1. Introduction

This project is a Flask-based stock analysis and prediction system. It enables users to search for tickers, fetch real-time and historical data from Yahoo Finance endpoints, visualize charts on the frontend, and predict next prices using multiple models (Linear Regression, Random Forest, ARIMA).

---

2. Objectives

- Collect real-time and historical stock data using Yahoo Finance public endpoints (without using the yfinance package).
- Preprocess data and engineer simple features such as lagged returns.
- Implement multiple models:
    - Linear Regression
    - Random Forest Regressor
    - ARIMA
- Evaluate model performance using RMSE (currently implemented).
- Provide APIs for charts and predictions intended for frontend visualization.

Planned (not yet included):

- RSI computation and charting
- Decision Tree Regressor
- R^2 Score
- SQLite persistence

---

3. Tools & Technologies

- Programming Language: Python
- Libraries:
    - Data Processing – pandas, numpy
    - Machine Learning – scikit-learn, statsmodels

    - Visualization – Chart.js (frontend)
    - Data Source – Yahoo Finance (HTTP endpoints via requests)
- Web Framework: Flask with flask-cors
- Environment Management: python-dotenv
- Database: None (SQLite not implemented yet)

---

4. System Architecture

1. User Input – Enter or select a stock ticker.
2. Data Collection – Fetch real-time and historical data via Yahoo Finance APIs.
3. Feature Engineering – Compute 5 lagged daily returns.
4. Preprocessing – Clean missing values, create supervised dataset, perform 90/10 split.
5. Model Training – Train models: Linear Regression and Random Forest on returns; ARIMA on close prices.
6. Prediction – Forecast next-day or multi-step to target date.
7. Visualization – Data is consumed by frontend (Chart.js) via Flask API endpoints.

---

5. Methodology

5.1 Data Collection

- Source: Yahoo Finance v8 chart API and v7 quote API via requests.
- Frequency: Daily.
- Features used: Close, Open, High, Low, Volume (primarily Close for modeling).

5.2 Relative Strength Index (RSI)

- Not implemented yet. Planned as a 14-day RSI with a dedicated chart overlay.

5.3 Data Preprocessing

- Convert timestamps to datetime.
- Build lagged daily returns (return_lag_1 ... return_lag_5).

- Drop rows with missing values created by lagging.
- Train/test split: 90% training and 10% testing.

5.4 Model Implementation

- Linear Regression – baseline model with lagged returns and iterative multi-step prediction.
- Random Forest Regressor – handles non-linearity on the same lagged features.
- ARIMA – applied on the closing price time-series (order = (5,1,0)).
- Decision Tree Regressor – not implemented yet.

5.5 Model Evaluation

- Current metric: RMSE (returned by /predict).
- Planned: R^2 Score for model interpretability.

---

6. Web Application

Frontend

- Static HTML/CSS pages.
- Chart.js demo (test_charts.html) shows line, bar, and multi-series charts.

Backend (Flask) Endpoints

- GET /search?q=<query> – Search ticker symbols (multi-region).
  Example: http://localhost:5000/search?q=AAPL
- GET /stock/<symbol> – Snapshot (current quote) and around one month of historical data.
  Example: http://localhost:5000/stock/AAPL
- GET /chart-data/<symbol> – Returns approximately six months of daily closing prices and volume data.
  Example: http://localhost:5000/chart-data/AAPL
- GET /predict?symbol=<symbol>&model=<lr|rf|arima>&from=YYYY-MM-DD&to=YYYY-MM-DD – Runs prediction using the selected model.
  Parameters:

- symbol: Stock ticker (e.g., AAPL)
- model: lr (Linear Regression), rf (Random Forest), arima
- from: (Optional) Start date for prediction
- to: Target date for prediction (if omitted, defaults to next trading day)

Example – Random Forest, next day prediction:
http://localhost:5000/predict?symbol=AAPL&model=rf

Example – ARIMA, targeted forecast:
http://localhost:5000/predict?symbol=AAPL&model=arima&from=2025-08-01&to=2025-08-28

Features Available Now

- Search and select tickers using /search.
- View stock price chart with /chart-data/<symbol>.
- Predict next-day or target-date price with evaluation metrics via /predict.

Pending Features

- RSI line graph.
- Watchlist persistence (SQLite).
- Decision Tree Regressor and multi-model comparison.

---

7. Results and Findings

- Linear Regression and Random Forest work reliably with lagged returns.
- ARIMA provides a classical baseline model.
- Prediction accuracy is modest due to stock market noise.
- Multi-exchange support and currency inference enhance international usability.
- RMSE provides a basic error metric; adding R^2 and RSI will improve interpretation.

---

8. Future Enhancements

- Add RSI (14-day) computation and visualization.

- Add Decision Tree Regressor and “Compare All” chart option.
- Compute and return R^2 alongside RMSE.
- Optional: evaluate using yfinance or continue with direct HTTP endpoints, with documentation of API rate limits.
- Add SQLite with Flask-SQLAlchemy for persistence (watchlists, history).
- Extend technical indicators (MACD, Bollinger Bands).
- Explore deep learning methods (LSTM/GRU).
- Deploy to cloud for real-time access.

---

9. Conclusion

The system provides a functional pipeline: search → fetch → preprocess → model → predict → visualize. Current accuracy is acceptable for educational purposes. The addition of RSI, R^2, and more models will enhance the analytical capabilities and make the system more aligned with technical analysis workflows.

---

10. References

- Yahoo Finance (alternative library): yfinance – pypi.org/project/yfinance
- Scikit-learn – scikit-learn.org
- Statsmodels (ARIMA) – statsmodels.org
- Flask – flask.palletsprojects.com
- Chart.js – chartjs.org