Gold Price Forecasting System

Assignment 2: Data System Implementation

Jaehee Kim, Jun Hyun Lim, James Davison

Group 20

Table of Contents

List of Figures

List of Tables

1. Introduction

The Gold Price Forecasting System (GPFS) is designed to provide real-time forecasting analysis of the volatility of gold prices based on financial market data and macroeconomic indicators. The system offers investors, analysts, and policymakers' valuable information that can enhance the quality of their economic decision-making processes.

1.1 Scope

The GPFS project strives to develop integrated data system to collect, transform, and persist data on gold, bitcoin, and economic metrics fetched from external APIs such as Yahoo Finance and FRED. The system employs machine learning models to model the data and thereby make predictions of gold prices, sell recommendations, and USD strength metrics. Outputs are provided through an interactive web-based application for various types of users, such as investors, analysts, and government representatives. Although user context is maintained, user credentials are no longer stored within the system.

1.2 Product Aim

The fluid nature of the stock market brings a need for a system that can fast track and deliver accurate information. GPFS strives to report this knowledge to an audience that relies on macroeconomic forecasts and gold price analytics without requiring persistent personal identification.

1.3 Purpose

This report serves to provide an in-depth illustration of the methods used to set up the data warehouse and reporting tools of GPFS. The report will cover the ETL design, step-by-step process in raising code to power the data warehouse, meeting the requirements of stakeholders, and the interface suited for users.

1.4 Definitions

Acronym	**Definition**
GPFS	Gold Price Forecasting System
FRED	Federal Reserve Economic Data
USD	United States Dollar
ETL	Extract, Transform, Load
API	Application Programming Interface
CPI	Consumer Price Index
DXY	U.S. Dollar Index
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
R^2	Coefficient of Determination

Table 1. Context Definitions

1.5 Contribution Table

Name	Student ID	Contributions
Jun Hyun Lim	25175391	1. Introduction, 2. System Architecture, 3. ETL Task Flow Design, 4. Reporting Application Task Flow Design, 6. Hardware & Software Tools, 7. System Testing, 8. Future Development, 9. Conclusion, 10. Appendix, Doc Editing
James Davison	14264194	7. Test case, 1. Introduction
JaeHee Kim	25175302	Implement the whole system, 5. Interface design, upload git, Doc Editing

Table 2. Contribution Table

2. System Architecture

The system architecture defines the overall arrangement of the GPFS, where the emphasis is on the backend infrastructure instead of the application for end users. This section addresses both the ETL framework and the reporting application framework. The ETL framework provides a high-level overview of the practices adopted for collecting, processing, and storing external data. The reporting application framework then explains how this processed data is utilised by the system to deliver insights and forecast outcomes for various types of users.

2.1 ELT Architecture

In out architecture, we utilise five core components from a technology stack: **Yahoo Finance** and **FRED API** for external data feeds, **Apache Airflow** for pipeline orchestration, **MinIO** for the local data lake for intermediate storage, **PostgreSQL** for the data warehouse in the middle, and **Streamlit** as the reporting and visualisation layer. These modules all function together to facilitate the end-to-end process of extracting, transforming, loading, and visualising economic and financial data. The modular architecture provides easy integration, portability through Docker, and scalability for future additions.

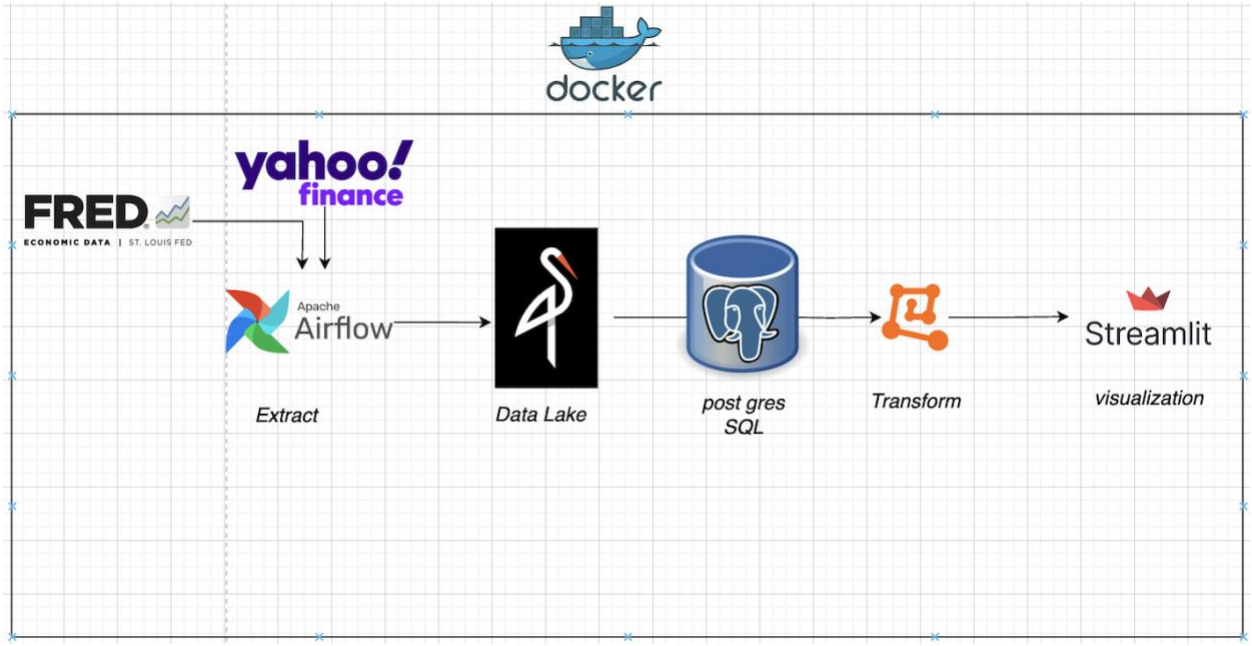


Figure 1. ETL Architecture

2.1.1 ExtractData

The extraction phase of the GPFS framework gathers raw information from various credible financial and economic sources.

The workflow utilises scheduled Python scripts controlled by **Apache Airflow** to get the monthly price of gold, bitcoin, and the USD index (open, close, high, low, volume) from the **Yahoo Finance API**. Meanwhile, the **FRED API** is used to get the most important macroeconomic indicators, including the **CPI**, **interest**, and **unemployment rates**. The data is stored in its raw form in the **MinIO** data to take for versatility and as a fail-safe for eventual transformations.

2.1.2 Transform & Load

After extraction, the raw data is transformed to become consistent, precise, and utilisable.

Transformations include handling missing values, normalising timestamps formats, renaming divergent field names, and joining price data with associated economic indicators. Once the data is cleaned and normalised, it is loaded into a structured **PostgreSQL** data warehouse and organised into star schema tables (fact and remaining dimension tables). The dimension table has been removed in the updated schema for simplification and security considerations. This enables efficient querying, forecasting, and reporting.

2.1.3 Model Implementation

The processed data stored in PostgreSQL is used as input for predictive modelling.

The predictive model, executed in Python and optionally XGBoost, extracts historical gold prices and the corresponding features from the database, trains previous trends, and makes a prediction for the closing price of the following day. These findings are inserted back into PostgreSQL for use in the reporting interface, where they can be viewed by users in real time without requiring persistent user records.

2.1.4 Visualisation

In the final stage, the **Streamlit** framework is utilised to create a simplified and interactive web-based reporting app.

Users can log in with their email and view real-time dashboards displaying gold price trends, USD strength levels, bitcoin comparisons, and machine learning-based forecast results. The app facilitates dynamic filtering by date, currency, and user role while providing an investor-, analyst-, and government-user-friendly interface. Hosted locally or optionally configured with Docker, the app provides secure and responsive access to all data.

2.2 Reporting Application Architecture

Our reporting features are implemented using Python and Streamlit library. The reporting application is a web-based system that gets data from PostgreSQL server to make accessibility better. These architectures include how to access four different pages.

2.2.1 Data Exploration Page

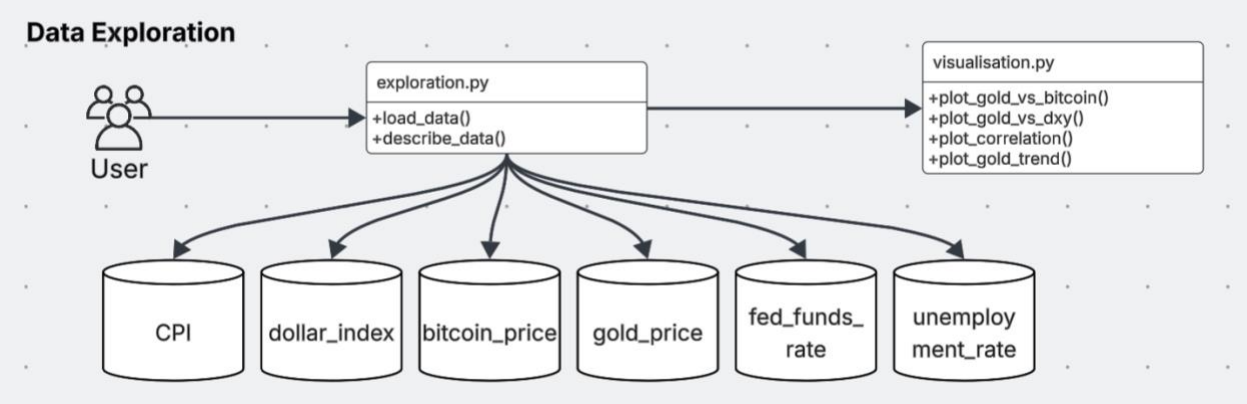


Figure 2. User Interface for Data Exploration

The page provides initial insights into the dataset. Users can preview raw data from various economic indicators and market sources (e.g., gold, bitcoin, USD index), as well as explore summary statistics and correlations using visual plots.

2.2.2 Model Training Page

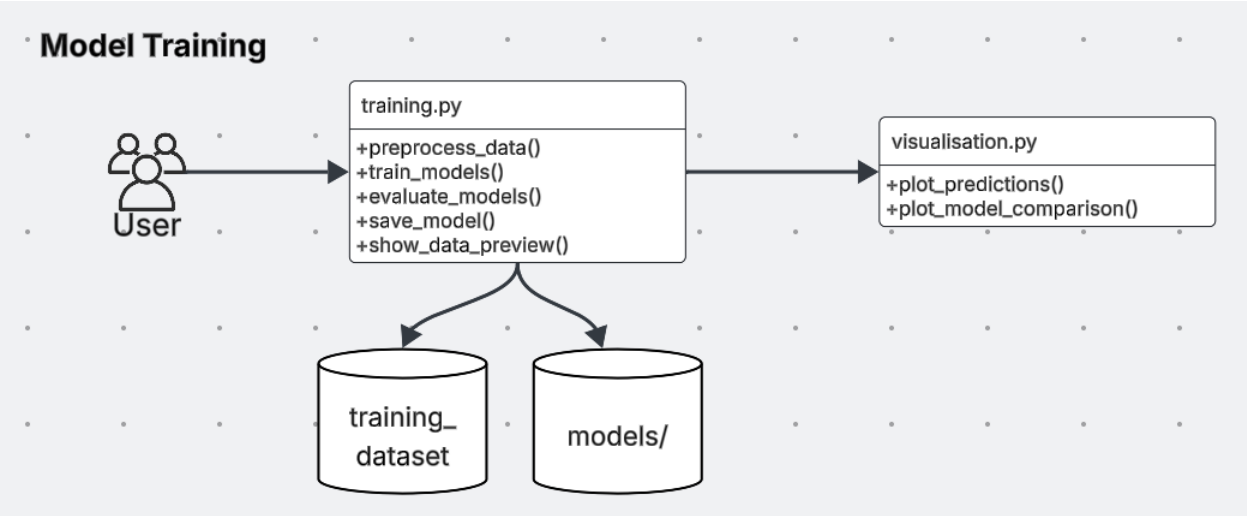


Figure 3. Model Training Interface

This page enables users to preprocess data, train machine learning models (Random Forest, Gradient Boosting, XGBoost), evaluate model performance, and save the best-performing models. Ket metrics such as RMSE, MAE, and R^2 are displayed alongside comparison plots.

2.2.3 Prediction Page

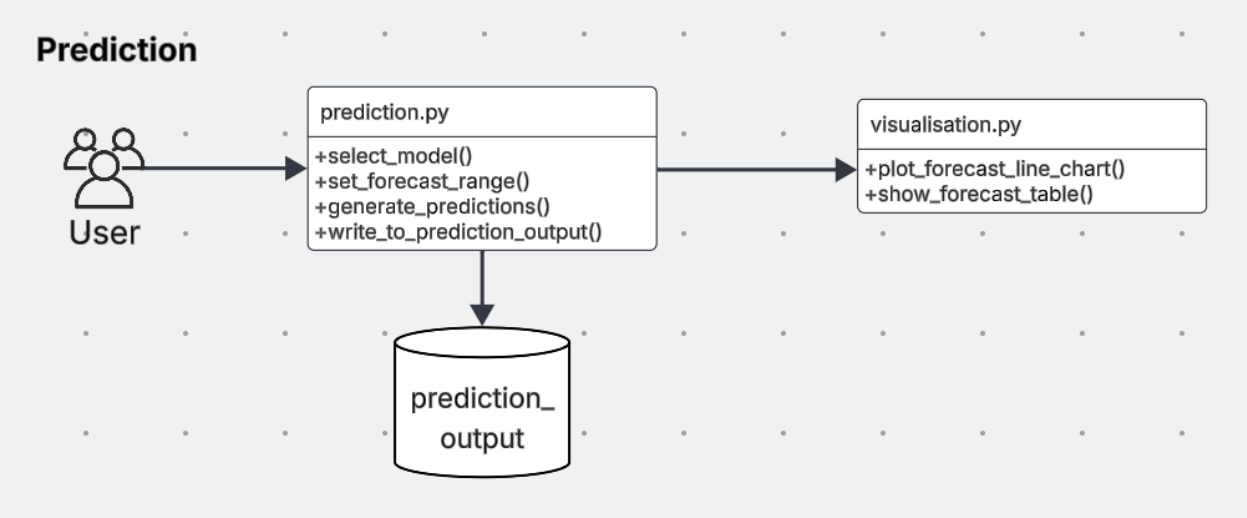


Figure 4. Forecasting Interface

Users can select a trained model, define the forecast range, and generate future gold price predictions. The results are saved and visualised through both a forecast chart and a detailed prediction table.

2.3 Schema Update

In the revised schema, the dim_User and dim_Calendar table have been removed. Former user history (name, email, role, and subscription) used to be persisted for login and for role-based dashboard personalisation. These features have been refactored to streamline data processing and enhance privacy. User login is now handled through external mechanisms, and role-based behavior is managed dynamically at runtime without storing personal user metadata in the data warehouse. Similarly, the dim_Calendar table, which previously supported date-based filtering and annotation (i.e., holidays, weekends, and seasons), has been eliminated. Instead, temporal logic is addressed through real-time preprocessing in the ETL process or date-filtering functionality at the dashboard level. This removes unnecessary joins, enhances query performance, and allows the schema to be dynamic enough to accommodate date-based analysis as needed.

2.3.1 Updated ERD Diagram

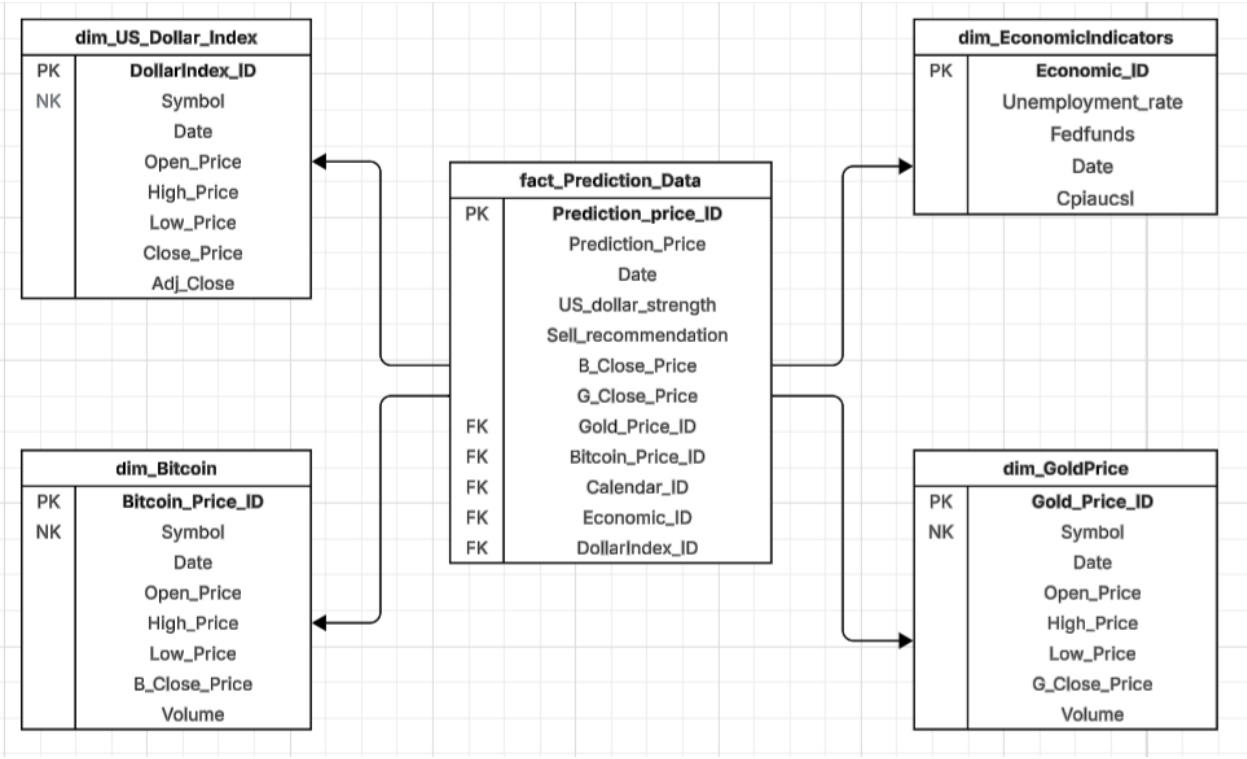


Figure 5. Updated Schema Diagram

2.3.2 Removal of dim_User

The dim_User dimension table has been dropped as part of schema optimization. Its previous function of login validation and support for dashboard role logic has been replaced by a more effective mechanism that isolates user authentication from the data warehouse. The alteration renders the schema simpler, reduces the storage of personally identifiable information (PII), and supports compliance with present-day privacy standards.

2.3.3 Removal of dim_Calendar

The dim_Calendar table has also been removed to further simplify the schema. It is used to capture structured date-related metadata such as Is_Holiday, Is_Weekend, and Season to facilitate temporal filtering and annotation. In the new system, this logic is either encoded directly into the fact tables or computed on the fly as part of the ETL process. Users are also able to filter by date directly in the dashboard using built-in calendar controls. This avoids a surrogate calendar join, minimises schema complexity, and is attuned to agile, event-driven data processing paradigms that operate over timestamped records in real-time.

3. ETL Task Flow Design

The ETL task flow design illustrates the main flow of the database and the system in gold price forecasting. Design table items in the database through appropriate extraction, transformation, or load, and illustrate proper data to supply comprehensive data to the prediction model. Each data flow represents the process through which the system is executed.

3.1 Extract

The ETL task flow for Extract outlines how external financial and economic data is collected using Airflow DAGs. It includes fetching gold, bitcoin, and US dollar index data from Yahoo Finance, as well as macroeconomic indicators from the FRED API. The process also includes basic validation and initial storage in MinIO.

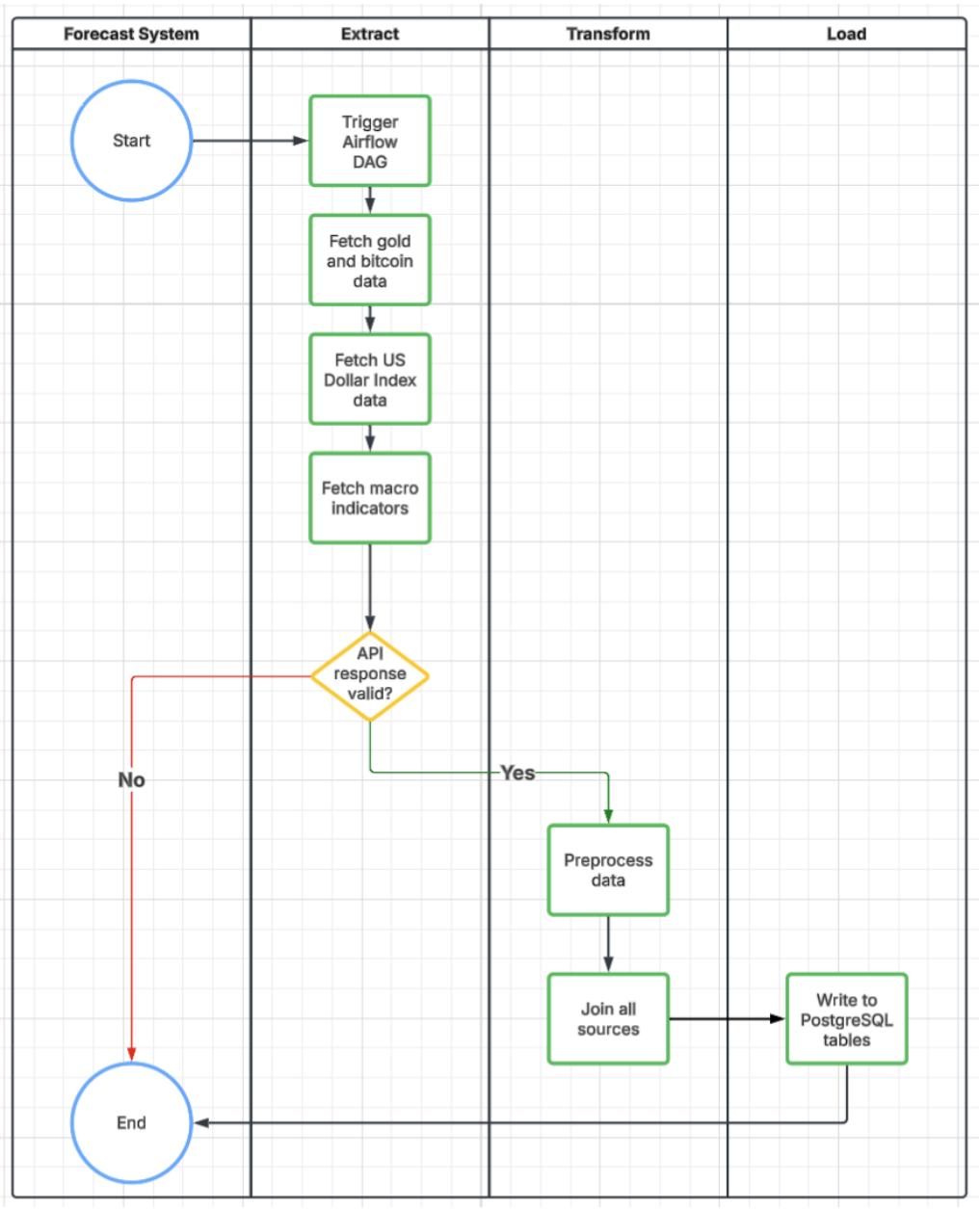


Figure 6. ETL Extract Task Flow

3.2 Transform & Load

The Transform & Load task flow shows how raw data from MinIO is cleaned, merged, and structured into a PostgreSQL star schema. This phase includes normalising formats, generating lag features, and joining financial with economic indicators before loading the final tables.

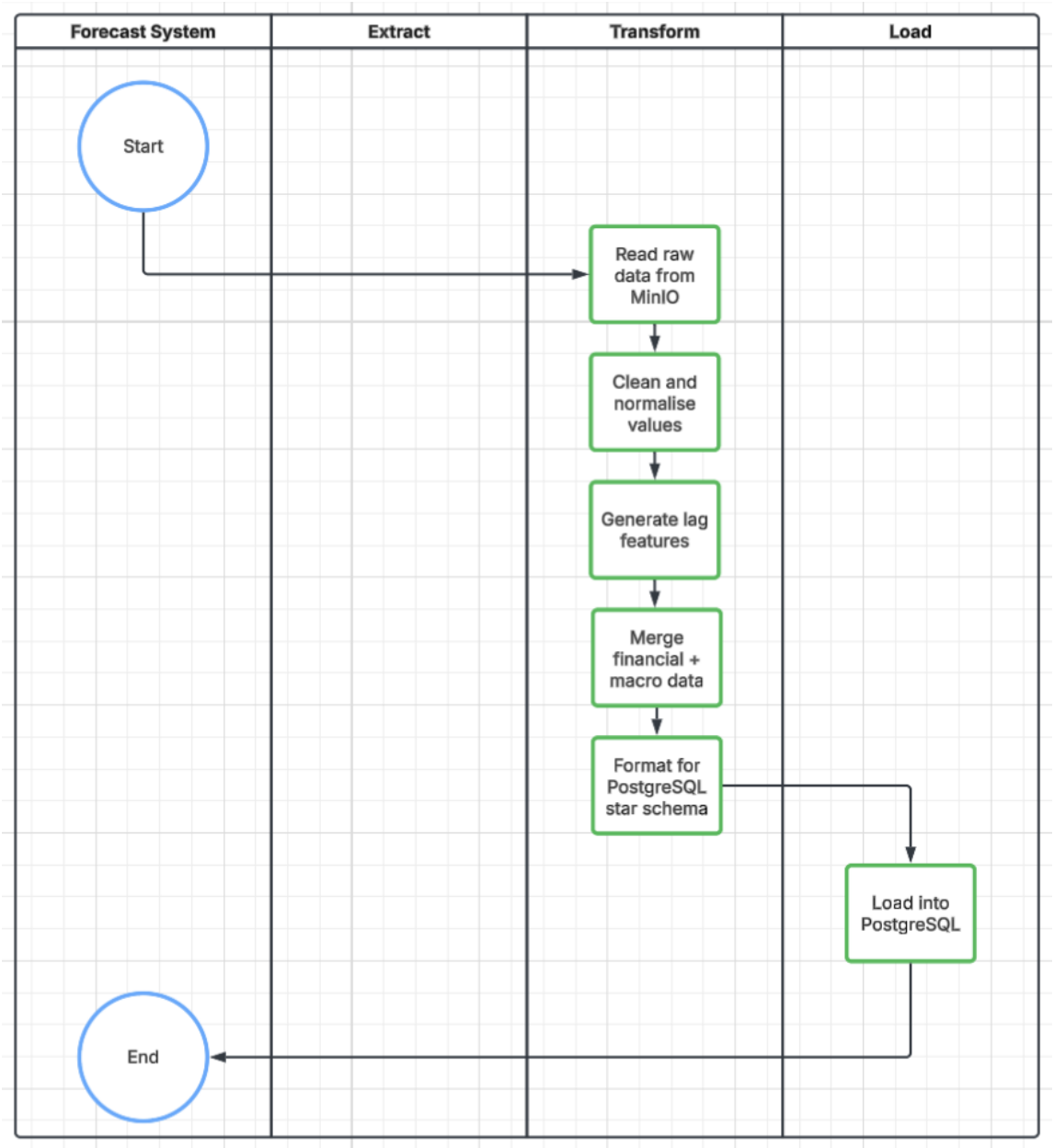


Figure 7. ETL Transform & Load Task Flow

3.3 Forecast & Output

The Forecast & Output task flow represents how trained machine learning models generate predictions based on processed data. It covers model training, evaluation, saving forecast outputs to PostgreSQL, and visualising the results in the Streamlit interface.

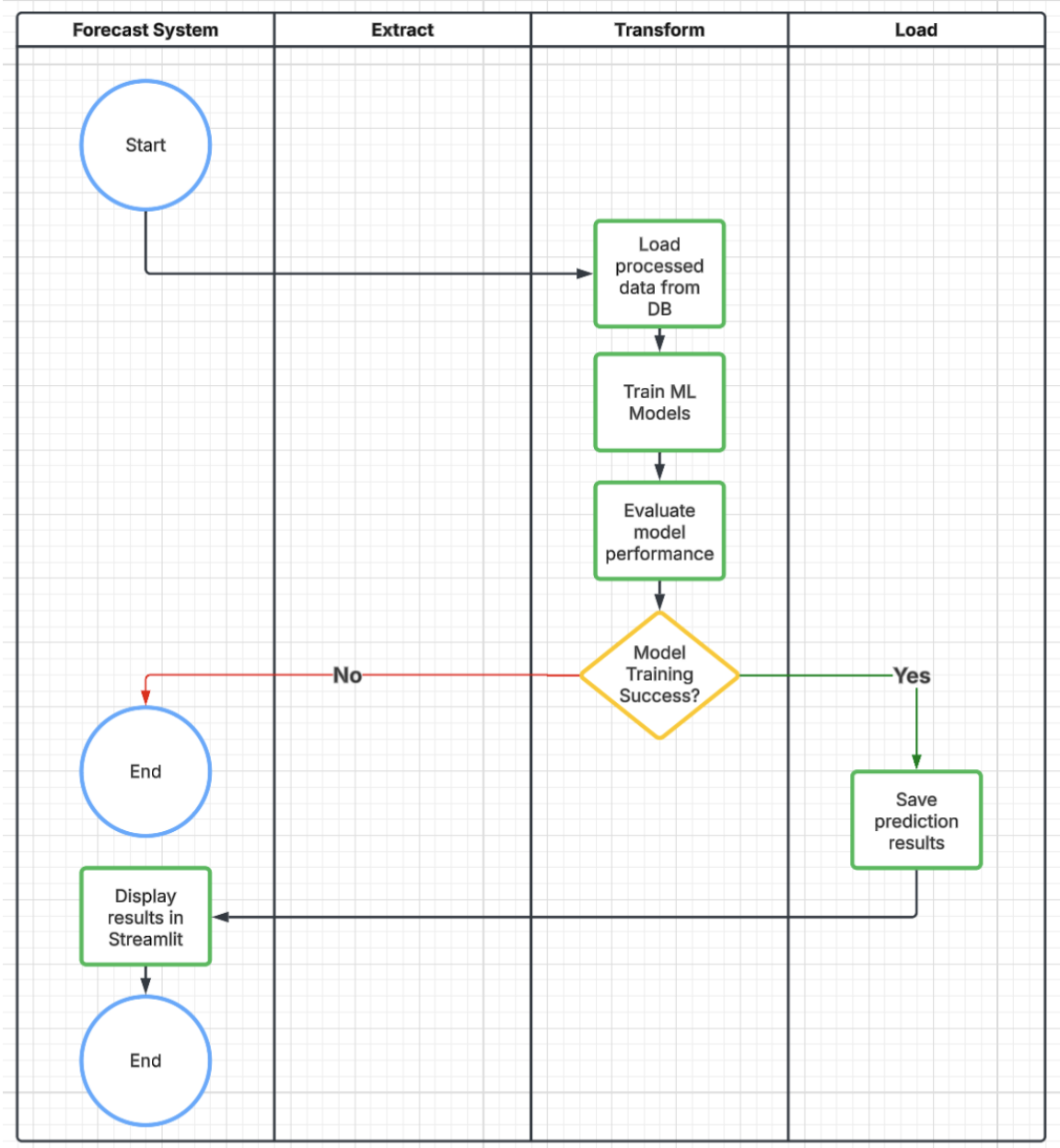


Figure 8. Forecast & Output Task Flow

4. Reporting Application Task Flow Design

This section covers the needs of the functional requirements and their associated functions. These needs are listed in the tables, and the applications task flow logic is shown in the swim lane diagram.

4.1 Data Exploration

[Design ID]: DI_4.1.1

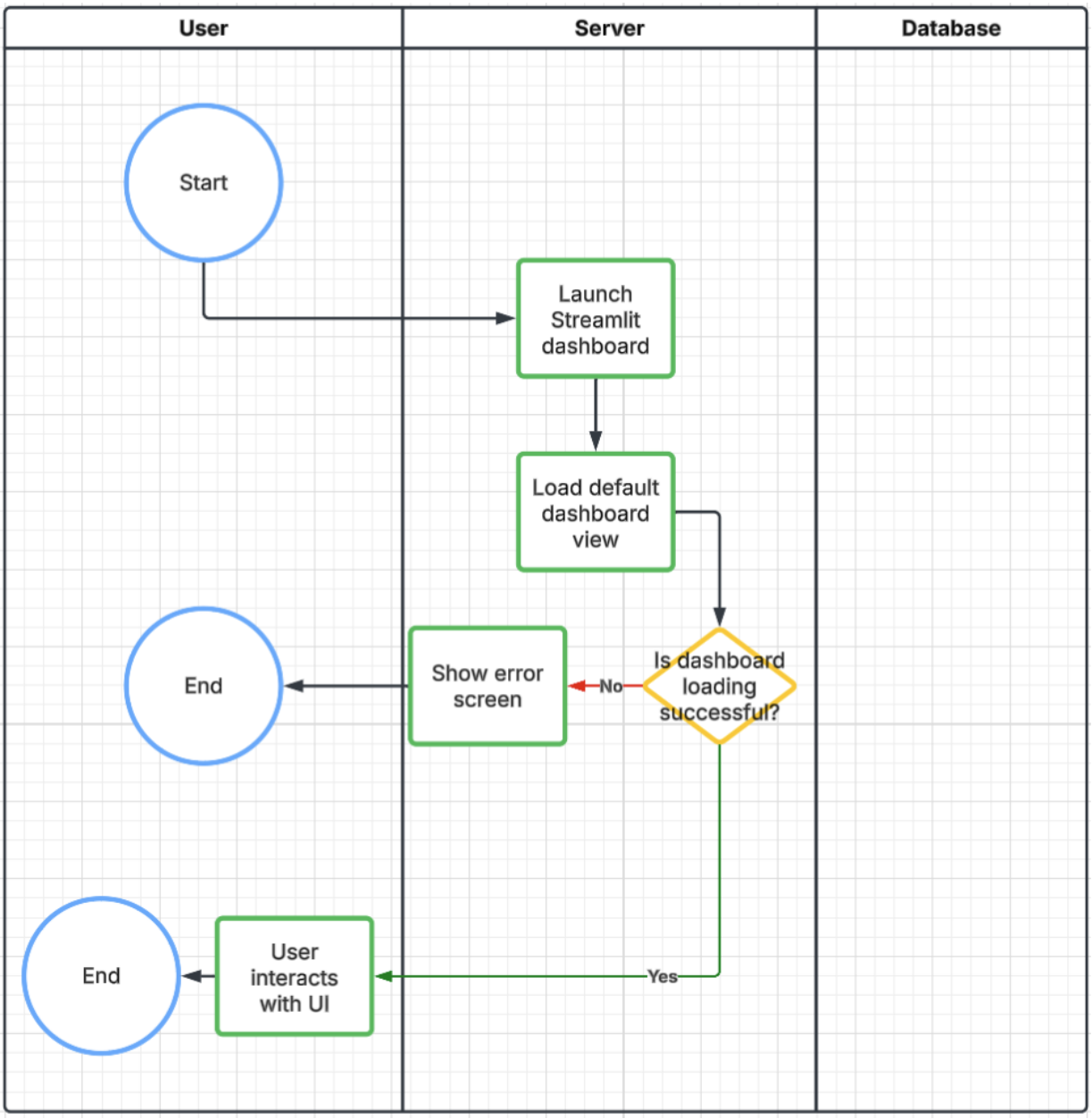


Figure 9. Data Exploration Flow

Case ID	Functional Requirements: FR.2.2.01
Case Name	Local Access Without Login

Case Description	Allows the user to access the dashboard locally without authentication. This improves accessibility and streamlines access for users who prefer local setups without credentials.
Basic Flow	- User launches the local Streamlit app. - The system starts and loads the dashboard interface. - The user interacts with visual elements and forecasts.
Alternatives	- If the system fails to load the dashboard correctly (e.g., missing files or port conflict), an error screen is displayed, and interaction is halted.
Assumptions	- The dashboard is only intended to be accessed on the local machine. - No user authentication or remote session is required.
Pre-conditions	- Streamlit is installed, deployed, and accessible via browser. - Local Python environment is operational.
Post-conditions	- The dashboard loads successfully. - Users can explore visualisations and real-time forecast components without any login barrier.

Table 3. Use Case Table: Local Access Without Login

4.2 Data Loading Pipeline

[Design ID]: DI_4.2.1

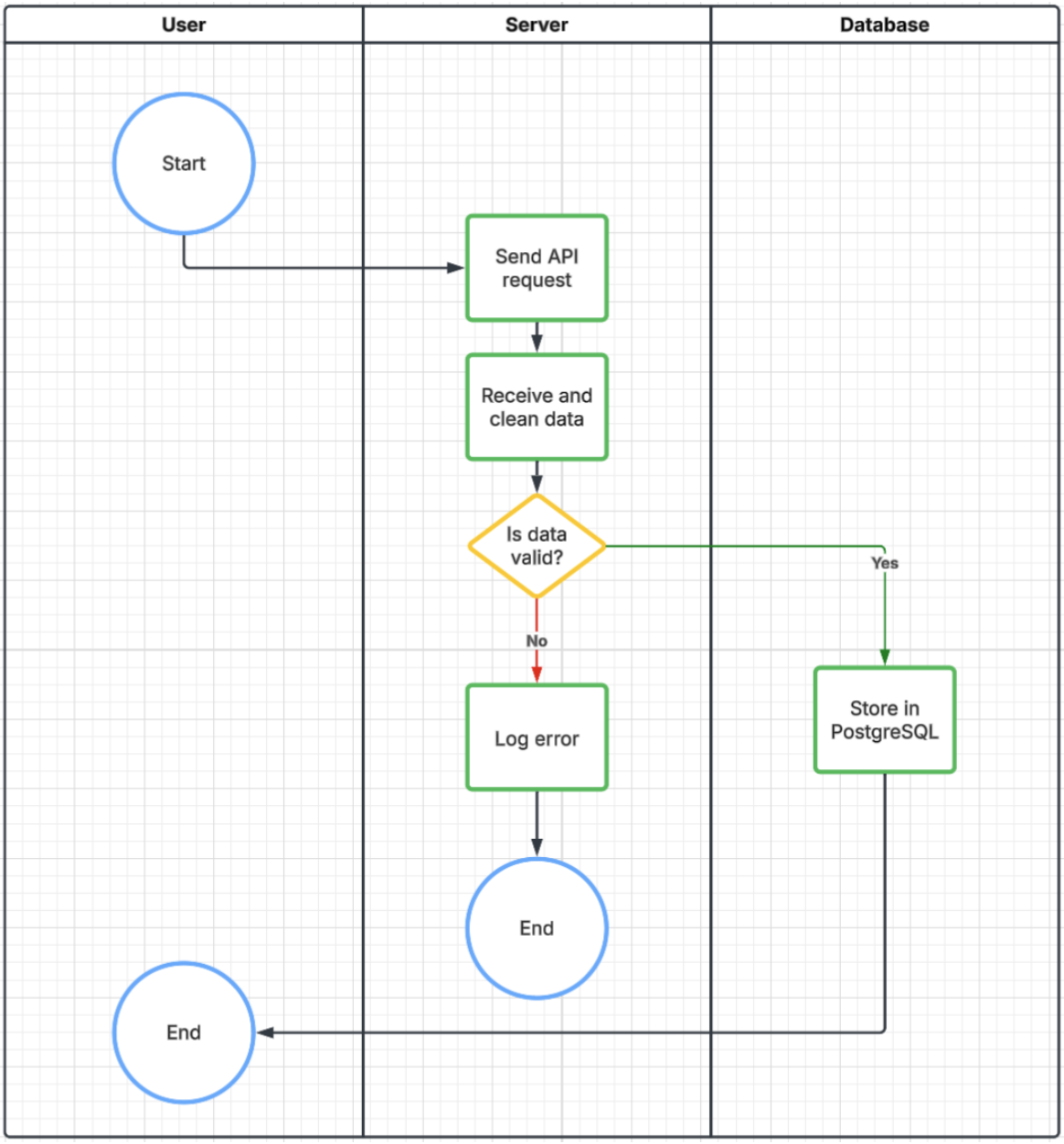


Figure 10. Real-Time data Fetching Task Flow

Case ID	**Functional Requirements: FR.2.2.02**
Case Name	Real-Time Data Fetching
Case Description	Periodically retrieves and stores market data (gold, Bitcoin, DXY) from Yahoo Finance APIs. This ensures up-to-date insights for model predictions and dashboard display.
Basic Flow	- Airflow triggers an API fetch task based on a schedule (e.g., hourly or daily). - The server sends a request to Yahoo Finance. - Data is received and cleaned (null checks, schema mapping). - Cleaned data is validated and stored in PostgreSQL.

Alternatives	- If the API is unreachable or data format is invalid, the system logs an error and skips the current fetch attempt. - Retry attempts may be configured via Airflow.
Assumptions	- Yahoo Finance APIs are online and reachable at the time of execution. - Data keys and endpoint structure are stable.
Pre-conditions	- Airflow DAGs are configured and enabled for periodic execution. - Internet connection is stable.
Post-conditions	- New market data is ingested and committed to PostgreSQL for future model training and visualisation.

Table 4. Use Case Table: Real Time Data Fetching

4.3 Model Training and Prediction

[Design ID]: DI_4.3.1

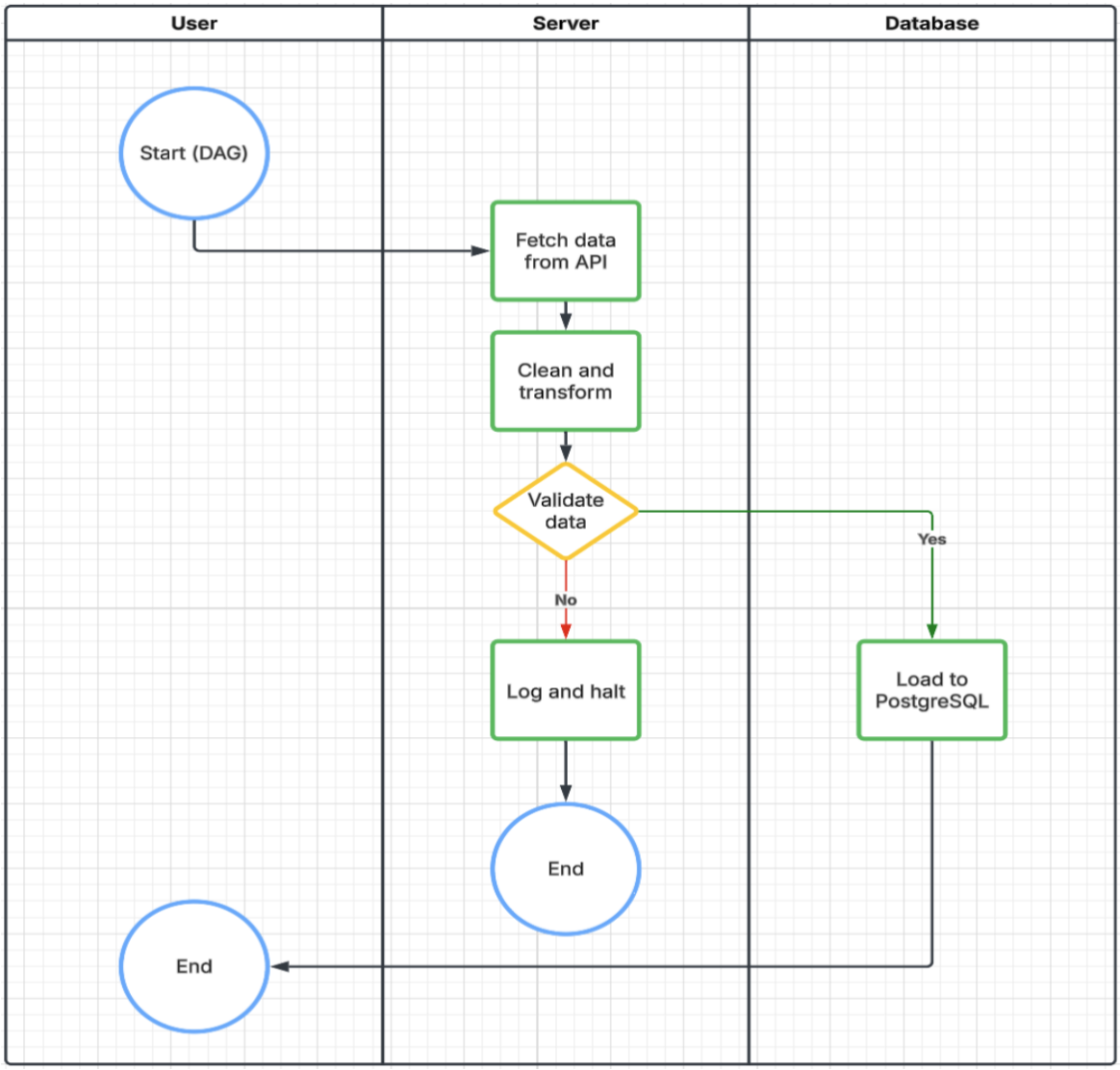


Figure 11. Automated ETL Pipeline Task Flow

Case ID	**Functional Requirements: FR.2.2.03**
Case Name	Automated ETL Pipeline
Case Description	Streamlines and automates the process of extracting raw financial data, transforming it, and loading it into the system database for storage and modelling.
Basic Flow	- An Airflow DAG initiates the ETL pipeline. - Server fetches data from APIs (e.g., gold, DXY, macro indicators). - Applies transformation (cleaning, renaming, null handling, feature extraction). - Validates structure (data types, missing values).

	- Clean data is pushed to PostgreSQL.
Alternatives	- If validation fails, the DAG halts and logs the issue. - The system notifies admin or retries depending on DAG settings.
Assumptions	- The pipeline scripts (Python, SQL) are complete and tested. - All transformation functions are reliable for the expected data shapes.
Pre-conditions	- ETL DAG must be active and scheduled. - Sources APIs must be functioning.
Post-conditions	- Validated and transformed data resides in PostgreSQL, ready for model prediction or visualisation.

Table 5. Use Case Table: Automated ETL Pipeline

4.4 Prediction

[Design ID]: DI_4.4.1

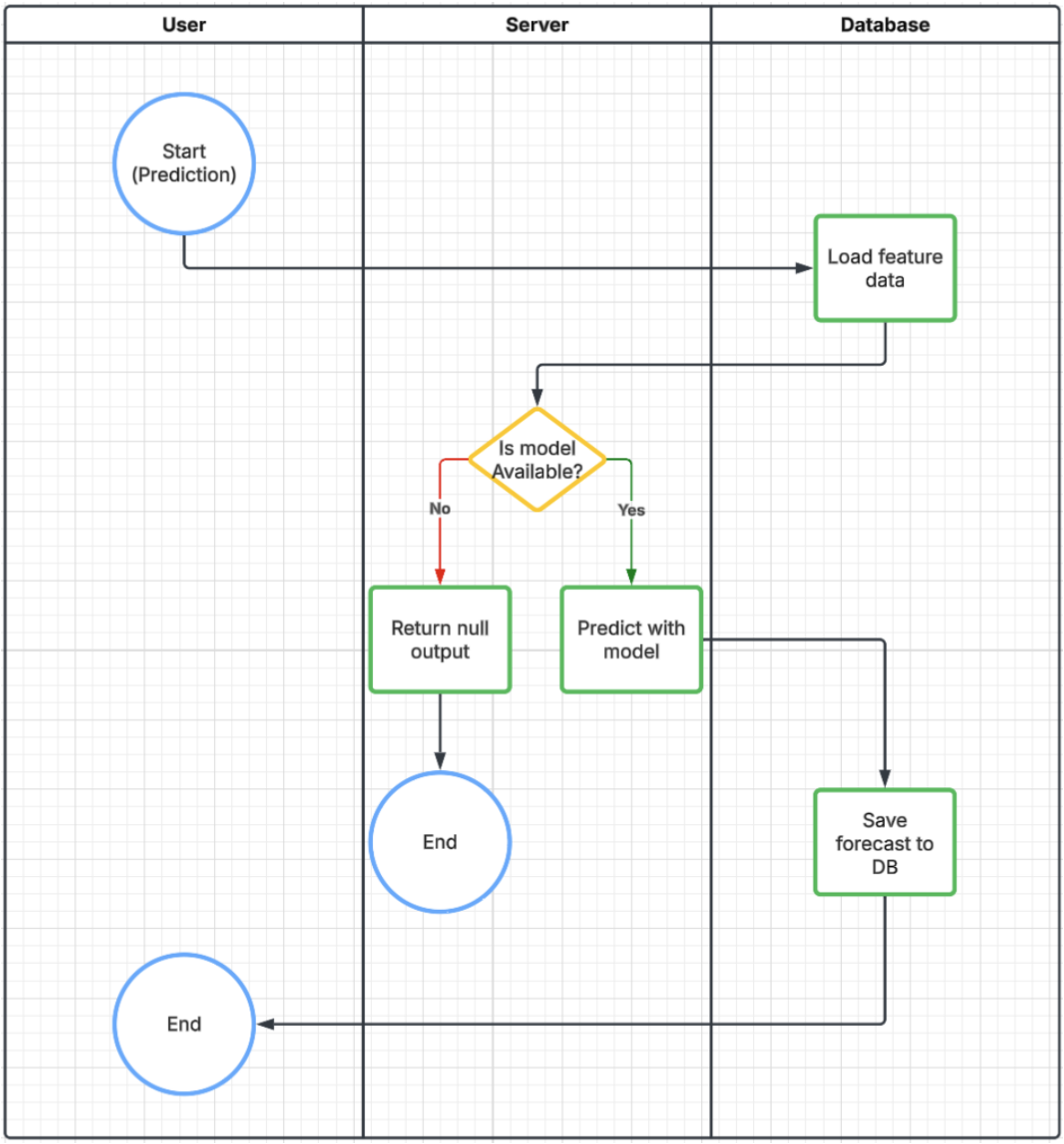


Figure 12. ML Prediction Task Flow

Case ID	**Functional Requirements: FR.2.2.04**
Case Name	ML Gold Price Prediction
Case Description	Uses a trained machine learning model to generate short-term forecasts for gold price movement based on financial indicators.
Basic Flow	- User or system scheduler triggers prediction. - Server queries the database for the latest features (Bitcoin, DXY, macro inputs). - Model predicts gold price for the next period. - Predicted value is written back to PostgreSQL.

Alternatives	- If the model is not available or input data is insufficient, the system returns a null output or placeholder message.
Assumptions	- The ML model has been trained, validated, and stored on disk or in memory. - Input features follow expected schema and ranges.
Pre-conditions	- Data exists in the PostgreSQL database for the required prediction window. - Model is loadable and operational.
Post-conditions	- Forecasted values are available in the database and ready for user access via dashboard or reporting.

Table 6. Use Case Table: ML Gold Price Prediction

4.5 Interactive Dashboard

[Design ID]: DI_4.5.1

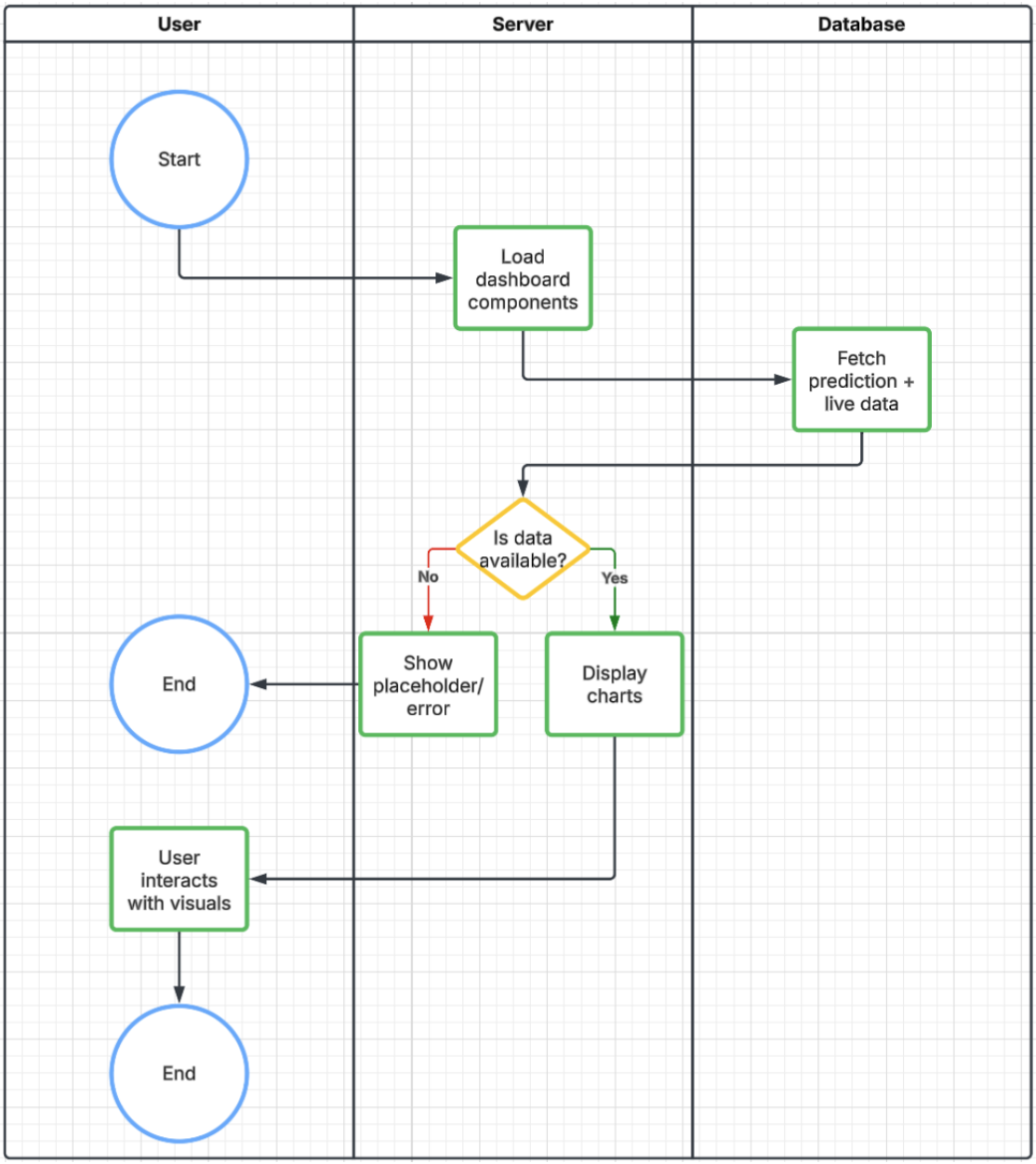


Figure 13. Interactive Dashboard Task Flow

Case ID	**Functional Requirements: FR.2.2.05**
Case Name	Interactive Dashboard
Case Description	Provides a real-time interface for users to explore gold price forecasts, economic indicators, and historical comparisons. Charts are interactive and dynamically updated.
Basic Flow	- User opens the dashboard via local browser. - Dashboard layout and widgets (tabs, filters, charts) are loaded by the server. - Database is queried for forecast data and real-time indicators.

	- If data is available, charts are generated and displayed. - User interacts with elements.
Alternatives	- If no data is available or query fails, dashboard shows fallback charts or error placeholders.
Assumptions	- Streamlit or other front-end is correctly connected to server and database. - Data is present and structured as expected.
Pre-conditions	- User launches the application. - ETL and ML pipelines have run at least once.
Post-conditions	- User is able to view and interact with all relevant visual elements. - Aids financial interpretation and insight gathering.

Table 7. Use Case Table: Interactive Dashboard

5. Interface Design

A user interface must exist to ensure that the data system is usable and open. For the GPFS, multiple interfaces are employed to support different user interactions. This section explains the design of these interfaces, particularly how users interact with the system in both data processing and reporting capabilities.

5.1 ELT Interface Design

The ELT (Extract, Load, Transform) interface is designed to support a modular and scalable pipeline for the Gold Price Forecasting System (GPFS). Unlike traditional ETL processes, the system loads raw data into storage first—then performs transformations later within the machine learning workflow. This architecture enhances flexibility for reprocessing, auditing, and iterative model development.

5.1.1 User Connects to Website and Views Data Insights

When the user connects to the website, the system automatically extracts data from external sources and loads it into the PostgreSQL table named gold_model_data. This process is handled during the system preparation phase and ensures that clean, structured data is available for visualisation.

Once this step is finalised, users accessing the website are presented with rich data insights, including correlation analysis and historical gold price trends. These insights are generated directly from the preloaded gold_model_data table, allowing the

dashboard to render visualisations instantly upon page load.

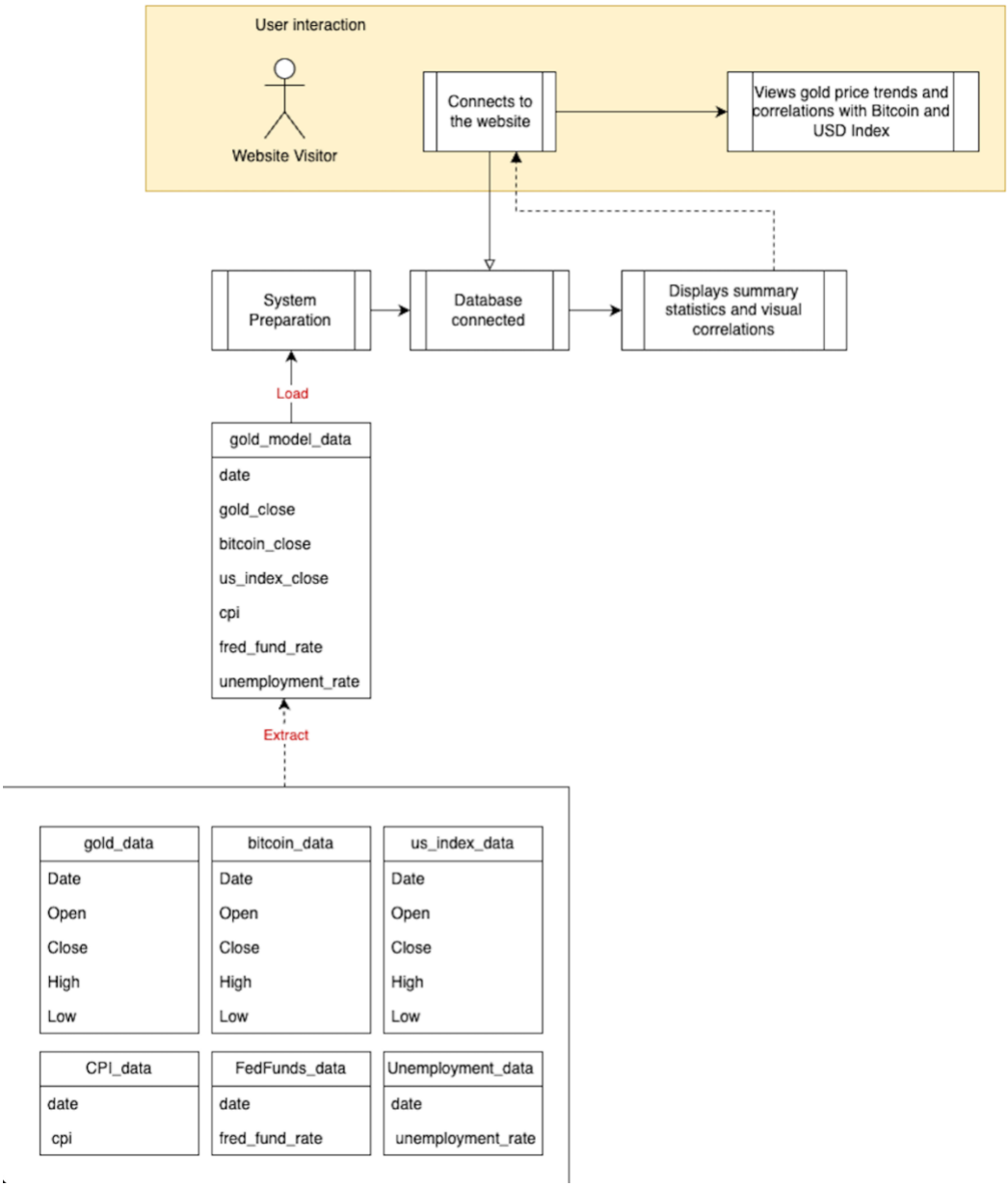


Figure 14. ELT Interface Flow: From Data Extraction to Instant Dashboard Insights

5.1.2 Model Training Trigger and Transformation Process

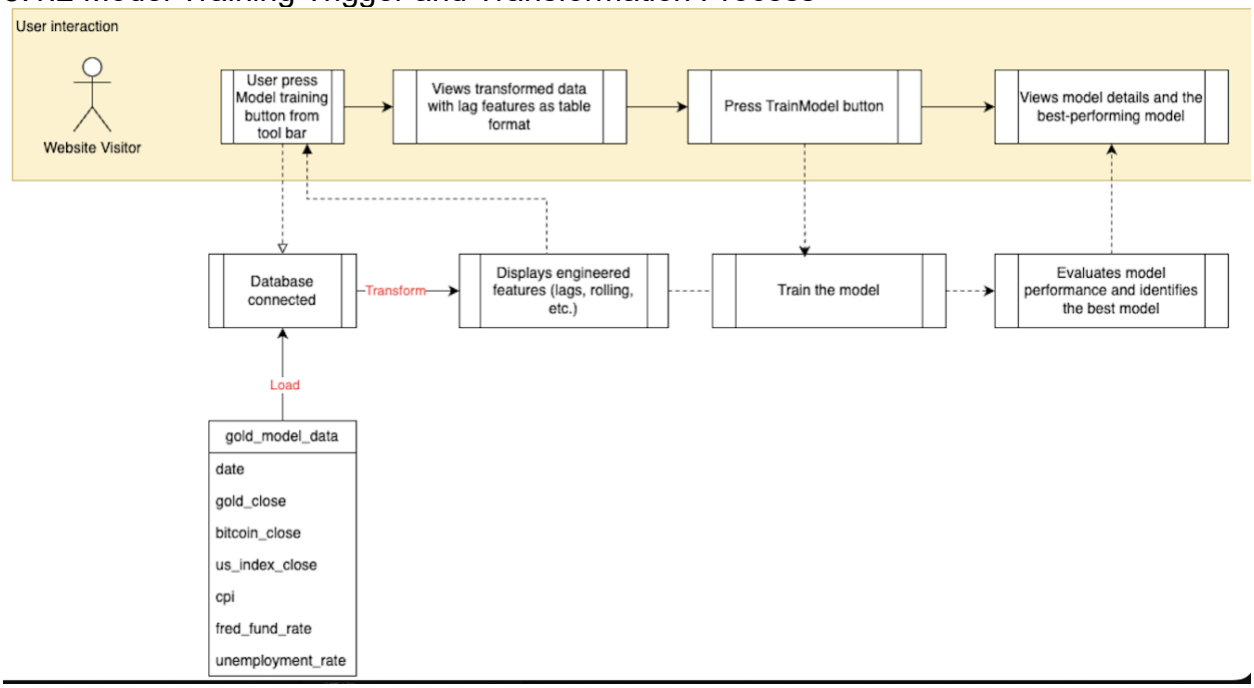


Figure 15. Model Training Process Triggered User and Transformed Data Display

When the user presses the Model Training button on the website, the system retrieves the cleaned dataset from the PostgreSQL table gold_model_data. It then applies feature engineering using Python, including 7-day lag features, rolling averages, and percentage changes. The transformed data is shown in table format, allowing the user to preview the processed input before training. After pressing the Train Model button, the system trains three machine learning models—Random Forest, Gradient Boosting, and XGBoost. Once training is complete, the system automatically evaluates model performance based on RMSE, MAE, and R^2, and highlights the best-performing model. The user can view the comparison results in both table and chart formats.

5.1.3 Prediction Execution Flow by Following User Selection

This section describes the execution flow of gold price prediction based on the user's model and forecast preferences. The process begins when the user presses the **Prediction** button from the navigation toolbar. The user can then select a trained machine learning model (saved in .pkl format) and specify the number of future days to predict.

After pressing the **Generate prediction** button, the system loads the selected .pkl model from the connected database and generates predictions. The results are presented as a 30-day line graph and a table showing predicted prices. Additionally, the interface provides a **Download Predictions as CSV** button, enabling the user to export the forecasted data for further analysis or record keeping.

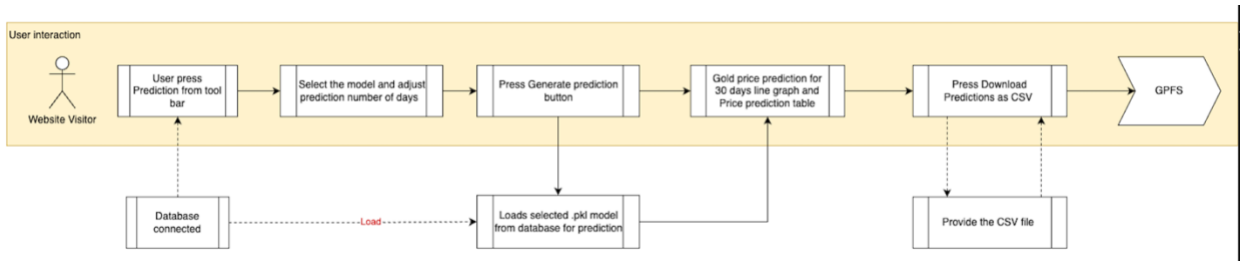


Figure 16. User-driven Prediction Process: From Model Selection to CSV Export

5.2 Application Navigation Design

This diagram presents the end-to-end navigation process of the gold price prediction application. After the database is successfully initialised, the user interacts with the dashboard to either explore existing prediction results or trigger new model training. Trained models are saved in .pkl format and registered back into the database, allowing updated predictions. The process supports continuous model improvement and user feedback integration.

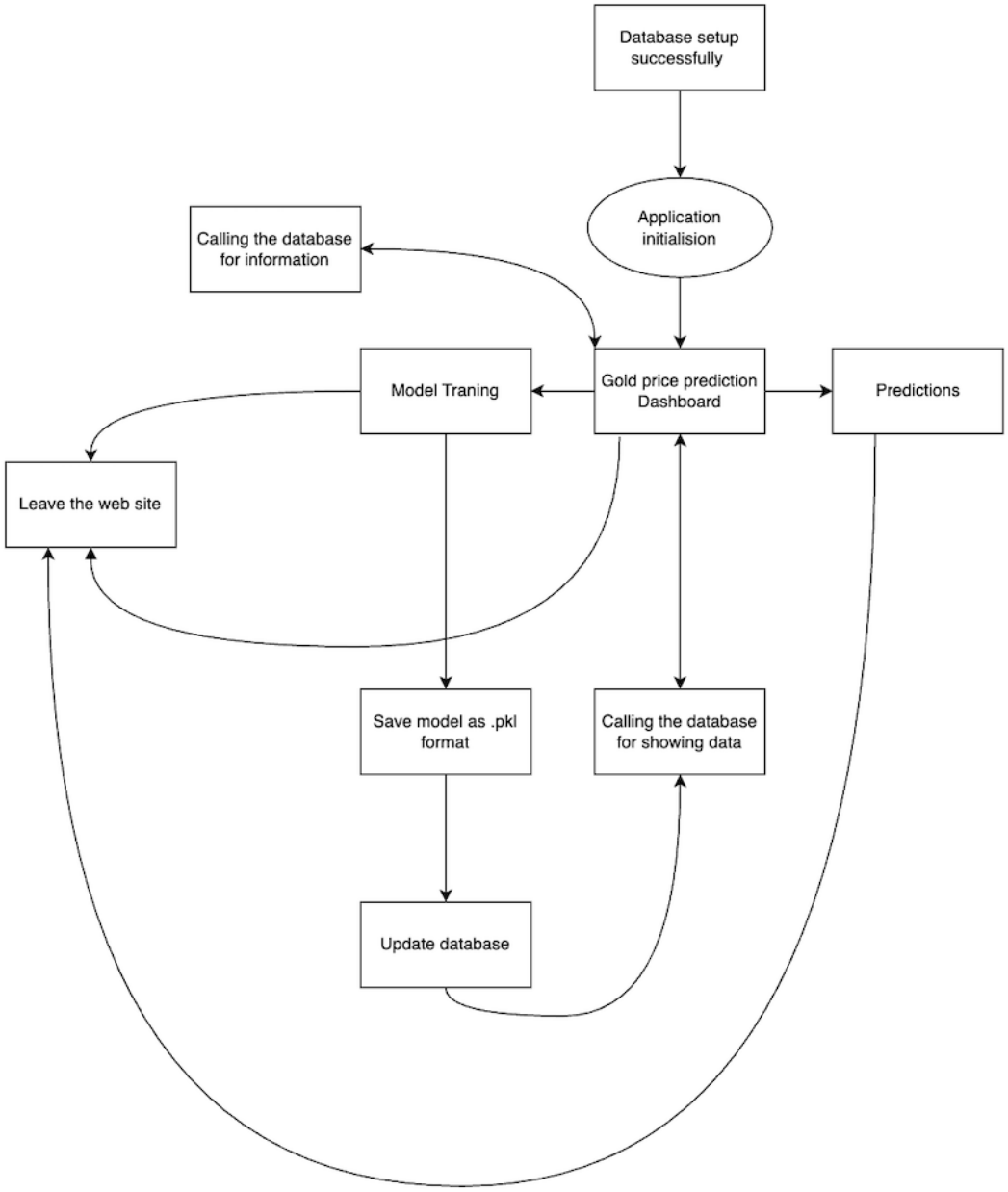


Figure 17. Application Navigation Flow: From Data Access to Forecast Generation

5.3 Application User Interface Design

5.3.1 Main Application Interface

This is the landing page displayed when users access the GPFS (Gold Price Forecasting System) website. The left-hand side toolbar shows three critical navigation options for the user. On the main page displays the historical trend of gold prices over time, offering insights into price fluctuations. In the implementation, the time series data

start from the year 2015 and extends to the present.

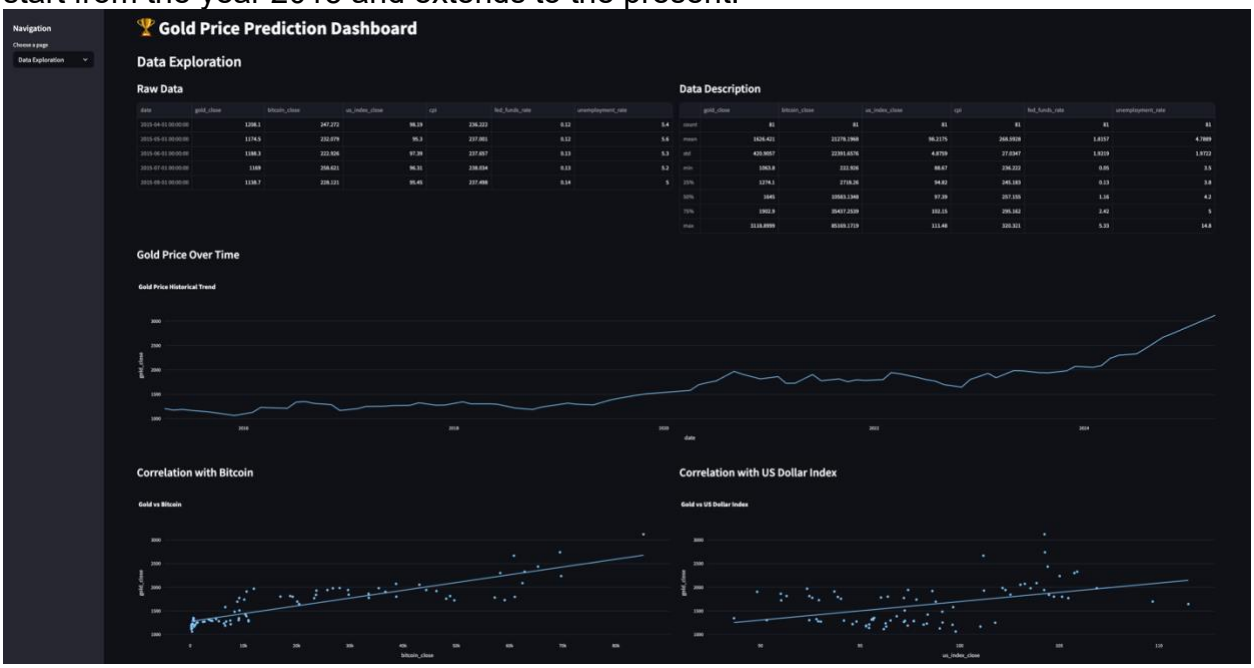


Figure 18. Main Application Interface Showing Historical Gold Price Trends

5.3.2 Interface of Navigation tool bar

The navigation toolbar provides users with streamlined access to the three main sections of the system. These sections include Data Exploration, Model Training, and Prediction. This interface is designed for intuitive interaction, enabling users to seamlessly switch between functionalities depending on their analytical or forecasting needs.

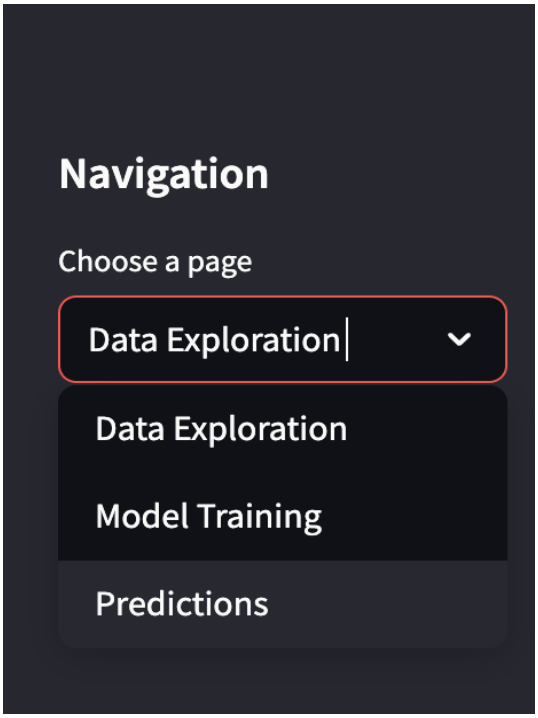
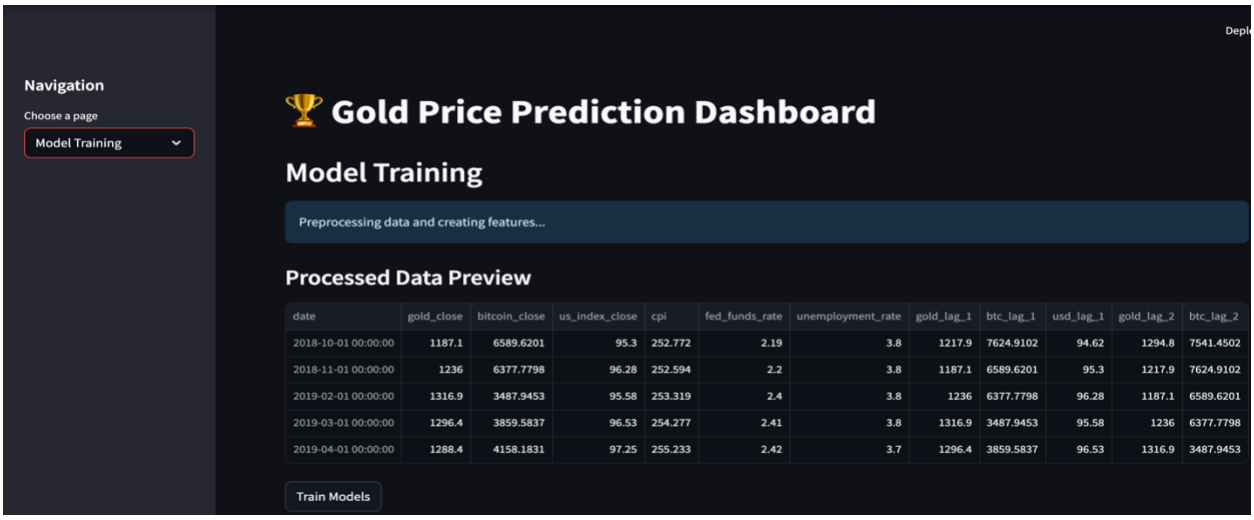


Figure 19. Interface of Navigation Toolbar: Three Main Analytical Sections

5.3.3 Model Training Interface

The Model Training Interface allows users to initiate the training process for three different machine learning models through a single action. By clicking the "Train Models" button, the system simultaneously trains all integrated models, streamlining the process for comparative evaluation and ensuring a consistent training workflow.



date	gold_close	bitcoin_close	us_index_close	cpi	fed_funds_rate	unemployment_rate	gold_lag_1	btc_lag_1	usd_lag_1	gold_lag_2	btc_lag_2
2018-10-01 00:00:00	1187.1	6589.6201	95.3	252.772	2.19	3.8	1217.9	7624.9102	94.62	1294.8	7541.4502
2018-11-01 00:00:00	1236	6377.7798	96.28	252.594	2.2	3.8	1187.1	6589.6201	95.3	1217.9	7624.9102
2019-02-01 00:00:00	1316.9	3487.9453	95.58	253.319	2.4	3.8	1236	6377.7798	96.28	1187.1	6589.6201
2019-03-01 00:00:00	1296.4	3859.5837	96.53	254.277	2.41	3.8	1316.9	3487.9453	95.58	1236	6377.7798
2019-04-01 00:00:00	1288.4	4158.1831	97.25	255.233	2.42	3.7	1296.4	3859.5837	96.53	1316.9	3487.9453

Figure 20. Model Training Interface with One-Click Execution for All Models

5.3.4 After interface when finish model train

After the user initiates model training, the system displays a comparative evaluation of the trained models. As shown in Figure 21, the interface shows a summary table that includes key performance metrics ; **RMSE, MAE (Mean Absolute Error), and R² Score** for each model (Random Forest, Gradient Boosting, and XGBoost.)

A status message confirms completion of the training process and identifies the best-performing model based on predefined criteria. This visual comparison allows users to easily interpret results and make informed decisions regarding which model to deploy for predictions.

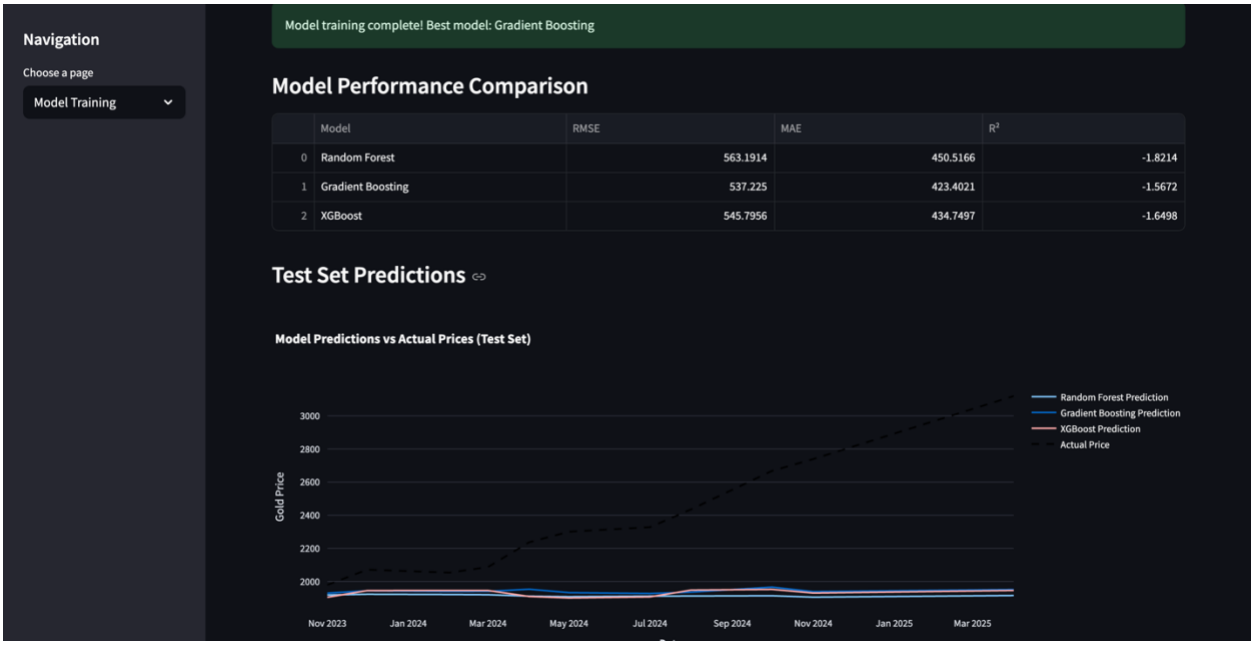


Figure 21. Model Performance Comparison Table in the Post-Training Interface

5.3.5 Prediction Interface

The Prediction Interface allows users to generate gold price forecasts by selecting one of the pre-trained machine learning models. As shown in Figure 22, users can choose from a dropdown menu that includes random_forest, gradient_boosting, and xgboost. Once a model is selected, the user clicks the "Generate Predictions" button to initiate the forecasting process. The system then applies the chosen model to the test data and displays the predicted gold prices. This design enables flexible model comparison and supports informed decision-making based on model performance.

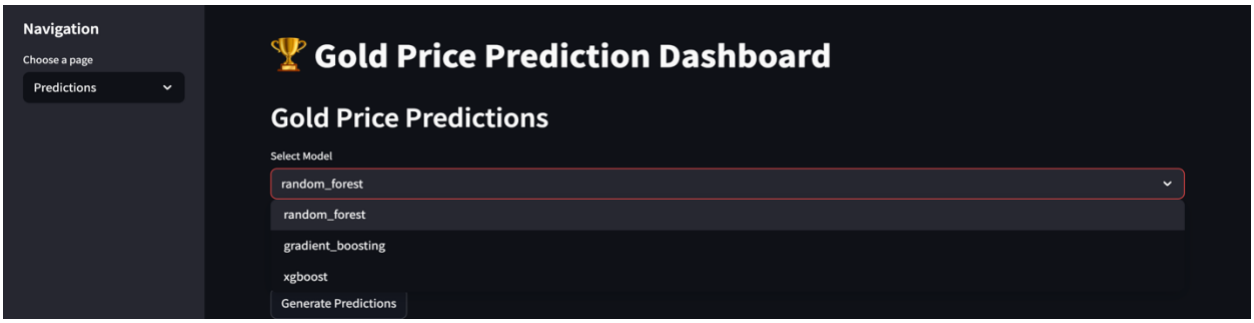


Figure 22. Prediction Interface with Dropdown Model Selection and Output Generation

5.3.6 Result of Prediction

After selecting the model, users can specify the number of future days to predict—ranging from 1 to 365 days. Once the prediction is generated, the interface displays a time series line chart, showing both historical gold prices and the predicted values for the selected future period. The chart clearly distinguishes between observed and forecasted data using color-coded lines, allowing users to visually assess the trend and model behavior. Additionally, the predicted results are presented in a tabular format and can be conveniently downloaded as a CSV file for further analysis or reporting purposes.

Figure 23. Prediction Output Display with Time Series Plot and Downloadable Results

6. Hardware and Software Tools

The GPFS is lightweight and can operate in typical computing environments. However, for it to realise the best possible performance, scalability, and reliability, the utilisation of specific hardware and software tools is recommended. These tools enhance the

efficiency of the ETL process, enable secure storage of data, and provide easy reporting through the user interface.

6.1 Hardware Tools

6.1.1 For the User

GPFS can run on a lightweight web browser-friendly platform. As such, users only require a basic computer or laptop with a modern web browser and Python installed. There is no requirement for a specialised GPU or a cloud-computing instance. This setup allows for convenience for a person like an analyst, economist, or investor so they can interact from their homes, offices, or any secure location without elaborate setups. The system runs on the Streamlit interface and bypasses the necessity of authenticating the user, thus allowing instantaneous exploration of the data and projections.

6.1.2 For the AI and ETL Processes

Most operations that make large computational demands—e.g., scheduling ETL operations, transformation of large datasets, and machine learning model training run on the server side or as part of the local set-up process through Apache Airflow and Python. Operations run on a machine with sufficient computational resources (e.g., a multi-core CPU machine with at least 8GB of RAM). Although the system may be capable of scaling out to cloud infrastructure in production setups, it is also possible to run it locally through Docker or virtual setups for research or personal use.

MinIO serves as a local object store solution mimicking cloud-blob store behavior to handle raw datasets. PostgreSQL serves as the relational database system for storing transformed data and prediction outputs, ensuring fast queries and long-term data persistence.

6.2 Software Tools

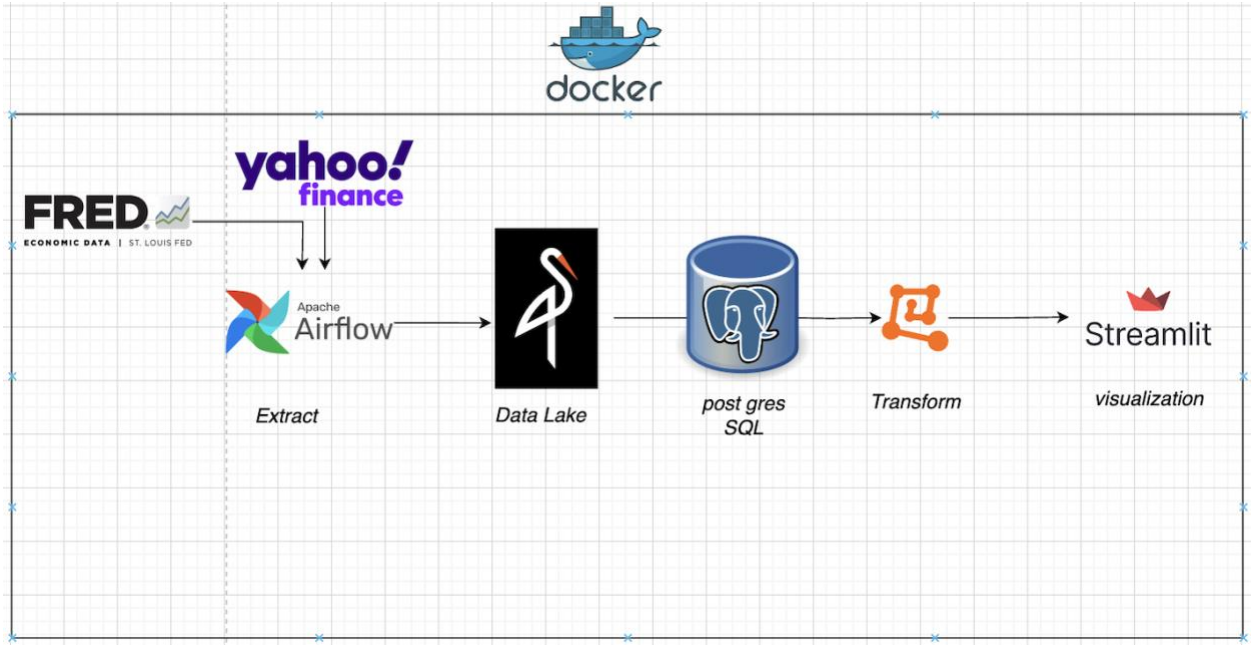


Figure 24. Overview of Software Tools Used in GPFS

Tool/Technology	Role in System	Description
Python	Core programming language	Python is used for data collection (via APIs), transformation, machine learning model development, and visualisation scripting. Libraries such as pandas, requests, scikit-learn, xgboost, and matplotlib form the backbone of data operations.
Apache Airflow	ETL Orchestration	Manages and schedules the automated ETL pipeline, ensuring fresh data from Yahoo Finance and FRED APIs are ingested and processed daily.
MinIO	Data Lake	Acts as a local data lake for raw CSV data (e.g., gold, bitcoin, CPI), providing traceability, fault tolerance, and scalability similar to cloud blob storage.
PostgreSQL	Data Warehouse	Structured storage of transformed data, predictions, and model evaluation metrics. Data is formatted in a star schema for efficient analytics.
FRED & Yahoo Finance API	External Data Sources	Used to fetch macroeconomic indicators and financial market data in real-time, powering forecasting models with up-to-date inputs.
Streamlit	Reporting and Dashboard UI	Builds the user-facing web interface, allowing real-time visualisations (trendlines, correlation plots), model training comparison (RMSE, MAE, R^2), and prediction display.

Table 8. Software Tools and Their Roles in the System

7. System Testing

7.1 Final System URL

To access the completed GPFS, follow the GitHub link provided:

https://github.com/JJaehEE1107/datasystem_2025

7.2 Test Procedure

Instructions for testing the system are below:

- Step 1: Check that all essential applications for the environment are installed
- Step 2: Download GitHub

- Step 3: Download all required dependencies through the terminal listed in the requirements.txt file
- Step 4: Create a .env file which contains the details for the PostgreSQL, FRED API, and Minio
- Step 5: Run the app.py application in the terminal
- Step 6: Follow the test cases with the needs of the environments met
- Step 7: Document all test cases in a neat structure
- Step 8: Close the program once testing is complete
- Step 9: Create a summary of the findings that were found

7.3 Test Cases

7.3.1 Test Case 1

Test Case Identifier	Test Case 1
User Requirement	FR 2.2.01
Test items	The system should allow users to acess the dashboard locally without authentication.
Environmental Needs	- The application is operational. - Streamlit is running locally.
Special procedural requirements	- No login or user data entry is required. - Dashboards must load automatically.

Table 9. Testing Case 1

7.3.2 Test Case 2

Test Case Identifier	Test case 2
User Requirement	FR 2.2.02
Test items	- System retrieves financial data (Gold, Bitcoin, DXY) from APIs at scheduled intervals. - Retrieved data is stored in PostgreSQL and timestamped correctly.
Environmental Needs	- Airflow DAG is active and scheduled. - API endpoints (Yahoo Finance) are accessible. - Database is operational.
Special procedural requirements	- Datetime format should be current and from the second. - Data should reflect current or most recent available records.

Table 10. Testing Case 2

7.3.3 Test Case 3

Test Case Identifier	Test case 3
User Requirement	FR 2.2.03
Test items	- Each step of the ELT pipeline (extraction, loading, transformation) should be executed automatically. - Raw data is fetched, cleaned, and stored without manual intervention.
Environmental Needs	- All ELT components (Airflow scheduler, scripts, PostgreSQL database) must be operational.
Special procedural requirements	- No user input required; job execution is fully automated.

Table 11. Testing Case 3

7.3.4 Test Case 4

Test Case Identifier	Test case 4
User Requirement	FR 2.2.04
Test items	- ML model should generate future gold price predictions using inputs such as Bitcoin price, DXY index, and macroeconomic indicators. - Model outputs must be saved to the database.
Environmental Needs	- ML model file must be pre-trained and loaded. - Database must be connected and accessible.
Special procedural requirements	- Model training must have been completed prior to execution. - Input features must match expected schema.

Table 12. Testing Case 4

7.3.5 Test Case 5

Test Case Identifier	Test case 5
User Requirement	FR 2.2.05
Test items	- Forecasts, real-time financial data, and recommendations should be displayed interactively on the dashboard. - Visual components must respond to user inputs (e.g., filters, toggles).
Environmental Needs	- Streamlit dashboard must be running.

	- Backend database must have forecast, and historical data loaded.
Special procedural requirements	- Forecasting and ETL pipelines must be completed successfully before launching.

Table 13. Testing Case 5

7.4 .1 Test results

Test Case ID	Test Status (Pass/Failed)	Incident	Tester	Test Date
Test Case 1	Pass	NTR	James	08/05/25
Test Case 2	Pass	NTR	James	08/05/25
Test Case 3	Pass	NTR	James	09/05/25
Test Case 4	Pass	NTR	James	09/05/25
Test Case 5	Pass	NTR	James	10/05/25

Table 14. Test Summary Results for All Cases Executed

7.4.2 Test Incident Report

Test Incident Identifier	Test Case Failed	Brief Description
Test Case 1	N/A	No incidents were identified during the execution of this test case. All test steps were completed successfully, and system behavior matched the expected outcomes.
Test Case 2	N/A	No incidents were identified during the execution of this test case. All test steps were completed successfully, and system behavior matched the expected outcomes.
Test Case 3	N/A	No incidents were identified during the execution of this test case. All test steps were completed successfully, and system behavior matched the expected outcomes.

Test Case 4	N/A	No incidents were identified during the execution of this test case. All test steps were completed successfully, and system behavior matched the expected outcomes.
Test Case 5	N/A	No incidents were identified during the execution of this test case. All test steps were completed successfully, and system behavior matched the expected outcomes.

Table 15. Test Incident Reports and Observations

7.4.3 Test Summary

This test summary looks at the tests conducted for the GPFS. A total of 5 test cases were executed, with 5 of 5 passing. However, due to time constraints, there were adjustments made to the system that reduced the number of tests required and thus the scope for the development of the product. The plans to improve the dashboard and customize it will be tested in future development once those features are realized. Those are mainly due to alterations to the ERD diagram in its updated form. After performing tests all the mandatory test cases passed. Meaning the functionality of the application was secured, and it is entirely operational.

8. Future Development

As part of the system evolution roadmap, we identified two high-impact enhancements for future development: automated daily summary reporting and customisable visualisation selection.

8.1 Daily Summary Forecast Reporting

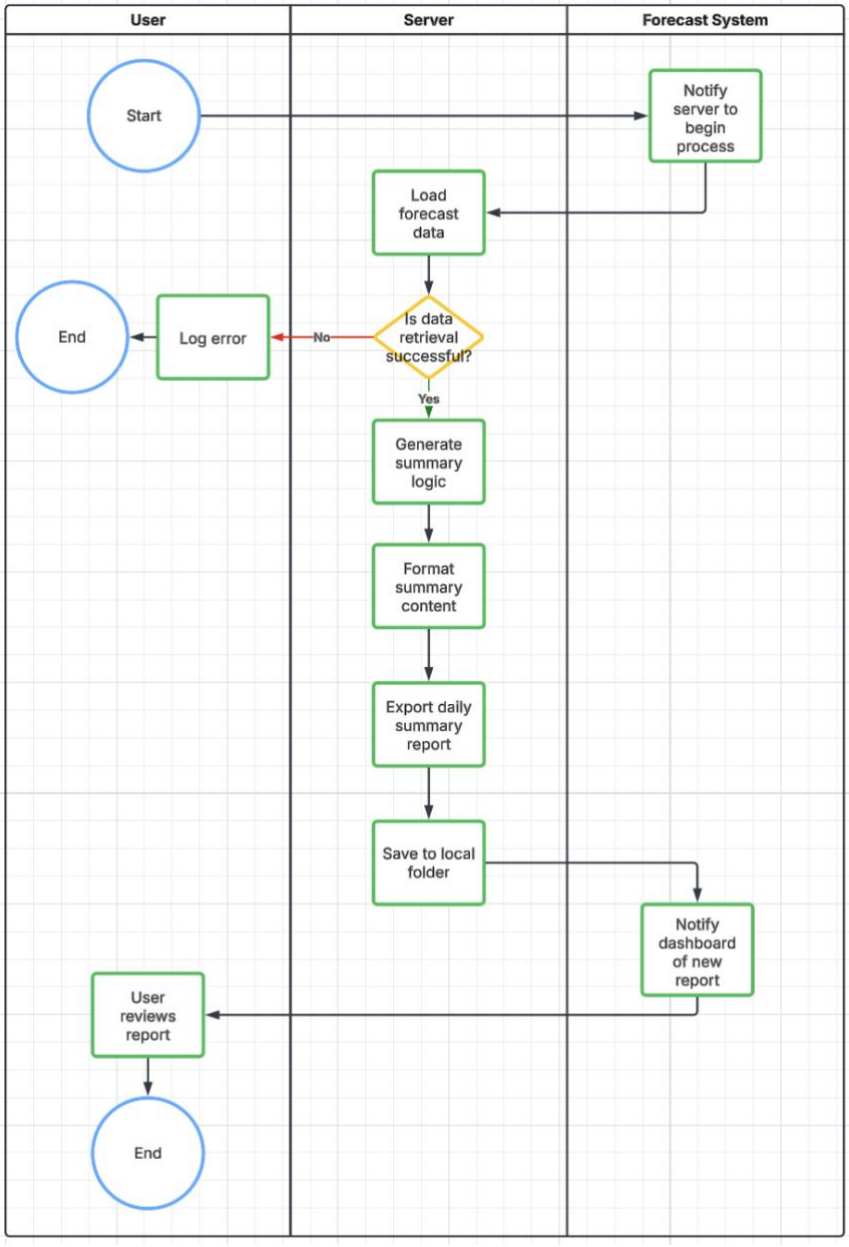


Figure 25. Daily Forecast Summary Report: Workflow and Output Access

Case ID	**Functional Requirements: FR 2.2.06**
Case Name	Daily Forecast Summary Report
Case Description	This feature automatically generates a daily summary report that presents forecast trends, economic signals, and action insights in plain language. It supports users who may find interpreting charts or raw data challenging.

Basic Flow	- The Forecast System triggers a scheduled task (e.g., via Airflow). - Forecast System notifies the Server to begin data retrieval. - Server loads gold, bitcoin, DXY, and other economic data from PostgreSQL. - If successful, Server generates the summary logic. - Summary content is formatted (e.g., price movement, action insight). - A summary file (TXT, HTML, or PDF) is exported and saved. - Forecast System notifies the dashboard of new file availability. - User accesses the report via the Streamlit dashboard.
Alternatives	- If PostgreSQL is unreachable or data is invalid, the system logs an error and halts summary generation. - If the model output is missing, fallback text (e.g., "Forecast unavailable") is inserted.
Assumptions	- The data and model predictions exist in PostgreSQL at the time of scheduled execution. - The user accesses the system locally. - The Streamlit dashboard is live and connected to local storage.
Pre-conditions	- Forecast System scheduler (e.g., Airflow) is running and properly configured. - At least one successful model training cycle has occurred.
Post-conditions	- A summary report is generated and saved in a local folder. - The user sees and optionally downloads the summary from the dashboard.

Table 16. Use Case Specification: Daily Forecast Summary Report

8.2 Customisable Visualisation Mode

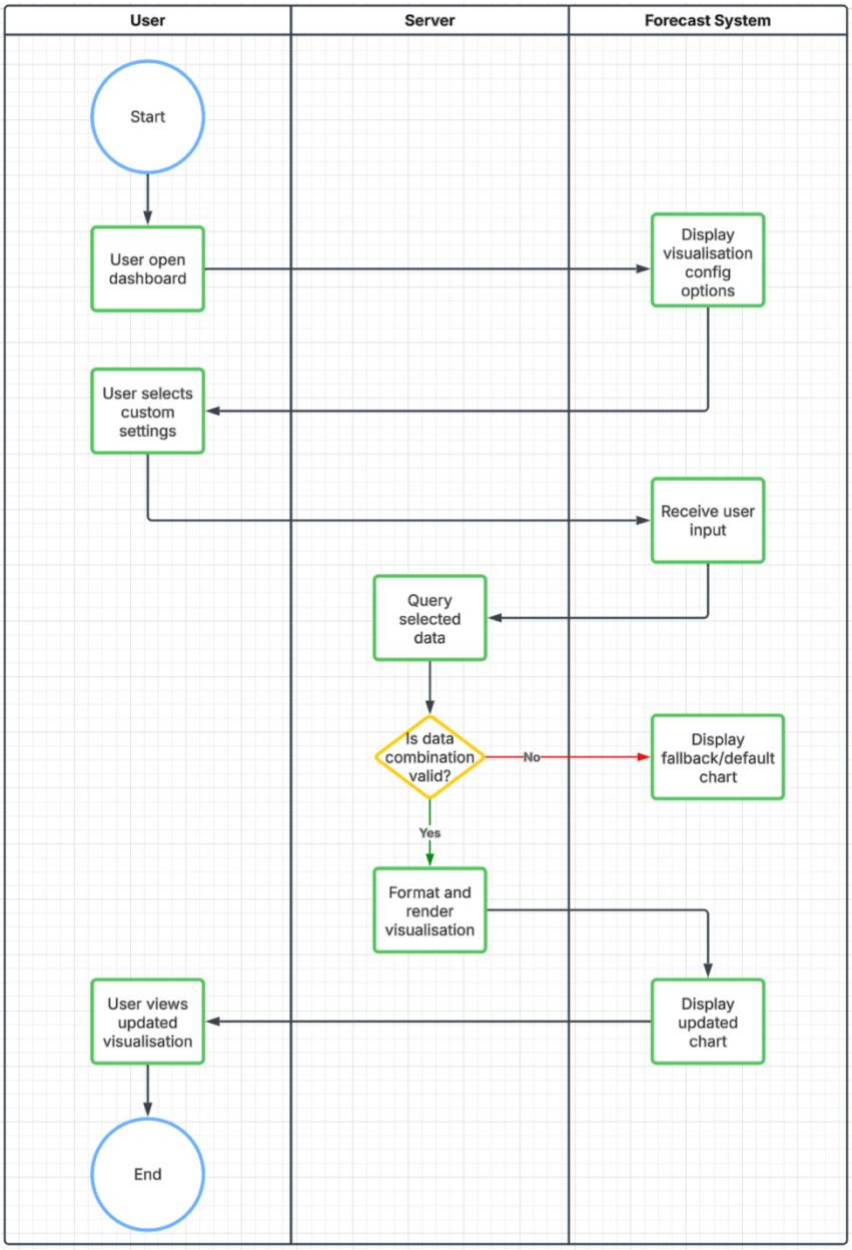


Figure 26. Customisable Visualisation Mode: Variable & Chart Type Selection

Case ID	**Functional Requirements: FR 2.2.07**
Case Name	Customisable Visualisation Mode
Case Description	This feature allows users to personalise their dashboard by selecting variables (e.g., Gold vs. Bitcoin), chart types (e.g., line, bar), and time frames (e.g., daily, monthly). It promotes flexible data exploration for users with different interests and levels of expertise.
Basic Flow	- User accesses the dashboard.

	- The system displays interactive visualisation settings. - User selects variables (e.g., Gold, CPI), chart type, and time range. - Server queries PostgreSQL for selected variables. - Server formats the data and renders the selected chart in real-time. - User sees the updated chart with custom settings.
Alternatives	- If incompatible variables are selected (e.g., different time resolutions), the system shows an error or default fallback chart. - If no user input is given, the dashboard defaults to the standard configuration.
Assumptions	- The data for selected variables exists and is regularly updated in PostgreSQL. - The dashboard UI supports dynamic chart rendering via Plotly/Matplotlib in Streamlit.
Pre-conditions	- The user is actively viewing the dashboard. - The dashboard is connected to the data backend and functioning properly.
Post-conditions	- The user sees a visualisation based on their selected preferences, enabling deeper data-driven exploration.

Table 17. Use Case Specification: Customisable Visualisation Mode

9. Conclusion

This report outlines the end-to-end setup of a Gold Price Forecast System, from design architecture to ETL pipelines, machine learning incorporation, and an interactive reporting dashboard. It was developed with Python, Streamlit, Apache Airflow, and PostgreSQL, and incorporates real-time financial indicators such as Gold, Bitcoin, and the US dollar index. The reporting and data flow designs were structured around functional requirements and visualised through swimlane diagrams, ERD models, and detailed use case tables. The system thus incorporates automated data ingestion, model-based prediction, and interactive user-facing dashboards, all tested and certified in a local runtime environment.

Some limitations were envisioned during the process, and areas for improvement have been listed in the future development section. While the current release includes support for the prediction of gold prices using various ML algorithms, enhancements in the generation of daily summaries and visualisation modes with customisation are planned to make it even more user-friendly and flexible. Overall, the system is a compromise between tutor guidance, technical feasibility, and team innovation with a scalable financial forecasting solution based on state-of-the-art data engineering principles.

10. Appendix

Appendix A – Informational Requirements

Requirement ID	Description	Tool/Technology	Priority
IR.2.1.01	The system should collect real-time gold, Bitcoin, and US Dollar Index data from the Yahoo Finance API.	Yahoo Finance API	Mandatory
IR.2.1.02	The system should schedule and manage automated ETL workflows for all data sources including FRED.	Apache Airflow	Mandatory
IR.2.1.03	Transformed and enriched data should be stored in PostgreSQL for long-term analysis.	PostgreSQL	Mandatory
IR.2.1.04	The system should support historical gold price forecasting using a machine learning model trained on multi-source data.	Python, XGBoost	Mandatory
IR.2.1.05	Forecast results, sell recommendations and US dollar strength should be displayed to end-users through an interactive web UI.	Streamlit	Mandatory
IR.2.1.06	Visualizations should reflect both real-time and historical gold, Bitcoin, and economic indicator trends.	Plotly / Matplotlib	Recommended
IR.2.1.07	The system should allow users to export predictions and historical data.	CSV, JSON Export	Recommended
IR.2.1.08	Role-based access should control user permissions (admin, analyst, investor).	Streamlit	Recommended
IR.2.1.09	Data pipeline logs, model performance, and error metrics should be stored and monitored.	Airflow Logs	Recommended

IR.2.1.10	External economic indicators (CPI, Fed Funds, Unemployment) from FRED should be integrated for forecasting.	FRED API (fredapi)	Mandatory
IR.2.1.11	Preprocessed data should be stored in PostgreSQL in a structured schema for prediction and analytics	PostgreSQL	Mandatory

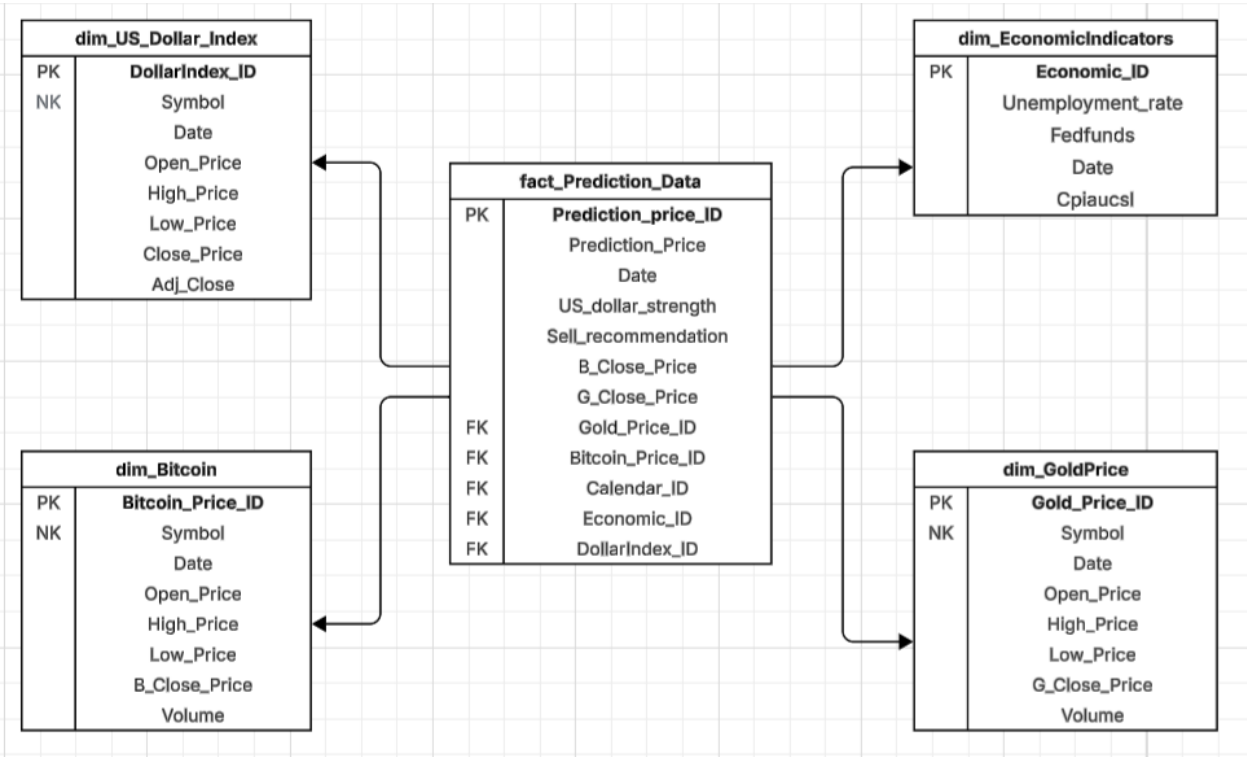
Appendix B – Functional Requirements

Requirement ID	Description	Priority
FR.2.2.01	The system should allow open local access without requiring login or registration. Role-based personalisation is not implemented to protect user privacy and simplify access.	Mandatory
FR.2.2.02	The system should fetch Real-time gold, Bitcoin, and DXY data from Yahoo Finance at regular intervals.	Mandatory
FR.2.2.03	An Airflow-based ETL pipeline should automate data fetching, cleaning, transformation, and storage.	Mandatory
FR.2.2.04	The ML model should predict future gold closing prices based on inputs like Bitcoin, DXY, and economic indicators.	Mandatory
FR.2.2.05	The system should be shown on an interactive dashboard the Forecast trends, real-time data, and recommendations.	Mandatory
FR.2.2.06	The system should automatically generate a daily summary report including key trends and predictions.	Recommended
FR.2.2.07	The system should allow users to customise the type of visualisations they wish to view on demand.	Recommended

Appendix C – Non-Functional Requirements

Requirement ID	Description	Purpose	Priority
NFR.2.3.01	The system must ensure encrypted transmission and secure storage of financial and user data.	Security	Mandatory
NFR.2.3.02	The system should support up to 100,000 requests per day without performance degradation.	Performance	Mandatory
NFR.2.3.03	99.9% uptime is required for continuous access to real-time market data.	Availability	Mandatory
NFR.2.3.04	The system should be scalable to support additional financial instruments in the future (e.g., ETFs).	Scalability	Recommended
NFR.2.3.05	Errors in ML predictions or pipeline failures should be logged and fixed within 2 business days.	Maintainability	Recommended
NFR.2.3.06	Real-time data should be updated every 5 minutes to reflect market volatility.	Real-Time Processing	Recommended
NFR.2.3.07	The UI must be intuitive and user-friendly, requiring minimal training for analysts and investors.	Usability	Mandatory
NFR.2.3.08	Maintenance should be scheduled during non-peak hours to reduce user impact.	Maintainability	Recommended
NFR.2.3.09	The UI should be accessible across multiple devices (desktop, tablet, mobile).	Usability	Recommended
NFR.2.3.10	Any corrupted or missing data should be flagged and either corrected automatically or reported to an admin.	Data Accuracy	Mandatory

Appendix D – Entity Relationship Diagram



Appendix E – Entity Key Relationship Table

Entity	Primary Key (PK)	Foreign Key (FK)
fact_Prediction_Data	Prediction_price_ID	User_ID, Gold_Price_ID, Bitcoin_Price_ID, Calendar_ID, Economic_ID, DollarIndex_ID
dim_GoldPrice	Gold_Price_ID	Nil
dim_Bitcoin	Bitcoin_Price_ID	Nil
dim_EconomicIndicators	Economic_ID	Nil
dim_US_Dollar_Index	DollarIndex_ID	Nil

Appendix F – Data Dictionary

Entity: **fact_Prediction_Data**

Attributes	Data Type	Description	Example
Prediction_Price_ID	Int	Unique identifier for each prediction record	1
Prediction_Price	Float	Predicted price of gold generated by the forecasting model	3,200.60
Date	Datetime	Date of prediction	2025-03-28
US_dollar_strength	Char	Describes the strength of the US dollar at the time of prediction	High or Low
Sell_recommendation	Char	System-generated recommendation on whether to sell gold	Sell
B_Close_Price	Float	Bitcoin closing price on prediction date	68000.50
G_Close_Price	Float	Gold closing price on prediction date	2445.30
Gold_Price_ID	Int	Reference to related gold price record	2001
Bitcoin_Price_ID	Int	Reference to related bitcoin price record	3001
Economic_ID	Int	Reference to economic indicators	5001
DollarIndex_ID	Int	Reference to dollar index dimension	6001

Entity: **dim_GoldPrice**

Attributes	Data Type	Description	Example
Gold_Price_ID	Int	Unique 4-digit number to identify each gold price record	2001
Symbol	Char	Trading symbol representing gold in USD market	XAU-USD
Date	Datetime	The date when the gold price was recorded	2025-03-28
Open_Price	Float	Gold price at the start of the trading session	3,099.30
High_Price	Float	Highest price of gold during the trading session	3,124.40

Low_Price	Float	Lowest price of gold during the trading session	3,096.30
Close_Price	Float	Gold price at the end of the trading session	3,111.60
Volume	Float	Total trading volume of gold for that date	78,052

Entity: **dim_Bitcoin**

Attributes	Data Type	Description	Example
Bitcoin_Price_ID	Int	Unique 4-digit number to identify each bitcoin price record	3001
Symbol	Char	Trading symbol representing bitcoin in USD market	BTC-USD
Date	Datetime	The date when the bitcoin price was recorded	2025-03-28
Open_Price	Float	Bitcoin price at the start of the trading session	87,201.91
High_Price	Float	Highest price of bitcoin during the trading session	87,477.88
Low_Price	Float	Lowest price of bitcoin during the trading session	84,818.96
Close_Price	Float	Bitcoin price at the end of the trading session	85,181.45
Volume	Float	Trading volume of bitcoin for that date	30,806,026,240

Entity: **dim_EconomicIndicators**

Attributes	Data Type	Description	Example
Economic_ID	Int	Unique ID assigned to each economic indicator record	5001
Unemployment_rate	Float	National unemployment rate at the time of record	3.8
Fedfunds	Float	U.S. federal funds target interest rate	5.25
Date	Datetime	Date the economic indicators were recorded	2025-03-28
Cpiaucsl	Float	U.S. Consumer Price Index (CPI)	304.0445

Entity: **dim_US_Dollar_Index**

Attributes	Data Type	Description	Example

DollarIndex_ID	Int	Unique ID for dollar index record	6001
Symbol	Char	Dollar index trading symbol	DXY
Date	Datetime	Date the index record was captured	2025-03-28
Open_Price	Float	Opening price of the dollar index on that day	104.28
High_Price	Float	Highest dollar index value during the day	104.50
Low_Price	Float	Lowest dollar index value during the day	104.21
Close_Price	Float	Final price of the index at closing	104.44
Adj_Close	Float	Adjusted closing price reflecting corporate actions	104.44