Stock Prediction App using Machine Learning

A Project Report

# Abstract

The Stock Prediction App is an interactive application developed using Streamlit and Python, aimed at predicting and visualizing stock market prices. Stock markets are inherently volatile, and forecasting them is a challenging task due to various economic and social factors. This project integrates stock data fetched from Yahoo Finance, performs feature engineering, and applies machine learning models such as Linear Regression, Decision Tree, and Random Forest to predict future stock prices. The application also includes advanced visualization features like candlestick charts, moving averages, and Bollinger Bands. Users can experiment with models and parameters to better understand financial data and its predictions.

Date of Submission: 29/08/2025

# Introduction

**Background of Stock Market Prediction**

The stock market is one of the most dynamic and unpredictable components of the global economy. Millions of investors, companies, and governments rely on stock prices to make financial decisions. Predicting stock price movements has always been of great interest because even a small increase in accuracy can result in substantial financial gains. Traditionally, investors have used fundamental analysis (company performance, balance sheets, market conditions) and technical analysis (chart patterns, indicators like moving averages) to estimate market movements. However, with the vast availability of data and advances in computing, **stock market prediction has become an area where technology, finance, and data science converge**.

**Challenges in Forecasting Stock Prices**

Despite its importance, stock price prediction remains a challenging task. Some of the key challenges include:

* **Market Volatility**: Stock prices are highly sensitive to news, global events, government policies, and investor sentiment, making them fluctuate unexpectedly.
* **Non-linearity**: Price movements are not always linear; they often follow complex patterns that traditional statistical models struggle to capture.
* **Data Overload**: The stock market produces a huge amount of data every second, making it difficult to filter out noise and identify meaningful patterns.
* **External Factors**: Unexpected events like pandemics, political conflicts, or natural disasters can drastically change the market, making predictions unreliable.

These challenges mean that **no model can guarantee absolute accuracy**, but advanced techniques can help improve the reliability of predictions.

**Why Machine Learning Models Are Useful Here**

Machine Learning (ML) offers powerful tools to tackle the challenges of stock market prediction. Unlike traditional statistical methods, ML can:

* **Identify complex patterns** hidden in historical data.
* **Adapt to new data quickly**, improving forecasts as more information becomes available.
* **Use multiple indicators simultaneously** (like moving averages, volume, lag features) to generate predictions.
* **Handle large datasets efficiently**, which is essential in stock market analysis.

Models like **Linear Regression, Decision Trees, and Random Forests** provide different perspectives on prediction. Linear Regression explains trends with simplicity, Decision Trees offer interpretability, and Random Forest improves accuracy through ensemble learning.

**Motivation for Building This Project**

The motivation behind developing this project stems from both academic and practical perspectives:

* **Learning Perspective**: Building this app provides hands-on experience in applying data science and machine learning techniques to a real-world problem. It bridges the gap between theory and practice.
* **Practical Utility**: Investors and students can interact with the app to visualize stock trends, experiment with models, and understand how forecasts are generated.
* **Personal Interest**: With growing curiosity in financial markets and AI, this project serves as a step toward exploring the intersection of **finance, technology, and machine learning**.

By developing this Stock Prediction App, the goal is not just to predict stock prices but also to **create an educational tool** that helps users understand the dynamics of stock data and the role of ML in financial forecasting.

# System Requirements

For developing and running the **Stock Prediction App**, both software and hardware requirements must be considered. The following specifications ensure smooth functioning of the project.

**Software Requirements**

* **Python 3.8 or above** – Programming language used to implement the app.
* **Streamlit** – Framework for building interactive web-based dashboards.
* **yfinance** – Library used to fetch historical stock data from Yahoo Finance.
* **scikit-learn** – Machine learning library for implementing models such as Linear Regression, Decision Tree, and Random Forest.
* **Plotly** – Visualization library used to generate candlestick charts, scatter plots, and forecast graphs.
* **Pandas** – Data manipulation and preprocessing library.
* **NumPy** – Library for numerical computations and array handling.

**Hardware Requirements**

* **Processor:** Intel i3 (or equivalent) and above.
* **RAM:** Minimum 4 GB (8 GB recommended for faster processing).
* **Storage:** At least 1 GB free space for dependencies and data storage.
* **Internet Connection:** Required for fetching stock data from Yahoo Finance API.

# System Design & Architecture

The **Stock Prediction App** follows a structured workflow consisting of multiple stages. Each stage plays a crucial role in ensuring accurate prediction and visualization of stock prices. The architecture can be represented as follows:

**Explanation of Each Stage**

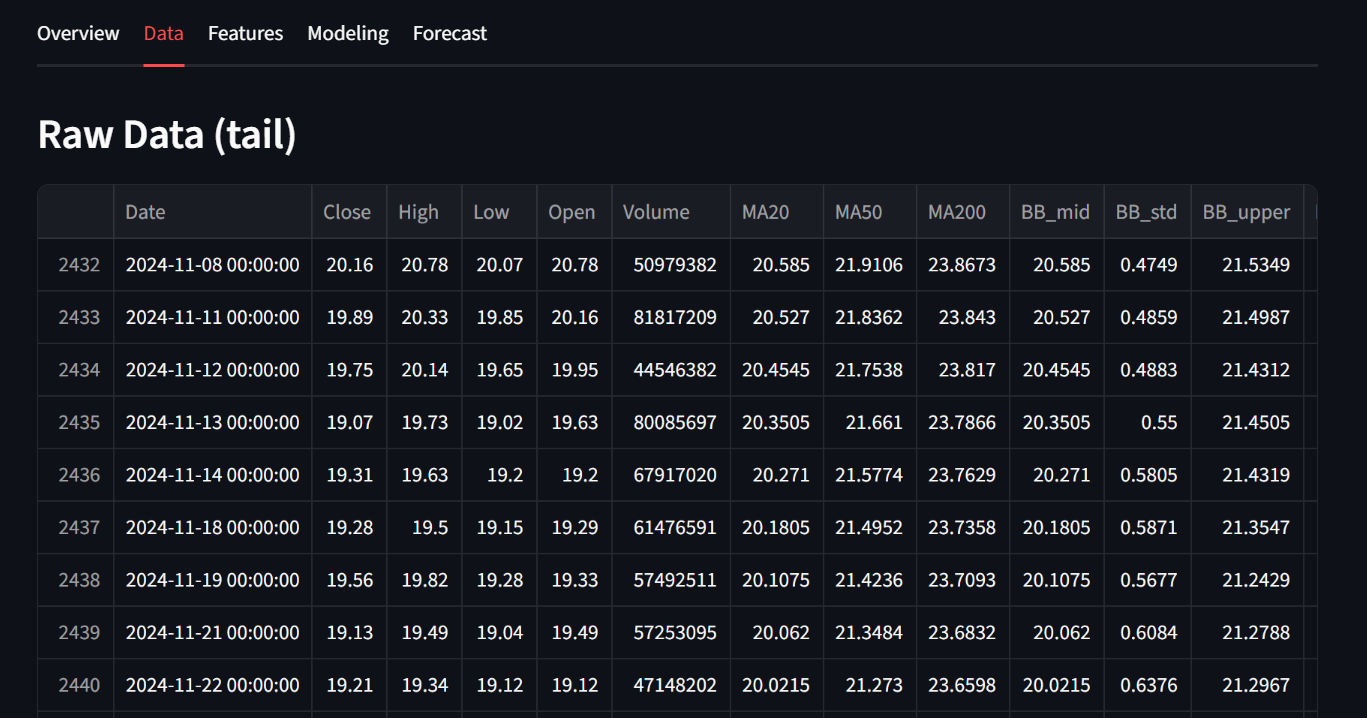
1. **Data Source (Yahoo Finance API)**
   * Historical stock market data is collected using the yfinance library.
   * Users can select the stock ticker (e.g., SUZLON, RELIANCE, TATAMOTORS) and define a date range.
   * The data includes **Open, High, Low, Close, Adjusted Close, and Volume** values.
2. **Data Preprocessing & Feature Creation**
   * Raw data is cleaned to handle missing values and ensure correct data types.
   * Technical indicators such as **Moving Averages (MA20, MA50, MA200)** and **Bollinger Bands** are generated.
   * Lag features (previous stock prices) are created to help models capture temporal dependencies.
3. **ML Model Training**
   * Data is split into training and testing sets using a **chronological split**.
   * Three machine learning models are implemented:
     + *Linear Regression (LR)* – simple baseline model.
     + *Decision Tree (DT)* – interpretable model capturing non-linear trends.
     + *Random Forest (RF)* – ensemble model for higher accuracy.
4. **Evaluation**
   * Model performance is measured using **Root Mean Squared Error (RMSE)**.
   * Visual comparisons between **Actual vs Predicted prices** are provided.
   * Scatter plots and residual analysis help in understanding accuracy.
5. **Forecasting & Visualization**
   * Once the model is trained, it generates forecasts for future stock prices (up to 4 years).
   * Forecasted data is displayed alongside historical data for comparison.
   * Interactive charts such as **candlestick plots, line charts, and feature importance graphs** allow users to analyze results intuitively.

# Implementation

The implementation of the **Stock Prediction App** was carried out in five main steps: **Data Collection, Preprocessing, Model Training, Evaluation, and Forecasting.** Each step is described below in detail.

**Step 1: Data Collection**

* The application uses the **Yahoo Finance API** through the yfinance Python library to fetch historical stock data.
* Users can select a stock ticker (e.g., SUZLON.NS, RELIANCE.NS, TATAMOTORS.NS) and specify a **date range** (start and end date).
* The collected dataset typically includes the following attributes:
  + **Open**: Opening price of the stock for the day.
  + **High**: Highest price of the stock for the day.
  + **Low**: Lowest price of the stock for the day.
  + **Close**: Closing price of the stock for the day.
  + **Adj Close**: Adjusted close price considering splits/dividends.
  + **Volume**: Number of shares traded during the day.
* The data is displayed in the **Data Tab** of the application.



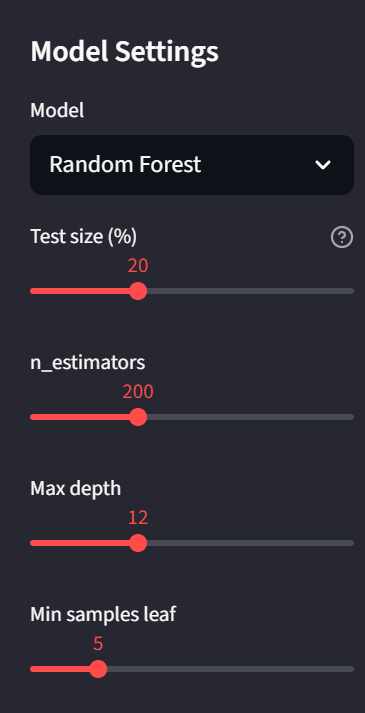
**Step 2: Preprocessing**

* The raw data collected often contains **missing values** or inconsistencies. These are handled by dropping invalid rows and converting columns into numeric format.
* **Feature Engineering** is performed to enrich the dataset:
  + **Lag Features**: Previous day(s) closing prices are added as new features (Close\_lag1, Close\_lag2, …). These help the model capture temporal dependencies.
  + **Moving Averages (MA20, MA50, MA200)**: Provide trend direction by smoothing stock price fluctuations.
  + **Bollinger Bands (Upper & Lower)**: Capture price volatility using rolling mean and standard deviation.
* The final dataset used for training consists of **lag features, moving averages, and the Open price (if available)**.



**Step 3: Model Training**

* To ensure realistic forecasting, a **chronological train-test split** is used (not random split).
  + Example: 80% of the earliest data is used for training, while the latest 20% is used for testing.
* Three ML models are implemented:
  + **Linear Regression**: Establishes a baseline by fitting a straight-line relationship between features and target price.
  + **Decision Tree**: Splits data into decision nodes to capture non-linear patterns.
  + **Random Forest**: An ensemble of multiple decision trees, reducing overfitting and improving accuracy.
* Model parameters (e.g., tree depth, number of estimators) can be tuned via the **sidebar controls**.



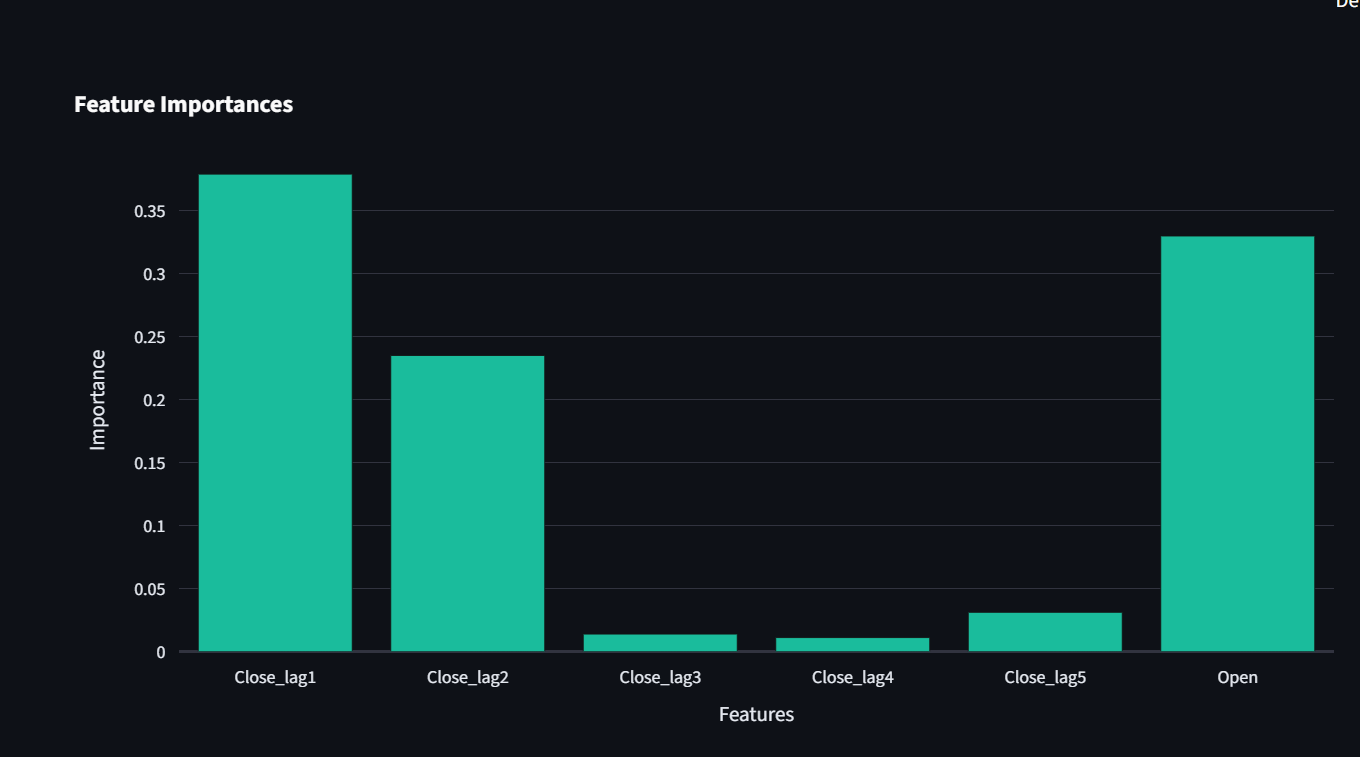
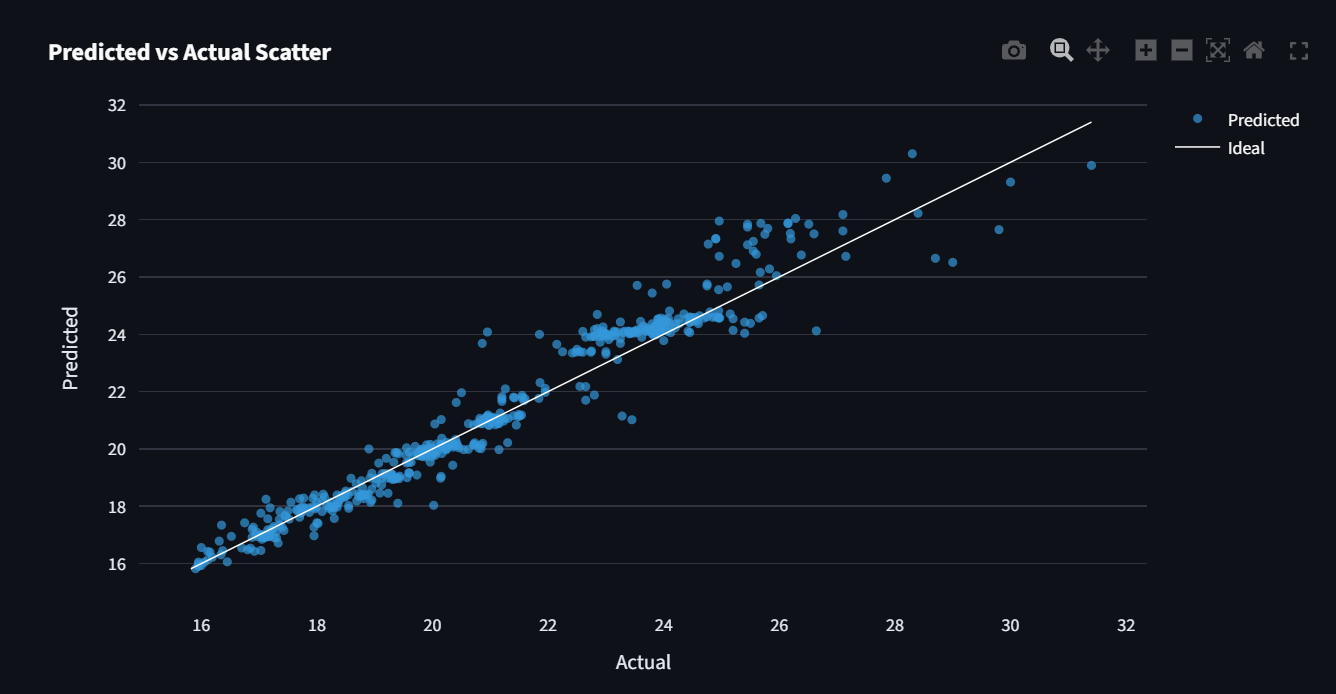
**Step 4: Evaluation**

* Model performance is assessed using **Root Mean Squared Error (RMSE)**:

RMSE=1n∑i=1n(yi−y^i)2RMSE = \sqrt{\frac{1}{n}\sum\_{i=1}^n (y\_i - \hat{y}\_i)^2}RMSE=n1​i=1∑n​(yi​−y^​i​)2​

where yiy\_iyi​ = actual values and y^i\hat{y}\_iy^​i​ = predicted values.

* Several visualizations are provided to interpret results:
  + **Actual vs Predicted Plot**: Shows how well the model’s predictions follow the actual stock price.
  + **Residual Plot**: Displays errors (difference between actual and predicted values) over time.
  + **Scatter Plot**: Compares predicted vs actual prices with an ideal line.
  + **Feature Importance Plot** (for Decision Tree and Random Forest): Shows which features contributed most to predictions.



**Step 5: Forecasting**

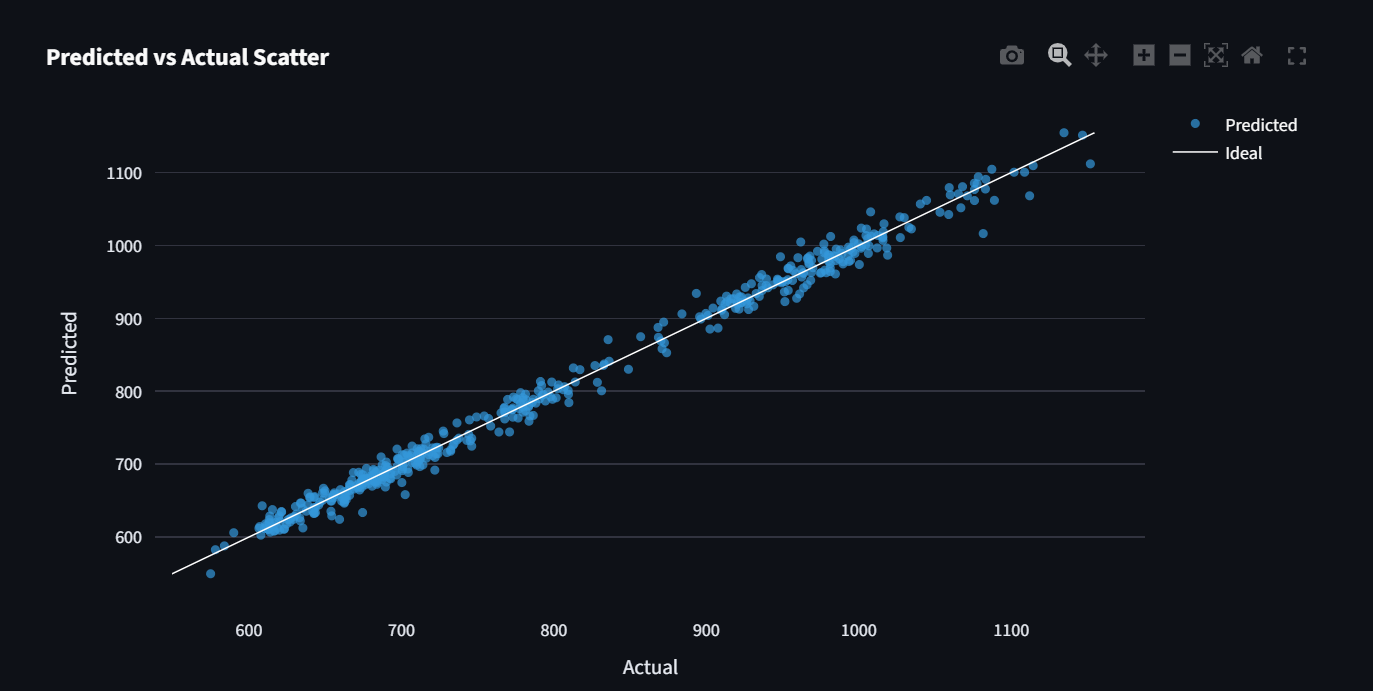
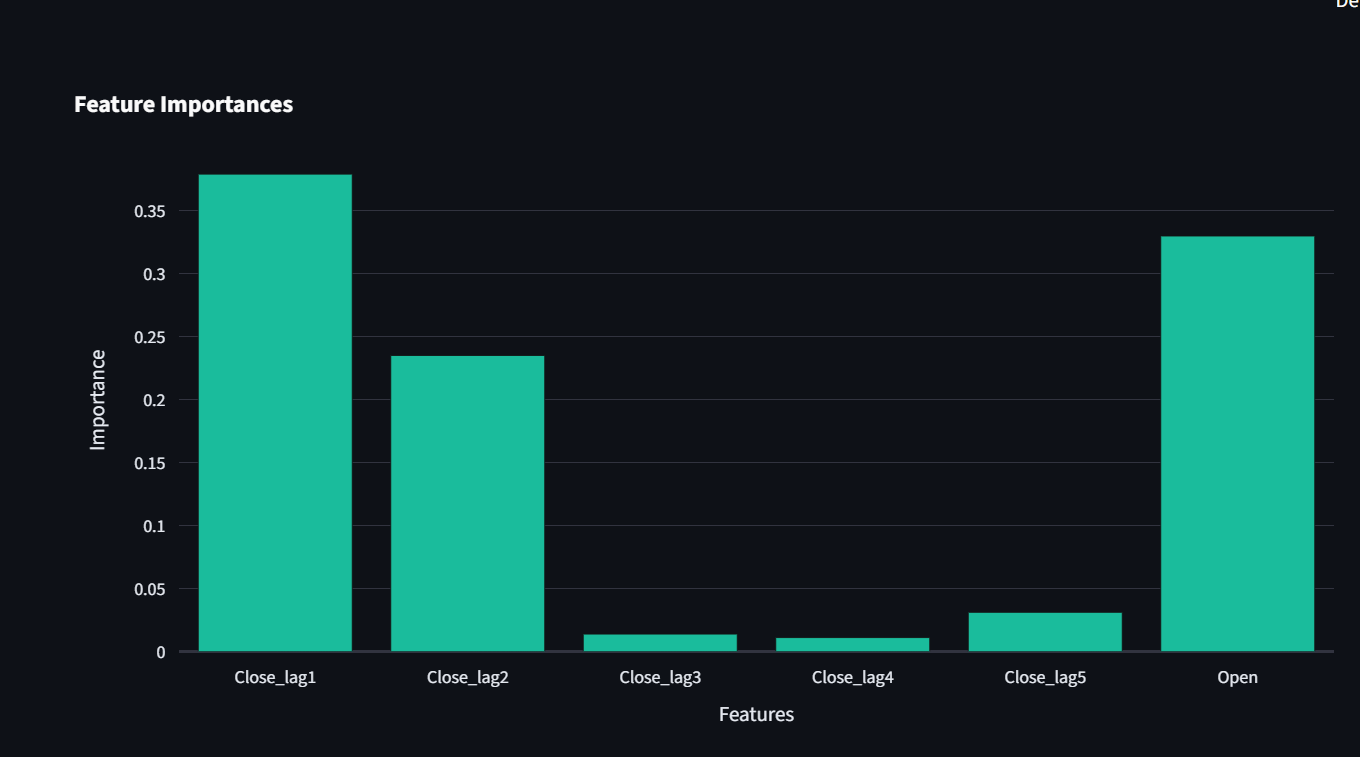
* Once the model is trained, it is refitted using the **entire dataset** to prepare for future forecasting.
* A **recursive prediction approach** is used:
  + The model predicts the next day’s stock price.
  + That prediction is fed back as an input feature to predict the following day.
  + This continues for the selected forecast horizon (1–4 years).
* Forecast results are displayed alongside the most recent actual data.
* The forecast chart shows:
  + **Recent Historical Prices** (in green).
  + **Predicted Future Prices** (in red dashed line).



# Results & Discussion

The **Stock Prediction App** provides a comprehensive platform for analyzing stock price data and applying machine learning models to forecast future movements. The results are presented through interactive charts and performance metrics, which help users evaluate the reliability and accuracy of the predictions.

**Visualization Results**

1. **Candlestick Chart with Indicators**
   * The candlestick chart provides a detailed view of stock price movements, showing Open, High, Low, and Close values.
   * Moving Averages (MA20, MA50, MA200) and Bollinger Bands are overlaid to indicate trend direction and volatility.
2. **Actual vs Predicted Graph**
   * This line chart compares the actual stock closing prices with the model’s predicted prices on the test dataset.
   * The closer the two lines overlap, the better the model’s accuracy.
3. **Feature Importance Chart**
   * For tree-based models (Decision Tree and Random Forest), feature importance scores highlight which features (e.g., lag features, moving averages) influenced predictions the most.
   * This helps interpretability and provides insights into which technical indicators were most significant.
4. **Forecast Line Chart**
   * The forecast visualization displays future predicted prices (dashed red line) alongside recent actual prices (green line).
   * This allows users to visually inspect expected trends over the selected forecast horizon (1–4 years).



**Discussion of Results**

* **Model Performance**  
  Among the three models tested — **Linear Regression, Decision Tree, and Random Forest** — the **Random Forest** consistently performed best. Its ensemble approach reduced overfitting and captured non-linear patterns better than the other two models.
  + *Linear Regression* was fast but oversimplified the stock price patterns.
  + *Decision Tree* captured non-linearity but tended to overfit when depth was not tuned properly.
  + *Random Forest* balanced accuracy and generalization, making it the most reliable choice.
* **RMSE Comparison**
  + The RMSE values varied depending on the stock and time period chosen.
  + On average, **Random Forest achieved the lowest RMSE**, followed by Decision Tree, with Linear Regression showing the highest error.
  + This indicates that Random Forest produced the most accurate short-term forecasts.
* **Forecast Reliability**
  + Forecasts generated by the models provide **approximate future trends**, not exact values.
  + While the short-term predictions were reasonably accurate, long-term forecasts showed increasing uncertainty, reflecting the real-world unpredictability of stock markets.
  + Users are advised to interpret results as **trend indications rather than financial advice**.

# Conclusion & Future Work

The Stock Prediction App successfully integrates data collection, machine learning, and visualization into a user-friendly platform. While predictions offer insights, they are not financial advice. In the future, the app can be enhanced by incorporating deep learning models such as LSTM, adding sentiment analysis from news and social media, and integrating portfolio-level analysis.

# References

* Streamlit Documentation: https://docs.streamlit.io/
* Yahoo Finance API (yfinance): https://pypi.org/project/yfinance/
* Scikit-learn Documentation: https://scikit-learn.org/
* Plotly Documentation: https://plotly.com/python/