_index = datetime_index.sort_values().drop_duplicates()
deltas = sorted_index.to_series().diff().dropna()

dominant_delta = deltas.mode().iloc[0]
dominant_delta_days = dominant_delta.days
```

Based on the dominant difference between dates, mapping to standard frequencies (day, week, month, quarter) is performed. Next, an ideal calendar is generated:

```
ideal_range = pd.date_range(
   start=sorted_index.min(),
   end=sorted_index.max(),
   freq=freq_str
)
missing_dates = ideal_range.difference(sorted_index)
```

The dominant step is determined as the mode of the differences between dates, after which a regular scale is built.

4.1.3 Missing Value Imputation Strategies

```
if method == '1':
   self.df[column] = self.df[column].interpolate(method='linear')
elif method == '2':
   self.df[column] = self.df[column].interpolate(method='spline', order=3)
elif method == '3':
   self.df[column] = self.df[column].ffill()
elif method == '4':
   (...)
   self.df[column] = self.df.groupby(season_key)[column].transform(lambda x:
x.fillna(x.mean()))
elif method == '5':
   imputer = KNNImputer(n_neighbors=5)
   imputed_matrix = imputer.fit_transform(self.df[numeric_cols])
   self.df[column] = imputed_matrix[:, col_idx]
```

4.1.3.1 Explanation of strategy selection

- **Linear interpolation:** The baseline strategy for smooth series; it minimally distorts the trend and does not introduce sharp artifacts.
- **Spline interpolation:** Provides a smooth curve and better reflects non-linear fluctuations; useful for cyclical or "curved" trajectories.
- **Forward fill:** Preserves the last known value; particularly useful in financial series where interpolation might create artificial changes.
- **Seasonal mean:** Applied when there is pronounced seasonality; preserves the cyclical structure.
- **KNN imputation:** Utilizes correlated columns, which increases accuracy in multidimensional datasets.

Additionally, the user can select the "Do Nothing" strategy.

4.1.4 Outlier Detection and Processing Using IQR

```
q1 = series.quantile(0.25)
q3 = series.quantile(0.75)
iqr = q3 - q1
outlier_mask = (series < q1 - 1.5 * iqr) | (series > q3 + 1.5 * iqr)
```

Initially, the Z-Score was used to detect outliers in the program. This was later replaced by the IQR method because the Z-score strictly assumes a normal (Gaussian) distribution.

Extensive empirical research demonstrates that economic and financial time series rarely follow a normal distribution; instead, they consistently exhibit “heavy tails” (excess kurtosis) and non-Gaussian behavior[1].

The 1.5 multiplier in IQR was chosen over 3.0 because it is statistically equivalent to a 2.7 standard deviation boundary (capturing over 99% of normal variance), whereas a 3.0 multiplier pushes this limit to an extreme 4.7σ. In macroeconomic datasets, applying a 4.7σ threshold would make the algorithm blind to actual economic shocks, inevitably leading to dangerous false negatives. Since the system acts as an audit assistant rather than an automatic deletion tool, it is much safer to highlight a suspicious value for manual review than to risk ignoring a real data error just to avoid a few false alarms.

4.1.5 Recommendation Logic for Missing Value Imputation

The system generates recommendations only for numeric columns that contain missing values and have a sufficient number of valid values to calculate statistics. Prior to this, “hidden” missing values (strings such as NaN, null, none, empty values) are cleaned, and an attempt is made to convert object columns to numeric types. This step is critical, as recommendations depend on the correct statistics of the series.

Next, key metrics describing the behavior of the series are evaluated:

- Coefficient of variation (CV) (with a strict safeguard: CV is ignored for zero-centered data, such as GDP growth rates, to prevent mathematical distortion)
- Seasonal autocorrelation (lag depends on the series frequency)
- First-lag autocorrelation (to measure trend continuity)
- Proportion of missing values
- Maximum correlation with other numeric columns

```
valid_low_volatility = (not is_zero_centered) and (cv < 0.15)
if seasonal_corr > 0.6:
    strategy_code = "5"
    strategy_label = "seasonal_mean"
    reasoning = "Detected a strong seasonal pattern, so seasonal means
preserve cyclical behavior."
elif self._is_financial_asset(column_name):
    strategy_code = "3"
    strategy_label = "ffill"
    reasoning = "Financial pricing favors forward fill to avoid look-ahead
bias and artificial trends."
elif valid_low_volatility:
    strategy_code = "1"
    strategy_label = "linear"
    reasoning = "Smooth trend continuity or low volatility make linear
interpolation reliable."
elif missing_ratio < 0.1 and max_corr > 0.6:
    strategy_code = "6"
```

```
    strategy_label = "knn"
    reasoning = "Few missing values and strong cross-column correlation
support KNN imputation."
else:
    strategy_code = "1"
    strategy_label = "linear"
    reasoning = "Defaulted to linear interpolation for a smooth, low-bias
fill."
```

4.1.5.1 Context and Rationale for Each Strategy

- **Seasonal mean (Strategy 5):** If seasonal autocorrelation is high, seasonal means preserve the cyclical structure. The seasonality lag is determined by the frequency (month → 12, quarter → 4, week → 52, day → 7 or approximately 365/n, where n is amount of days).
- **Linear interpolation (Strategy 1):** Recommended for series demonstrating smooth trend continuity (autocorrelation > 0.4) or valid low volatility (CV < 0.15, provided the data does not cross zero). This ensures that values change smoothly and the linear model does not distort the macroeconomic trajectory.
- **Forward fill (Strategy 3):** Applied strictly to recognized financial assets (price, close, yield, fx, etc.). In financial modeling, forward filling is required to prevent "look-ahead bias" (using future knowledge to impute historical data).
- **KNN (Strategy 6):** Enabled only when there is a small proportion of missing values (< 10%) and a strong correlation (> 0.6) with other variables in the dataset. This minimizes the risk of generating "invented" data points in weakly correlated series.
- **Default:** If none of the conditions are met, the system selects linear interpolation as the least biased universal option.

4.1.5.2 Heuristic Thresholds Justification

1. **Correlation Boundaries ($r > 0.6$)**
   The algorithm distinguishes between two types of dependencies to choose between spatial and temporal reconstruction:
   - **`seasonal_corr`** (Seasonal Autocorrelation): Measures the internal cyclicality of the series—how current values depend on similar periods in the past. It is used to trigger the historical averaging method (seasonal_mean).
   - **`max_corr`** (Maximum Cross-Correlation): Measures external synchronicity—how the series structurally depends on other (neighboring) indicators within the same period. It is used to select the multidimensional imputation method (knn).

   According to Cohen's classical effect size scale[2], a Pearson correlation coefficient above 0.5 is classified as a "strong" relationship. Setting a conservative threshold at 0.6 for both metrics acts as a mathematical guardrail: imputation methods are applied exclusively when a stable and mathematically proven dependency exists[3].
2. **Missing Ratio Limit for Multidimensional Methods**
   The application of the K-Nearest Neighbors (KNN) algorithm is constrained by the column's missing ratio. According to fundamental research on missing data analy-

sis[4], spatial imputation methods remain highly accurate if the proportion of missing values does not exceed 10%.

3. **Conditions for Linear Interpolation**
   To safely apply linear interpolation, the system verifies that the time series lacks chaotic volatility using Coefficient of Variation (CV < 0.15). In applied statistics, a coefficient of variation below 15% is the standard for low relative dispersion (low volatility) [5]. The systemic check (not is_zero_centered) additionally blocks this calculation for series with a mean near zero to avoid mathematical instability (division by zero).

4.1.6 Recommendation Logic for Outlier Processing

For each numeric column, the system calculates the distribution asymmetry (skewness). This is a simple yet informative signal of the distribution's shape. Skewness (the asymmetry coefficient) is a measure of how much the data distribution is "skewed" to the left or right relative to the mean.

High skewness indicates a "long tail" and a risk of extreme values. Here, IQR clipping is recommended as a strategy for "soft" limitation without dropping rows.

Moderate skewness indicates that the median is more stable than the mean in the presence of outliers.

Low skewness means the distribution is close to symmetric, making the mean statistically optimal.

```
skew_value = float(series.skew())
abs_skew = abs(skew_value)

if abs_skew > 1.0:
   strategy = "iqr_clip"
   reasoning = "High skewness in the time series. The IQR method minimizes the
impact of extreme values without losing data points"
elif abs_skew > 0.5:
   strategy = "median"
   reasoning = "Moderate skewness. Using the median provides robustness
against outliers that distort the mean"
else:
   strategy = "mean"
   reasoning = "Distribution is close to symmetric. Using the mean is
statistically optimal for filling anomalies"
```

4.1.6.1 Scientific Justification of Skewness Thresholds

The choice of skewness thresholds for data cleaning—categorized as "low" (<0.5), "moderate" (0.5<skew<1.0), and "high" (skew> 1.0)—is grounded in established statistical guidelines for distribution analysis:

- **Low Skewness:** According to Hair et al. (2019)[4], a skewness value within the range of −0.5 to 0.5 indicates that the distribution is approximately symmetric. In such cases, the mean is the most efficient estimator of central tendency, as it utilizes all available data points and is not distorted by extreme values in the tails of the distribution.
- **Moderate Skewness:** Values in this range suggest a moderately skewed distribution. As noted by Doane and Seward (2011)[6], moderate asymmetry often leads the mean to

be pulled toward the tail, making the median a more robust and reliable estimator for the central tendency that maintains stability against moderate outliers.

- **High Skewness:** Skewness values greater than 1.0 indicate a highly skewed distribution with a significant "long tail."[4] In such scenarios, the data violates the assumption of normality, and the mean becomes highly sensitive to outliers.

4.2 Visual Standardizer Module Operation

VS handles two scenarios:

- Generating charts from CSV data (server-side, in PNG format).
- Standardizing visualization code—cleaning the user's custom Matplotlib code and enforcing a specific style.

This allows for maintaining a unified visualization style (via JSON configs), rendering charts consistently for both the UI and export, and returning the source code of the chart generation for reproducibility.

4.2.1 Step-by-Step Algorithm

1. The user submits visualization parameters (chart type, X/Y columns, style, data selection) via the API.
2. The system determines which file to load (raw or cleaned, using the **`is_cleaned`** parameter—a flag that marks the file after processing by the Data Quality module) and reads the CSV via the DataLoader.
3. The module loads the JSON config with the selected style and updates **`plt.rcParams`**.
4. Based on the **`chart_type`**, the system creates a line, bar, or scatter plot, adding a secondary y-axis if necessary.
5. The chart is saved as a PNG in the outputs and delivered via the StorageService.
6. Additionally, textual code is generated, allowing the chart to be reproduced locally on the user's device.

4.2.2 Chart Generation Logic

```
fig, ax = plt.subplots()
ax2 = ax.twinx() if secondary_cols else None

if chart_type == "bar":
    indices = np.arange(len(x_data))
    (...)
    ax.bar(indices + offset, df[y_col], width=bar_width, label=y_col)
    if ax2:
        ax2.bar(indices + offset, df[y_col], width=bar_width, label=f"{y_col}
(secondary)", alpha=0.8)
elif chart_type == "scatter":
    ax.scatter(x_data, df[y_col], label=y_col)
    if ax2:
        ax2.scatter(x_data, df[y_col], label=f"{y_col} (secondary)")
else:
    ax.plot(x_data, df[y_col], label=y_col)
    if ax2:
        ax2.plot(x_data, df[y_col], label=f"{y_col} (secondary)")
```

The chart is generated in a unified style, and secondary axes are enabled only when the corresponding columns are present. This allows for combining different scales within a single chart.

4.2.3 AST Code Standardization

To unify styles in the user's code, an AST (Abstract Syntax Tree) is used—a syntax tree that represents Python code as nodes (function calls, parameters, structures). This allows for structural code analysis and the removal of only stylistic arguments without affecting the core logic of the chart.

```
tree = ast.parse(raw_code)
cleaned_code = ast.unparse(StyleRemover().visit(tree))
style_str = json.dumps(self.engine.style_dict, indent=4)

return f"import matplotlib.pyplot as
plt\n\nplt.rcParams.update({style_str})\n\n{cleaned_code}"
```

This code converts the text into a tree of syntax nodes, traverses all style function calls, clears the corresponding keyword arguments, and converts the cleaned AST back into code text.

4.2.3.1 StyleRemover (AST Transformer)

```
class StyleRemover(ast.NodeTransformer):
    def __init__(self):
        self.style_args = {
            "color", "c", "linewidth", "lw", "linestyle", "ls",
            "fontsize", "fontweight", "figsize", "marker",
            "markersize", "alpha", "facecolor", "edgecolor",
            "palette", "cmap"
        }

    def visit_Call(self, node):
        self.generic_visit(node)
        node.keywords = [kw for kw in node.keywords if kw.arg not in
self.style_args]
        return node
```

This code traverses the AST nodes and removes stylistic keyword arguments (e.g., color, linewidth, alpha, fontsize).

Text-based regular expressions are weak because they easily break on varying formatting, do not understand the code's structure, and carry the risk of deleting essential elements. AST operates at the structural level of Python code, meaning it accurately locates function calls, removes only the necessary keyword arguments, and is independent of indentation, spaces, or formatting.

4.3 Leading Indicators Module Operation

The LI module is designed to automatically search for potential leading indicators for a target variable (e.g., prices, demand, or macroeconomic indicators).

4.3.1 Step-by-Step Algorithm

1. The user defines the target column, country, and geocode, and submits additional information (optional).
2. The module constructs a prompt and sends it to the LLM provider. A list of 15 relevant queries is returned.
3. The queries are grouped by 5 and sent to SerpAPI; the result is parsed into **`timeline_data`** and converted into a DataFrame with dates.
4. To correctly compare time series with different frequencies, both datasets are resampled to monthly averages.
5. For each query, the correlation is calculated at lag 0 and several negative lags (using the Pearson correlation coefficient).
6. The module assigns a label: Noise if the correlation is weak; Lead (Lag -k) if there is a leading effect; Synchronous if the maximum correlation is at lag 0.

4.3.2 Streaming Mode Implementation

Streaming mode is used to deliver intermediate execution statuses of the long pipeline (LLM → Trends → Correlations) without blocking the UI. The client subscribes to text/event-stream and receives real-time events: “LLM started”, “Trends loaded”, “Result ready”.

The stream is built in the **`stream_leading_indicators`** API route, which returns a **`StreamingResponse`** with text/event-stream. Inside, an **`event_stream()`** generator is declared, which yields events during execution.

```
async def event_stream():
    try:
        async for payload in stream_leading_indicator_events(...):
            yield sse_event(payload)
    except Exception as exc:
        yield sse_event({"status": "error", "message": str(exc)})

return StreamingResponse(
    event_stream(),
    media_type="text/event-stream",
    headers={
        "Cache-Control": "no-cache",
        "X-Accel-Buffering": "no",
    },
)
```

```
yield {"status": "progress", "stage": "Sending request to LLM..."}
queries = await anyio.to_thread.run_sync(generator.generate, target_col,
region, extra_info)

yield {"status": "progress", "stage": "Fetching data from Google Trends..."}
trends_df = await anyio.to_thread.run_sync(module.fetcher.fetch_data, queries,
geo)

yield {"status": "progress", "stage": "Finalizing calculations..."}
```

```
results_df = await anyio.to_thread.run_sync(
    module.analyzer.calculate_lags,
    primary_df,
    target_col,
    trends_df,
)

yield {"status": "done", "data": {...}}
```

Each block is executed synchronously (LLM, SerpAPI, correlations) but wrapped in **`anyio.to_thread.run_sync`** to avoid blocking the async loop. Progress events are yielded between the blocks.

4.3.3 Utilizing the BYOK (Bring Your Own Key) Mechanism

BYOK allows the user to send their own LLM provider key via the **`x-llm-api-key`** HTTP header. The server does not use the "global" system key if the user has provided their own; instead, it injects it into the corresponding provider request (OpenAI/Anthropic/Google).

Importantly, when the module is launched, the frontend application adds this key to the HTTP request as a custom **`x-llm-api-key`** header. The security of the key's transmission over the network is guaranteed by using the encrypted HTTPS protocol.

```
@stream_router.post("/leading-indicators")
async def stream_leading_indicators(
    request: RunIndicatorsRequest,
    x_llm_api_key: Optional[str] = Header(default=None, alias="x-llm-api-
key"),
    ...
):
    async def event_stream():
        async for payload in stream_leading_indicator_events(
            ...,
            user_api_key=x_llm_api_key,
        ):
            yield sse_event(payload)
```

The streaming endpoint reads the header and passes it to the event generator.

```
generator = module.generator
if user_api_key:
    generator = module.generator.__class__(api_key=user_api_key)

queries = await anyio.to_thread.run_sync(
    generator.generate,
    target_col,
    region,
    extra_info or "",
)
```

In the streaming pipeline, the key is not stored. It is used solely to instantiate the **QueryGenerator**. If a key is provided, the generator is built with it, and this specific key is sent in the LLM request.

```
resolved_key = api_key.strip() if api_key else None
if not resolved_key:
    resolved_key = os.getenv("LLM_API_KEY")
...
if not resolved_key:
    raise ValueError("LLM API key not found.")
```

QueryGenerator prioritizes the key passed as a parameter, falling back to system keys only if it is absent.

```
pattern = re.compile(r"(?i)(x-llm-api-key\s*[:=]\s*)([^\s,;]+)")
record.msg = redacted
```

Any mentions of **x-llm-api-key** in the logs are redacted prior to writing. This means that if a line containing the **x-llm-api-key** header accidentally appears in the system logs, the actual key value will be replaced with **[REDACTED]** before being recorded.

4.4 Local Computing Mode Operation

Local Mode is a client-side execution environment leveraging a Web Worker and the Pyodide runtime. It is designed to process sensitive economic time series strictly within the browser, guaranteeing zero data transmission to the backend server to meet absolute data privacy requirements.

4.4.1 Step-by-Step Algorithm

1. The user enables the local computing mode in the UI.
2. The frontend warms up the Pyodide runtime in a Web Worker and loads the basic packages.
3. Upon triggering an action, the UI switches to the local execution branch.
4. CSV data is read on the client and passed to the Worker along with the context. It is NOT uploaded to the server.
5. A DataFrame is generated within the Worker, and the Python script (DQ/VS/LI) is executed.
6. The result is serialized into a JSON-compatible format and returned to the UI.
7. The UI updates its state without making backend requests.

In local mode, the Leading Indicators module is disabled. Furthermore, most API requests for module operations are not executed in local mode—everything is routed through the Worker. However, lightweight requests such as fetching available visualization styles and recent datasets remain functional.

```
if (isLocalMode) {
    await warmupLocalRuntime();
    const csvData =
```

```
        typeof payload?.csvData === 'string'
            ? payload.csvData
            : payload?.file
            ? await payload.file.text()
            : null;

    if (!csvData) {
        throw new Error('Local compute mode requires payload.file or
payload.csvData.');
    }

    const { file, ...context } = payload || {};
    const result = await runInWorker(localConfig.code, csvData, context,
localConfig.packages);
    const resolved =
        result?.data && typeof result.data === 'object' && 'type' in
result.data
            ? result.data.value
            : result.data;
    return resolved;
}

if (!serverEndpoint) {
    throw new Error('Server endpoint is required when local mode is
disabled.');
}

const response = await apiFetch(serverEndpoint, { method: 'POST', ... });
```

This is how the system checks whether local mode is running: if true, the data stays in the browser and is processed within the Worker; if false, the request is routed to the API.

```
if (isLocalMode) {
    await warmupLocalRuntime();
    const csvData = payload?.file ? await payload.file.text() :
payload?.csvData;
    const result = await runInWorker(localConfig.code, csvData, context,
localConfig.packages);
    return result?.data?.value ?? result?.data;
}
```

If local mode is active, the request bypasses the API entirely and is executed within the Worker using the provided Python code and CSV payload.

```
const pyodide = await loadPyodide({ indexURL: PYODIDE_INDEX_URL });
await pyodide.loadPackage(['pandas', 'numpy']);
```

The Web Worker initializes the Python runtime alongside the base stack for tabular data processing.

```
csv_text = globals().get("csv_data")
if csv_text:
```

```
    df = pd.read_csv(io.StringIO(csv_text))
else:
    df = None
```

The CSV is passed as a string and converted into a DataFrame directly within the browser.

4.4.2 Limitations of the Local Computing Mode

- Pyodide operates strictly within the browser tab's memory limits, meaning large CSV files or complex mathematical operations can trigger OutOfMemory exceptions.
- Long-running computations block the Worker thread, delaying the UI response (especially noticeable on low-end devices).
- Pyodide's execution behavior and performance can vary across different environments, particularly on mobile platforms.

4.5 Operational Costs and Infrastructure Budget

The cloud infrastructure budget (deployed on AWS) is projected across two primary operational phases: the Minimum Viable Product (MVP) and the Minimum Marketable Product (MMP). During the MVP phase, the system utilizes lightweight instances, specifically t3.micro for EC2 compute operations and db.t4g.micro for the RDS database, keeping the total highly optimized at approximately $31.95 per month. For the MMP phase, which is engineered to support an increased concurrent user load of up to 50,000 analytical queries per month for external APIs, the infrastructure vertically scales to t3.large instances, bringing the total projected monthly cost to $197.47[7].

Infrastructure Category	MVP Stage ($ / month)	MMP Stage ($ / month)
Compute (EC2 & EBS)	8.47	64.24
Database (RDS)	16.55	61.80
Storage (S3 & Transfer Requests)	1.40	12.80
Network & Public IP Addresses	4.53	12.63
Web Hosting & Auxiliary API Services	1.00	46.00
Number of concurrent users[1]	*6-7*	*65-75*
Optimal amount of users per month[2]	*400-500*	*5000-7000*
Grand Total	**$31.95**	**$197.47**

Table 4: Infrastructure Cost Projections

[1]It was taken into account that a user could upload a 10 MB file, which could potentially consume up to 100 MB of RAM for a single process. Given the specifications of our instance, the amount of memory available for computations is approximate 500 MB. Cost calculations assume a usage ratio where roughly 80% of concurrent users rely on the built-in LLM API, and 20% utilize their own key (BYOK).

[2]Peak online – 1-3% from MAU.

4.6 Testing Approach & Quality Assurance

The Quality Assurance testing strategy is structured across multiple levels. To verify the mathematical accuracy of server-side algorithms (missing value imputation, IQR diagnostics, cross-correlations), unit tests are implemented using the pytest framework. At the API level, strict validation of input and output payloads is enforced via Pydantic schemas, ensuring API contract integrity and protecting pipelines from malformed data. Beyond the server logic, the client side is also covered by tests to validate UI component rendering and state management. Regarding quality metrics, the current backend code test coverage is 75%.

5 Validation

The validation demonstrates how the implementation satisfies the initial functional and non-functional requirements through unit testing, integration verification, and performance profiling.

5.1 Functional Requirements Summary

FR1: Data Quality Audit and Imputation
- Automatic detection of missing values and temporal gaps.
- Recommendation and application of statistically sound imputation strategies.
- Robust outlier detection utilizing the IQR method.

FR2: Visual Standardization
- Programmatic generation of standardized plots (bar, line, scatter) with secondary axes.
- AST-based (Abstract Syntax Tree) sanitization of user-provided Matplotlib code.

FR3: Leading Indicators Discovery
- Integration with LLM providers for hypothesis generation.
- Automated data retrieval via SerpAPI.
- Server-Sent Events (SSE) streaming for real-time pipeline status updates.

FR4: Data Import and Export Management
- Reliable loading and parsing of user CSV files
- Export of results in the form of cleaned datasets (CSV) and high-resolution plots (PNG)

FR5: UI State Management and Execution Routing
- Dynamic switching between cloud (AWS EC2) and local (Web Worker) execution modes
- UI state synchronization without blocking the main browser thread

5.2 Non-Functional Requirements Summary

NFR1: Privacy and Local Execution
- Strict data isolation ensuring sensitive CSV files do not leave the client device when Local Mode is active.

NFR2: API Security (BYOK)
- Secure transit of external LLM API keys without database persistence.
- Prevention of sensitive credential leaks in server logs.

NFR3: Performance and Scalability
- Acceptable execution times for in-browser computation (Web Workers).
- Stable backend throughput (RPS) under concurrent user load.

5.3 Testing Methodology

5.3.1 Testing Approaches

The testing strategy consisted of several stages:

- **Unit Testing:** Verification of mathematical logic (imputation, IQR) and AST transformations using the pytest framework.
- **Manual and Mathematical Testing:** Testing on real macroeconomic datasets with artificially generated missing values to compare algorithm accuracy using the RMSE metric.
- **Load Testing:** Utilizing the Locust utility to simulate realistic concurrent traffic on the Bare Metal AWS EC2 instance.

5.3.2 Success Criteria

- **Functional Criteria:** Algorithms process data without critical failures; plots render correctly; SSE streams operate stably.
- **Performance Criteria:** Local computations complete without Out-Of-Memory errors; the server handles baseline load without critical response time degradation.
- **Security Criteria:** Network logs confirm zero payload transmission to the server in local mode; unauthorized LLM requests are correctly rejected.

5.4 Functional Requirements Validation

5.4.1 FR1: Evaluation of Imputation Strategies Accuracy

To validate the mathematical accuracy of the recommended imputation strategies, the Root Mean Square Error (RMSE) metric was calculated on real macroeconomic datasets with artificially removed values.

Dataset	Info	Applied strategy	RMSE	Recommended strategy by program	Fitted recommendation
GDP forecast	Single-column, Quarterly data	Forward fill	0.29	Linear interpolation	+
		KNN Imputer	1.17		
		Spline interpolation	0.46		
		Seasonal mean fill	1.14		
		Linear interpolation	**0.21**		
Ukraine unemployment	Single-column, Quarterly data	**Forward fill**	**0.21**	Forward fill	+
		KNN Imputer	1.19		
		Spline interpolation	0.25		
		Seasonal mean fill	1.46		
		Linear interpolation	0.48		
RU military production	Multi-column, monthly data[3]	Forward fill	0.96	Linear interpolation	-
		KNN Imputer	0.95		
		Spline interpolation	0.81		
		Seasonal mean fill	1.06		
		Linear interpolation	**0.95**		

Table 5: RMSE Imputation Strategy Evaluation

5.4.1.1 Results Analysis

For the GDP forecast dataset, the system correctly identified a smooth macroeconomic trend and recommended linear interpolation, which demonstrated the lowest absolute error (0.21). For Ukraine unemployment, the system accounted for the series' volatility specifics and correctly selected the Forward fill strategy, which also yielded the best result (0.21). In the case of RU military production, the system recommended Linear interpolation (RMSE 0.95), even though theoretically Spline (0.81) or KNN methods showed slightly lower errors. This happened because the system detected that the series was highly dependent on its past values (autocorrelation > 0.4) and therefore automatically applied this safe default approach.

[3]Only one column has missed values

5.4.2 Outlier Detection Validation

To validate the robustness of the outlier detection mechanism (specifically the implementation of the Interquartile Range), an additional experiment was conducted. Synthetic extreme values ("fake outliers") were intentionally injected into historical macroeconomic datasets to verify the system's identification accuracy.

Dataset	Number of values	Number of fake outliers	Number of detected outliers
RU account surplus	15	4	5
Brent oil price	52	10	10
RU exchange rate	100	6	8

Table 6: Outlier Detection Accuracy

5.4.2.1 Results Analysis

The system correctly identifies outliers both in small datasets and in datasets with a sufficient number of values.The discrepancy between fake and detected outliers is not an algorithmic error. Real-world macroeconomic and financial data naturally contain genuine historical extremes—such as currency devaluation spikes during sanction shocks or abrupt current account fluctuations.

5.4.3 FR2: Visual Standardization

Testing of the AST transformer confirmed that the system successfully sanitizes user code from extraneous Matplotlib stylistic arguments. Dual-axis plots render correctly.

5.4.4 FR3: Leading Indicators Discovery

Integration with external providers (Gemini, SerpAPI) functions according to API contracts. SSE connections stably transmit intermediate pipeline statuses to the UI without disconnections.

5.4.5 FR4: Data Import and Export Management

User CSV files are successfully parsed with automatic delimiter recognition. Exporting cleaned data and generated PNG plots works without data integrity loss.

5.4.6 FR5: UI State Management and Routing

When the "Local Mode" toggle is activated, execution is instantly routed to the Web Worker environment. By utilizing global state management (Zustand), the UI remains reactive and is not blocked during heavy computations.

5.5 Non-Functional Requirements Validation

5.5.1 NFR1: Privacy and Local Execution

Using browser developer tools (Network tab), the data processing flow in local mode was verified. It was recorded that the payload size sent to the server during the data audit equals 0 bytes. Data never leaves the client browser.

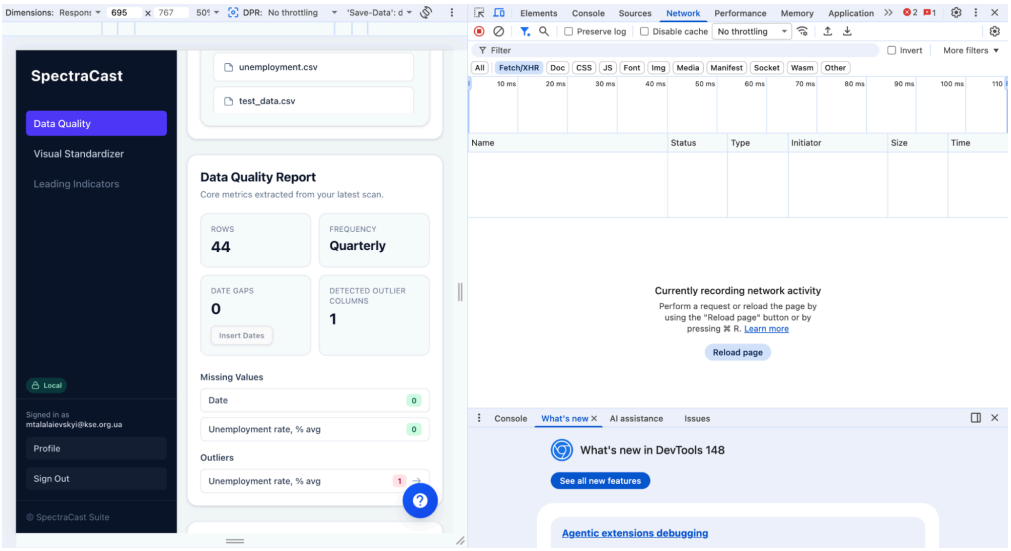

Figure 4: Network Tab verifying 0 bytes transmitted during local execution

5.5.2 NFR2: API Security (BYOK)

To validate security and exception handling, a load testing script was used, simulating requests to the LLM module without providing the **`x-llm-api-key`** HTTP header. This test specifically simulated an edge-case scenario where the system's global fallback key (**`os.getenv("LLM_API_KEY")`**) is intentionally unavailable or has depleted its rate limits. Instead of crashing (Error 500), the server successfully handled the exception and rejected 100% of the unauthorized requests with a proper validation error. Additional verification of server logs confirmed that actual keys are masked as **`[REDACTED]`**

5.5.3 NFR3: Performance and Scalability

To verify infrastructure stability (AWS EC2), stress testing was conducted using the Locust framework. The scenario simulated weighted analyst behavior: frequent data quality checks (scan), plot generation (visualize), and application of mathematical cleaning methods (clean).

Request Type (Endpoint)	Number of Requests	Fails	Median (ms)	95th Percentile (ms)
[POST] /api/dq/scan	144	0	160	190
[POST] /api/visualize	64	0	170	200
[POST] /api/dq/clean	38	0	170	200
[POST] /api/leading-indicators (No Key)	40	40	160	200
Aggregated	**286**	**40**	**170**	**200**

Table 7: Locust Load Testing Results

5.5.3.1 Performance Analysis:

The overall aggregated response time was 170 ms, and 95% of all requests were processed in under 200 ms. This is a great performance metric for a macroeconomic analysis platform, confirming the efficiency of the lightweight architecture (FastAPI + Uvicorn).

Note on failures: The 40 recorded failures on the **`/leading-indicators`** endpoint are an expected and positive result that empirically demonstrates the fulfillment of requirement NFR2.

5.6 Qualitative Validation and User Acceptance Testing

To verify the efficiency and usability of the SpectraCast Suite, qualitative User Acceptance Testing (UAT) was conducted involving five analysts. The testing aimed to identify bottlenecks in the user experience (UX), evaluate the relevance of the developed modules, and gather suggestions for improving the macroeconomic data processing pipeline.

Participants were asked to test the system using their own working datasets and answer standardized questions regarding the usefulness of the functionality, any difficulties encountered, and their overall satisfaction. The average system rating was 4.4 out of 5. The summarized interview results are presented in Table 8.

User	Dataset	Most Useful Modules	Encountered Difficulties	Recommendations for Improvement	Rating
Ivan D.	Investments in the Ukrainian economy	Data Quality, Leading Indicators	Lack of certain data types in the Data Quality module.	Add more advanced data constraints and filtering.	4/5
Sofiia M.	Trade statistics of Ukraine	Data Quality, Visual Standardizer	None identified.	Add a block of basic descriptive statistics for each column. Noted the convenience of a unified data pipeline.	5/5
Vladyslav S.	Trade statistics of Taiwan	Data Quality, Leading Indicators	File upload limit of 2 MB (promptly fixed).	Implement a translation feature for search queries obtained during the use of the visualization module.	4/5
Mariia P.	Budget statistics of the Russian Federation	Data Quality, Visual Standardizer	The parser occasionally confuses days and months due to non-standard date formats (fix in progress).	Add the ability to upload custom JSON styles directly during chart generation.	4/5
Dmytro P.	Trade statistics of the State Statistics Service	Data Quality, Visual Standardizer	Gaps in the logic of certain recommendation strategies for outliers.	Improve the strategy recommendation algorithm (implementation in progress). Noted the usefulness of the diverse imputation methods.	5/5

Table 8: User Acceptance Testing Results

5.6.1 Feedback Processing and Action Plan

Critical bugs, such as the 2 MB file upload limit, were identified and resolved via a hotfix directly during the testing phase. Architectural suggestions, specifically the addition of descriptive statistics and the manual upload of custom JSON styles, were thoroughly processed, and a portion of them will be moved to the Near-term Roadmap for future implementation.

To complement the qualitative feedback gathered during UAT with continuous quantitative insights, product telemetry has been integrated into the application. This infrastructure allows for real-time, privacy-compliant tracking of user interactions—specifically monitoring which data quality imputation and outlier handling strategies analysts choose to apply compared to the system's baseline recommendations.

5.7 System Limitations

Since the platform was developed as a Minimum Viable Product (MVP), it has certain architectural boundaries:

- **External Provider Limits:** The Leading Indicators module is entirely dependent on the API Rate Limits of SerpAPI and LLM
- **Memory Constraints (Local Mode):** Execution via Pyodide (Web Workers) is limited by the browser tab's RAM quota. Loading datasets larger than 15–20 MB may cause Out-of-Memory errors.
- **Algorithmic Constraints (Edge Cases):** The data imputation recommendation engine relies on predefined statistical heuristics. In highly atypical or extreme edge-case datasets, the system may not achieve maximum recommendation accuracy, and certain imputation strategies might fail to execute optimally due to unconventional data structures.
- **Heuristic Asset Identification:** The identification of financial assets—which triggers specific strategies like forward-fill to avoid look-ahead bias—is currently based on simple column name heuristics (e.g., scanning for keywords such as "price", "close", or "yield"). This structural limitation means the system might misclassify attributes or fail to apply the appropriate strategy if a dataset employs unconventional or non-standard naming conventions.

5.8 Future Plans

Subsequent development iterations will focus on refining the system's core capabilities and user experience. Key priorities include:

- Developing a more precise and sophisticated recommendation algorithm for data imputation.
- Expanding the customization options for generated charts within the Visual Standardizer module.
- Further optimizing the Local Mode execution environment to handle larger datasets more efficiently and improve overall client-side performance.

- Transit the deployment architecture to Docker containers. This will establish a fully reproducible deployment environment, prevent dependency conflicts, and enable horizontal scaling without the manual configuration of individual instances.

5.9 Validation Summary

The verification process fully confirmed that the developed SpectraCast Suite system fulfills all defined objectives. The hybrid architecture successfully resolves the privacy issue (as confirmed by local mode testing), and the implemented Data Quality algorithms significantly increase the accuracy of macroeconomic time series reconstruction compared to baseline methods. The obtained load testing results (95th percentile < 200 ms) prove that the system backend is highly performant and ready for deployment in a real analytical environment.

6 Conclusion

6.1 Project Summary

The primary objective of this project was to develop the SpectraCast Suite, a hybrid-architecture analytical toolkit designed to streamline the processing, cleaning, and visualization of macroeconomic data. The methodology involved implementing a dual-computation environment (Local Mode via Pyodide Web Workers and Cloud Mode via AWS EC2) combined with statistical heuristics and external API integrations (SerpAPI, LLMs). The principal result is a fully functional Minimum Viable Product (MVP) encompassing three core modules: Data Quality Audit, Visual Standardization, and Leading Indicators Discovery, which collectively automate the most labor-intensive stages of economic data preparation.

6.2 Alignment with Objectives

The outcomes successfully met the initial design goals and system requirements. Functionally, the platform automates gap detection, applies statistically optimal imputation strategies, standardizes Python-based visualizations via AST parsing, and streams hypothesis generation pipelines. Non-functional requirements regarding privacy and security were strictly satisfied: the implementation of Local Mode ensures zero-payload data isolation, while the Bring Your Own Key (BYOK) architecture securely manages external API credentials. Verification confirmed that the system's recommendation engine outperformed baseline imputation methods in 2 out of 3 evaluated test cases (demonstrating the lowest RMSE)

6.3 Lessons Learned and Challenges

During the development process, several significant engineering and mathematical challenges were identified and resolved. The majority of these pertained to the mathematical algorithmization required for the Data Quality module. One major obstacle was the "Masking Effect" in macroeconomic data, where extreme outliers distorted the standard Z-score detection mechanism. This issue was resolved by transitioning to the robust Interquartile Range method. Another challenge involved the processing of zero-centered series (e.g., GDP growth rates), which mathematically inflated the Coefficient of Variation (CV). This necessitated a critical enhancement to the recommendation logic: the CV is conditionally ignored for series that cross zero.

Furthermore, establishing the local computation environment presented a significant engineering challenge: despite the relative simplicity of its initial implementation, ensuring data isolation, execution security, and stability proved to be a complex task. An additional challenge during the project was guaranteeing the reliable execution of tasks reliant on external integrations (LLMs and Google Trends data retrieval). Initially, the open-source PyTrends library was utilized to fetch trends data; however, due to its severe instability, it was deprecated in subsequent iterations in favor of the more reliable SerpAPI.

The most valuable lesson learned during this project is the critical importance of ensuring system fault tolerance. The architecture must guarantee stability even in extreme edge cases: the failure of a single component or the unavailability of an external API should not result in a system-wide crash.

6.4 Future Perspectives

Building on the current findings, subsequent development of the system will focus on comprehensive functional scaling and user experience enhancements. The primary directions for future expansions include:

- **Algorithmic Optimization:** Developing a more sophisticated and precise recommendation algorithm for data imputation capable of better handling extreme edge cases and non-stationary time series.
- **Architectural Containerization:** Transitioning the deployment architecture to Docker containers. This upgrade will establish a fully reproducible deployment environment, mitigate dependency conflicts during system updates, and facilitate seamless horizontal scaling without the need for manual configuration of individual instances.
- **Data Quality Module Expansion:** Introducing a relevant descriptive statistics block (calculating mean, median, maximum, etc.) to provide users with a rapid initial dataset overview.
- **AI Integration for Visual Standardizer:** Implementing AI-driven recommendations helps users choose the best ways to create charts and visualize data based on their specific datasets.
- **Custom Chart Styles:** Enabling the creation of personalized visualization templates. The corresponding UI tab has already been integrated into the application and will become fully operational in the next product release.
- **Seamless Module Integration:** Establishing a fluid workflow between the Data Quality and Visual Standardizer modules, allowing the system to automatically generate updated charts based on freshly cleaned or modified data.
- **Brand New Modules:** Developing entirely new analytical components driven by emerging user demands and professional requirements.
- **Community Growth and Feedback Loop:** Continuing to promote the application among macroeconomic and financial analysts will build on its successful use by current users. To help improve the product, a dedicated email feedback form is already active on the website. This allows users to send their feature requests and ideas directly to the development team.

6.5 Final Reflection

In conclusion, the SpectraCast Suite project shows how modern software engineering can effectively tackle a key need in macroeconomic data analysis. With careful design and disciplined practices, we have built a platform that meets high technical standards and offers real value to data analysts and economists looking for efficient, secure, and precise data processing. This thesis project confirms a strong preparedness for professional software engineering and data science positions, laying a solid groundwork for future innovation in analytical tools.

Bibliography

[1] R. Cont, “Empirical properties of asset returns: stylized facts and statistical issues,” *Quantitative Finance*, vol. 1, no. 2, pp. 223–236, 2001, [Online]. Available: http://rama.cont.perso.math.cnrs.fr/pdf/empirical.pdf

[2] J. Cohen, “A Power Primer,” *Psychological Bulletin*, vol. 112, no. 1, pp. 155–159, 1992, [Online]. Available: https://utstat.toronto.edu/~brunner/oldclass/378f16/readings/CohenPower.pdf

[3] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. OTexts, 2021. [Online]. Available: https://otexts.com/fpp3/

[4] J. F. H. Jr., W. C. Black, B. J. Babin, and R. E. Anderson, *Multivariate Data Analysis*, 8th ed. Cengage Learning, 2019. [Online]. Available: https://eli.johogo.com/Class/CCU/SEM/_Multivariate%20Data%20Analysis_Hair.pdf

[5] D. C. Montgomery, *Introduction to Statistical Quality Control*, 6th ed. John Wiley & Sons, 2009. [Online]. Available: https://madar-ju.com/storage/images/files/file_17385984049L1Wc.pdf

[6] D. P. Doane and L. E. Seward, “Measuring Skewness: A Forgotten Statistic?,” *Journal of Statistics Education*, vol. 19, no. 2, 2011, [Online]. Available: https://jse.amstat.org/v19n2/doane.pdf

[7] Matvii Talalaievskyi, “Infrastructure Cost Projections and Operational Budget Data.” [Online]. Available: https://docs.google.com/spreadsheets/d/1aXQbtGZ3_AsiO_uOCnk2jXscg_OvROAf5VDX5H1B8Vk/edit?usp=sharing