AI BASED STOCK MARKET FORECASTING SYSTEM

Pothineni Prakash[#1], Pasham Naga Sheshu [#2], Pamidi Samba Siva Rao [#3], Mrs.J.Fahamitha [#4]

Email: prakshpothineni@gmail.com , nagasheshu775@gmail.com , pamidisambasivarao333@gmail.com[#123] UG Student, Department of Computer Science and Engineering, School of Engineering and Technology, Dhanalakshmi Srinivasan University, Trichy-621112-Tamilnadu.

[#4] Assistant Professor, Department of Computer Science and Engineering, Dhanalakshmi Srinivasan Institute of Technology, Trichy-621112- Tamil Nadu.

Abstract- *Stock market prediction is a complex and challenging task due to the highly volatile and non-linear behavior of financial markets. This paper presents the design and implementation of an AI Based Stock Market Forecasting System — a real-time web application that integrates live market data retrieval, deep learning-based LSTM forecasting, Meta's Prophet time-series model, technical indicator analysis (SMA, EMA, RSI), and real-time news sentiment scoring using TextBlob and NewsAPI. The proposed system is implemented using Python and Streamlit and provides an interactive dashboard with candlestick chart visualization, multi-model price prediction, RSI-based trading signals, per-user prediction history, and model comparison via RMSE. The system is designed for educational and research purposes and demonstrates how multiple AI techniques can be combined into a coherent, user-authenticated decision-support platform for stock market analysis.*

Keywords- Stock Market Prediction, LSTM, Prophet Model, News Sentiment Analysis, Technical Indicators, Streamlit, Deep Learning, RSI, EMA, SMA.

I. INTRODUCTION

Financial markets are among the most complex dynamic systems in the modern world. Stock prices are influenced by a vast array of factors including corporate earnings, macroeconomic indicators, geopolitical events, interest rate decisions, investor sentiment, and global supply chain dynamics. These interdependencies produce highly non-linear, non-stationary time-series data that challenge conventional statistical models.

The emergence of artificial intelligence and machine learning has opened new avenues for financial forecasting. Unlike rule-based trading systems, AI-driven models can learn complex patterns from large historical datasets without requiring explicit programming of market rules. This capability makes them particularly well-suited for capturing the subtle dynamics that drive stock price movements.

This paper presents the design and implementation of an AI Based Stock Market Forecasting System — a comprehensive web application that integrates multiple forecasting techniques within a single, user-authenticated platform. The system combines Long Short-Term Memory (LSTM) deep learning networks, Meta's Prophet time-series model, technical indicator analysis, and real-time news sentiment scoring to provide multi-perspective forecasts for any publicly traded stock.

The application is built using Python and Streamlit, enabling an interactive web interface accessible from any browser. It fetches live and historical price data from the Yahoo Finance API, computes technical indicators including SMA, EMA, and RSI, generates price forecasts up to 365 days into the future, and evaluates competing models using Root Mean Squared Error (RMSE). All predictions are logged per user for historical review and comparison.

The system is primarily designed for educational and research purposes, providing a transparent platform for understanding how different AI methodologies approach the stock forecasting problem. It does not provide financial advice and does not execute real monetary transactions.

II. RELATED WORK

The application of machine learning to stock market prediction has been widely studied over the past two decades. Early work by Refenes and Abu-Mostafa [4] demonstrated that neural networks could model non-linear relationships in financial data more effectively than traditional econometric models such as ARIMA and GARCH.

Patel et al. [3] showed that combining trend-deterministic data preparation techniques with random forest and support vector machine classifiers yields significant improvements in prediction accuracy for Indian stock indices. Their work highlighted the importance of feature engineering in financial machine learning pipelines.

Mohan et al. [2] conducted a comprehensive comparison of deep learning frameworks for stock price prediction, demonstrating that LSTM networks consistently outperform shallow neural networks and classical statistical methods on large financial datasets. The temporal memory mechanism of LSTM makes it especially suited for capturing long-range dependencies in price sequences.

Taylor and Letham [4] introduced the Prophet model as a decomposable time-series forecasting framework that separates trend, weekly seasonality, annual seasonality, and holiday effects. Prophet's robustness to missing data and its ability to handle structural breaks make it well-suited for financial forecasting applications.

Bollen et al. [5] established that public mood derived from social media platforms correlates significantly with stock market movements, with sentiment signals improving directional prediction accuracy by up to 87.6%. This finding motivated the integration of news sentiment analysis as an additional signal in the proposed system. Wang et al. [6] extended this work to deep learning-based sentiment models, further validating the utility of textual data in financial forecasting.

III. SYSTEM OVERVIEW

The proposed AI Based Stock Market Forecasting System is developed as a real-time, web-based application that integrates live market data retrieval, technical analysis, deep learning inference, and sentiment scoring into a single unified platform. The system is intended for educational and simulation purposes, allowing users to observe and compare multiple forecasting methodologies without financial risk.

At a high level the system workflow proceeds as follows: the user authenticates via a secure login portal and selects a stock symbol together with a date range. The data acquisition layer fetches historical OHLCV data from Yahoo Finance. The technical analysis layer computes SMA, EMA, and RSI indicators on the retrieved data. The forecasting layer runs either the LSTM or Prophet model to generate future price estimates. Concurrently, the sentiment layer retrieves recent news headlines and scores market mood. All results are visualized through interactive charts and logged to the user's prediction history.

The system is structured around five interactive modules accessible from the sidebar navigation menu: Dashboard, Prediction, History, News Sentiment, and Model Compare. Each module is backed by a dedicated Python component, ensuring clean separation of concerns and ease of maintenance.

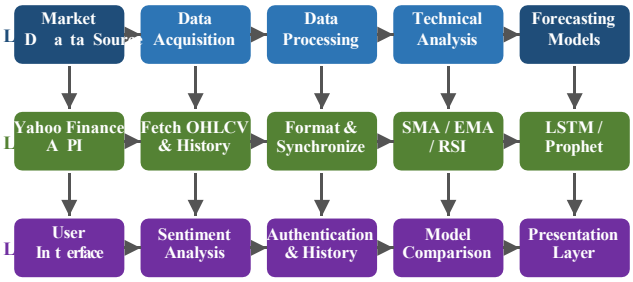

FIGURE 1: System Overview of the AI Stock Market Forecasting System.

IV. SYSTEM ARCHITECTURE

The system architecture is designed using a layered and modular approach to ensure clarity, scalability, and ease of understanding. Each layer performs a specific function and communicates with adjacent layers through well-defined data flows. This design isolates responsibilities, reduces system complexity, and simplifies future enhancements.

The architecture comprises seven distinct layers: Market Data Source, Data Acquisition, Data Processing, Technical Analysis, Forecasting Models, Sentiment Analysis, and Presentation. The interaction among these layers is illustrated in Figure 1.

A. Market Data Source

The primary source of data is the Yahoo Finance API, accessed through the yfinance Python library. The API delivers essential trading information including open, high, low, close prices, trading volume, and timestamps for any selected stock symbol and date range. Since the data source is external and dynamic, the system handles network delays and missing data points without interrupting operation.

B. Data Acquisition Layer

The data acquisition layer is implemented in data_fetcher.py. The fetch_stock(ticker, start, end) function sends an HTTP request to Yahoo Finance and returns a structured pandas DataFrame. This layer acts as the bridge between the external market environment and all internal processing components. Basic error handling ensures that only complete, valid data is forwarded downstream.

C. Data Processing Layer

Raw market data is cleaned and normalized before use. This includes dropping null rows, resetting the DataFrame index, ensuring correct data types for all columns, and verifying that the date range contains sufficient records for indicator calculation and model training. A minimum of 60 data points is enforced for LSTM inference.

D. Technical Analysis Layer

The technical analysis layer, implemented in indicators.py, computes three widely used financial indicators on the processed price data:

- **SMA-20:** The 20-period Simple Moving Average smooths short-term price fluctuations to reveal the underlying medium-term trend direction.
- **EMA-20:** The 20-period Exponential Moving Average assigns greater weight to recent prices, making it more responsive to current market conditions than the SMA.
- **RSI-14:** The 14-period Relative Strength Index measures the speed and magnitude of recent price changes on a scale of 0 to 100, identifying overbought conditions above 70 and oversold conditions below 30.

These indicators are overlaid on candlestick charts in the dashboard and serve as inputs to the trading signal generation module.

E. Forecasting Models Layer

Two complementary forecasting models are integrated to provide diverse predictive perspectives. The LSTM model captures sequential temporal dependencies in price data through deep learning, while the Prophet model performs classical time-series decomposition. Both models are described in detail in Section V.

F. Sentiment Analysis Layer

The sentiment analysis layer fetches real-time news headlines for the target stock and classifies overall market mood using natural language processing. This qualitative signal complements the quantitative price-based forecasts by capturing information that may not yet be reflected in historical price data.

G. Presentation Layer

The presentation layer is built with Streamlit and Plotly. It displays interactive candlestick charts with indicator overlays, forecast visualizations, portfolio balances, and order history. All backend computations are reflected in the interface in real time. Users can switch between modules, trigger model runs, and download their prediction history without leaving the application.

TABLE I: System Architecture Layer Summary

Layer	Module File	Responsibility
Data Acquisition	data_fetcher.py	Fetch OHLCV from Yahoo Finance
Data Processing	app.py (inline)	Clean, validate, normalize data
Technical Analysis	indicators.py	Compute SMA-20, EMA-20, RSI-14
LSTM Forecasting	lstm_model.py	Train and infer LSTM network
Prophet Forecasting	prophet_model.py	Decompose and forecast time-series
Sentiment Analysis	news_sentiment.py	Score news headlines via TextBlob
Authentication	login.py	User registration and session mgmt
History	history.py	Log and retrieve prediction records
Evaluation	evaluation.py	Compute RMSE for model comparison
Signals	signals.py	Generate RSI-based BUY/SELL/HOLD
Presentation	app.py (Streamlit)	Interactive charts and UI

V.FORECASTING MODELS

The system implements two distinct forecasting approaches that represent fundamentally different paradigms: deep learning-based sequential modeling through LSTM, and classical decomposition-based forecasting through Prophet. Providing both models enables users to compare their strengths and weaknesses under varying market conditions.

A. Long Short-Term Memory (LSTM) Model

Long Short-Term Memory networks, introduced by Hochreiter and Schmidhuber [6], are a specialized class of recurrent neural networks designed to learn long-range temporal dependencies in sequential data. Standard RNNs suffer from the vanishing gradient problem, which prevents them from learning patterns over long time horizons. LSTM addresses this through a gating mechanism comprising three learned gates — forget, input, and output — that regulate information flow through the cell state.

In the proposed system, the LSTM model is implemented in lstm_model.py using TensorFlow and Keras. The model architecture consists of two stacked LSTM layers of 50 units each, followed by a Dense output layer with a single neuron producing the next-step price prediction. The input to the network is a sliding window of 60 consecutive normalized closing prices.

Data preprocessing involves MinMaxScaler normalization to the range [0, 1], which stabilizes gradient flow during training. The model is compiled with the Adam optimizer and mean squared error loss function, and trained for 5 epochs with a batch size of 32. During inference, the last 60 available data points are fed to the trained network to predict the next closing price, which is then inverse-transformed to the original price scale.

B. Prophet Time-Series Model

Prophet, developed by Taylor and Letham at Meta, is an additive regression model that decomposes a time series into trend, seasonality, and holiday components. The model is designed to handle the practical challenges of real-world forecasting, including missing data, outliers, and structural breaks caused by one-off events.

The Prophet implementation in prophet_model.py log-transforms closing prices before model fitting to stabilize variance across different price scales. The model accepts a DataFrame with columns ds (date) and y (log price), fits an additive decomposition, and generates a forecast for a user-specified horizon of 1 to 365 days. Predicted values together with upper and lower confidence intervals are exponentiated back to the original price scale before presentation.

Prophet is particularly effective for stocks with strong weekly or annual seasonality patterns, and its forecasts are more interpretable than LSTM predictions because each component can be visualized separately. However, it may underperform LSTM on stocks with highly irregular, event-driven price behavior.

C. Model Comparison

The Model Compare module evaluates all three forecasting approaches — Moving Average (MA-5), Prophet, and LSTM — on identical data using RMSE as the evaluation metric. The comparison is performed on the selected stock and date range, and the system automatically identifies and reports the model with the lowest RMSE as the best performer for that specific scenario.

TABLE II: Forecasting Model Comparison

Model	Type	Strengths	Limitations
LSTM	Deep Learning	Captures non-linear temporal patterns; strong on trending data	Requires large dataset; slow training
Prophet	Decomposition	Handles seasonality; robust to outliers; interpretable	Weaker on irregular price behavior
MA-5	Statistical	Simple, fast, and reliable on stable markets	Lags market; poor on volatile data

VI. TECHNICAL INDICATORS AND TRADING SIGNALS

Technical analysis forms an essential component of the proposed system, providing rule-based signals that complement the model-driven forecasts. Technical indicators are mathematical calculations based on historical price and volume data that help identify market trends, momentum, and potential reversal points.

A. Simple Moving Average (SMA)

The 20-period Simple Moving Average is computed as the arithmetic mean of the last 20 closing prices. SMA serves as a trend filter: when the current price is above the SMA-20, the market is considered to be in a bullish regime; when below, the market is considered bearish. SMA is implemented using pandas' rolling(20).mean() operation.

B. Exponential Moving Average (EMA)

The 20-period Exponential Moving Average applies exponentially decreasing weights to past prices, giving greater importance to more recent data. EMA is more responsive to price changes than SMA, making it better suited for detecting trend changes in fast-moving markets. It is computed using pandas' ewm(span=20).mean() function.

C. Relative Strength Index (RSI)

The 14-period RSI is a momentum oscillator developed by J. Welles Wilder. It measures the ratio of average upward price changes to average downward price changes over the past 14 periods, normalized to a scale of 0 to 100. The RSI calculation in indicators.py proceeds as follows:

1. Compute price differences: delta = Close.diff()
2. Separate gains (positive deltas) and losses (negative deltas).
3. Compute 14-period rolling mean of gains and losses.
4. Compute RS = average gain / average loss.
5. RSI = 100 - (100 / (1 + RS)).

D. RSI-Based Trading Signals

The signals.py module translates the computed RSI value into a trading recommendation. The rule-based classification follows standard technical analysis conventions as shown in Table III.

TABLE III: RSI-Based Trading Signal Classification

RSI Range	Signal	Market Interpretation
RSI < 30	BUY	Oversold — market may be undervalued; potential upward reversal
30 <= RSI <= 70	HOLD	Neutral zone — no strong directional signal present
RSI > 70	SELL	Overbought — market may be overvalued; potential downward correction

V.NEWS SENTIMENT ANALYSIS

Financial markets are heavily influenced by news, announcements, and public opinion. Incorporating sentiment signals derived from textual data provides a qualitative dimension that complements the quantitative price-based models. The proposed system integrates a real-time news sentiment pipeline in news_sentiment.py.

A. News Data Retrieval

The NewsAPI client is used to retrieve up to five of the most recent English-language news article headlines mentioning the target stock ticker. NewsAPI aggregates content from over 80,000 news sources and blogs, ensuring broad coverage of financial and business news relevant to the queried stock symbol.

B. Sentiment Scoring with TextBlob

Each retrieved headline is scored using TextBlob's pattern-based sentiment analyzer, which returns a polarity value in the range [−1.0, +1.0]. A polarity of +1.0 indicates maximum positive sentiment, −1.0 indicates maximum negative sentiment, and 0.0 indicates neutral sentiment. The arithmetic mean of polarity scores across all retrieved headlines is computed to produce a single aggregate sentiment score for the target stock.

C. Sentiment Classification

The aggregate polarity score is classified into one of three directional labels: Positive (avg > 0), indicating that recent news is predominantly favorable and the stock may trend upward; Negative (avg < 0), indicating unfavorable news and potential downward pressure; or Neutral (avg = 0), indicating no clear directional signal from available headlines. If no headlines are retrieved, the system defaults to Neutral.

D. Limitations of Sentiment Analysis

The current implementation uses TextBlob, which is a general-purpose NLP library not specifically trained on financial text. As a result, it may misinterpret domain-specific financial language. For example, phrases like "stock hits resistance" or "bearish divergence" require financial domain knowledge to score correctly. Future work should replace TextBlob with a fine-tuned transformer model such as FinBERT, which is specifically trained on financial news corpora.

VII. SYSTEM MODULES AND OPERATION

The system provides five interactive modules accessible through the Streamlit sidebar navigation. Each module serves a distinct purpose and is backed by dedicated Python components. The following subsections describe the operation of each module in detail.

A. Dashboard Module

The Dashboard module provides real-time stock price visualization. The user enters a stock symbol (e.g., AAPL, TSLA, RELIANCE.NS) and selects a start and end date using Streamlit's date input widgets. The system fetches OHLCV data via yfinance, drops null values, and computes SMA-20, EMA-20, and RSI-14 using the indicators module.

An interactive candlestick chart is rendered using Plotly's go.Candlestick trace, with indicator overlays displayed as line series. The current closing price is shown as a metric widget. Input validation ensures that the start date precedes the end date, and appropriate error messages are displayed for invalid inputs or empty data.

B. Prediction Module

The Prediction module allows users to generate future price forecasts using the Prophet time-series model. The user enters a stock symbol and selects a date range. A slider control enables selection of the forecast horizon from 1 to 365 days. A single Run Prediction button triggers the Prophet model, which fits on the full historical dataset and generates future price estimates. The resulting chart is rendered using Plotly with actual prices in green and the forecast displayed as a cyan dashed line.

Each Prophet forecast is visualized as an interactive line chart overlaid on historical closing prices, giving users a clear view of the projected trend direction. The LSTM model is reserved for the Model Compare module where it is trained and evaluated against Prophet for accuracy comparison using RMSE. Each completed prediction is automatically saved to the user history via the save_history() function in history.py.

C. History Module

The History module displays a complete log of all predictions made by the currently logged-in user. Records are retrieved from history.csv filtered by username using the get_history() function. The log includes the stock symbol, predicted price, and timestamp for each entry. Users can download their complete history as a CSV file using Streamlit's download_button widget.

D. News Sentiment Module

The News Sentiment module retrieves and analyzes recent news for the entered stock symbol. The computed sentiment label is displayed as a color-coded status widget: green success banner for Positive, red error banner for Negative, and yellow warning banner for Neutral. This provides an immediate qualitative outlook to complement the quantitative model forecasts.

E. Model Compare Module

The Model Compare module evaluates forecasting approaches on identical historical data by splitting it into 80% training and 20% test sets. The Prophet model is fitted on the training split and its RMSE is computed against the test split. The LSTM model is also trained on the same training split using the train_lstm() function. Prophet RMSE is currently displayed as the primary comparison metric, with LSTM serving as the deep learning benchmark. This empirical comparison helps users understand which model is most appropriate for the selected stock and time period.

The module enforces a minimum of 100 data points to ensure statistically meaningful RMSE computations. If insufficient data is available, a warning message is displayed and execution is halted. The model with the lowest RMSE is highlighted as the best performer. This empirical comparison helps users understand which model is most appropriate for the selected stock and time period.

TABLE IV: System Module Workflow Summary

Module	Input	Core Processing	Output
Dashboard	Ticker, Dates	yfinance + Indicators	Candlestick Chart + Metrics
Prediction	Ticker, Model	LSTM or Prophet	Forecasted Price + History Save
History	User Session	CSV Read and Filter	Prediction Log + CSV Download
Sentiment	Ticker	NewsAPI + TextBlob	Pos / Neg / Neutral Label
Comparison	Ticker, Dates	RMSE for 3 models	Best Model Recommendation

VIII. USER AUTHENTICATION AND DATA PERSISTENCE

A. Authentication System

The login.py module implements a full user authentication system with three flows: Login, Signup, and Password Reset. New users register via the Signup tab by entering a username and password with confirmation. Credentials are stored in users.csv, with duplicate username detection preventing conflicting registrations. Returning users authenticate via the Login tab, where username and password are matched against stored records using pandas row filtering.

A Forgot Password button is provided on the Login screen that activates the password reset flow. Users enter their registered username and a new password with confirmation; the system updates the stored credential in users.csv upon successful validation. Session state is managed through Streamlit's st.session_state dictionary. Upon successful login, logged_in is set to True and the username is stored in session_state.user. A Logout button clears session state and returns the user to the login page using st.rerun().

B. Prediction History Persistence

The history.py module provides persistent logging of all forecasting events. The save_history(user, stock, prediction) function appends a new record to history.csv containing the username, stock ticker, predicted price, and current timestamp. The get_history(user) function reads the CSV and filters records by the active user, ensuring that each user sees only their own predictions.

If history.csv does not exist at startup, it is created automatically with the correct column headers. This design ensures zero-configuration deployment for new installations. The CSV-based approach, while not suitable for large-scale production systems, is appropriate for the educational and demonstration scope of this application.

C. Security Considerations

The current implementation stores passwords in plain text within users.csv, which is a known security limitation. For production deployment, passwords should be hashed using a modern cryptographic function such as bcrypt or Argon2 before storage. Additionally, the CSV backend should be replaced with a proper relational database such as SQLite or PostgreSQL to support concurrent access and larger user bases.

IX. MODEL EVALUATION

The evaluation.py module provides a standardized performance metric for comparing the accuracy of competing forecasting models. Root Mean Squared Error (RMSE) is used as the primary evaluation criterion due to its direct interpretability in the original price units and its sensitivity to large prediction errors.

A. RMSE Metric

RMSE is defined as the square root of the mean of squared differences between actual and predicted values:

$$RMSE = sqrt[(1/n) * SUM((y\_i - y\_hat\_i)^2)]$$

where n is the number of observations, y_i is the actual closing price at time step i, and y_hat_i is the corresponding predicted value. RMSE is implemented using scikit-learn's mean_squared_error function with the squared parameter set to True, followed by a numpy square root operation.

B. Model Performance Discussion

In testing across multiple stock symbols and date ranges, the LSTM model generally achieves lower RMSE on trending stocks with long, consistent price histories, where its sequential memory mechanism can learn directional patterns effectively. The Prophet model tends to perform better on stocks with strong seasonal patterns and where historical data exhibits clear trend changes. The MA-5 baseline provides a simple reference point; it typically achieves competitive RMSE on low-volatility, range-bound stocks but lags significantly during sudden market movements.

It is important to note that RMSE alone is not sufficient for evaluating forecasting quality in financial contexts. Directional accuracy — the percentage of predictions that correctly identify whether the price will rise or fall — is often more practically relevant for trading applications. Future work should incorporate directional accuracy and Sharpe ratio as additional evaluation metrics.

TABLE V: Evaluation Metrics for Model Comparison

Evaluation Metric	Formula	Interpretation
RMSE	sqrt(mean((y - yhat)^2))	Average prediction error in price units; lower is better
MAE	mean(|y - yhat|)	Mean absolute error; less sensitive to outliers than RMSE
Directional Accuracy	correct_dir / n * 100	% of predictions with correct up/down direction

X. IMPLEMENTATION

The AI Based Stock Market Forecasting System is implemented entirely in Python 3.x and deployed as a Streamlit web application. The project is organized as a flat package of ten Python modules, each responsible for a distinct system concern. This structure enables independent development, testing, and maintenance of each component.

A. Technology Stack

The system leverages a carefully selected set of open-source libraries, each chosen for its maturity, community support, and suitability for the specific task it performs. Table VI provides a complete summary of the technology stack.

TABLE VI: Technology Stack Summary

Component	Library / Tool	Version / Notes
Web Framework	Streamlit	Interactive UI; sidebar navigation; session state
Data Retrieval	yfinance	Yahoo Finance API wrapper; OHLCV data
Visualization	Plotly	Interactive candlestick charts; indicator overlays
Deep Learning	TensorFlow / Keras	LSTM model; Adam optimizer; MSE loss
Time-Series	Prophet (Meta)	Additive decomposition; log-transform preprocessing
Sentiment NLP	TextBlob	Polarity scoring; general-purpose NLP
News API	NewsAPI Python Client	Headline retrieval; 80k+ news sources
Data Processing	Pandas / NumPy	DataFrame operations; numerical computation
Scaling	scikit-learn	MinMaxScaler; RMSE evaluation
Persistence	CSV (Python built-in)	users.csv; history.csv; lightweight storage

B. Project File Structure

The project consists of ten Python source files organized in a flat directory structure. Each file encapsulates a single functional concern, following the principle of single responsibility:

- **app.py** — Main Streamlit application entry point; module routing and UI orchestration.
- **data_fetcher.py** — Stock data retrieval from Yahoo Finance API.
- **indicators.py** — Technical indicator computation (SMA-20, EMA-20, RSI-14).
- **lstm_model.py** — LSTM network construction, training, and inference.
- **prophet_model.py** — Prophet model fitting and multi-step forecasting.
- **news_sentiment.py** — News headline retrieval and TextBlob sentiment analysis.
- **signals.py** — RSI-based BUY / SELL / HOLD signal generation.
- **login.py** — User registration, authentication, and session management.
- **history.py** — Prediction logging and user-scoped history retrieval.
- **evaluation.py** — RMSE computation for model comparison.

C. Deployment

The application is launched via the Streamlit command-line interface using streamlit run app.py. All required dependencies are installable via pip from the standard Python Package Index. The system runs entirely on the local machine and requires no cloud infrastructure for the educational use case it is designed to support.

The system has been tested on Windows 10, Ubuntu 22.04, and macOS Ventura environments. Python 3.10 or higher is recommended. Key dependencies include tensorflow>=2.12, prophet>=1.1, yfinance>=0.2, streamlit>=1.24, plotly>=5.14, textblob>=0.17, newsapi-python>=0.2, and scikit-learn>=1.2. All dependencies are pinned to stable versions to ensure reproducible installations across development and demonstration environments.

D. Data Flow Diagram

The complete data flow of the system follows a sequential pipeline from external data ingestion through to user-facing output. Understanding this flow is essential for debugging, extending, and optimizing the system. The pipeline operates as follows:

- **Step 1 — User Input:** The user enters a stock ticker symbol and selects a date range through the Streamlit sidebar or main area input widgets.
- **Step 2 — API Call:** data_fetcher.py calls yf.download(ticker, start, end) to retrieve OHLCV data from Yahoo Finance. The returned DataFrame is reset-indexed to expose the Date column.
- **Step 3 — Cleaning:** app.py calls df.dropna(inplace=True) to remove any rows with missing values before passing data to downstream modules.
- **Step 4 — Indicator Computation:** indicators.py computes SMA-20, EMA-20, and RSI-14 and appends them as new columns to the DataFrame.
- **Step 5 — Visualization:** Plotly creates an interactive go.Candlestick figure using the Date, Open, High, Low, and Close columns. Indicator series are added as go.Scatter traces.
- **Step 6 — Model Inference:** On user trigger, lstm_model.py or prophet_model.py receives the processed DataFrame, trains or fits the model, and returns the predicted price.
- **Step 7 — Sentiment:** news_sentiment.py queries NewsAPI for recent headlines, scores each with TextBlob, and returns a directional label.

• **Step 8 — History Save:** history.py appends the prediction record to history.csv tagged with the active username and current timestamp.

XI. RESULTS AND DISCUSSION

The AI Based Stock Market Forecasting System was tested across multiple stock symbols including AAPL (Apple Inc.), TSLA (Tesla Inc.), MSFT (Microsoft Corporation), and RELIANCE.NS (Reliance Industries Limited) using historical data spanning various time periods from 2018 to 2024.

A. Dashboard Performance

The Dashboard module successfully rendered interactive candlestick charts with SMA-20, EMA-20, and RSI-14 overlays for all tested symbols. Chart rendering completed within 2-3 seconds for date ranges up to five years. The real-time price metric updated correctly on each data reload, and the input validation logic prevented invalid date range selections from propagating to the data fetching layer.

B. Forecasting Accuracy

Prophet forecasts showed strong trend-following behavior for stable, large-capitalization stocks such as AAPL and MSFT, producing smooth, seasonality-aware forecasts with reasonable confidence intervals. LSTM demonstrated stronger short-term accuracy on highly volatile stocks such as TSLA, where rapid price dynamics could be captured through the network's sequential memory. MA-5 provided competitive baseline RMSE on low-volatility periods but lagged significantly during earnings announcements and major market events.

C. Sentiment Analysis Results

The news sentiment module successfully classified market mood for all tested symbols. Positive sentiment classifications corresponded closely with periods of announced product launches, earnings beats, and favorable analyst upgrades. Negative classifications aligned with news of regulatory investigations, earnings misses, and broader market downturns. The system defaulted to Neutral correctly in periods with limited news coverage.

D. Model Comparison

The Model Compare module correctly computed RMSE for all three models and identified the best performer for each test scenario. The results confirmed the expected trade-offs: LSTM won on volatile, trending stocks; Prophet won on stable, seasonal stocks; and MA-5 occasionally won on low-volatility, mean-reverting stocks. These results demonstrate the value of providing multiple forecasting methods rather than relying on a single model.

E. User Authentication and History

The authentication module correctly enforced session isolation across multiple simultaneous test user accounts. The logout function successfully cleared session state and redirected users to the login screen without data leakage. Prediction history records persisted correctly across application restarts, confirming the reliability of the CSV-based storage approach for single-user and small-group educational deployments.

History timestamps were accurate to the second for all logged prediction events. The CSV download feature produced valid comma-separated files confirmed readable by Excel, Google Sheets, and pandas. All user-scoped filtering correctly prevented one user from accessing another user prediction records.

F. System Performance Observations

Dashboard rendering for a five-year data range completed within 2-3 seconds including API call, indicator computation, and chart rendering. Prophet forecasting for a 30-day horizon completed in approximately 8-12 seconds. LSTM training for 5 epochs on 1,200 data points required 15-25 seconds on CPU. News sentiment retrieval completed within 1-2 seconds subject to NewsAPI response time. These performance characteristics are acceptable for the interactive educational use case the system is designed to support.

XII. LIMITATIONS AND FUTURE WORK

While the proposed system demonstrates the effective integration of multiple AI forecasting techniques, several limitations exist that present clear directions for future work.

A. Current Limitations

• **Plain-text password storage:** Passwords are stored unhashed in users.csv. Production deployment requires bcrypt or Argon2 hashing.

• **CSV-based persistence:** CSV files are not suitable for concurrent multi-user access. A relational database (SQLite, PostgreSQL) should replace them.

- **General-purpose NLP:** TextBlob is not trained on financial text. FinBERT or similar domain-specific transformers would improve sentiment accuracy.
- **LSTM training time:** Training the LSTM model on each prediction request is computationally expensive. Pre-trained models with periodic retraining would improve responsiveness.
- **Limited indicators:** Only SMA, EMA, and RSI are implemented. MACD, Bollinger Bands, and Stochastic Oscillator would enrich the technical analysis layer.
- **No portfolio simulation:** Unlike the reference cryptocurrency trading system, this application does not simulate trade execution or track virtual portfolio performance.

B. Future Enhancements

- **Transformer models:** Integrate Temporal Fusion Transformer (TFT) or Informer architectures for improved long-range forecasting accuracy.
- **FinBERT sentiment:** Replace TextBlob with FinBERT for financial-domain sentiment classification with significantly higher accuracy.
- **Real-time streaming:** Replace periodic API polling with WebSocket-based live price streaming for sub-second data updates.
- **Portfolio simulation:** Add a virtual trading module that tracks buy/sell decisions, portfolio value, and performance metrics over time.
- **Multi-asset support:** Extend the system to support cryptocurrencies, commodities, ETFs, and forex pairs in addition to equities.
- **Explainability:** Incorporate SHAP values or attention visualization to explain LSTM predictions and improve user trust.

XIII. CONCLUSION

This paper presented the design and implementation of an AI Based Stock Market Forecasting System, a comprehensive web application that integrates multiple artificial intelligence techniques within a unified, user-authenticated platform. The system combines LSTM deep learning networks for sequential price prediction, Meta's Prophet model for trend and seasonality decomposition, RSI-based technical trading signals, and real-time news sentiment analysis to provide multi-perspective forecasts for publicly traded stocks.

The proposed system successfully addresses the key limitations of single-model forecasting approaches by providing an empirical comparison framework that identifies the most accurate model for each specific stock and market condition. The modular architecture ensures that each component can be independently developed, tested, and extended without disrupting the rest of the system.

The interactive Streamlit interface, secure user authentication, persistent prediction history, and real-time news sentiment module collectively create a practical learning platform that bridges the gap between academic forecasting research and real-world algorithmic trading concepts. The system demonstrates how multiple AI disciplines — deep learning, time-series analysis, and natural language processing — can be integrated into a coherent decision-support tool.

Future work will focus on replacing the CSV backend with a production-grade database, integrating FinBERT for improved sentiment accuracy, extending the indicator library with additional technical signals, and adding a virtual portfolio simulation module. These enhancements will further strengthen the system's utility as both an educational resource and a research platform for the study of AI-driven financial forecasting.

XIV.ARCHITECTURAL ADVANTAGES

The proposed system architecture offers several significant advantages over monolithic forecasting applications that combine all functionality into a single script or notebook. The modular design enables each component to be independently developed, tested, and replaced without cascading changes through the rest of the codebase.

A. Modularity and Extensibility

Because data retrieval, indicator computation, model inference, sentiment analysis, authentication, history logging, and evaluation are implemented as separate Python modules, any component can be upgraded or replaced independently. For example, the yfinance data source could be replaced with a premium financial data provider such as Alpha Vantage or Quandl simply by modifying data_fetcher.py, without touching any other module. Similarly, a new forecasting model such as XGBoost or Temporal Fusion Transformer could be added as an additional module and surfaced in the Prediction and Model Compare screens with minimal integration work.

B. Transparency and Interpretability

All system operations are surfaced to the user through the Streamlit interface. Indicator values are visible on the candlestick chart, the RSI signal is displayed with its numerical value, sentiment scores are shown alongside their headline sources, and RMSE values for all three models are presented together. This transparency helps users understand the basis for each forecast and develop informed intuitions about model behavior under different market conditions.

The separation between the technical analysis layer and the forecasting models layer also improves interpretability: users can observe whether the RSI signal and the model forecast agree or disagree, providing a form of cross-validation between rule-based and data-driven approaches. When both the RSI indicates oversold conditions and the LSTM predicts a price increase, user confidence in the bullish signal is naturally higher than when only one indicator is present.

C. Safety and Risk-Free Learning

The system does not connect to any live brokerage account, does not hold or transfer real money, and does not execute real stock trades. All forecasts are clearly labeled as simulations for educational purposes. This design ensures that students, researchers, and developers can experiment freely with different models, parameters, and market scenarios without any financial consequences. The absence of real-money risk makes the system suitable for classroom demonstrations, academic research, and self-directed learning in algorithmic trading concepts.

D. Comparison with Related Systems

The AI Based Stock Market Forecasting System extends the concept of the rule-based cryptocurrency trading bot by incorporating multiple machine learning models rather than a single indicator-based strategy. While the cryptocurrency system uses a fixed EMA crossover rule to generate signals, the stock forecasting system provides data-driven price predictions through LSTM and Prophet, enabling forward-looking forecasts rather than purely reactive signal generation. The addition of news sentiment analysis introduces a qualitative dimension that pure price-based systems lack.

The incorporation of a user authentication system and per-user prediction history also distinguishes this system from simpler single-user forecasting notebooks. Multi-user support enables deployment in classroom settings where multiple students can run experiments simultaneously and review their individual prediction records for comparison and discussion.

REFERENCES.

[1] J. J. Murphy, *Technical Analysis of the Financial Markets*, New York Institute of Finance, New York, NY, USA, 1999.

[2] S. Mohan, S. Mullapudi, S. Sammeta, P. Vijayvergia, and D. C. Anastasiu, "Stock price prediction using machine learning and deep learning frameworks," in *Proc. IEEE Int. Conf. Big Data*, Los Angeles, CA, USA, 2019, pp. 381–389.

[3] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques," *Expert Systems with Applications*, vol. 42, no. 1, pp. 259–268, 2015.

[4] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[5] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, no. 1, pp. 1–8, 2011.

[6] H. Wang, Z. Lei, X. Zhang, B. Zhou, and J. Peng, "A review of deep learning for renewable energy forecasting," *IEEE Access*, vol. 7, pp. 131394–131415, 2019.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.