DEPI
IBM Data Scientist R3
Team 79
Graduation Project

Team Members

Abdelrahman Sameeh
Habiba Mohamed
Menna Fakharany
Malak Khaled
Mohamed Adham
Sherouq Eldanaf

1. Project Planning & Management

- **Project Proposal**

This project aims to analyze the S&P 500 stock market data and apply machine learning techniques to predict the next-day stock movement. The system provides automated technical indicators and presents results through an interactive dashboard to support investors in making data-driven decisions.

- **Project Plan**

Task	Deliverables	Milestones	Timeline
Data collection and Preprocessing	1.Select Yahoo Finance and Kaggle for datasets 2.Cleaning missing values, fixing date formats.	1.Raw data collected and stored in project directory 2.Final cleaned dataset	Week 1-2
Data visualization and feature engineering	1.visual the data 2.Creating technical indicators (MA, EMA, RSI, ..)	1.show the data 2.dataset ready for modeling	Week 3-4
Machine learning model training	1.Implement ML models (Random Forest, LightGBM, XGBoost). 2.Hyperparameter tuning and model optimization.	Best-performing model selected.	Week 5-6
Evaluation & tunning	Performance metrics	final evaluation	Week 7
Deployment	Streamlit application UI	Deploy project on Streamlit Cloud	Week 8-9
testing & documentation	1.user inputs 2.complete technical documentation 3.README file	Final project submission.	Week 9-10

Table 1 shows a project plan.

• **Task Assignment & Roles**

	Member	Responsibilities
Data collection	Mohamed Adham	1.Gather the project idea and define the stock analysis scope. 2.Collect historical stock price datasets. 3.Identify appropriate data sources (Yahoo Finance, Kaggle).
preprocessing	Abdelrahman Sameeh	Clean and format the raw data , handle missing values.
Data visualization	Shrouq	visual to understand the data
feature engineering	Mohamed Adham	1.Generate technical indicators and prepare modeling features.
Machine learning model training	Habiba, Abdelrahman	Implement ML algorithms (Random Forest, LightGBM, XGBoost).
evaluation & Tunning	Malak	1.Evaluate model accuracy using. 2.Decide which model performs best for deployment. 3.Optimize hyperparameters. 4.Compare performance across models.
Deployment & testing	Menna	1.Build the Streamlit dashboard. 2.Integrate model predictions. 3.Deploy the final app. 4.Test app functionality, fix issues, and validate user inputs.
Documentation	Habiba	1.Prepare detailed project documentation, diagrams, and README. 2.Ensure reproducibility for any future work.

Table 2 shows a task assignment and roles.

- **Risk Assessment & Mitigation Plan**

Risk	Solutions
Low model accuracy	Improve feature engineering and tune hyperparameters
Time constraints	Distribute work and prioritize critical tasks
Accurate match real-time stocks	Feed the model the latest real-time stock data using only the features it was trained on, reshape the data into the correct format while handling errors gracefully.
Streamlit in colab need ngrok	Use streamlit cloud
Hug Dataset in GitHub	Use Google Drive

Table 3 shows a risk assessment and mitigation.

• KPIs (Key Performance Indicators)

– Prediction accuracy = 45 %

An accuracy of 54% in stock movement prediction is expected due to the inherent volatility and unpredictability of financial markets. Still, it performs better than random guessing (50%), showing that the model captures some real market patterns.

– Low system response time
– High dashboard usability
– Stable performance during repeated requests

2. Literature Review

• Feedback & Evaluation

Previous studies have applied machine learning for stock prediction using indicators such as RSI, SMA, and volatility. This project extends similar approaches by integrating real-time data and an interactive dashboard.

• Suggested Improvements

- Deploying on AWS EC2, Azure App Service, or Google Cloud Run.

- Combine historical stock data with global events and news using NLP and advanced models to improve the accuracy of future market movement predictions.
- Automated Alerts: Notify users of predicted trends or unusual market movements.
- Portfolio Optimization: Suggest buy/sell strategies based on model predictions.
- Interactive Visualizations: Real-time charts, heatmaps, and comparative dashboards.

3. Requirements Gathering

• Stakeholder Analysis

Stakeholders include investors and financial analysts.They need reliable stock data, accurate predictions, and a simple interface.

• User Stories & Use Cases

As a user, I want to select a stock so I can view its indicators and prediction.

• Functional Requirements

– Fetch live stock data
– Compute RSI, SMA, volatility, and daily return
– Predict next-day movement using ML model
– Display results on an interactive dashboard

• Non-functional Requirements

– Low response time
– User-friendly interface
– Handling of sensitive information

4. System Analysis & Design

i. Problem Statement

The stock market contains large amounts of real-time data that are difficult to analyze manually. The goal of this project is to automate technical analysis and provide accurate predictions to support investor decision-making.

Objectives

- Enable users to fetch live stock data.
- Compute key technical indicators automatically.

- Generate next-day stock movement predictions using ML.
- Present results on an interactive dashboard for decision-making.

1. Use Case Diagram & Description

The system lets a user select a stock, fetch live prices from YFinance, compute technical indicators (daily return, RSI 14, SMA 20/50, Volatility 20/50), and predict the next-day movement using a pre-trained ML model. The results are displayed on the Streamlit dashboard. Actors include the User and the live data provider (YFinance). Core use cases:Request Live Price,Select stock , Fetch Live data, Calculate Indicators, Predict Up/Down, and Display Results.

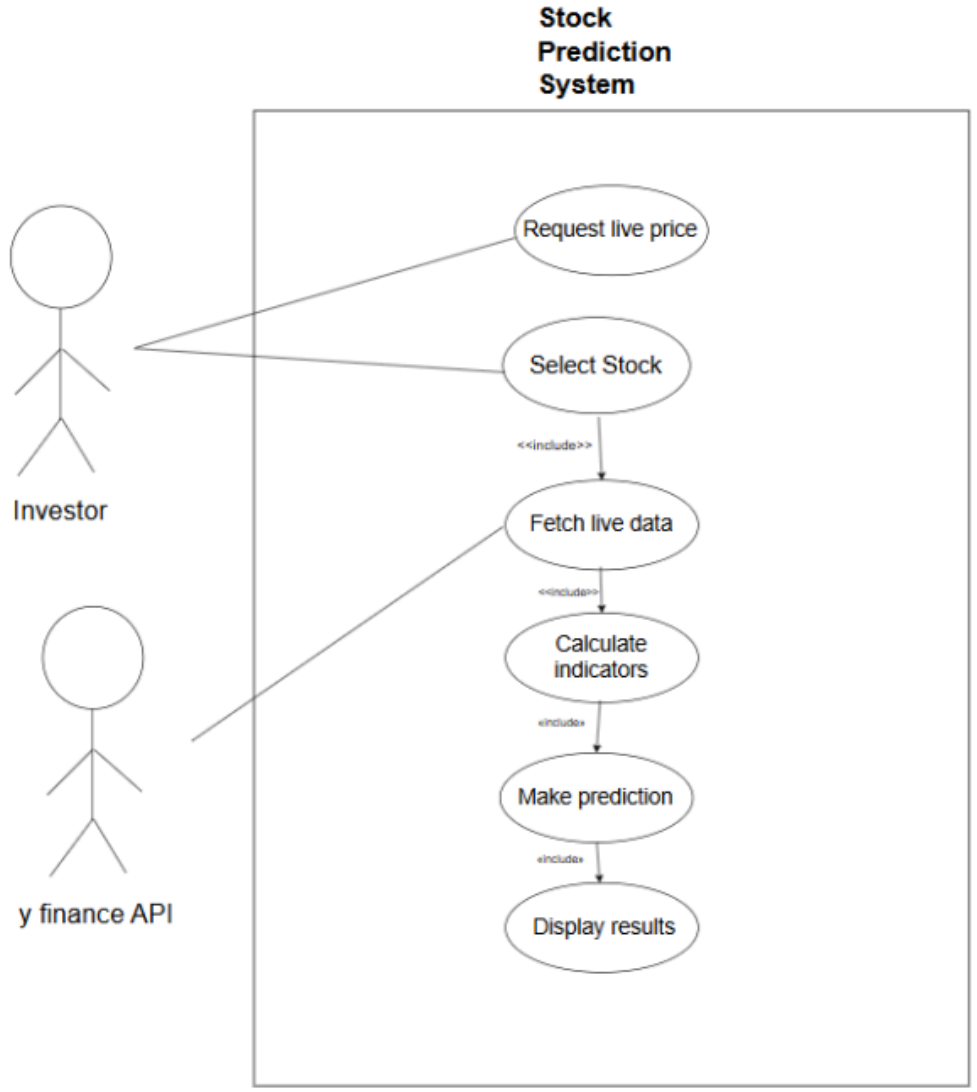

Figure 1 shows Use Case Diagram

- **Functional & Non-Functional Requirements**

Functional Requirements (FR):

1. Fetch live stock data from Yahoo Finance API.
2. Compute technical indicators: daily return, RSI (14), SMA (20/50), Volatility (20/50).
3. Generate next-day stock movement prediction using the pre-trained ML model.
4. Display results on a real-time interactive dashboard.
5. Allow users to select different stock tickers for analysis.

Non-Functional Requirements (NFR):

1. **Performance:** Low time system responds for stock queries.
2. **Usability:** Dashboard is simple and easy to navigate.
3. **Scalability:** System can handle multiple simultaneous user requests (default small scale).

- **Software Architecture**

1. **Data Fetching Module:** Connects to Yahoo Finance API and retrieves live stock data.
2. **Indicator Calculation Module:** Computes RSI, SMA, Volatility, and daily returns.
3. **Prediction Module:** Uses ML model on S&P 500 data to predict next-day movement.
4. **Dashboard (View):** Displays stock prices, indicators, and prediction results interactively.
5. **Controller:** Manages user input (stock selection) and coordinates between modules.
6. **Optional Storage Module:** Can store historical data or prediction logs for reference.

High-Level Flow:

The user selects a stock from the Streamlit dashboard, and the controller sends a request to fetch live data from YFinance. The system then calculates the technical indicators (RSI, SMA, Volatility), after which the ML model generates the prediction. Finally, the results are displayed back on the interactive dashboard.

User → Dashboard → Controller → Data Fetching → Indicator Calculation → Prediction → result.

ii. Database Design & Data Modeling

A traditional database was not used in this project. The system relies on a pre-existing dataset from Kaggle, which contains historical S&P 500 stock data. The dataset is loaded directly during the model training phase and processed using Python without the need to store it in a database.

iii.Data Flow & System Behavior

• DFD (Data Flow Diagram)

Context-level(Level 0)

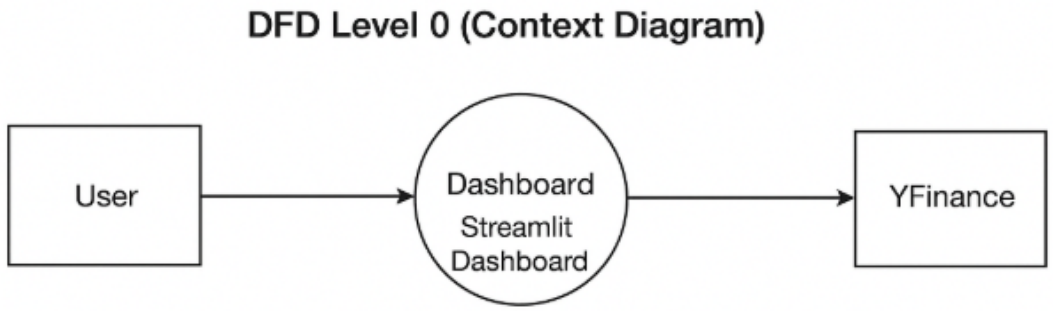

Figure 2 shows DFD Level 0.

Detailed levels(Level 1)

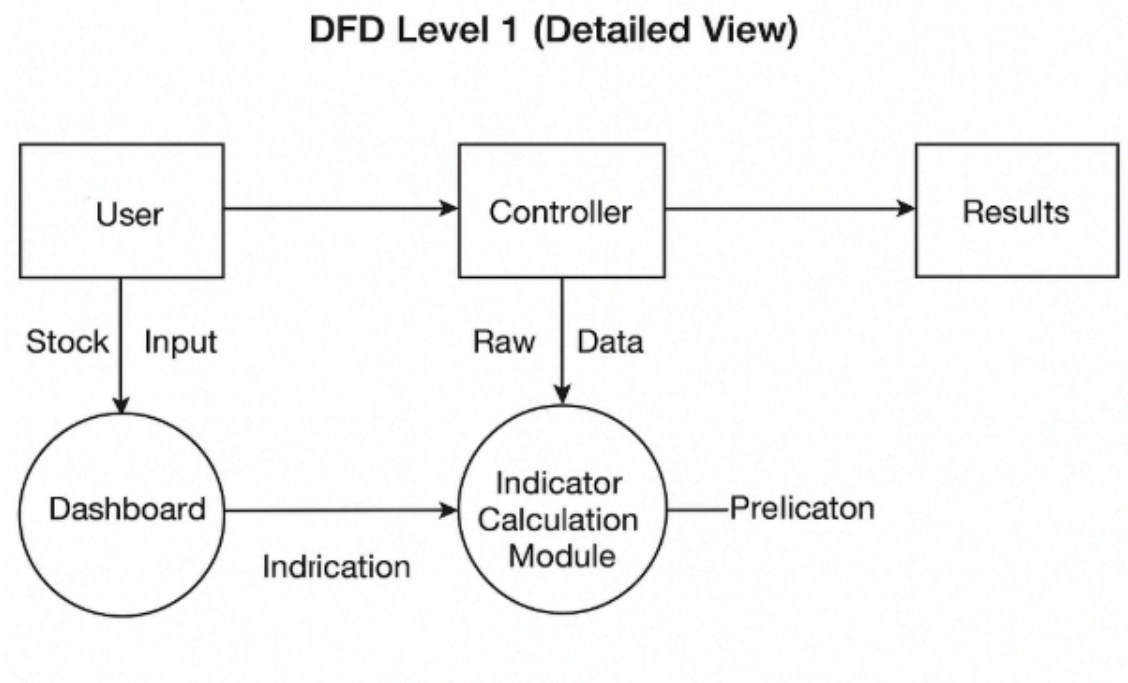

Figure 3 shows DFD Level 1.

• Sequence Diagrams

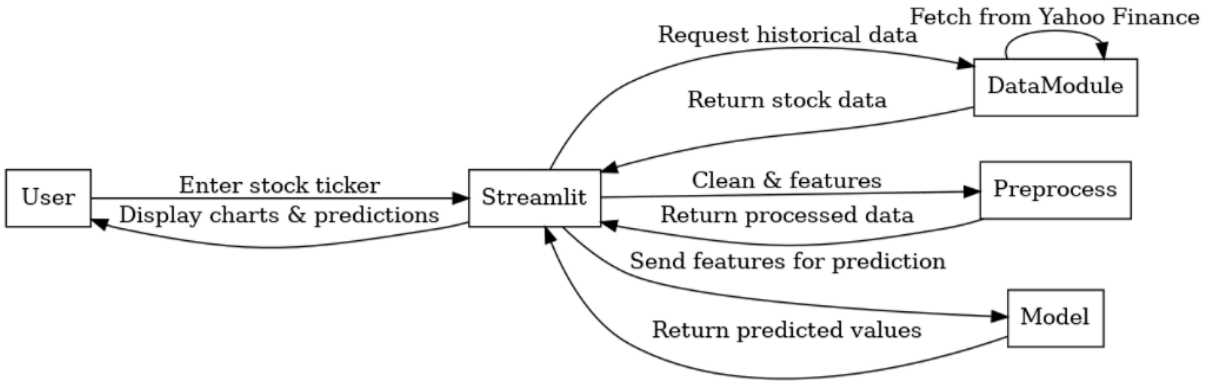

Figure 4 shows Sequence Diagram.

• Activity Diagram

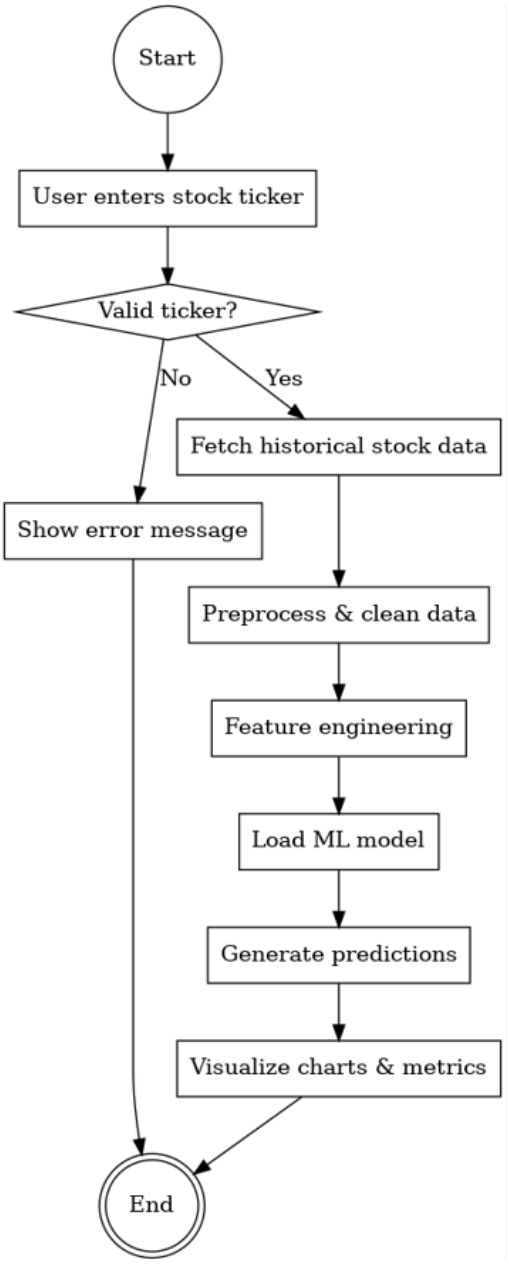

Figure 5 shows Activity Diagram.

iv. UI/UX Design & Prototyping

• Wireframes & Mockups

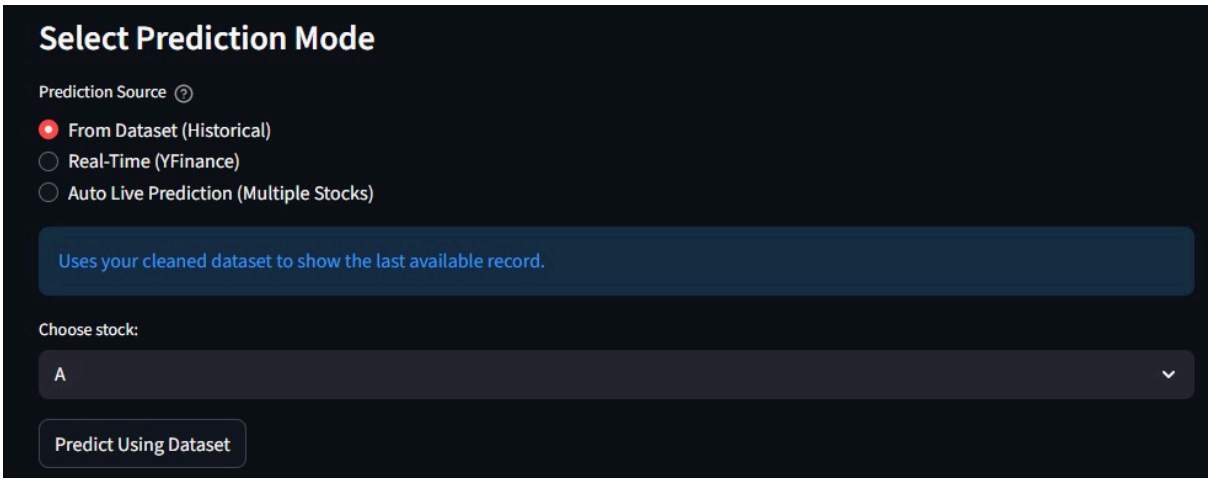

Figure 6 shows UI.

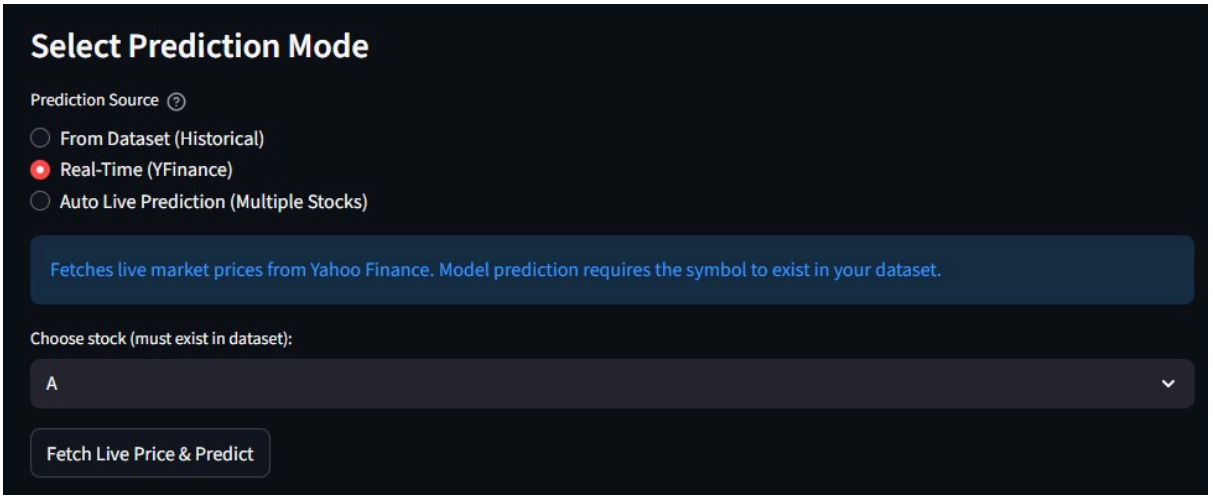

Figure 7 shows UI.

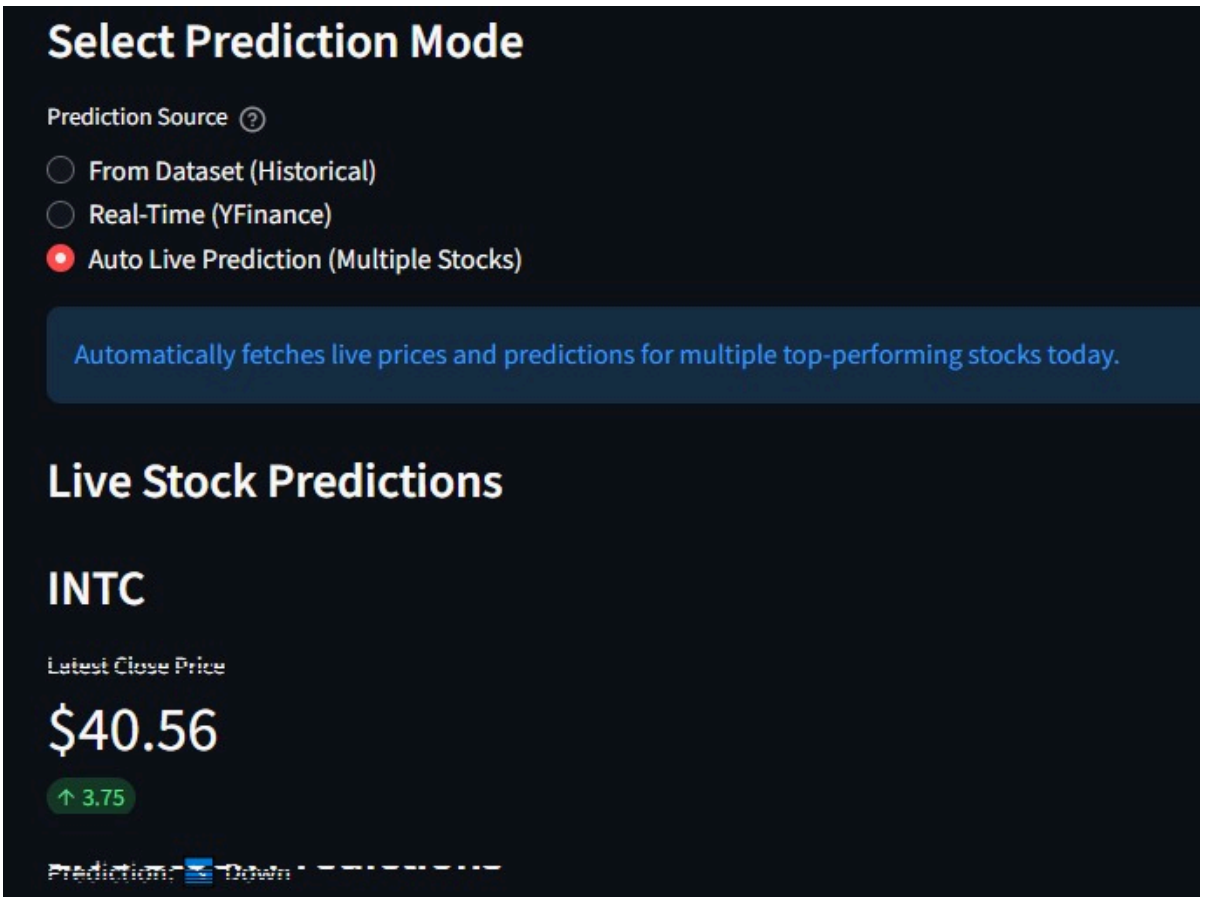

Figure 8 shows UI.

v. System Deployment & Integration

• Technology Stack

- Python: For data processing, analysis, and machine learning.
- Pandas & NumPy: For cleaning and analyzing stock data.
- Scikit-learn / LightGBM / XGBoost: For building and training prediction models.
- Matplotlib, Seaborn & Plotly: For visualization and plotting charts.
- yFinance: To fetch stock data from Yahoo Finance.
- Streamlit: To build an interactive dashboard for end-users.
- Joblib / Pickle: For saving and loading trained machine learning models.

• **Deployment Diagram**

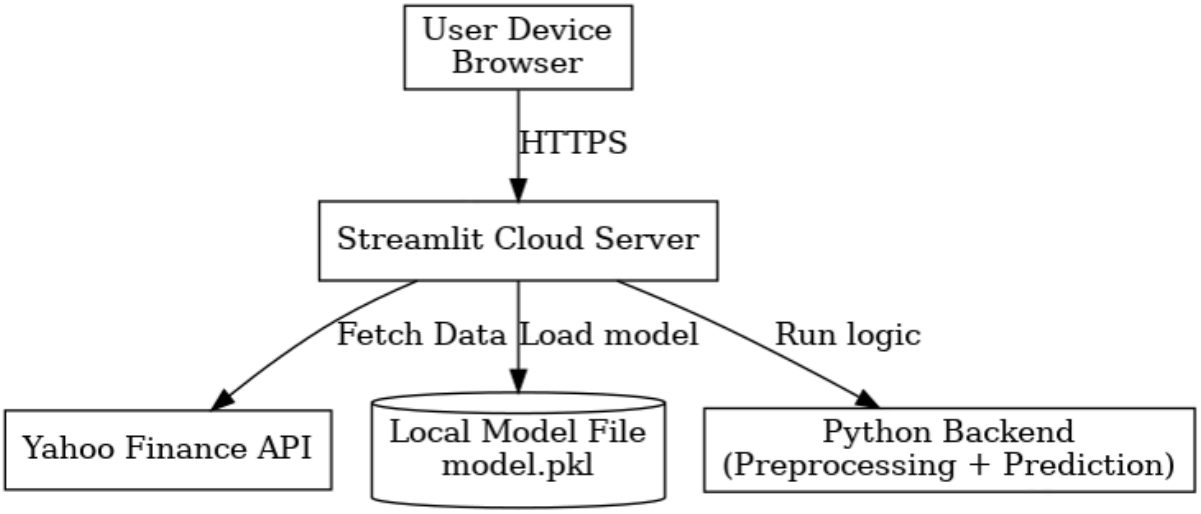

Figure 9 shows Deployment Diagram.

• **Component Diagram**

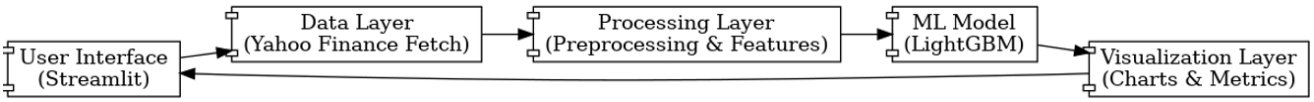

Figure 10 shows Component Diagram.

vi. Additional Deliverables

• **API Documentation**

◄ **External APIs Used (Yahoo Finance API (via `yfinance` library))**

- Fetch historical stock prices
- Retrieve company info
- Get real-time market data

◄ **Internal API (Streamlit Functions)**

- Data Fetching Methods
  - Input: user-selected ticker
  - Output: DataFrame of stock prices
- Model Prediction Endpoint
  - Input: cleaned dataset
  - Output: future predictions

• **Testing & Validation**

- **Unit Tests**

- Validate data loading functions
- Ensure indicators (RSI, ..) compute correct values
- Confirm model prediction functions run successfully

- **Integration Tests**
- Full pipeline testing:
    - Data → Preprocessing → Model → Visualization

- Test Streamlit integration with prediction outputs
- Verify handling of invalid user input
- **Model Validation**
- Evaluate ML models
- Time-series split validation
- **User Acceptance Testing (UAT)**
- Test usability of the Streamlit dashboard
- Validate that charts, predictions, and indicators load properly
- Ensure the dashboard works across multiple devices and browsers

• Deployment Strategy

- **Hosting Environment**

The project is deployed on Streamlit Cloud, providing:

- Automatic dependency installation
- Fast web hosting for Python dashboards
- Easy sharing via public URL
- **Deployment Pipeline**
- Code pushed to GitHub
- Streamlit Cloud automatically pulls the latest version
- Rebuilds the environment using `requirements.txt`
- App updates instantly after each successful deployment

• Scaling Considerations

Although Streamlit Cloud is sufficient for single-user or small-team usage, scaling options include:

- Deploying on AWS EC2, Azure App Service, or Google Cloud Run
- Using containerization (Docker) for more stable performance
- Enabling caching for heavy models to reduce computation time
- Using asynchronous APIs for faster data fetching

5. Implementation (Source Code & Execution)

1. Source Code

[colab]

• **Security & Error Handling**

◄ **Secure Coding Practices**

- External data is fetched only from trusted sources (Yahoo Finance).
- No sensitive user data is stored or processed.

◄ **Validation Checks**

- **Data Validation**

- Check if the API returned actual price data.
- Detect missing values
- Validate that data types match expected formats.

- **Model Validation**

- Confirm that the training data has sufficient rows before fitting a model.
- Validate feature engineering steps (e.g., check if indicators are calculable).

◄ **Proper Exception Handling**

- **Try/Except Blocks**

API data fetching wrapped with try/except to prevent crashes and provide helpful error messages.

- **Streamlit Error Messages**

When errors occur, the system displays:

- Clear, user-friendly alerts (e.g., “Error fetching or predicting real-time data:”)

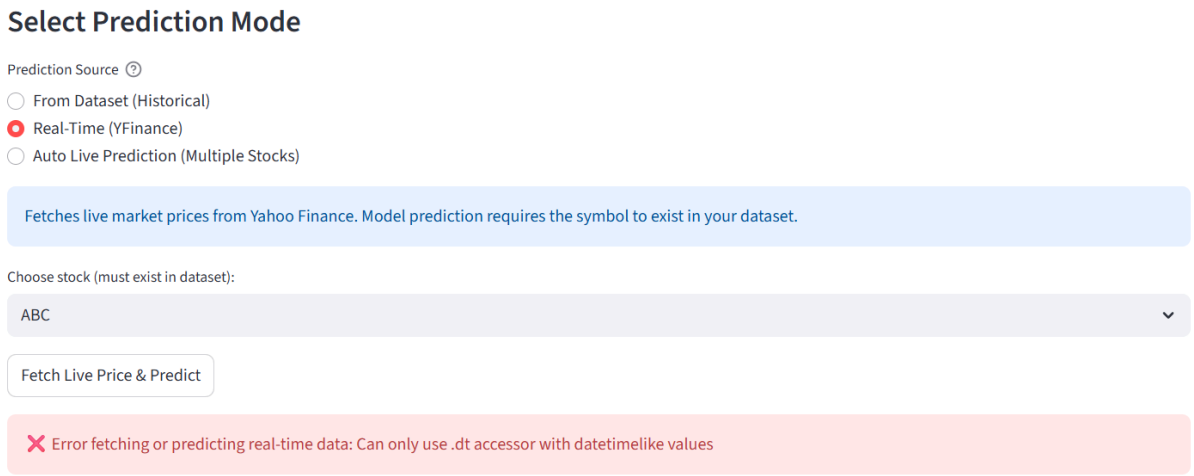

Figure 11 shows Handle Errors.

2. Version Control & Collaboration

• Version Control Repository

The project is managed using Git and hosted on:

- **GitHub Repository**
  The full source code, documentation files, and project assets are uploaded to a public GitHub repository.

The repository contains:

- Main project folder structure
- Source code scripts (data collection, preprocessing, modeling, visualization)
- Streamlit deployment files
- Requirements file `requirements.txt`
- Documentation assets

This allows easy access, sharing, and continuous updates.

• Branching Strategy

Main Branch

- Contains the stable production-ready code
- Reflects the deployed version of the Streamlit app

• Commit History & Documentation

- Clear and meaningful commit messages were maintained to track project progress

3. Deployment & Execution

• README File – Includes:
• Installation steps
• System requirements (software dependencies)
• API documentation

• Executable Files & Deployment Link

Streamlit

6. Testing & Quality Assurance

• Test Cases & Test Plan

A structured test plan was created to ensure the accuracy, performance, and reliability of both the machine learning pipeline and the Streamlit dashboard.
The test cases covered:

- **Data Validation Tests:**
  - Checking for missing values, incorrect data types, outliers.
  - Ensuring correct loading of CSV files and API responses.

- **Feature Engineering Tests:**
  - Verifying correct calculation of technical indicators (e.g., SMA, EMA, RSI, …).
  - Ensuring no data leakage between training and testing splits.

- **Model Performance Tests:**
  - Confirming that LightGBM model training runs without errors.
  - Validating expected performance metrics (Accuracy, Precision, Recall, F1-score).
  - Testing model stability.

- **Dashboard Functionality Tests:**
  - Ensuring charts load correctly and update with selected stock symbols.
  - Testing UI components (filters, date selectors, buttons).
  - Validating that model predictions display correctly for user-selected periods.

• Bug Reports

Throughout development, several issues were identified and documented:

- Data leakage in train-test split → resolved using TimeSeriesSplit instead of random split.
- UI rendering issues on Streamlit Cloud → resolved by adjusting chart sizes and layout responsiveness.
- Large dataset size prevented upload to GitHub → GitHub rejected the dataset due to file size limits, so the dataset was hosted on Google Drive instead, and the project includes a download link with instructions to load the data locally.

7. Final Presentation & Reports

• Project Presentation (PPT/PDF)

presentation

• Video Demonstration

Demo

8.Model Performance Discussion

While our model achieved approximately **55% accuracy**, this is actually typical in financial prediction tasks, as stock markets are highly random and noisy.

Research shows that across a wide range of stock markets and machine learning methods, the average directional accuracy is around 51% (Aparicio et al., 2025). Therefore, achieving 55% accuracy places our model above the broad average, representing a meaningful improvement over baseline chance. This result highlights that predicting markets is inherently difficult, but our approach demonstrates how machine learning can extract patterns even within noisy and complex financial data.

9.Reference

Aparicio, D., et al. (2025). Machine Learning, Stock Market Forecasting, and Market Efficiency: A Comparative Study. Springer Nature. **https://link.springer.com/article/10.1007/s41060-025-00854-4?u**