1 Introduction

The accurate prediction of stock market movements remains one of the most challenging problems in financial analysis. Modern financial markets are characterized by complex dynamics, where stock prices are influenced by numerous factors including market sentiment, economic indicators, and global events. This project addresses these challenges through the implementation of advanced machine learning techniques, specifically Long Short-Term Memory (LSTM) networks, to analyze historical stock data and forecast future price movements. By leveraging deep learning capabilities, we aim to identify patterns and relationships in temporal data that traditional analytical methods might overlook.

Motivation

The development of reliable stock price prediction models has significant implications for investment strategy and risk management. Our research is motivated by the growing need for sophisticated analytical tools that can process vast amounts of financial data and generate actionable insights. Through the application of LSTM networks and comprehensive data analysis, this project seeks to enhance the accuracy of stock price forecasting while providing meaningful confidence intervals for predictions.

Existing Methods

Traditional stock price prediction methods include fundamental and technical analysis. Fundamental analysis evaluates a company's financial health and market conditions, while technical analysis uses historical price data and indicators like Moving Averages and RSI to identify trends. Statistical models such as ARIMA and linear regression have also been employed, but they often struggle with the non-linear and dynamic nature of stock data.Machine learning approaches, including SVM and Random Forests, offer improvements but still face challenges in handling sequential data and long-term dependencies. LSTM networks, a type of Recurrent Neural Network (RNN), excel in this domain [1]. LSTMs are designed to capture long-term dependencies in time series data through their unique architecture, which includes memory cells and gating mechanisms. These features allow LSTMs to:

- Retain relevant information over long sequences
- Learn complex, non-linear patterns in stock data
- Adapt to changing market dynamics

In our project, LSTMs are employed to process normalized stock prices, leveraging their ability to model temporal dependencies and improve prediction accuracy. This approach enables the model to effectively learn from historical data and provide reliable forecasts, addressing the limitations of traditional methods.

Dataset Description

For this project, we rely on historical stock price data sourced from Yahoo Finance [2], a widely respected provider of financial information. Our dataset includes daily records for several prominent companies, focusing primarily on the 'Close' and 'Adjusted Close' prices, which are essential for accurate financial forecasting. The dataset covers an extensive time period, offering a detailed view of market trends and price fluctuations. Key elements of the dataset include:

- Date: The specific trading day for each entry
- Open, High, Low, Close Prices: These daily metrics capture the stock's trading range and provide insights into market behavior
- Adjusted Close Price: This is the closing price adjusted for dividends and stock splits, giving a more accurate picture of a stock's true value over time.
- Volume: The number of shares traded each day, reflecting market activity and investor interest.

In our analysis, we focus on the 'Close' prices, applying a log transformation and normalization to prepare the data for modeling. These steps help stabilize the data and enhance the model's ability to detect patterns. The dataset is divided into training and testing sets, typically with 80% used for training and 20% reserved for testing. This approach ensures that our model learns from a substantial amount of data while being evaluated on new, unseen data to verify its predictive accuracy.

Data preprocessing

The preprocessing pipeline implemented in this project involves several sophisticated transformations to prepare the time-series data for the LSTM model. The initial preprocessing step applies a logarithmic transformation to the closing prices, expressed mathematically as:

$$y_i = \log(x_i)$$

where X_i is the original stock price at time i. This transformation helps address the exponential growth patterns typically observed in stock prices and stabilizes the variance across the time series.
The transformed data undergoes normalization using `MinMaxScaler` with a feature range of `(0,1)`. This scaling operation is crucial for neural network training, as it

ensures all input features are on the same scale and helps prevent gradient explosion during the training process. The normalization is applied to the entire dataset before sequence creation to maintain consistency across all temporal segments.

Sequence generation is performed using a sliding window approach with a lookback period of 60 days. This process creates input sequences X and target values y, where each input sequence contains 60 consecutive days of price data, and y represents the price of the following day. The sequences are structured as three-dimensional arrays with shape `(n_samples, lookback_period, n_features)` to match the LSTM input requirements. This transformation is accomplished through numpy operations, reshaping the data as:

```
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
```

The preprocessed dataset is then partitioned into training and testing sets using an 80-20 split ratio. This division is performed chronologically rather than randomly to preserve the temporal structure of the data. The training set is further subdivided to create a validation set (15% of training data) used during the model training process for hyperparameter tuning and early stopping decisions.

2 Methodology

Our approach integrates two distinct components: the `StockAnalyzer` and `StockPredictor` classes, each serving specific analytical purposes. The `StockAnalyzer` class performs preliminary market analysis, calculating technical indicators such as Simple Moving Averages (SMA) for multiple periods (10, 20, and 50 days) and daily returns. This multi-period analysis helps identify different time-scale trends and market momentum. The core predictive functionality is implemented in the `StockPredictor` class, which employs a sequential approach to future price prediction. The model incorporates a rolling-window prediction mechanism, where each prediction becomes part of the input sequence for the next forecast, enabling multi-step predictions up to 10 business days into the future. This is implemented through an iterative process where:

```
current_sequence = np.roll(current_sequence, -1, axis=1)
current_sequence[0, -1, 0] = next_pred
```

The prediction confidence is quantified using a statistical approach that calculates prediction intervals based on the standard deviation of historical prediction errors:

```
lower_bound = price - 2*std_dev
upper_bound = price + 2*std_dev
```

Hyperparameters

The model's performance is governed by key hyperparameters carefully tuned for stock price prediction. The network architecture uses two LSTM layers (100 units each) with dropout rates of 0.2, followed by dense layers (50 units, 1 unit) for feature extraction and prediction. The temporal aspect is handled by a 60-day lookback period, capturing relevant market patterns while maintaining computational efficiency. Training parameters include a batch size of 32, up to 150 epochs with early stopping (patience=15), and a 0.15 validation split. The learning rate adapts through `ReduceLROnPlateau` (factor=0.2, patience=5, min_lr=0.0001). The training-test split is set at 0.8, and future predictions extend to 10 days with confidence intervals at two standard deviations. These parameters balance model complexity, training stability, and prediction accuracy while preventing overfitting. The learning rate is dynamically adjusted using `ReduceLROnPlateau`

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=5,
    min_lr=0.0001
)
```

3 Evaluation and Visualization

Our model's performance was evaluated through a comprehensive set of metrics and visualizations that provide insights into prediction accuracy and model behavior. The evaluation metrics show strong predictive performance, with an R-squared value of 0.9925 indicating that our model explains 99.25% of the variance in stock price movements. The Mean Absolute Percentage Error (MAPE) of 3.31% suggests high accuracy in price predictions, while the Root Mean Squared Error (RMSE) of $13.37 provides a scale-dependent measure of prediction accuracy.
The training visualization suite offers four key perspectives on model performance:

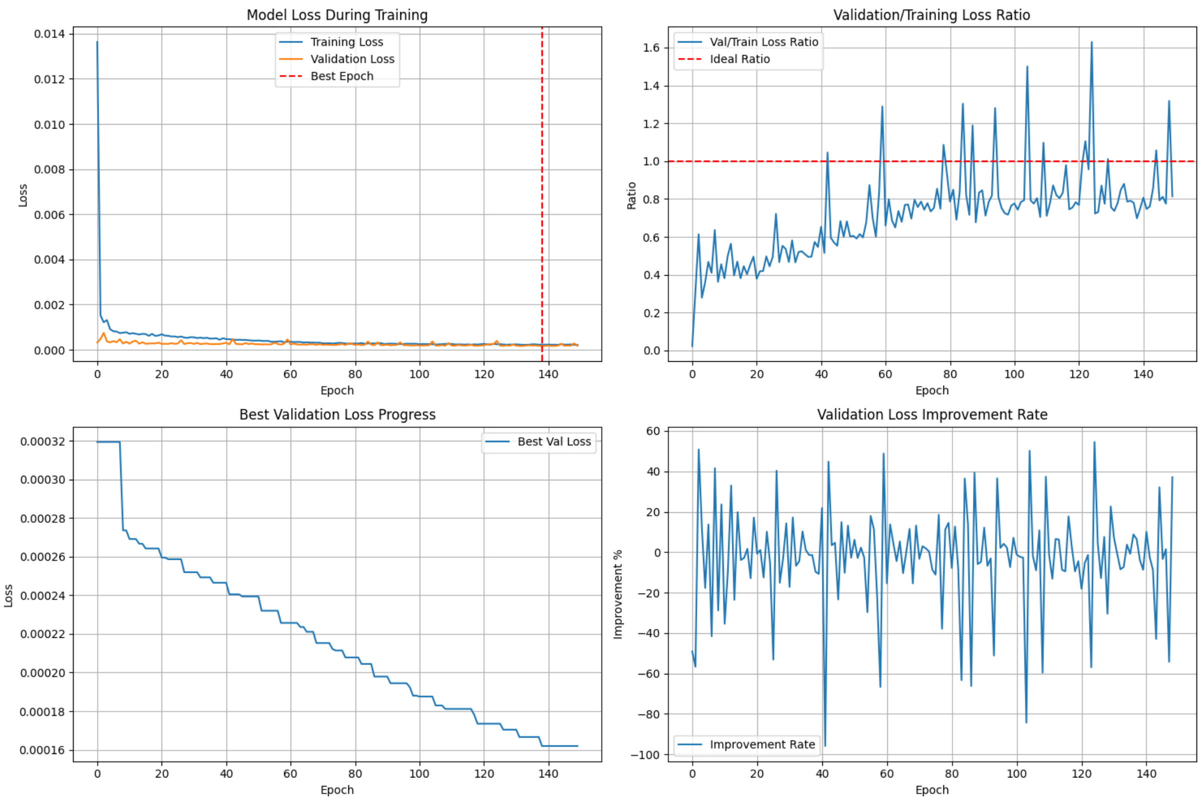

Figure 1: Model Training and Validation Loss Analysis

Model Loss During Training (top-left): Shows the convergence of training and validation loss, with the red dashed line indicating the best epoch. The close tracking between training and validation loss suggests good generalization without overfitting.

Validation/Training Loss Ratio (top-right): Tracks the relationship between validation and training loss, with the red dashed line at 1.0 representing ideal balance. The ratio staying below 1.0 indicates the model maintains good generalization capacity.

Best Validation Loss Progress (bottom-left): Demonstrates consistent improvement in model performance, with the curve showing diminishing returns as training progresses, suggesting effective learning.

Validation Loss Improvement Rate (bottom-right): Shows the percentage improvement in validation loss between epochs, with fluctuations indicating the dynamic nature of the learning process while maintaining an overall positive trend.

The negative Mean Error of -4.98 with a Standard Deviation of 12.41 suggests a slight conservative bias in predictions, which could be beneficial for risk-averse investment strategies. This comprehensive evaluation framework provides confidence in the model's predictive capabilities while highlighting areas for potential improvement.

The prediction analysis is visualized through four complementary plots that provide deep insights into our model's performance characteristics:

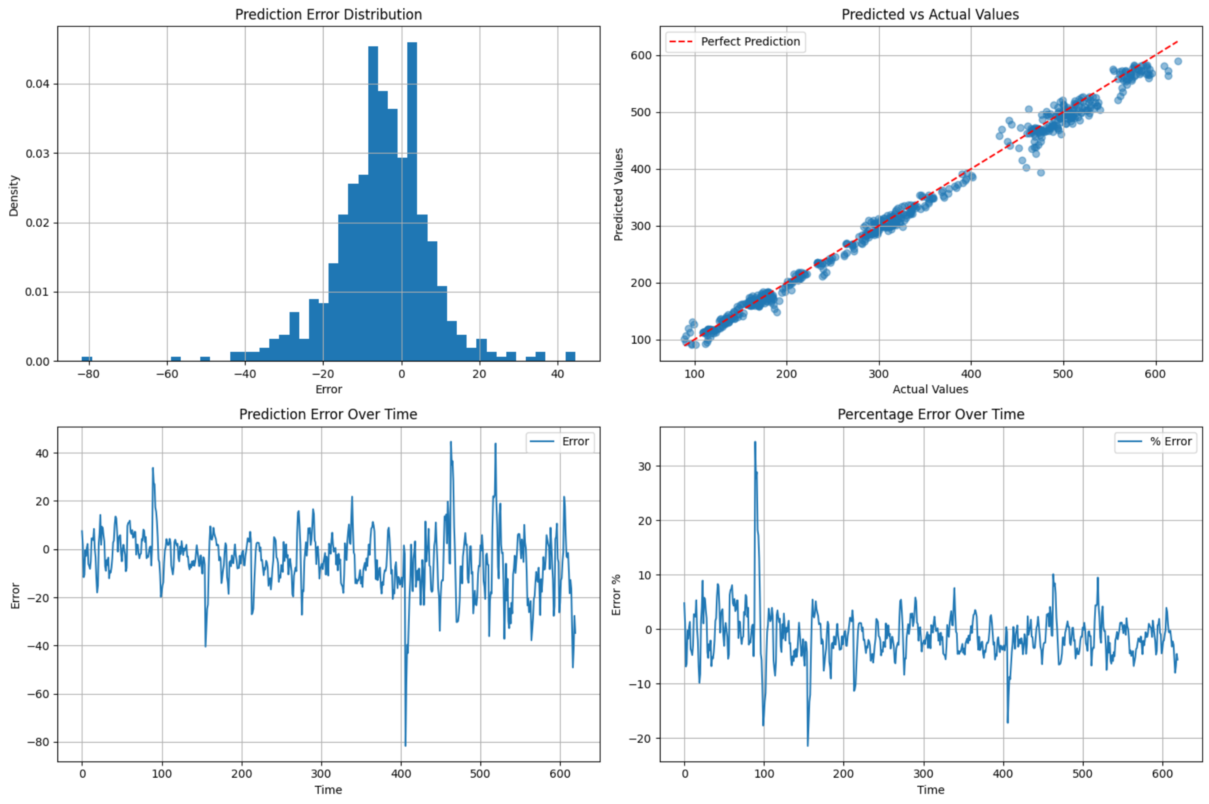

Figure 2: Prediction Error and Model Evaluation

Prediction Error Distribution (top-left) shows a bell-shaped distribution centered near zero, indicating our model's errors follow a roughly normal distribution. The slight leftward skew aligns with our negative mean error (-4.98), suggesting a tendency to underestimate prices slightly – a conservative bias that could be valuable for risk management.

Predicted vs Actual Values (top-right) demonstrates the model's accuracy through a scatter plot where points closely follow the red dashed ”perfect prediction” line. The tight clustering around this line, particularly in the lower and middle price ranges, confirms our high R-squared value (0.9925). The slight dispersion at higher prices suggests increased uncertainty in predicting extreme market movements.

Prediction Error Over Time (bottom-left) tracks absolute error across the testing period. While most errors stay within ±20 dollars, occasional spikes highlight periods of market volatility where prediction becomes more challenging. This temporal view helps identify market conditions where the model might need additional attention.

Percentage Error Over Time (bottom-right) provides a scale-independent view of prediction accuracy. The majority of percentage errors remain within ±5%, with occasional outliers reaching ±20%. This stability in percentage error across different price levels indicates consistent relative performance, supporting the model's reliability for practical applications.

Historical price prediction analysis

The visualization below compares META's actual stock prices (solid blue line) against our model's predictions (dashed red line) over the historical period. The graph demonstrates remarkable prediction accuracy, particularly in capturing both long-term trends and short-term price movements. Looking at specific price points from our validation data, we can observe the model's performance across different price ranges. At lower price levels (around \$155-170), the predictions closely track actual prices with minimal deviation. The model shows consistent performance through the middle range, and even during the significant price surge from \$400 to \$600, it maintains strong predictive capability. However, examining the tabulated results reveals some interesting patterns. For instance, in recent high-price scenarios (¿\$600), the model tends to be more conservative, with predictions slightly underestimating actual prices (e.g., predicting \$588.94 when the actual price was \$623.77). This conservative bias aligns with our earlier error analysis and could be beneficial for risk-averse investment strategies. The model's ability to capture META's substantial price appreciation while maintaining prediction accuracy is particularly noteworthy. The smooth tracking of the prediction line suggests that our LSTM architecture, combined with the 60-day lookback period, effectively captures both the momentum and trend changes in META's stock price movements.

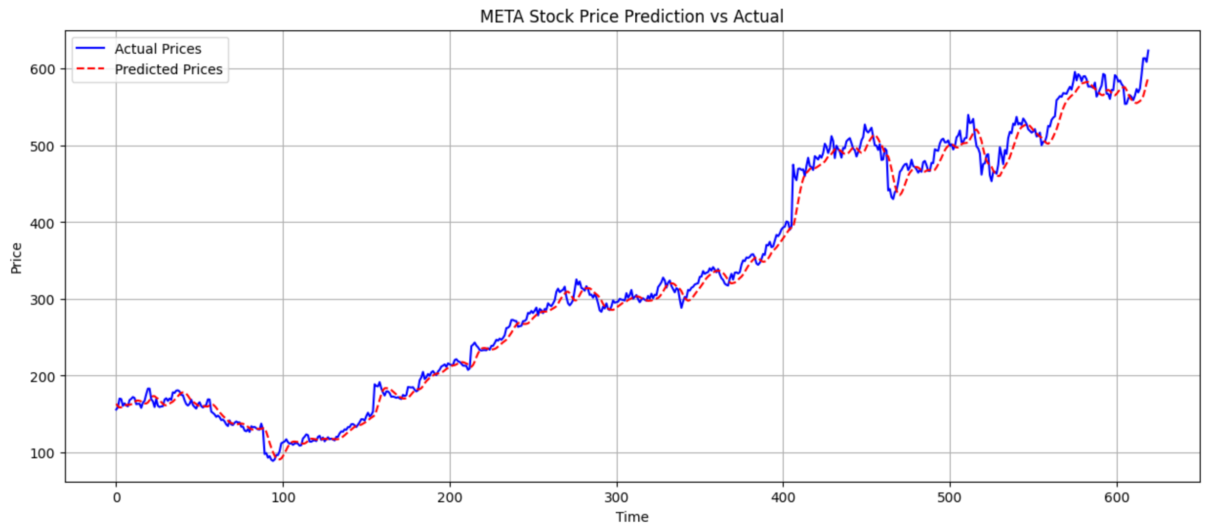

Figure 3: META Stock Price Prediction vs Actual Describes the comparison between the actual stock prices and predicted stock prices for META.

Future price prediction analysis

The model's future forecast for META stock demonstrates an interesting price trajectory over the next 10 business days. Starting from the last actual price of \$623.77, the model predicts an initial adjustment to \$596.88, followed by a modest upward movement reaching \$601.76 by December 11th, before gradually declining to \$591.42 by December 20th. The prediction mechanism, implemented through a rolling-window approach (lines 167-178 in `StockPredictor` class), generates sequential predictions while maintaining temporal dependencies. Each prediction becomes input for the next forecast, creating a coherent price trajectory:

```
current_sequence = np.roll(current_sequence, -1, axis=1)
current_sequence[0, -1, 0] = next_pred
```

The model incorporates uncertainty through 95% confidence intervals, calculated using historical prediction error standard deviation. These intervals widen slightly as predictions extend further into the future, with an average range of approximately ±25\$, reflecting increased uncertainty over time. For instance, the December 20th prediction of \$591.42 has a confidence interval of \$566.61 to \$616.23. The visualization includes a green-shaded confidence band and dotted prediction line, providing a clear distinction between historical performance and future projections. This conservative prediction pattern, showing a slight downward trend after initial stability, aligns with the model's demonstrated tendency to be prudently conservative in its estimates, particularly at higher price levels.

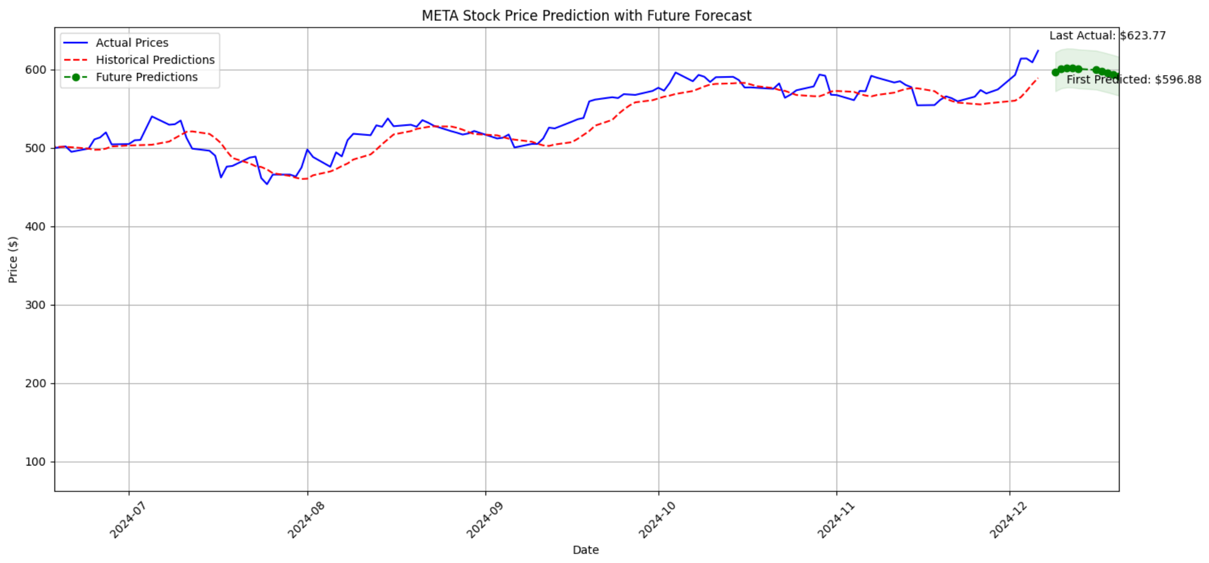

Figure 4: META Stock Price Prediction vs Actual (with 10-Day Forecast)

Table 1: Future Price Predictions

Date	Price ($)
2024-12-09	596.88
2024-12-10	600.54
2024-12-11	601.76
2024-12-12	601.59
2024-12-13	600.60
2024-12-16	599.14
2024-12-17	597.40
2024-12-18	595.49
2024-12-19	593.48
2024-12-20	591.42

Table 2: Prediction Intervals (95% confidence)

Date	Lower ($)	Upper ($)
2024-12-09	572.07	621.70
2024-12-10	575.73	625.35
2024-12-11	576.95	626.57
2024-12-12	576.78	626.40
2024-12-13	575.79	625.41
2024-12-16	574.33	623.95
2024-12-17	572.59	622.21
2024-12-18	570.68	620.30
2024-12-19	568.67	618.29
2024-12-20	566.61	616.23

Project source code

The full project source is available as a Google Colab notebook. It can be accessed by clicking the following link: **Google Colab Notebook**.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. pages 1

[2] Yahoo Finance. Yahoo finance api documentation. `https://finance.yahoo.com`, 2024. Accessed: 2024. pages 2