Title: Time Series-Based Stock Market Forecasting Using Machine Learning Models

Submitted by: Swayam Tripathi
Program: B.Tech CSE (AI & DS)
Project Duration: 12 Weeks
Submission Date: 27 May 2025

Abstract:
This project explores the use of time series forecasting techniques for predicting stock market movements. Leveraging statistical and deep learning models including ARIMA, SARIMA, Prophet, and LSTM, we aim to extract meaningful patterns from historical data. The project provides comparative insights into these methods, emphasizing their applicability and accuracy for financial data forecasting. It serves as a practical exposure to time series modeling, data preprocessing, and trend analysis.

Objectives:

- To comprehend the core principles of time series data such as trend, seasonality, and irregular components.
- To collect and preprocess real-time historical stock market data.
- To design and implement ARIMA, SARIMA, Prophet, and LSTM models.
- To analyze and compare the forecasting performance of different models.
- To present results through visual reports and dashboards.

Project Synopsis:
In this project, historical stock data is analyzed to detect patterns and make short- and long-term predictions using time series models. It bridges theoretical understanding with hands-on application, offering experience in financial analytics, statistical modeling, and neural network-based forecasting.

Tools and Technologies:

- **Languages:** Python
- **Libraries:** Pandas, NumPy, Matplotlib, Seaborn, Plotly, Scikit-learn, Statsmodels, Facebook Prophet, TensorFlow/Keras
- **Optional Deployment Tools:** Streamlit, Flask
- **Data Sources:** Yahoo Finance, Alpha Vantage, Kaggle

Dataset Description:
The datasets comprise daily stock market indicators such as Opening price, Closing price, Highest and Lowest prices of the day, and Volume traded. These were obtained from open financial APIs and repository called YFinance.

Work Plan and Methodology:

- **Week 1-2:** Study fundamentals of time series analysis; gather datasets.
- **Week 3-4:** Preprocess data and perform exploratory visualization.
- **Week 5-6:** Implement ARIMA, SARIMA, LSTM and Prophet models.
- **Week 7-8:** Develop and train LSTM model using Keras.
- **Week 9:** Evaluate all models on common metrics and fine-tune hyperparameters.
- **Week 10-12:** Finalize reporting, create visual dashboards, and optionally deploy a web app for predictions.

Fundamentals of Time Series Analysis:

Time Series Analysis is the process of analyzing data points collected or recorded at specific time intervals to identify patterns, trends, and future forecasts. These datasets typically include components such as *trend* (the long-term upward or downward movement in the data), *seasonality* (regular and repeating patterns at fixed intervals like daily, weekly, or yearly), *cyclic behavior* (non-fixed periodic fluctuations often linked to economic or natural cycles), and *noise* (random or irregular variations that do not follow a specific pattern). A key concept in time series is *stationarity*, where the statistical properties like mean and variance remain constant over time—an important requirement for many forecasting models such as ARIMA. Analysts often use *lag values* (previous time points) and study *autocorrelation* (correlation of current values with past ones) to improve prediction accuracy. Common models include AR (AutoRegressive), MA (Moving Average), ARIMA (combination of AR and MA with differencing for non-stationary data), SARIMA (seasonal extension of ARIMA), Facebook's Prophet model (for handling complex trends and seasonality), and LSTM (a deep learning model suited for capturing long-term dependencies). To assess the accuracy of these models, metrics such as MAE (Mean Absolute Error), RMSE (Root Mean Square Error), and MAPE (Mean Absolute Percentage Error) are used. Visualization techniques like line plots, autocorrelation plots, and seasonal decomposition charts are essential tools in identifying patterns and structures in time series data. Applications of time series analysis span across various domains including stock market prediction, weather forecasting, sales and demand forecasting, and economic modeling.

Terminologies used in Forecasting project:

- **High**: The highest value recorded in a given time period (e.g., highest stock price in a day).
- **Low**: The lowest value recorded in a given time period.
- **Seasonality**: Regular, repeating patterns or cycles in data occurring at fixed intervals (e.g., daily, weekly, yearly).
- **Trend**: The long-term upward or downward movement in the data over time.
- **Noise**: Random variations or fluctuations in the data that do not follow a pattern.
- **Stationarity**: When the statistical properties of the series (mean, variance) remain constant over time.
- **Lag**: Previous time steps used as predictors for current values.
- **Autocorrelation**: Correlation of the time series with its own past values.

Dataset Description

- **Source**: Public financial datasets (e.g., Yahoo Finance, Kaggle, or other financial APIs).
- **Type**: Time series data representing stock market prices.
- **Features/Columns**:
    - **Date**: Time index (daily, weekly, or monthly intervals).
    - **Open**: Price at which the stock opens.
    - **High**: Highest price during the time interval.
    - **Low**: Lowest price during the time interval.
    - **Close**: Price at which the stock closes.
    - **Volume**: Number of shares traded during the interval.
- **Target Variable**: Usually the **'Close' price**, which is forecasted using different models.
- **Nature of Data**:
    - Sequential, time-indexed, and numeric.
    - Includes trend, seasonality, and noise — ideal for time series modeling.
- **Preprocessing Applied**:
    - Handled missing values.
    - Converted dates to datetime format.
    - Normalized or transformed data (where required).
    - Created lag features and rolling statistics.

Python Libraries Used

- **NumPy**
  Used for numerical operations, array manipulation, and handling large datasets efficiently.
- **Pandas**
  Ideal for data loading, cleaning, transformation, and time-indexed manipulation using DataFrames.

- **Matplotlib / Seaborn**
  For data visualization and plotting line graphs, trends, autocorrelations, and seasonal components.
- **Statsmodels**
  Used to build and evaluate statistical models like ARIMA and SARIMA.
- **pmdarima**
  For auto-tuning ARIMA/SARIMA parameters using the `auto_arima()` function.
- **Prophet (by Facebook)**
  A simple, intuitive library for trend and seasonality-based forecasting.
- **scikit-learn**
  Used for scaling, splitting the data, and evaluating models using metrics like MAE, RMSE.
- **TensorFlow / Keras**
  Deep learning frameworks used to build and train LSTM models for sequence prediction.

Preprocessing of Data in Time Series Analysis

- **Handling Missing Values**:
  Fill gaps caused by missing records using techniques like forward fill, backward fill, or interpolation.
- **Timestamp Formatting**:
  Ensure the date and time column is correctly formatted and sorted to maintain proper sequence.
- **Outlier Detection and Removal**:
  Identify and remove abnormal values that can distort model predictions.
- **Resampling**:
  Adjust the frequency of the data (e.g., hourly to daily or monthly) to suit the analysis and model requirements.
- **Transformation**:
  Apply transformations (like log or square root) to stabilize variance or reduce skewness in the data.
- **Normalization/Standardization**:
  Scale the data to bring different features to a common range, especially useful in machine learning models.
- **Stationarity Check and Differencing**:
  Use differencing to remove trends and seasonality if the data is non-stationary, as required by models like ARIMA.
- **Creating Lag Features**:
  Generate lagged versions of the time series to help models learn from past values.
- **Rolling Statistics (Optional)**:
  Calculate moving averages or rolling means to smooth the data and understand short-term trends.

Project Architecture – Time Series Stock Forecasting :

1. Data Collection
   └──> Yahoo Finance API / Kaggle Dataset

2. Data Preprocessing
   └──> Fill missing values
   └──> Set datetime index and frequency
   └──> Normalize or scale (if needed)

3. Exploratory Data Analysis (EDA)
   └──> Line plots (trend, seasonality)
   └──> Statistical summaries

4. Model Implementation
   ├──> ARIMA
   ├──> SARIMA
   ├──> Prophet
   └──> LSTM (Deep Learning)

5. Model Evaluation
   └──> Use metrics like MAE, RMSE, MAPE
   └──> Compare model predictions vs actuals

6. Visualization
   └──> Forecast plots with confidence intervals
   └──> Training vs test comparison charts

7. Deployment (Optional)
   └──> Streamlit or Flask Web App

8. Final Report & Insights
   └──> Interpretation of results
   └──> GitHub Repo, PDF Report, Video Demo

Four models used in Time Series Forecasting project — ARIMA, SARIMA, Prophet, and LSTM:

1. ARIMA (AutoRegressive Integrated Moving Average)

- **Purpose**: Suitable for univariate time series data that shows trends but no strong seasonality.
- **Components**:
  - **AR (AutoRegressive)**: Uses past values to predict the current one.
  - **I (Integrated)**: Differencing the data to make it stationary.
  - **MA (Moving Average)**: Uses past forecast errors in a regression-like model.
- **Strengths**:
  - Well-suited for linear and trend-based data.
  - Performs well on stationary data.
- **Limitations**:
  - Doesn't handle seasonality by default.
  - Requires careful parameter tuning and stationarity checks.

2. SARIMA (Seasonal ARIMA)

- **Purpose**: Extends ARIMA to handle **seasonal** data patterns (e.g., monthly or quarterly cycles).
- **Components**:
  - Adds **seasonal parameters (P, D, Q, s)** to the original ARIMA model.
- **Strengths**:
  - Excellent for datasets with strong seasonal trends.
  - Incorporates both non-seasonal and seasonal behaviors.
- **Limitations**:
  - More complex due to additional seasonal parameters.
  - Can be slow to tune for large datasets.

3. Prophet (by Facebook)

- **Purpose**: Designed for **automatic and flexible forecasting** of time series with trends and seasonality.
- **Components**:
  - Models trend, seasonality (weekly, yearly), and holidays as separate components.
- **Strengths**:
  - Very user-friendly, requires minimal tuning.
  - Handles missing data, outliers, and irregular time series efficiently.
  - Good for business and economic forecasting.
- **Limitations**:
  - Can underperform on data with highly complex or non-standard patterns.
  - Less accurate than deep learning models on very complex datasets.

4. LSTM (Long Short-Term Memory - a type of RNN)

- **Purpose**: A deep learning model capable of learning complex, long-term dependencies in time series data.
- **Components**:
  - Uses memory cells to retain information across long time sequences.
- **Strengths**:
  - Works well with multivariate and non-linear data.
  - Can capture long-term trends and relationships that traditional models may miss.
- **Limitations**:
  - Requires large datasets and more computational power.
  - Longer training time and more complex to implement compared to statistical models.

Evaluation Metrics in Time Series Forecasting:

1. MAE (Mean Absolute Error)

- Definition: Average of the absolute differences between actual and predicted values.
- Formula: MAE = (1/n) * Σ |actual - predicted|
- Interpretation: Tells how much, on average, the predictions differ from actual values.
- Pros: Simple and less sensitive to outliers than RMSE.

2. RMSE (Root Mean Square Error)

- Definition: Square root of the average of squared differences between predicted and actual values.
- Formula: RMSE = sqrt((1/n) * Σ (actual - predicted)^2)
- Interpretation: Penalizes large errors more heavily than MAE.
- Pros: Better when large errors are undesirable.

3. MAPE (Mean Absolute Percentage Error)

- Definition: Average of the absolute percentage errors between actual and predicted values.
- Formula: MAPE = (100/n) * Σ |(actual - predicted) / actual|
- Interpretation: Expresses forecast error as a percentage.
- Pros: Easy to interpret and useful for comparing across datasets.
- Cons: Can give misleading results when actual values are close to zero.

4. R2 Score (Coefficient of Determination)

- Definition: Indicates how well the model's predictions approximate the actual data.
- Interpretation:
  - R2 = 1: perfect prediction
  - R2 = 0: model does no better than the average
  - R2 < 0: model performs worse than the mean
- Pros: Good for understanding the overall fit of the model.

Use Case:

- Use MAE for simplicity.
- Use RMSE when large errors matter more.
- Use MAPE for percentage-based evaluation.
- Use R2 to understand the goodness of fit.

In our project,we have used **RMSE (Root Mean Square Error) .**

Tuning Hyperparameters in Time Series Forecasting:

Hyperparameter tuning is the process of selecting the best combination of parameters for a model to achieve optimal performance. In time series forecasting, each model has its own set of parameters that need to be fine-tuned to minimize forecasting errors.

Hyperparameter tuning means finding the best settings for a model to improve its prediction accuracy. Each model has different parameters that affect how well it learns from data.

1. ARIMA / SARIMA

- Key parameters:
  - p: Number of lag observations (AR part)
  - d: Number of times data is differenced to become stationary
  - q: Size of moving average window (MA part)
  - For SARIMA, seasonal parameters: P, D, Q, and s (season length)
- How to tune:
  - Use tools like auto_arima() to automatically select the best parameters
  - Look at ACF and PACF plots to guess p and q
  - Use information criteria like AIC or BIC to choose the best model

2. Prophet

- Key parameters:
  - changepoint_prior_scale: Controls how much the trend can change
  - seasonality_mode: Can be additive or multiplicative
  - seasonality_prior_scale: Controls how strong seasonality effects are
  - holidays_prior_scale: Controls effect of holidays on forecast
- How to tune:
  - Adjust parameters manually or use grid search
  - Check model fit by visualizing predictions and errors

3. LSTM (Deep Learning)

- Key parameters:
    - Number of LSTM layers and units (neurons)
    - Sequence length (number of time steps used as input)
    - Batch size and epochs (training rounds)
    - Learning rate and optimizer choice
    - Dropout rate to prevent overfitting
- How to tune:
    - Use grid search or random search with Keras Tuner
    - Monitor training and validation loss to avoid overfitting
    - Use early stopping to stop training when performance stops improving

General Tips

- Always keep a separate validation dataset to test performance
- Start with default settings and tweak step-by-step
- Use evaluation metrics like MAE, RMSE, or MAPE to compare models
- Time-based cross-validation can give better estimates on time series data

General Tips for Tuning

- Always split data into training and validation sets to evaluate hyperparameter effects.
- Use cross-validation (time-based) where possible.
- Start with default values and incrementally adjust based on results.
- Track metrics like MAE, RMSE, and MAPE to compare model performance.

Reporting Results

- Summarize the forecasting model's performance using key evaluation metrics like MAE, RMSE, MAPE, and R^2.
- Present findings clearly with tables and charts comparing actual vs predicted values.
- Highlight insights about trends, seasonality, and anomalies found in the data.
- Discuss limitations and possible improvements for future work.
- Include visualizations such as line plots, residual plots, and error distributions to support your analysis.

Creating Visual Dashboards

- Use tools like **Tableau**, **Power BI**, or Python libraries (matplotlib and Seaborn) for interactive dashboards.
- Display real-time or updated forecasts alongside historical data for easy comparison.
- Incorporate charts like time series plots, bar graphs for error metrics, and seasonal decomposition visuals.
- Add filters to explore different time periods, metrics, or stock symbols.

- Make dashboards user-friendly to help stakeholders understand and trust the predictions.

Deploying a Web App for Predictions (Optional)

- Build a simple web app using frameworks like **Flask**, **Django**, **Streamlit**, or **Dash** to serve your forecasting model.
- Allow users to input parameters or select dates to get on-demand predictions.
- Integrate visualizations for predicted vs actual values within the app.
- Host the app on cloud platforms like **Heroku**, **AWS**, or **Google Cloud** for easy access.
- Ensure the app is responsive and includes error handling for a smooth user experience.

Deliverables:

- Cleaned datasets and preprocessing notebooks.
- Complete model implementations and evaluation metrics.
- Comparative analysis plots and interpretative visualizations.
- Documented GitHub repository.

Results and Insights:

Each model was evaluated on standard performance metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). The LSTM model showcased superior pattern recognition ability, especially for complex non-linear data. Prophet provided intuitive trend forecasts with minimal tuning.

Skills and Knowledge Gained:

- In-depth understanding of statistical and machine learning models for time series.
- Enhanced proficiency in Python programming and data visualization.
- Practical exposure to real-world financial data analytics.
- Experience in communicating insights through dashboards and reports.

Conclusion:
The comparative approach in this project highlights that while statistical models are efficient for simpler data, deep learning techniques like LSTM excel in capturing intricate dependencies. This project underscores the relevance of data science in financial markets and reinforces the importance of choosing appropriate models based on data characteristics and forecasting goals.

References:

- Forecasting: Principles and Practice by Hyndman and Athanasopoulos
- Official documentation of libraries: Scikit-learn, Statsmodels, Prophet, Keras
- Data sources: Yahoo Finance, Kaggle
- For Code/Repository and Working refer to the **(.ipynb)** file.

THANK YOU