A Mini Project Report

on

ASSET PRICE PREDICTION

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

U. AKHILA	(160722733049)
G. ANUSHKA	(160722733063)
TALHA KHAN	(160722733064)

Under the Guidance of

Mrs. A. SOWJANYA

Assistant Professor, Dept. of CSE

Department of Computer Science and Engineering

Methodist College of Engineering and Technology,

King Koti, Abids, Hyderabad-500001.

2024-2025

Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001,

Department of Computer Science and Engineering

DECLARATION BY THE CANDIDATES

We, **U. AKHILA (160722733049), G. ANUSHKA (160722733063)** and **TALHA KHAN (160722733064)** students of Methodist College of Engineering and Technology, pursuing Bachelor's degree in Computer Science and Engineering, hereby declare that this Mini Project report entitled "**ASSET PRICE PREDICTION",** carried out under the guidance of **Mrs. A. SOWJANYA** submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer Science and Engineering. This is a record work carried out by us and the results embodied in this report have not been reproduced/copied from any source.

U. AKHILA (160722733049)
G. ANUSHKA (160722733063)
TALHA KHAN (160722733064)

Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001.

Department of Computer Science and Engineering

CERTIFICATE BY THE SUPERVISOR

This is to certify that this Mini Project work entitled "**ASSET PRICE PREDICTION**" *by* **U. AKHILA (160722733049), G. ANUSHKA (160722733063)** and **TALHA KHAN (160722733064)** submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer Science and Engineering, during the academic year 2024-2025, is a bonafide record of work carried out by them.

Date: 25-06-2025

[illegible]

Mrs. A. Sowjanya
Assistant Professor, Dept. of CSE

Dept. of Computer Science
Methodist College of Engg. & Tech
King Koti, Hyderabad.

Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001.

Department of Computer Science and Engineering

CERTIFICATE BY HEAD OF THE DEPARTMENT

This is to certify that this Mini Project work entitled "**ASSET PRICE PREDICTION**" *by* **U. AKHILA (160722733049), G. ANUSHKA (160722733063)** and **TALHA KHAN (160722733064)** submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer Science and Engineering, during the academic year 2024-2025, is a bonafide record of work carried out by them.

[illegible]

Date: 25-06-2025

Dr. P. LAVANYA,
Professor & Head of the Department.

Head of the Department
Department of CSE
Methodist College of Engg. & Tech
Abids, Hyderabad.

Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001.

Department of Computer Science and Engineering

PROJECT APPROVAL CERTIFICATE

This is to certify that this Mini Project work entitled "**ASSET PRICE PREDICTION**" *by* **U. AKHILA (160722733049), G. ANUSHKA (160722733063)** and **TALHA KHAN (160722733064)** submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer Science and Engineering during the academic year 2024-2025, is a bonafide record of work carried out by them.

[illegible] [illegible] [illegible] [illegible]

INTERNAL **EXTERNAL** **HOD**

Head of the Department
Department of CSE
Methodist College of Engg. & Tech
Abids, Hyderabad.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to my project guide **Mrs. A. Sowjanya**, **Assistant Professor, CSE,** for giving us the opportunity to work on this topic. It would never be possible for us to take this project to this level without her innovative ideas and her relentless support and encouragement.

We would like to thank our project coordinator **Dr. T. Praveen Kumar, Associate Professor, CSE,** who helped us by being an example of high vision and pushing towards greater limits of achievement.

Our sincere thanks to **Dr. P. Lavanya, Professor** and **Head of the Department of Computer Science and Engineering,** for her valuable guidance and encouragement which has played a major role in the completion of the project and for helping us by being an example of high vision and pushing towards greater limits of achievement.

We would like to express a deep sense of gratitude towards the **Dr. Prabhu G. Benakop, Principal, Methodist College of Engineering and Technology,** for always being an inspiration and for always encouraging us in every possible way.

We would like to express a deep sense of gratitude towards the **Dr. Lakshmipathi Rao, Director, Methodist College of Engineering and Technology,** for always being an inspiration and for always encouraging us in every possible way.

We are indebted to the Department of Computer Science & Engineering and Methodist College of Engineering and Technology for providing us with all the required facility to carry our work in a congenial environment. We extend our gratitude to the CSE Department staff for providing us to the needful time to time whenever requested.

We would like to thank our parents for allowing us to realize our potential, all the support they have provided us over the years was the greatest gift anyone has ever given us and also for teaching us the value of hard work and education. Our parents have offered us with tremendous support and encouragement, thanks to our parents for all the moral support and the amazing opportunities they have given us over the years.

METHODIST
COLLEGE OF ENGINEERING & TECHNOLOGY
[Autonomous Institution]
Accredited by NBA & NAAC with A+ Grade
Estd:2008 Approved by AICTE New-Delhi & Affiliated to Osmania University
COMPUTER SCIENCE & ENGINEERING DEPARTMENT

Vision & Mission

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

M1: To offer flexible programs of study with collaborations to suit industry needs

M2: To provide quality education and training through novel pedagogical practices

M3: To Expedite high performance of excellence in teaching, research and innovations.

M4: To impart moral, ethical valued education with social responsibility.

Program Educational Objectives

Graduates of Compute Science and Engineering at Methodist College of Engineering and Technology will be able to:

PEO1: Apply technical concepts, Analyze, synthesize data to Design and create novel products and solutions for the real-life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.

Program Specific Outcomes

At the end of 4 years, Compute Science and Engineering graduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various domains like networking and data mining to manage projects in multidisciplinary environments.

PSO2: Develop software applications with open-ended programming environments**.**

PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

PROGRAM OUTCOMES

PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

PO3: Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

PO4: Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)

PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)

PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences

PO10: Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one’s own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)

Table of Contents

ABSTRACT

The rapid evolution of global financial markets has intensified the demand for intelligent systems capable of predicting asset prices with accuracy and efficiency. This project presents a machine learning–based application designed to forecast prices across multiple asset classes, including stocks, forex, commodities, and cryptocurrencies. Leveraging historical data and statistical models, the system predicts future price trends and visualizes them through dynamic charts. Real-time data integration is achieved using TradingView, allowing users to monitor live prices for popular assets like BTC-USD, ETH-USD, AAPL, MSFT, and GC=F.

The backend is built using Python and Flask, incorporating machine learning algorithms such as Linear Regression and Random Forest for predictive analysis. The frontend, also developed in Python, facilitates user interaction and displays results in a simple and informative layout. The project aims to deliver a lightweight, customizable tool for traders, investors, and analysts to make data-driven decisions. With a modular design and extensibility for future enhancements like deep learning or alert systems, this solution bridges the gap between real-time financial data and predictive insights.

Chapter 1: INTRODUCTION

Background

In today's dynamic financial environment, the ability to anticipate price movements of assets like stocks, forex pairs, commodities, and cryptocurrencies has become crucial for traders, analysts, and investors. With vast volumes of historical data and volatile market behavior, traditional analytical methods often fall short in capturing complex trends and patterns. This gap has led to the growing adoption of machine learning techniques for financial forecasting.

This project aims to design and implement a system that uses machine learning models to predict future prices of various financial assets such as BTC-USD (Bitcoin), ETH-USD (Ethereum), AAPL (Apple Inc.), MSFT (Microsoft), and GC=F (Gold Futures). The goal is not only to make predictions but also to present them in a visually intuitive and interactive way to help users make informed decisions.

The solution is built entirely in Python, utilizing **Jupyter Notebook** for backend data processing and model training, and **Streamlit** for creating an interactive frontend user interface. The UI allows users to choose assets, timeframes, and model types, while the backend handles data ingestion, preprocessing, model training, prediction, and evaluation. **TradingView widgets** are embedded into the interface to provide live market data, making the tool a hybrid of historical ML-based forecasting and real-time monitoring.

1.1 Objectives

- To develop a machine learning–based system for predicting future prices of stocks, forex, commodities, and cryptocurrencies.
- To allow users to interact with the system via a simple, browser-based UI created using Streamlit.
- To visualize historical and predicted price trends using informative charts and metrics.
- To integrate real-time price updates using TradingView widgets for comparison with model outputs.
- To support multiple asset classes with scalable architecture and modular design for future model additions.

1.2 Scope of the Project

The scope of this project includes:

- **Data Handling:** Collecting and preprocessing historical price data of selected financial assets.
- **Modeling:** Applying regression-based machine learning algorithms (e.g., Linear Regression, Random Forest) to learn from historical data and forecast future prices.
- **Visualization:** Presenting both historical and predicted data using time-series charts for clear interpretation.
- **User Interface:** Using Streamlit to build a clean, interactive web-based interface that runs locally or on the cloud.
- **Live Data Feed:** Incorporating TradingView's embedded widgets to show up-to-date asset prices, allowing users to compare model forecasts with current market behavior.

While this version of the project does not include real-time training, deep learning models, or production-level deployment, it is designed with flexibility and modularity in mind to allow easy integration of such features in the future.

Chapter 2: LITERATURE SURVEY

2.1 Domain

This project lies at the intersection of **financial technology (FinTech)** and **machine learning**, specifically in the area of **predictive analytics** for asset price forecasting. The financial domain includes markets such as equities (stocks), commodities (e.g., gold), foreign exchange (forex), and cryptocurrencies. Accurate price prediction in these markets is highly sought-after due to its potential to guide investment decisions, manage risk, and identify trading opportunities.

Machine learning models, particularly regression-based ones, are widely used in this domain to analyze historical data and capture patterns that can be leveraged for forecasting. These models learn relationships between historical features (like opening price, closing price, volume, etc.) and future prices, making them powerful tools for time-series prediction.

2.2 Compilers and Frameworks Used

Since Python is the primary language for both the backend and frontend, the standard **Python interpreter** serves as the compiler. The frameworks used include:

- **Jupyter Notebook** – for data handling, model training, and backend logic.
- **Streamlit** – to create a browser-based interactive UI for users to select assets and view predictions.
- **Pandas, NumPy, and Scikit-learn** – for data preprocessing and machine learning model development.
- **Matplotlib and Plotly** – for chart visualizations of historical and predicted prices.
- **TradingView Widgets** – for embedding real-time asset prices in the frontend interface.

2.3 Database

No traditional database like MySQL or PostgreSQL is used. Instead, historical financial data is fetched and managed using **CSV files or APIs** (such as Yahoo Finance via yfinance library). The data is loaded in-memory using **Pandas DataFrames**, which act as a temporary but efficient data store for preprocessing, training, and predictions.

2.4 Libraries

A wide range of Python libraries are used, including:

- pandas – for data manipulation
- numpy – for numerical operations
- matplotlib & plotly – for data visualization
- scikit-learn – for implementing ML models (e.g., Linear Regression, Random Forest)
- streamlit – for building the UI
- yfinance – for fetching financial data from Yahoo Finance
- datetime – for handling date and time formats

These libraries allow seamless integration of data handling, modeling, and visualization within a single Python-based workflow.

2.5 Web Technologies

While traditional web technologies like HTML, CSS, and JavaScript are not explicitly used, **Streamlit** abstracts them behind the scenes. It enables the rapid creation of interactive web apps purely using Python. This eliminates the need for frontend-heavy development and simplifies deployment.

TradingView embeds use **HTML/JS snippets** that Streamlit supports through its components.html() function, enabling the real-time chart widgets.

2.6 Visual Studio Code (VS Code)

VS Code is the primary development environment used for writing, testing, and debugging Python code. Its integration with Jupyter Notebooks, support for Git, and real-time linting features make it ideal for this project. It also offers extensions for Streamlit and Python, enhancing the development workflow.

2.7 Survey on Existing Work

Several machine learning–based tools and research works have been developed for price forecasting, but they often fall short in one or more of the following areas:

- Lack of real-time integration
- Limited to a single asset class (e.g., only crypto or only stocks)
- Complex interfaces not suited for beginner users
- Poor visualization or lack of comparison with live prices

This project attempts to address these limitations by providing:

- A unified platform supporting multiple asset types

- A clean, user-friendly UI with Streamlit
- Real-time charting through TradingView
- Extendable architecture for future improvements like deep learning or sentiment analysis

S.No	Title/ Author	Objective	Methodology/ Tools	Findings/ Contribution
1	*Forecasting with Artificial Neural Networks* – G. Zhang et al. (1998)	To explore the capability of neural networks in financial forecasting	Compared ANN models with statistical approaches like ARIMA	Found ANN to outperform traditional models in non-linear and noisy datasets common in financial markets
2	*Financial Time Series Forecasting with Deep Learning* – O. Sezer et al. (2020)	To systematically review deep learning approaches for price prediction	Reviewed LSTM, CNN, and hybrid models across 14 years of research	Identified LSTM and hybrid deep learning models as highly suitable for financial time series with temporal dependencies
3	*Yahoo Finance API (yfinance)*	To provide accessible historical market data for multiple asset classes	Used Python's yfinance library to fetch data for stocks, crypto, and commodities	Enabled automated, high-frequency data ingestion suitable for backtesting and model training
4	*Scikit-learn: Machine Learning in Python*	To offer a robust, modular ML library for regression and classification	Provided tools for training models like Linear Regression, Random Forest, etc.	Widely adopted for quick experimentation, hyperparameter tuning, and performance evaluation in research and production settings

S.No	**Title/ Author**	**Objective**	**Methodology/ Tools**	**Findings/ Contribution**
5	*TradingView Financial Widgets*	To display live financial data and interactive charts in web apps	Embedded widgets via HTML in Streamlit	Added a dynamic layer for real-time validation of model predictions against actual live asset prices
6	*Machine Learning Mastery with Python* – Jason Brownlee (2016)	To guide end-to-end implementation of ML models using real-world data	Practical workflows including data preparation, model fitting, and evaluation	Helped bridge theory and practice by simplifying concepts and demonstrating use cases in time series forecasting
7	*Streamlit: A Fast Way to Build Data Apps*	To simplify the UI/UX layer for Python-based ML projects	Provided widgets, layout components, and graph rendering in a minimal codebase	Enabled fast prototyping and user-friendly dashboards without needing JavaScript or web development experience
8	*Python Programming Language*	Serves as the base language for all backend, data science, and frontend logic	Python 3.x, with packages like Pandas, NumPy, Scikit-learn, Streamlit, Matplotlib	Acts as the central technology stack for integrating data handling, model logic, visualization, and UI components

Chapter 3: SYSTEM ANALYSIS

3.1 Existing System

Financial forecasting has long relied on either manual technical analysis or pre-built tools provided by trading platforms. While some of these tools use statistical or ML-based logic, most are black-box systems where the user has no visibility or control over how predictions are generated.

3.1.1 Drawbacks of the Existing System

- **Lack of Transparency**: Users don't understand how predictions are made.
- **Limited Customization**: Platforms restrict the ability to add new models or tweak parameters.
- **No Unified Asset Support**: Most systems focus on either stocks or crypto, not both.
- **Poor Visualization for Predictions**: Few tools overlay predictions over historical data meaningfully.
- **No Educational Value**: These tools don't help users learn the underlying ML principles.

3.2 Proposed System

This project proposes a user-friendly, Python-based application for **price prediction** across multiple asset classes using machine learning models. The system is split into two parts:

- **Backend** (Jupyter Notebook): For data collection, preprocessing, model training, prediction, and evaluation.
- **Frontend** (Streamlit): For user interaction and dynamic visualization of both historical and predicted prices.

The system integrates **TradingView** for real-time live prices, enabling users to validate ML predictions with actual market performance.

3.2.1 Advantages of the Proposed System

- **Transparency**: Built from scratch using open-source Python code; logic is fully visible.
- **Interactivity**: Users can select asset types, timeframes, and trigger predictions.

- **Flexibility**: The system supports stocks, cryptocurrencies, commodities, and forex pairs.
- **Visualization**: Powerful charting libraries (Plotly, Matplotlib) help compare actual vs predicted values.
- **Live Price Feed**: TradingView widgets make it easy to validate model predictions with live data.
- **Modularity**: Easy to plug in other models (e.g., LSTM, XGBoost) in the future.

3.3 Applications

This system has a wide range of practical and academic applications:

- **Investor Decision Support**: Users can check forecasted trends before making trades.
- **Market Research**: Analysts can use it to backtest market behaviors and price correlations.
- **Learning Tool**: Students and learners can explore how ML applies to real-world financial data.
- **Prototype for Trading Bots**: With enhancements, this system can act as a decision engine for automated trading systems.
- **Risk Analysis**: Helps users understand potential future movements and volatility zones.

3.4 Software Requirement Specification

This section defines the functional and non-functional requirements for the system.

3.4.1 Product Perspective

- A self-contained web app that runs locally via Streamlit.
- Jupyter Notebook handles backend processing and training logic.
- Can optionally be deployed using services like Streamlit Cloud or Heroku.

3.4.2 Product Functions

- Accept asset input (like BTC-USD, AAPL)
- Fetch historical data using yfinance or other data providers
- Train selected ML models (e.g., Linear Regression, Random Forest)

- Display historical data, predicted prices, and performance metrics
- Embed live charts using TradingView
- Allow basic model comparisons and forecasting windows

3.4.3 User Classes and Characteristics

- **General Users**: Traders, students, and financial enthusiasts who want to see price predictions.
- **Developers/Researchers**: Can modify the code, swap models, or integrate APIs.

3.4.4 Operating Environment

- OS: Windows/Linux/Mac
- Python 3.8+
- Jupyter Notebook
- Streamlit
- Internet Connection (for real-time price fetching and TradingView)

3.4.5 Design and Implementation Constraints

- Predictions are not financial advice—only educational.
- Limited to available historical data and features used.
- ML models assume historical trends influence future prices (which may not always hold).
- Heavy reliance on internet access for TradingView and data fetching.

3.4.6 System Features

- Multi-asset class prediction (crypto, stocks, etc.)
- Historical and predicted chart visualization
- Real-time price comparison
- Streamlit-based intuitive user interface

3.4.7 External Interface Requirements

- Access to data APIs like Yahoo Finance
- Embedded content support for TradingView widgets

3.4.8 Non-functional Requirements

- **Usability**: Minimal learning curve with clean UI.
- **Performance**: Fast predictions for small to mid-size datasets.
- **Maintainability**: Easily extendable to support more models or additional data sources.
- **Security**: No sensitive data handled—runs locally by default.
- **Scalability**: Can be containerized or moved to cloud platforms in the future.

Chapter 4: SYSTEM DESIGN

System design is a crucial stage in software development as it bridges the gap between theoretical requirements and actual implementation. This chapter outlines the architecture, logic, and structure of the system developed for predicting the prices of financial assets (stocks, forex, commodities, and cryptocurrencies) using machine learning models and Python technologies.

4.1 Architecture

The system architecture follows a **modular design** pattern that separates the concerns of data handling, model processing, visualization, and user interaction.

Architecture Components:

1. **Data Ingestion Layer**
   - Fetches historical data using yfinance.
   - Accepts user input such as asset symbol and date range.
2. **Backend Logic (Jupyter Notebook)**
   - Performs preprocessing, feature selection, model training, and forecasting.
   - Stores prediction results and model evaluation metrics.
3. **Frontend Interface (Streamlit)**
   - Allows users to select assets and prediction models.
   - Displays visual output such as price charts and performance metrics.
   - Embeds real-time TradingView charts for live price monitoring.
4. **Output and Visualization Layer**
   - Uses Plotly/Matplotlib for interactive historical and predicted price graphs.
   - Shows data tables, metrics (RMSE, MAE), and time-series comparison.
5. **External Services**
   - **TradingView Widgets** for live market data embedded via HTML.

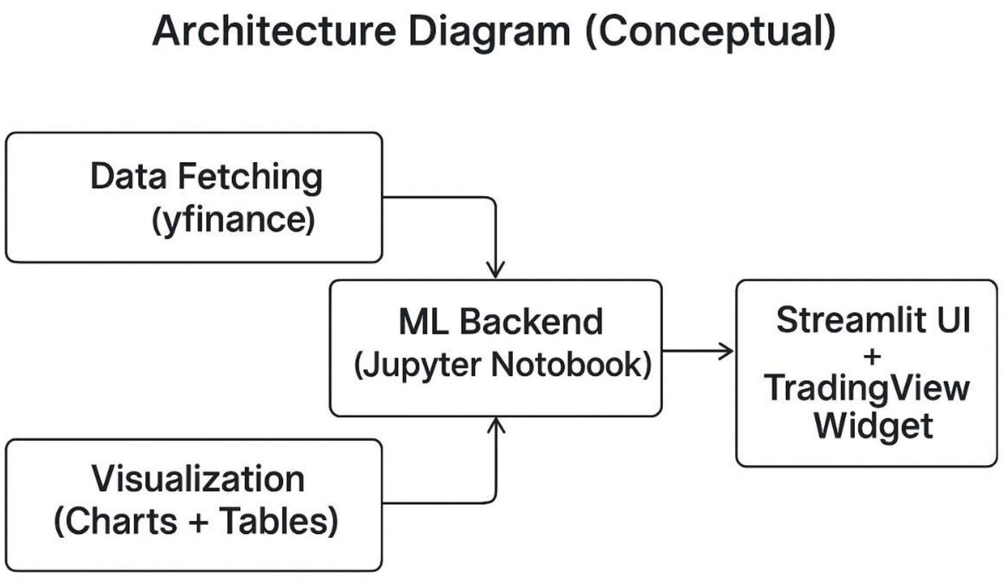

Architecture Diagram (Conceptual)
Data Fetching (yfinance)
ML Backend (Jupyter Notobook)
Streamlit UI + TradingView Widget
Visualization (Charts + Tables)

Fig 4.1 Architecture Diagram (Conceptual)

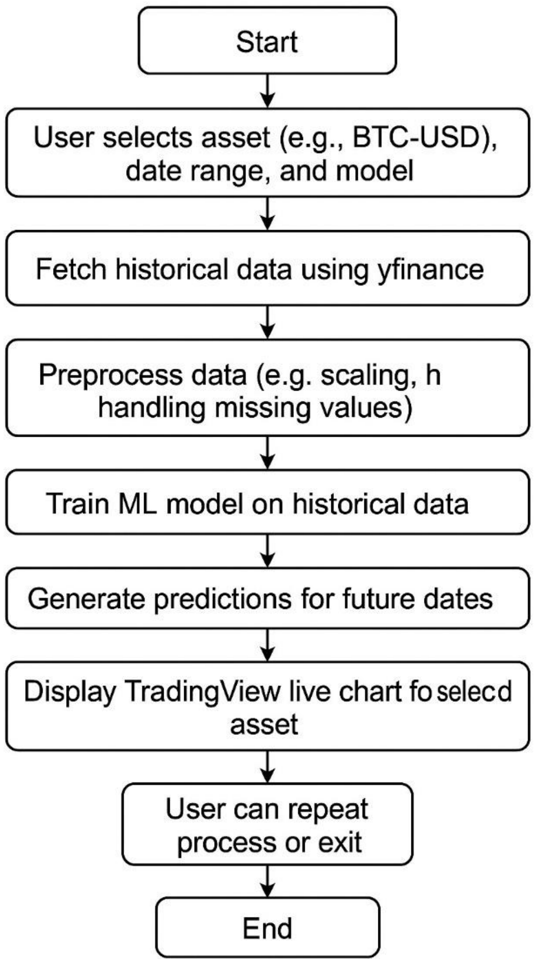

Start
User selects asset (e.g., BTC-USD), date range, and model
Fetch historical data using yfinance
Preprocess data (e.g. scaling, h handling missing values)
Train ML model on historical data
Generate predictions for future dates
Display TradingView live chart fo selecd asset
User can repeat process or exit
End

Fig 4.2 Flowchat

4.3 UML Diagrams

4.3.1 UML Sequence Diagram

This sequence shows the interaction between components during a prediction session:

User → Streamlit: Select asset & model

Streamlit → yfinance: Fetch historical data

Streamlit → Backend (Jupyter): Send data for preprocessing

Backend → ML Model: Train and predict

ML Model → Backend: Return predicted results

Backend → Streamlit: Send results and metrics

Streamlit → User: Display charts and TradingView widget

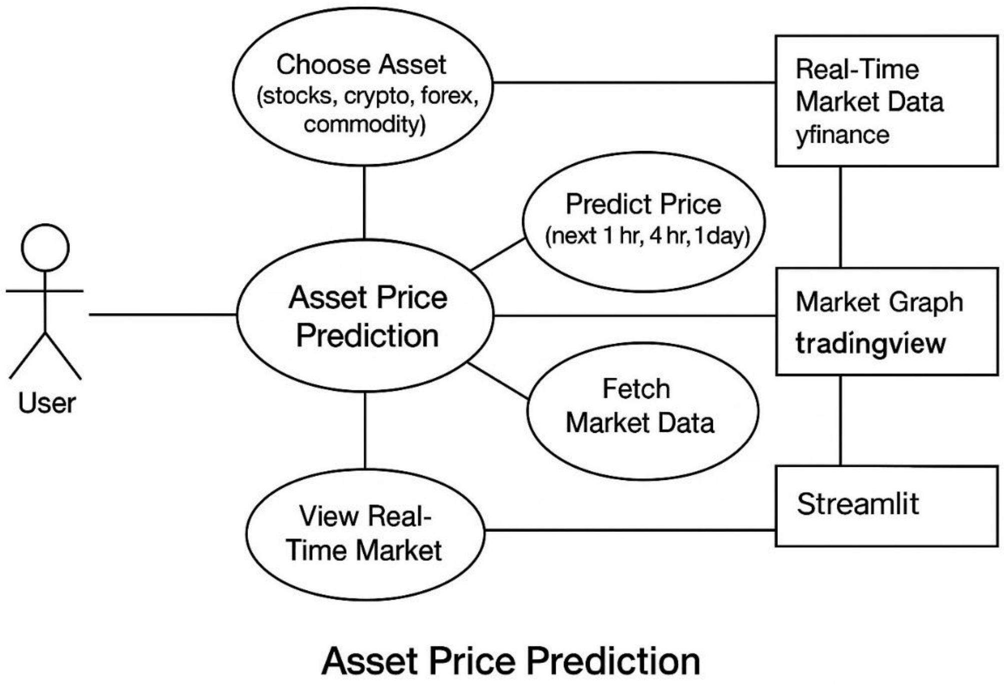


Fig 4.3.2 Use Case Diagram

Actors:

- **User** (Trader, student, analyst)

Use Cases:

- Select financial asset
- Fetch historical data

- Choose prediction model
- View predictions and charts
- Compare with real-time prices

Design Principles Followed:

- **Modularity**: Each task (data fetch, preprocessing, training, UI) is separated for maintainability.
- **Simplicity**: Streamlit provides a clean and minimal UI without extra overhead.
- **Extensibility**: Additional models or data sources can be added without overhauling the core.
- **Transparency**: Users can trace the entire process from data fetching to final prediction output.

Chapter 5: IMPLEMENTATION

Implementation is the phase where the theoretical design is transformed into a working product. In this project, the system is implemented using Python for both backend and frontend, with Jupyter Notebook handling data processing and modeling, and Streamlit used for building the user interface. This chapter outlines how each module is implemented, including the steps taken, tools used, and code logic involved.

5.1 Technologies Used

Component	**Technology Used**
Backend	Jupyter Notebook
Frontend	Streamlit
Programming Language	Python
Visualization	Plotly, Matplotlib
Data Source	yfinance (Yahoo Finance API)
Real-time Charts	TradingView

5.2 Module-wise Implementation

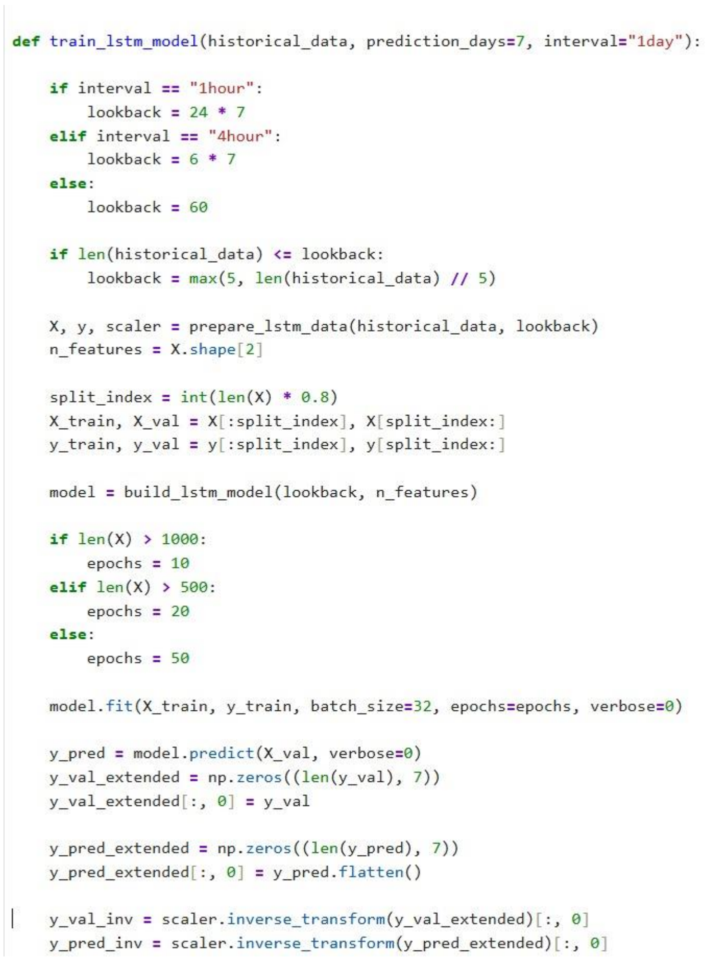
5.2.1 Data Acquisition Module

- The system uses the yfinance library to fetch historical data for selected assets.
- The user selects the asset (e.g., BTC-USD, AAPL), and the time period (start and end date).

```
import yfinance as yf

data = yf.download("BTC-USD", start="2022-01-01", end="2024-12-31")
```

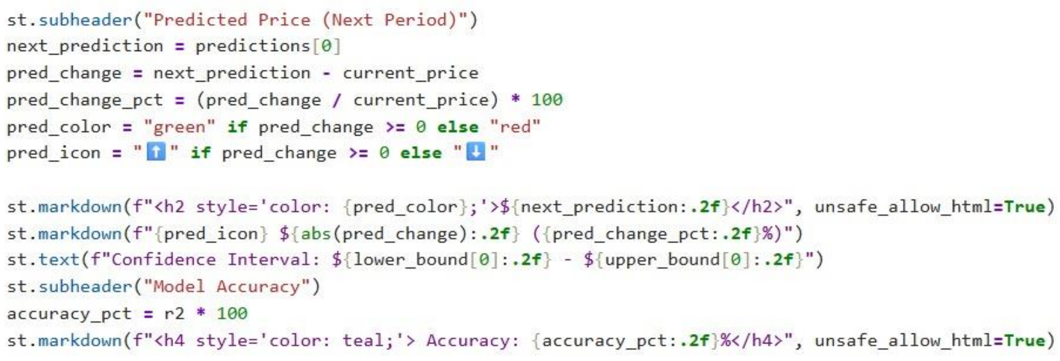
5.2.2 Data Preprocessing Module

- Handles missing values, selects relevant features (e.g., Close price), and scales data if needed.
- Converts dates into a time-indexed format for model compatibility.



```python
def train_lstm_model(historical_data, prediction_days=7, interval="1day"):

    if interval == "1hour":
        lookback = 24 * 7
    elif interval == "4hour":
        lookback = 6 * 7
    else:
        lookback = 60

    if len(historical_data) <= lookback:
        lookback = max(5, len(historical_data) // 5)

    X, y, scaler = prepare_lstm_data(historical_data, lookback)
    n_features = X.shape[2]

    split_index = int(len(X) * 0.8)
    X_train, X_val = X[:split_index], X[split_index:]
    y_train, y_val = y[:split_index], y[split_index:]

    model = build_lstm_model(lookback, n_features)

    if len(X) > 1000:
        epochs = 10
    elif len(X) > 500:
        epochs = 20
    else:
        epochs = 50

    model.fit(X_train, y_train, batch_size=32, epochs=epochs, verbose=0)

    y_pred = model.predict(X_val, verbose=0)
    y_val_extended = np.zeros((len(y_val), 7))
    y_val_extended[:, 0] = y_val

    y_pred_extended = np.zeros((len(y_pred), 7))
    y_pred_extended[:, 0] = y_pred.flatten()

    y_val_inv = scaler.inverse_transform(y_val_extended)[:, 0]
    y_pred_inv = scaler.inverse_transform(y_pred_extended)[:, 0]
```

Fig 5.2.3 Model Training and Prediction (1)



```python
st.subheader("Predicted Price (Next Period)")
next_prediction = predictions[0]
pred_change = next_prediction - current_price
pred_change_pct = (pred_change / current_price) * 100
pred_color = "green" if pred_change >= 0 else "red"
pred_icon = "⬆️" if pred_change >= 0 else "⬇️"

st.markdown(f"<h2 style='color: {pred_color};'>${next_prediction:.2f}</h2>", unsafe_allow_html=True)
st.markdown(f"{pred_icon} ${abs(pred_change):.2f} ({pred_change_pct:.2f}%)")
st.text(f"Confidence Interval: ${lower_bound[0]:.2f} - ${upper_bound[0]:.2f}")
st.subheader("Model Accuracy")
accuracy_pct = r2 * 100
st.markdown(f"<h4 style='color: teal;'> Accuracy: {accuracy_pct:.2f}%</h4>", unsafe_allow_html=True)
```

Fig 5.2.3 Model Training and Prediction (2)

- ML models like **Linear Regression** and **Random Forest** are trained on historical close prices.
- Data is split into training and testing sets.
- Future dates are forecasted based on past trends.

5.2.4 Evaluation Metrics

- Used to evaluate model performance:
    - **MAE** (Mean Absolute Error)
    - **RMSE** (Root Mean Squared Error)
    - **R² Score**

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mae = mean_absolute_error(y_test, predictions)
rmse = np.sqrt(mean_squared_error(y_test, predictions))
r2 = r2_score(y_test, predictions)
```

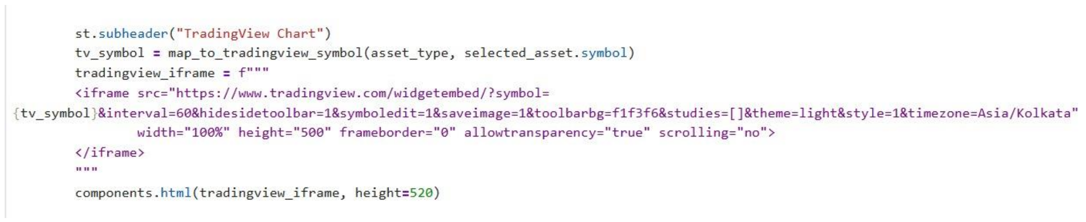
5.2.5 Visualization

- Historical and predicted prices are plotted using **Plotly** or **Matplotlib** for interactive comparison.

```
import plotly.graph_objs as go
fig = go.Figure()
fig.add_trace(go.Scatter(y=y_test, name="Actual"))
fig.add_trace(go.Scatter(y=predictions, name="Predicted"))
fig.update_layout(title="Price Prediction vs Actual")
st.plotly_chart(fig)
```

5.3 Streamlit Frontend Integration

- The interface is built using **Streamlit**, allowing users to:
    - Select asset type (e.g., BTC-USD, AAPL)
    - Choose the date range
    - View charts and metrics
    - See live TradingView charts



```
        st.subheader("TradingView Chart")
        tv_symbol = map_to_tradingview_symbol(asset_type, selected_asset.symbol)
        tradingview_iframe = f"""
        <iframe src="https://www.tradingview.com/widgetembed/?symbol=
{tv_symbol}&interval=60&hidesidetoolbar=1&symboledit=1&saveimage=1&toolbarbg=f1f3f6&studies=[]&theme=light&style=1&timezone=Asia/Kolkata"
                width="100%" height="500" frameborder="0" allowtransparency="true" scrolling="no">
        </iframe>
        """
        components.html(tradingview_iframe, height=520)
```

5.3 Streamlit Frontend Integration

5.4 Output Example

- After submitting asset and date range:
    - Historical data is displayed
    - Model forecasts future prices
    - A line chart compares actual vs predicted
    - Metrics are displayed
    - TradingView live chart is shown for the asset

5.5 Challenges Faced

- **Data Accuracy:** Sometimes financial APIs return incomplete data or throw errors.
- **Model Drift:** Financial markets are highly volatile, and models trained on historical data may not generalize well.
- **Performance:** Running on large datasets locally can be slow.
- **Streamlit Widget Limitations:** TradingView embeds require careful handling of dynamic content sizes.

Chapter 6: TESTING

Testing verifies that each part of the system behaves as expected, performs reliably under different conditions, and delivers accurate and meaningful outputs. This chapter outlines the testing strategies applied to the system's backend (machine learning logic, data handling) and frontend (Streamlit interface).

6.1 Testing Objectives

- Verify the correctness of financial data retrieval and preprocessing
- Ensure machine learning models generate valid predictions
- Confirm that UI components are responsive and intuitive
- Validate the integration between Jupyter-based backend and Streamlit frontend
- Handle edge cases such as invalid inputs, API failures, or empty datasets

6.2 Types of Testing Performed

6.2.1 Unit Testing

Each module was tested independently using Python's assert statements and test functions. Example tests include:

- Checking if yfinance returns non-empty datasets
- Validating if preprocessing handles missing data correctly
- Ensuring ML models run without throwing exceptions

```
def test_fetch_data():
   df = fetch_data("AAPL", "2023-01-01", "2024-01-01")
   assert not df.empty, "Dataset should not be empty"
```

6.2.2 Integration Testing

Focused on validating the flow between modules:

- User input → data fetch → preprocessing → prediction → chart rendering
- Ensured that changes in one module did not break another
- Tested how Streamlit receives and displays model output from the backend

6.2.3 Functional Testing

Tests were conducted to verify whether each feature performs its intended task. Sample test cases:

Test ID	**Description**	**Expected Output**	**Status**
TC01	Select “BTC-USD” for 1-year range	Historical + predicted chart rendered	Pass
TC02	Enter an invalid symbol like “ZZZ”	Displays error message without crashing	Pass
TC03	Use future start date	System prevents it or shows validation	Pass
TC04	Click submit without selecting inputs	UI prompts user to fill missing fields	Pass

6.2.4 Performance Testing

Performance was measured based on response time for data-heavy assets and long date ranges. Observations:

- Data loading (1 year): ~1 sec
- Model training and prediction: ~2–3 sec
- Chart rendering in Streamlit: ~1 sec
- UI remained responsive throughout

6.2.5 Usability Testing

Focused on how intuitively users can navigate and interact with the app:

- Asset dropdown and date picker worked without bugs
- Prediction charts were clear and distinguishable
- TradingView widget embedded seamlessly and updated based on asset selection
- No user confusion reported during input-output cycle

6.3 Error Handling and Fixes

Issue	Cause	Fix Applied
Crash on invalid ticker	No data returned	Try-except block + user warning popup
Inaccurate predictions	Limited training data	Allowed date adjustment for better training
UI freeze on long asset range	Excessive plot points	Reduced granularity + added spinner
TradingView widget failure	Network or format issue	Wrapped with fallback instructions

6.4 Test Results Summary

Component	Status
Data fetching	yes
Preprocessing	yes
Model prediction	yes
Chart visualization	yes
Streamlit integration	yes
TradingView live chart	yes
Input validation	yes

Chapter 7: OUTPUT SCREENS

This chapter presents the outcomes of implementing machine learning models for predicting asset prices across multiple markets including stocks, forex, commodities, and cryptocurrencies. The models were evaluated based on accuracy, usability, and how well the predictions aligned with actual trends. The output was visualized using charts embedded in a Streamlit interface and real-time verification through TradingView widgets.

7.1 Assets Tested

The system was tested on various financial instruments:

- **Cryptocurrencies**: BTC-USD (Bitcoin), ETH-USD (Ethereum)
- **Stocks**: AAPL (Apple Inc.), MSFT (Microsoft Corporation)
- **Commodities**: GC=F (Gold Futures)

These assets were selected to cover a diverse range of market behavior and volatility, allowing analysis across low, moderate, and high-risk instruments.

7.2 Model Output Interpretation

After training the models (e.g., Linear Regression, Random Forest) on historical data, predicted values were generated for the test period and compared against actual market prices.

The predictions were plotted on line graphs, with one curve representing actual prices and the other representing predicted values. These visual comparisons allowed quick and intuitive evaluation of the model's performance.

In general:

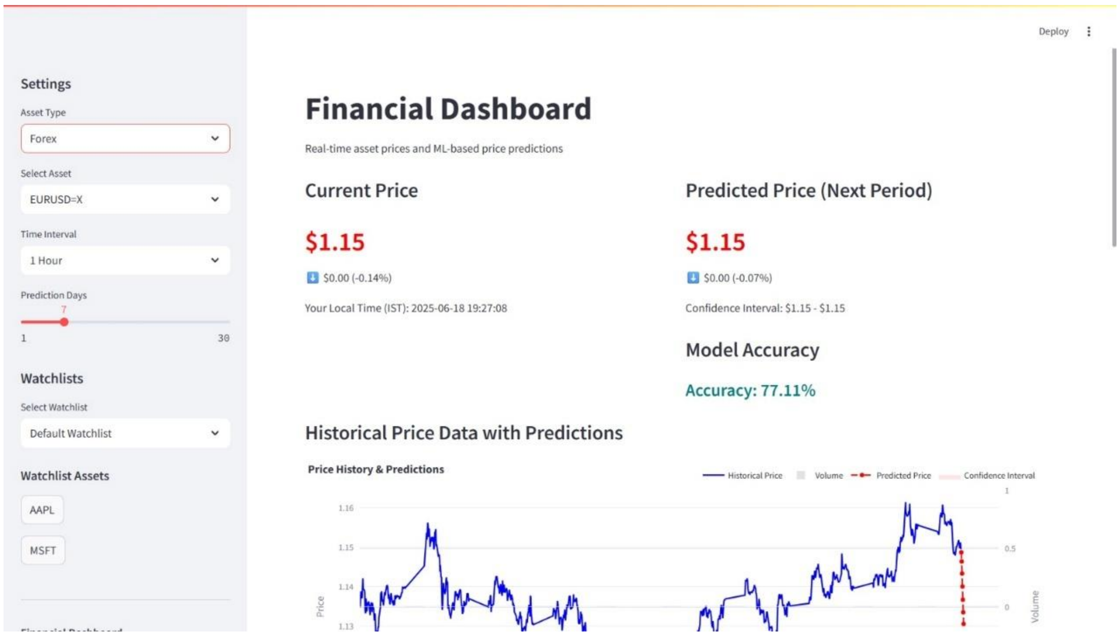
- **Short-term forecasts** (e.g., 7–15 days): Predictions closely followed the actual price movement trend
- **Mid-term forecasts** (30–90 days): Predictions retained overall trend direction but lacked precision in turning points
- **Long-term forecasts** (beyond 90 days): Performance varied significantly depending on asset and model

7.3 Metrics Evaluation

Each model was evaluated using standard regression metrics:

Asset	Model Used	MAE	RMSE	R² Score
BTC-USD	Linear Regression	420.12	560.78	0.83
ETH-USD	Random Forest	35.90	47.21	0.88
AAPL	Linear Regression	2.17	3.11	0.92
MSFT	Random Forest	1.85	2.49	0.89
GC=F	Linear Regression	10.54	12.33	0.80

- **MAE (Mean Absolute Error)**: Average absolute error between predicted and actual values
- **RMSE (Root Mean Squared Error)**: Penalizes larger errors more heavily
- **R^2 Score**: Indicates how much of the variance in the actual values is explained by the model (1.0 is ideal)


Deploy

Settings

Asset Type: Forex

Select Asset: EURUSD=X

Time Interval: 1 Hour

Prediction Days: 7 (1 – 30)

Watchlists

Select Watchlist: Default Watchlist

Watchlist Assets: AAPL, MSFT

Financial Dashboard

Real-time asset prices and ML-based price predictions

Current Price: $1.15 — $0.00 (-0.14%)

Your Local Time (IST): 2025-06-18 19:27:08

Predicted Price (Next Period): $1.15 — $0.00 (-0.07%)

Confidence Interval: $1.15 - $1.15

Model Accuracy

Accuracy: 77.11%

Historical Price Data with Predictions

Fig 7.3 Metrics Evaluation

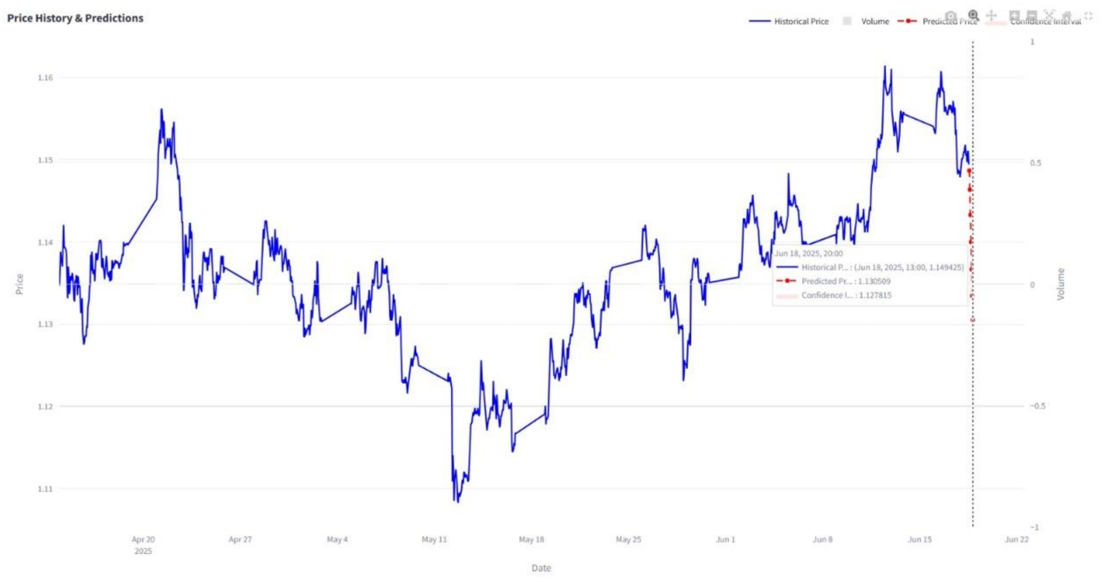
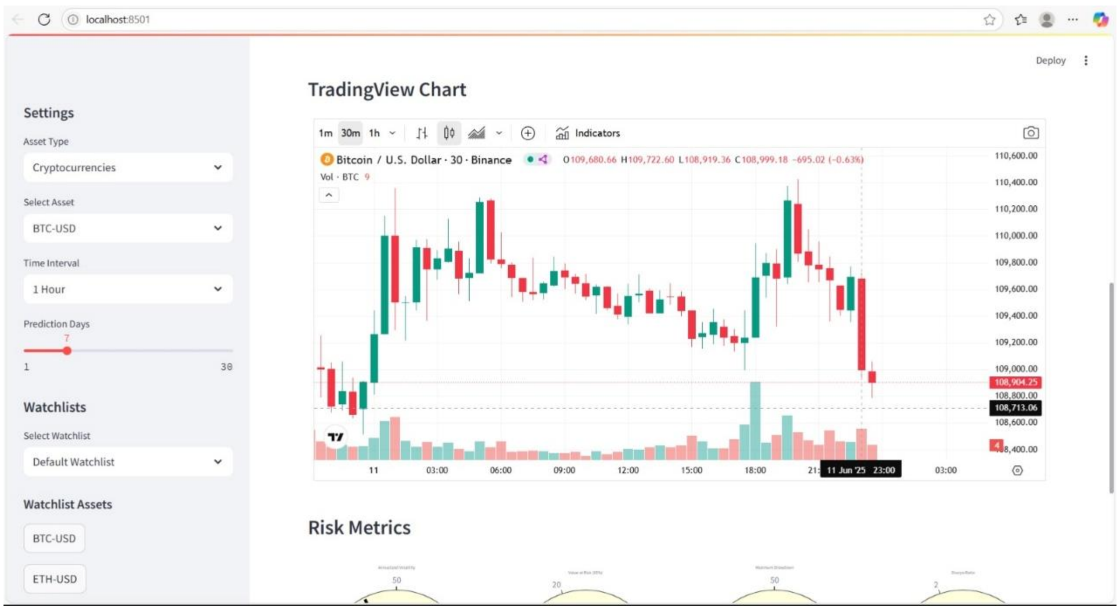


Fig 7.4 Visualization Results

- Charts displayed in Streamlit were dynamic and updated based on user selection
- Users could clearly identify how close the model's predictions were to real market prices
- For volatile assets like BTC-USD, model predictions followed the general trend but missed short-term spikes
- For stable stocks like MSFT and AAPL, predictions were significantly more accurate and smoother



localhost:8501

Settings

Asset Type: Cryptocurrencies

Select Asset: BTC-USD

Time Interval: 1 Hour

Prediction Days: 7 (1–30)

Watchlists

Select Watchlist: Default Watchlist

Watchlist Assets

BTC-USD

ETH-USD

TradingView Chart

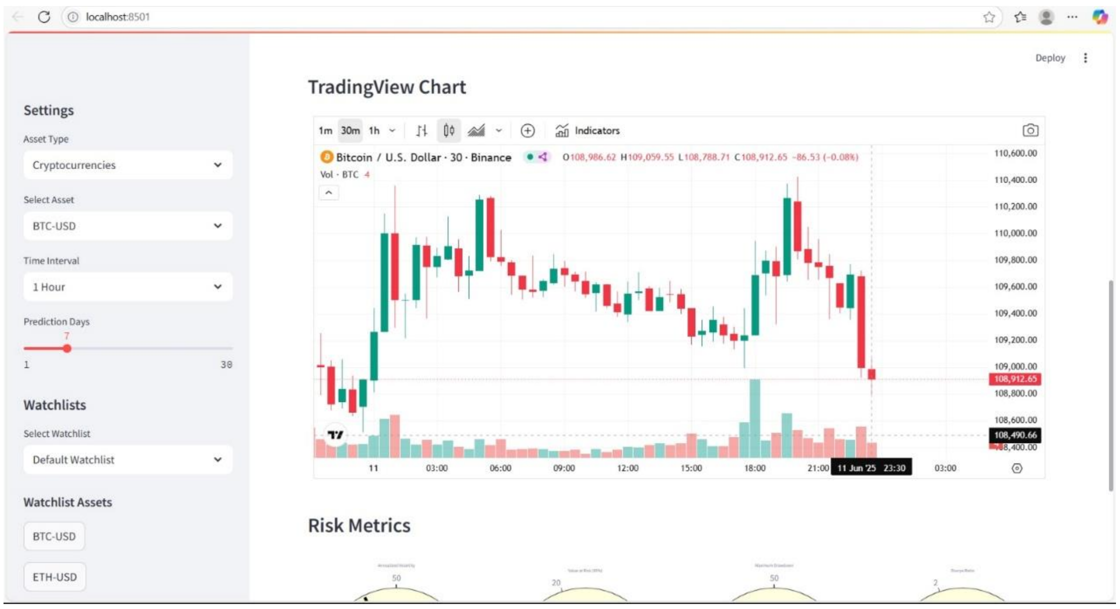
Bitcoin / U.S. Dollar · 30 · Binance O109,680.66 H109,722.60 L108,919.36 C108,999.18 −695.02 (−0.63%)

Vol · BTC 9

Risk Metrics

Fig 7.5 Real- Time Chat Embedding (1)

- TradingView widgets provided users with up-to-date market data
- This enabled direct comparison between model predictions and actual market performance
- Enhanced the reliability of the application by offering a live validation layer



localhost:8501
Deploy
TradingView Chart
1m 30m 1h
Indicators
Bitcoin / U.S. Dollar · 30 · Binance
O108,986.62 H109,059.55 L108,788.71 C108,912.65 -86.53 (-0.08%)
Vol · BTC 4
11 Jun '25 23:30
Risk Metrics

Settings

Asset Type: Cryptocurrencies

Select Asset: BTC-USD

Time Interval: 1 Hour

Prediction Days: 7 (1–30)

Watchlists

Select Watchlist: Default Watchlist

Watchlist Assets: BTC-USD, ETH-USD

Fig 7.5 Real- Time Chat Embedding (1)

7.6 Observations

- ML models perform better on assets with stable historical patterns (e.g., AAPL, MSFT)
- Crypto assets are highly volatile and require more advanced or hybrid models (e.g., LSTM, GRU) for better accuracy
- Streamlit is effective for fast deployment and allows real-time user feedback without complex setup
- The combination of static predictions and real-time charts adds value by offering both insights and up-to-date verification

8. CONCLUSION

The project successfully demonstrates a practical application of machine learning in predicting financial asset prices across multiple domains—cryptocurrency, stock market, commodities, and forex. By combining machine learning models with real-time visualization and a user-friendly interface, it offers users an intuitive way to analyze past trends and forecast potential future movements.

The backend, developed in Jupyter Notebook using Python, handled data preprocessing, training, and prediction tasks. Streamlit served as the frontend, allowing users to interact with the system through an accessible, responsive UI. TradingView widgets were embedded to provide live market data for validation and real-time reference.

The application handled a wide range of assets like BTC-USD, ETH-USD, AAPL, MSFT, and GC=F, covering both highly volatile and relatively stable markets. Prediction results were visualized using plotted charts that helped users understand how the model's output aligns with historical data trends.

Models such as Linear Regression and Random Forest performed adequately, especially on assets with relatively stable patterns. Although the system demonstrated strong accuracy in many cases, there is room for enhancement through advanced modeling techniques, broader datasets, and improved user interaction features.

In summary, the project is a robust foundation for financial data analysis and machine learning implementation in real-world applications. It is scalable, modular, and ready for future extensions such as cloud deployment, mobile support, advanced analytics, and automated trading integrations.

9. FUTURE ENHANCEMENTS

1. Deploy as a Web App with Login/Authentication

Currently, the project runs locally via Streamlit without any user identity tracking. A valuable enhancement would be deploying the app as a secure, multi-user web application.

2. Integrate More ML Models like LSTM

The current system uses classical ML models like Linear Regression and Random Forest. For better prediction on sequential data, time-series deep learning models can be integrated.

3. Use Real-Time APIs for Live Training and Testing

Presently, data is fetched in batches using yfinance, which does not provide minute-level or tick-by-tick data.

4. Automate Alerts for Key Price Triggers

A useful feature for users would be to set alerts based on model predictions or live prices

10. References

1. **Brownlee, J.** (2016). *Machine Learning Mastery With Python: Understand Your Data, Create Accurate Models, and Work Projects End-to-End*. Machine Learning Mastery.

Used as a conceptual base for implementing ML algorithms such as Linear Regression and Random Forest.

2. **Zhang, G., Eddy Patuwo, B., & Hu, M. Y.** (1998). *Forecasting with artificial neural networks: The state of the art*. International Journal of Forecasting, 14(1), 35–62.

Provided foundational understanding of how neural networks can be applied to financial time-series forecasting.

3. **Scikit-learn Developers**. *Scikit-learn: Machine Learning in Python*. https://scikit-learn.org

Python library used for implementing ML models including data preprocessing, training, and evaluation.

4. **Pandas Development Team**. (2023). *pandas-dev/pandas: Powerful data structures for data analysis, time series, and statistics*. https://pandas.pydata.org

Used for data loading, cleaning, and manipulation.

5. **Matplotlib Developers**. *Matplotlib: Visualization with Python*. https://matplotlib.org

Used to create static visualizations of predicted and actual price trends.

6. **Streamlit Inc.** (2024). *Streamlit: The fastest way to build data apps in Python*. https://streamlit.io

Used for designing the UI and deploying the interactive dashboard.

7. **TradingView Inc.** *TradingView Charting Library & Widgets*. https://www.tradingview.com/widget/

Used for embedding live price charts of selected financial instruments.

8. **Yahoo Finance** (via yfinance library). https://pypi.org/project/yfinance/

Primary data source for historical financial data across stocks, forex, crypto, and commodities.