ChatGPT

AI-Powered Stock Analysis and Prediction Platform

Bachelor of Technology (CSE-AI&ML) – Capstone Project Phase 03
October 2025

Acknowledgements

We would like to thank our project supervisor, faculty, and peers for their guidance and support. Special thanks to the Computer Science & Engineering department for providing resources and encouragement throughout this project.

Abstract

This project presents a comprehensive **AI-powered stock analysis and prediction platform**. It combines real-time and historical market data with advanced analytics to help users make informed investment decisions. The platform automatically fetches stock prices and fundamentals (using the `yfinance` library [1]), computes technical indicators (e.g. moving averages, RSI, MACD, Bollinger Bands), and analyzes news sentiment via NLP (TextBlob and GPT-4). Multiple prediction models (Random Forest, ARIMA, BiLSTM, CNN) are available for forecasting future prices [2]. Additional features include portfolio optimization (using Markowitz mean-variance theory [3]) and graph-based discovery of related stocks. All functionality is exposed through a user-friendly Streamlit dashboard and a Flask REST API. The platform's design and capabilities make it a versatile tool for investors and analysts.

Table of Contents

List of Figures and Tables

Introduction / Project Overview

This project implements a comprehensive **AI-powered stock analysis and prediction platform**. It provides users with tools to retrieve real-time and historical market data, perform advanced analyses, and visualize results via an interactive dashboard. Key capabilities include:

- **Data Fetching:** Automatic retrieval of stock price histories, volume, and financial fundamentals from Yahoo Finance (using the `yfinance` library [1]).
- **Technical Analysis:** Calculation of 20+ indicators (Moving Averages, RSI, MACD, Bollinger Bands, etc.) to identify trends and momentum.
- **Sentiment Analysis:** Processing of financial news articles to gauge market mood. Sentiment scores are obtained via simple libraries (TextBlob) and advanced AI (OpenAI's GPT-4) for nuanced interpretation.
- **Predictive Models:** A suite of forecasting models: Random Forest (feature-based), ARIMA (statistical time-series) [2], BiLSTM (neural sequential model), and 1D CNN (convolutional time-series model). Users can choose or auto-select models based on data.
- **Portfolio Optimization:** Calculation of an efficient portfolio allocation under Markowitz's mean-variance framework [3].
- **Related Stocks Discovery:** Graph-based analysis to suggest similar stocks, helping users diversify or identify new opportunities.
- **User Interface & API:** A Streamlit web app for interactive use, plus a Flask REST API for programmatic access.

By integrating traditional finance techniques with modern AI, the platform delivers institutional-grade analysis in an accessible format. All stock and indicator data (e.g., OHLC prices) are sourced via Yahoo Finance, a widely used financial data provider [1]. The result is a powerful tool for both novice and experienced investors.

Literature Review

Prior work in stock forecasting spans many methods. Traditional time-series models like ARIMA have long been used for financial prediction [2]. Machine learning techniques (SVM, Random Forest, etc.) have also been applied to market data [6]. Recent research incorporates deep learning (LSTM, BiLSTM, CNN) to capture complex temporal patterns in stock prices [6]. Meanwhile, sentiment analysis of news and social media is increasingly used to inform predictions. Some multi-modal approaches combine price charts with text data for improved accuracy. This project builds on these insights by fusing technical indicators, NLP-based sentiment, and multiple modeling approaches in one platform. For example, Shahzad et al. (2019) used SVR for short-term index prediction, and Ao & Zhu (2018) applied SVM to stock forecasting [6], illustrating the variety of techniques that inform our design.

System Architecture

The platform uses a modular architecture with clear separation of components:

- **Frontend (Streamlit Dashboard):** The main user interface for input and visualization. Users can search for stocks, view charts, and adjust settings.
- **Backend API (Flask):** A RESTful API (`api.py`) allows external programs to query data and predictions.
- **Data Collection:** The `stock_data.py` module handles all market data retrieval. It connects to Yahoo Finance via `yfinance` [1] to download historical prices, volumes, and fundamentals. It also obtains company information (industry, sector, financial ratios) and global market indicators (major indices, VIX).
- **News & Sentiment:** The `news_analysis.py` module fetches recent news articles (from Financial Modeling Prep or Yahoo News) for each stock. It then analyzes text sentiment, using TextBlob (fast polarity scoring) and optionally GPT-4 for deeper context. It can also summarize news impact with AI.
- **Analysis & Modeling:** Processed data flows into analytical modules (`model.py`):
- **Technical Indicators:** `stock_data.py` computes indicators like moving averages, RSI, MACD, Bollinger Bands, etc. These features feed the models.
- **Prediction Models:** The system supports multiple models. A Random Forest regressor trains on the engineered features. An ARIMA model is used for classical time-series forecasting [2]. Deep learning models (Bidirectional LSTM and 1D CNN) capture complex temporal patterns in the data.
- **Portfolio Optimization:** The `portfolio_optimization.py` module implements Markowitz's mean-variance algorithm to find the efficient frontier and optimal asset weights [3].
- **Related Stocks:** The `generate_related_stocks_graph.py` module analyzes stock correlations/graph structure to suggest similar stocks.
- **Visualization:** The `visualization.py` module generates plots (candlestick charts, line graphs, indicator overlays, portfolio frontiers, etc.) using Plotly and Matplotlib. For instance, Figure 1 below is a candlestick chart produced by this module using Yahoo data [4].

Data flows from collection to analysis to presentation. When a user requests a stock, the backend calls `stock_data.py` and `news_analysis.py` to gather inputs. These feed into the chosen models, whose outputs are then sent to `visualization.py` and the frontend.

Workflow / Use Case

A typical user interaction proceeds as follows:

1. **Stock Query:** The user enters a stock ticker or selects stocks in the Streamlit app.
2. **Data Fetching:** The system calls `stock_data.py` to download historical price/volume data and financial info via Yahoo Finance (using `yfinance` [1]). Simultaneously, `news_analysis.py` retrieves recent news headlines for the stock.
3. **Indicator Computation:** Technical indicators (e.g. moving averages, RSI, MACD, Bollinger Bands) are computed on the price history. News headlines are processed by TextBlob (giving polarity scores between -1 and +1) and by GPT-4 (for nuanced sentiment and summaries).

4. **Price Prediction:** The user selects a model (or uses default). The prepared features (prices + indicators + sentiment scores) are fed into the model:
5. **Random Forest:** Trained on the feature set to predict future price.
6. **ARIMA:** A statistical time-series model fitted on the price history to forecast short-term future values [2].
7. **BiLSTM:** A deep learning network that processes the sequence of past prices (forward and backward) to capture trends.
8. **CNN:** A 1D convolutional network extracting local temporal patterns.
   The `predict_prices()` function in `model.py` chooses and runs the model.
9. **Visualization:** Results are displayed. Figure 1 (below) shows an example candlestick chart with Apple's price history; Figure 2 shows an example efficient frontier from portfolio optimization. Prediction outputs (future price curve) are overlaid on charts.
10. **Advanced Analysis:** The user can switch to portfolio mode, inputting multiple tickers. The system calls `portfolio_optimization.py` to compute optimal weights, plotting the Markowitz efficient frontier. For related stocks, `generate_related_stocks_graph.py` uses graph analysis to suggest similar companies.
11. **Output:** All charts, tables, and summaries appear in the dashboard. The user can also query the results via the REST API (`api.py`). Data is cached to speed up repeated requests.

Overall, the workflow is fully automated: user input → data collection → analysis/prediction → output visualization. Key steps use well-known libraries and concepts (e.g. Yahoo Finance via yfinance [1], ARIMA time-series forecasting [2], and Markowitz portfolio optimization [3]).

Project Structure

The repository is organized into modules by functionality:

- `run_streamlit_app.py`: Launches the Streamlit web application and defines the main UI layout.
- `api.py`: Implements a Flask REST API exposing core functions (data fetch, predictions, optimization) for programmatic use.
- `stock_data.py`: Fetches all stock-related data. Uses **yfinance** to download historical OHLCV data and fundamentals [1]. Computes technical indicators and pre-processes data.
- `news_analysis.py`: Retrieves financial news articles and computes sentiment scores. Integrates TextBlob and OpenAI GPT-4 for sentiment and summaries.
- `model.py`: Core predictive models. Contains functions to create features, train/test models, and make forecasts. Supports Random Forest, ARIMA, BiLSTM, CNN.
- `portfolio_optimization.py`: Implements mean–variance optimization using Markowitz's model [3]. Computes efficient frontier and optimal asset allocations.
- `generate_related_stocks_graph.py`: Builds and analyzes a graph of stock relationships (e.g., correlations). Identifies related/similar stocks.
- `visualization.py`: Charting utilities (Plotly/Matplotlib). Generates candlestick plots, line charts, bar charts, scatter plots (efficient frontier), etc.
- `utils.py`: Miscellaneous helper functions (e.g. for data cleaning, indicator formulas, date handling).
- **Datasets/Assets**: CSV files like `stock_industry_sector_dataset.csv` for industry metadata, and precomputed JSON files (e.g. `related_stocks_AAPL.json`) caching related-stock data.

- `requirements.txt`: Lists core Python dependencies. Additional files `requirements_advanced.txt` and `requirements_torch.txt` include extra libraries (e.g. TensorFlow/PyTorch, advanced NLP packages) for optional features.

Module Descriptions

Data Collection (`stock_data.py`)

This module retrieves and preprocesses market data. It uses the Python **yfinance** library to fetch historical price data (open, high, low, close, volume) and company fundamentals from Yahoo Finance [1]. Key functions include:
- `get_stock_data()`: Downloads daily price series and computes 20+ technical indicators (moving averages, RSI, MACD, Bollinger Bands, etc.). These enrich the feature set for models.
- `get_company_info()`: Retrieves fundamental info such as industry, sector, business summary, P/E ratio, dividend yield, etc.
- `get_historical_earnings()`, `get_quarterly_earnings()`: Fetch earnings history to provide financial context.
- `get_market_sentiment()`: Gathers broad market sentiment via indices (e.g., S&P 500, VIX).

By centralizing data collection and feature engineering, this module feeds cleaned, enriched data into the modeling pipeline.

Sentiment Analysis (`news_analysis.py`)

This module captures the "voice of the market" by analyzing news sentiment. It has two main components:
- **News Retrieval**: It fetches recent news articles for a given stock (using APIs like Financial Modeling Prep or Yahoo News).
- **Sentiment Scoring**: Each article is analyzed for sentiment. By default, the module uses **TextBlob**, a simple NLP library that outputs a polarity score from -1 (negative) to +1 (positive). If an OpenAI API key is provided, it also queries **GPT-4** to get a more nuanced sentiment assessment and summary of the news's impact.
The result is a list of sentiment scores and summaries associated with each news item. These scores are used as additional features to indicate market mood.

Machine Learning & Deep Learning Models (`model.py`)

This module implements the core prediction engines. It consists of:
- **Feature Engineering:** A function constructs a feature matrix from price history, including all technical indicators and previous price changes.
- **Model Training/Prediction:** Several algorithms are supported:
- **Random Forest**: Trains a `RandomForestRegressor` on the engineered features for a quick, feature-based forecast.
- **ARIMA**: A statistical time-series model (ARIMA) fitted on the closing prices to predict future values [2]. This serves as a classical benchmark.
- **BiLSTM**: A Bidirectional LSTM network that processes the sequence of past prices (both forward and backward) to capture temporal dependencies. LSTM architectures have been shown effective in stock prediction tasks [7].

- **CNN**: A 1D Convolutional Neural Network that applies convolutional filters over the time axis to extract local patterns in the price series.

The function `predict_prices()` manages model selection: it chooses an appropriate model based on data length (e.g. very short history uses ARIMA by default) or user preference. Models are trained (or a pre-trained model is loaded) and used to forecast the next N days' closing prices.

Visualization (`visualization.py`)

This module generates all the charts and plots for the dashboard. Key functions include:
- **Candlestick Charts:** Plotting OHLC candlesticks over time (see *Figure 1*).
- **Line Charts:** Plotting price history, predicted future prices, or indicator curves.
- **Bar/Scatter Charts:** For example, a scatter plot of portfolios on risk-return axes (*Figure 2*).
- **Dashboard Figures:** Combining data into summary visuals (e.g., sentiment over time, performance comparisons).

Plots are created using Plotly for interactivity (in the web app) and Matplotlib for static reports. For instance, Figure 1 below shows a candlestick chart of Apple Inc.'s stock (data from Yahoo Finance [4]) with key price movements highlighted.

Figure 1. Candlestick chart of Apple Inc.'s stock price (March 2009–May 2011). Green candles indicate days where the closing price was higher than the opening price. Data is sourced from Yahoo Finance [4].

Portfolio Optimization (`portfolio_optimization.py`)

This module applies **Markowitz's mean–variance optimization** to user-selected stocks. Given expected returns and the covariance matrix of returns, it computes the efficient frontier – the set of portfolios that maximize expected return for a given risk level. The `optimize_portfolio()` function determines the optimal weights for a target return or risk. Figure 2 (below) illustrates a sample Pareto efficient frontier, where each red point is an optimal portfolio for a given risk [5]. This approach is grounded in Modern Portfolio Theory, as originally developed by Harry Markowitz [3].

Figure 2. Example of a Pareto efficient frontier from portfolio optimization. Each red point represents a portfolio that offers the maximum expected return for its level of risk (standard deviation). This concept follows Markowitz's mean–variance theory [5].

Related Stocks Discovery (`generate_related_stocks_graph.py`)

To help users find new investment ideas, this module analyzes relationships between stocks. It constructs a graph where nodes represent companies and edges encode similarity (e.g. correlation of price movements, common industry). Using network analysis, it identifies clusters of related stocks. For example, given a tech stock, it might suggest other tech firms with correlated behavior. The graph-based approach enables multi-hop discovery (e.g. friends-of-friends in the stock market graph). In future work, this could be enhanced with graph neural networks or knowledge graphs to capture more nuanced relationships.

How to Run the Project

1. **Clone the repository:**

```
git clone <repository-url>
cd <repository-directory>
```

2. **Create a Python virtual environment:**

```
python -m venv venv
source venv/bin/activate        # On Windows: venv\Scripts\activate
```

3. **Install dependencies:**

```
pip install -r requirements.txt
```

   Note: For advanced features (deep learning, GPT), also install from `requirements_advanced.txt` and `requirements_torch.txt`.
4. **Set up API keys (optional but recommended):**
5. **Financial Modeling Prep (FMP) API key:** Needed for premium news data.
6. **OpenAI API key:** Enables GPT-4 sentiment analysis and summaries.
   Set them in the environment:

```
export FMP_API_KEY="your_fmp_api_key"
export OPENAI_API_KEY="your_openai_api_key"
```

7. **Run the Streamlit application:**

```
streamlit run run_streamlit_app.py
```

8. **Access the dashboard:** Open a web browser at `http://localhost:8501`. The interactive UI will appear for inputting tickers, viewing charts, etc.
9. **Use the API (optional):** Start the Flask API by running `api.py`. Endpoints like `/predict`, `/sentiment`, etc., allow you to retrieve data and analysis programmatically (e.g. via `curl` or Python requests).

Make sure the virtual environment is active whenever running the app. The first run may take time as data downloads and models are initialized.

Results / Example Outputs

The platform produces rich visual outputs. Examples include: candlestick charts of stock prices, overlaid with technical indicators; line plots of actual vs. predicted prices; and graphs of portfolio risk-return. Figures

1 and 2 above illustrate sample outputs. In the dashboard, users will see interactive versions of these plots, along with tables of indicator values and news sentiment scores. For instance, after querying "AAPL", one sees Apple's recent price chart, a predicted price curve (from the selected model), sentiment bar graphs for recent headlines, and a summary of the chosen portfolio allocation. (In a PDF or report, these interactive charts would appear as static images.)

Conclusion and Future Work

The AI-Powered Stock Analysis and Prediction Platform demonstrates how multiple data modalities and AI models can be combined to provide deep insights into financial markets. By integrating real-time data collection, technical analysis, NLP-based sentiment scoring, and diverse predictive models, it offers a one-stop solution for stock analysis. Users can not only see price forecasts but also understand the factors (indicators and news) influencing those forecasts, and optimize portfolios under risk constraints.

Future enhancements could include:
- **Refactor Feature Engineering:** Consolidate technical indicator calculations into a shared utility to avoid duplication between modules.
- **Expand Data Sources:** Add social media sentiment (Twitter, Reddit) or alternative financial APIs for broader coverage.
- **Advanced Models:** Incorporate graph neural networks to better model inter-stock relationships (harnessing `graph_neural_networks.py`), and explore reinforcement learning for automated trading strategies (as hinted by `reinforcement_learning.py`).
- **User Personalization:** Implement user accounts to save portfolios, preferences, and analysis history.
- **Real-time Alerts:** Add a notification system (email/SMS) for custom alerts on price thresholds or sentiment shifts.
- **Deployment Enhancements:** Dockerize the app for scalable, cloud deployment; set up continuous integration pipelines.

With these improvements, the platform could serve as an even more powerful assistant for investors and analysts, bringing cutting-edge AI tools to everyday financial decision-making.

References

- GeeksforGeeks, *"What is YFinance library?"* (Updated May 2025) [1].
- Wikipedia, *"Autoregressive integrated moving average"* (ARIMA) [2].
- Wikipedia, *"Markowitz model"* (Modern Portfolio Theory) [3].
- Sang *et al.*, "A Stock Prediction Method Based on Heterogeneous Bidirectional LSTM" (*Appl. Sci.* 14(20):9158, 2024) [7] [6].
- Wikimedia Commons, *"File: Apple Inc Candle Stick Chart Mar 2009 to May 2011.png"* (Chart by Yndesai, 2011) [4].
- Wikimedia Commons, *"File: Pareto Efficient Frontier for the Markowitz Portfolio selection problem..png"* (Chart by Marcuswikipedian, 2012) [5].

---

[1] What is YFinance library? - GeeksforGeeks
https://www.geeksforgeeks.org/machine-learning/what-is-yfinance-library/

2 Autoregressive integrated moving average - Wikipedia
https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

3 Markowitz model - Wikipedia
https://en.wikipedia.org/wiki/Markowitz_model

4 File:Apple Inc Candle Stick Chart Mar 2009 to May 2011.png - Wikimedia Commons
https://commons.wikimedia.org/wiki/File:Apple_Inc_Candle_Stick_Chart_Mar_2009_to_May_2011.png

5 File:Pareto Efficient Frontier for the Markowitz Portfolio selection problem..png - Wikimedia Commons
https://commons.wikimedia.org/wiki/File:Pareto_Efficient_Frontier_for_the_Markowitz_Portfolio_selection_problem..png

6 7 A Stock Prediction Method Based on Heterogeneous Bidirectional LSTM
https://www.mdpi.com/2076-3417/14/20/9158