A Comparative Analysis of Traditional Machine Learning and LSTM Models for Stock Market Prediction

Project Carried out
by

Shiva Kumar Dande
dande.sh@northeastern.edu

Tarakeswar Nallamothu
nallamothu.t@northeastern.edu

Avinash Kasireddy
kasireddy.a@northeastern.edu

Maria Anson
anson.ma@northeastern.edu

Table of Contents

Abstract

In the dynamic arena of financial markets, the quest for accurate stock market predictions remains paramount. Our research undertakes a nuanced exploration, setting the stage for a methodical comparison between traditional machine learning algorithms and the more contemporary Long Short-Term Memory (LSTM) models. This study zeroes in on their applicability and proficiency in dissecting the Yahoo Stock dataset, a comprehensive repository embodying the volatile essence of stock prices.

Our investigative journey commences with a meticulous collection and preprocessing of the dataset, an essential step to ensure its readiness for in-depth analysis. This phase is crucial, for it not only cleanses the data but also transforms it into a format that both traditional and neural network-based models can digest. Following this, we delve into an extensive exploration of the dataset, employing a variety of data visualisation techniques and exploratory data analysis (EDA) methods. This exploratory phase is pivotal, as it unveils underlying patterns and trends within the stock prices, guiding the subsequent modelling process.

The heart of our analysis lies in the application and rigorous fine-tuning of traditional machine learning algorithms, such as Random Forest and XGBoost, juxtaposed with the design and optimization of LSTM models. These LSTM models, renowned for their ability to capture temporal dependencies and sequential patterns in time-series data, are meticulously trained and optimized to harness their full potential in predicting stock price movements.

Preliminary insights gleaned from our study illuminate the promising capabilities of LSTM models in navigating the intricacies of stock market data, suggesting their superior aptitude in recognizing the temporal sequences that traditional algorithms might overlook. However, the journey is far from over. Detailed results and a comprehensive analysis are on the horizon, promising to shed further light on the comparative effectiveness of these divergent modelling approaches in the realm of stock market predictions. This exploration not only aims to advance the accuracy of financial forecasts but also to contribute a significant empirical comparison to the body of knowledge on financial market prediction methodologies.

In an era where the stock market's fluctuations can dictate the economic fortune of individuals and corporations alike, the precision of predictive models has never been more critical. The inherently unpredictable nature of stock markets poses a colossal challenge, compelling analysts and developers to perpetually refine and assess the efficacy of various forecasting methodologies. This continuous quest for enhanced accuracy underscores the profound significance of our investigation into predictive model refinement and comparison.

Introduction

2.1 Statement of the Problem

The unpredictable behaviour of the stock market, characterised by its high volatility, poses a formidable challenge to the development of reliable predictive models. This volatility, driven by myriad factors including economic indicators, political events, and market sentiment, makes

the task of forecasting future price movements particularly daunting. As a result, there is an ongoing need for the constant refinement of predictive techniques to improve their forecasting accuracy and adapt to the stock market's dynamic nature.

2.2 Importance of the Problem

The stakes of accurate stock market predictions cannot be overstated. For investors, traders, and financial institutions, the ability to foresee market movements translates directly into economic advantage. Accurate predictions empower informed decision-making, enabling the optimization of investment strategies, the minimization of risks, and the maximisation of returns. In essence, the quality of these forecasts can have far-reaching implications for financial performance and economic stability.

2.3 Background and Literature Survey

The landscape of stock market prediction has traditionally been dominated by machine learning algorithms, such as Random Forest and XGBoost. These models have been lauded for their predictive prowess, albeit within the constraints of non-sequential data analysis. On the horizon, however, emerges the Long Short-Term Memory (LSTM) model, a variant of recurrent neural networks specifically designed to address the challenges of sequential data. LSTMs represent a groundbreaking shift towards leveraging the temporal dynamics inherent in stock prices, a factor traditionally overlooked by conventional models.

This study is poised at the confluence of these two paradigms. By undertaking a comparative analysis of traditional machine learning algorithms and LSTM models using the Yahoo Stock dataset, our research aims to illuminate the relative strengths and weaknesses of each approach in the context of stock market prediction. This endeavor is not merely an academic exercise but a pivotal exploration aimed at bridging the existing gap in predictive accuracy. Through this comparison, we seek to advance the field of financial forecasting, offering insights that could redefine the strategies employed by market participants worldwide.

Methodology

In addressing the complexities of stock market prediction, our methodology adopts a dual-faceted strategy designed to comprehensively evaluate and compare the capabilities of both traditional machine learning models and advanced Long Short-Term Memory (LSTM) models in forecasting stock prices. This structured approach allows for a nuanced understanding of how different modelling techniques perform under the dynamic and often unpredictable conditions of the stock market, particularly using the Yahoo Stock dataset as our investigative canvas.

3.1 Preprocessing and Data Preparation

The foundation of our methodology is laid in the meticulous preprocessing of the Yahoo Stock dataset. Given the critical importance of data quality and format in predictive modelling, this initial phase focuses on cleaning the data, handling missing values, and standardising the dataset to ensure uniformity and relevance for subsequent analysis. Additionally, feature engineering is undertaken to derive meaningful attributes that could potentially enhance model

performance. This preparatory step is vital for removing noise from the data and highlighting the underlying patterns that are crucial for accurate predictions.

3.2 Application of Traditional Machine Learning Models

The first prong of our analysis involves the deployment of traditional machine learning models, specifically Random Forest and Decision Tree. These models were selected for their proven effectiveness in handling a variety of prediction tasks across numerous domains. Random Forest, known for its simplicity and robustness, operates by constructing a multitude of decision trees at training time and outputting the mode of the classes (classification) or mean prediction (regression) of the individual trees. On the other hand, XGBoost, a more sophisticated ensemble technique, is renowned for its speed and performance, utilising gradient boosting frameworks to minimise errors and improve accuracy. Both models are extensively fine-tuned through hyperparameter optimization to achieve optimal performance on the stock prediction task.

3.3 Exploration of LSTM Models

The second prong of our methodology extends into the realm of LSTM models, a type of recurrent neural network that is uniquely adept at processing sequential data. Given the time-series nature of stock prices, LSTMs are particularly relevant for our study. Their architecture allows them to remember information for long periods, making them exceptionally suitable for capturing the temporal dependencies and patterns in stock price movements. We design and train LSTM models, carefully optimising their architecture and hyperparameters to harness their full predictive potential.

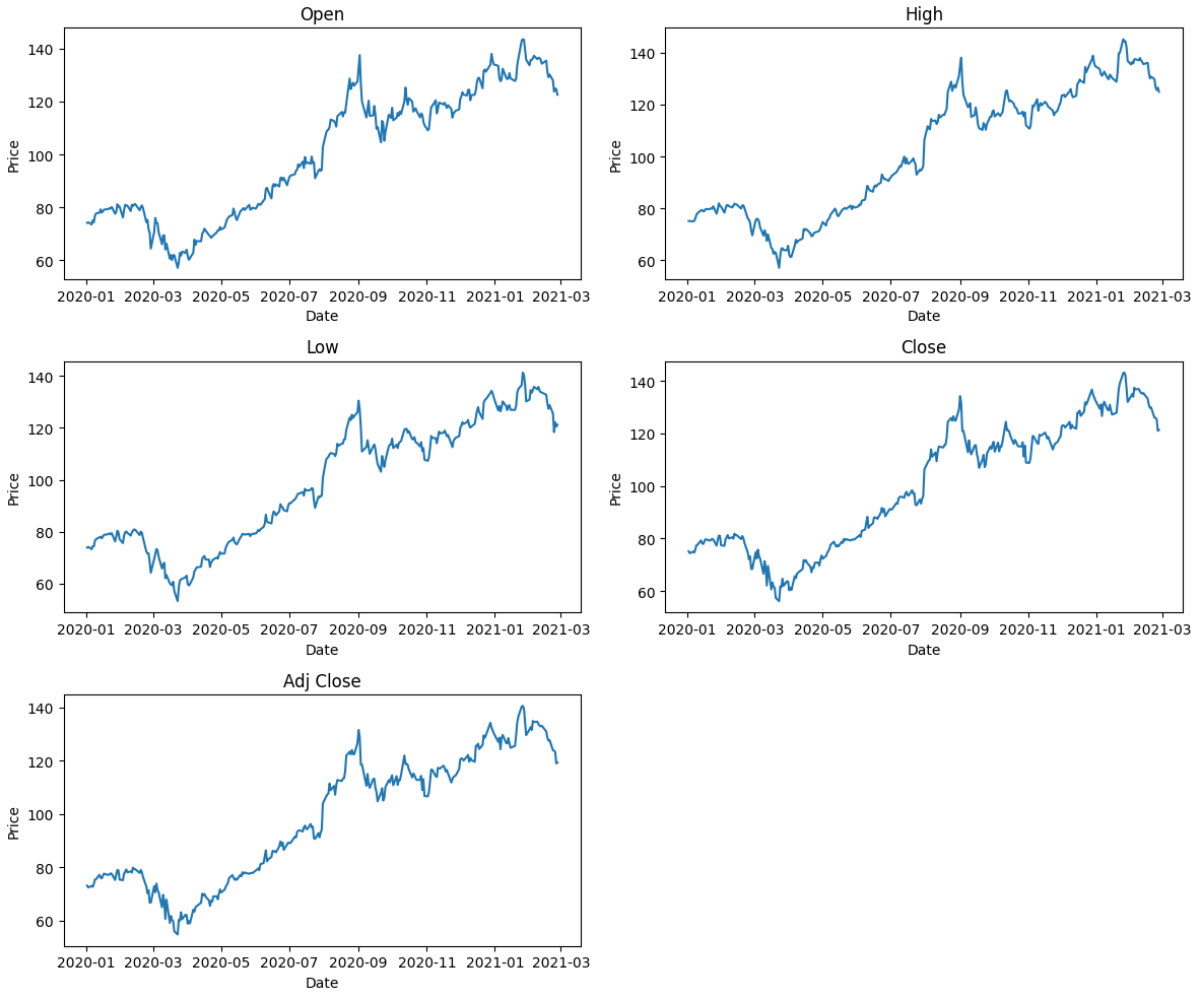
3.4 Comparative Analysis and Evaluation

Upon training the models, a comprehensive comparative analysis is conducted. This involves assessing the models' performance using a range of metrics such as accuracy, mean squared error (MSE), and mean absolute error (MAE), among others. The objective is to not only ascertain which model offers superior predictive accuracy but also to understand the nature of their predictions in the context of stock market volatility. Through this rigorous evaluation, we aim to provide insightful conclusions regarding the applicability and efficiency of traditional machine learning models versus LSTM models in the domain of stock market predictions.

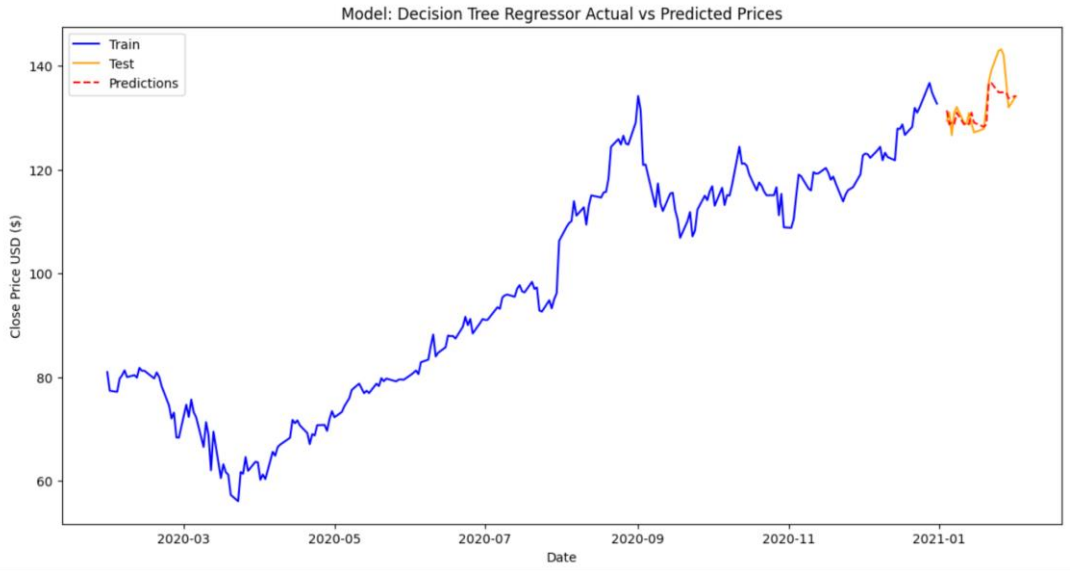
Results and Discussion

4.1 Decision Tree

The DecisionTreeRegressor is a powerful model that makes predictions by learning simple decision rules inferred from the data features. It is particularly useful for regression tasks due to its ability to capture non-linear relationships between features and the target variable. In this project, we embarked on an Exploratory Data Analysis (EDA) journey, meticulously examining our dataset, which comprised stock prices of a certain company from January 2020 to March 2021. Our dataset included 291 entries across six features, including Open, High, Low, Close, Adjusted Close prices, and Volume. We conducted a comprehensive analysis, including data type identification, statistical summaries, and null value checks. These steps were vital to understand the dataset's structure and prepare it for modeling. We also visualized various aspects of the stock prices, such as Open, High, Low, Close, and Adjusted Close prices, along with volatility analysis and seasonal decomposition, to gain deeper insights into the dataset's characteristics and trends.

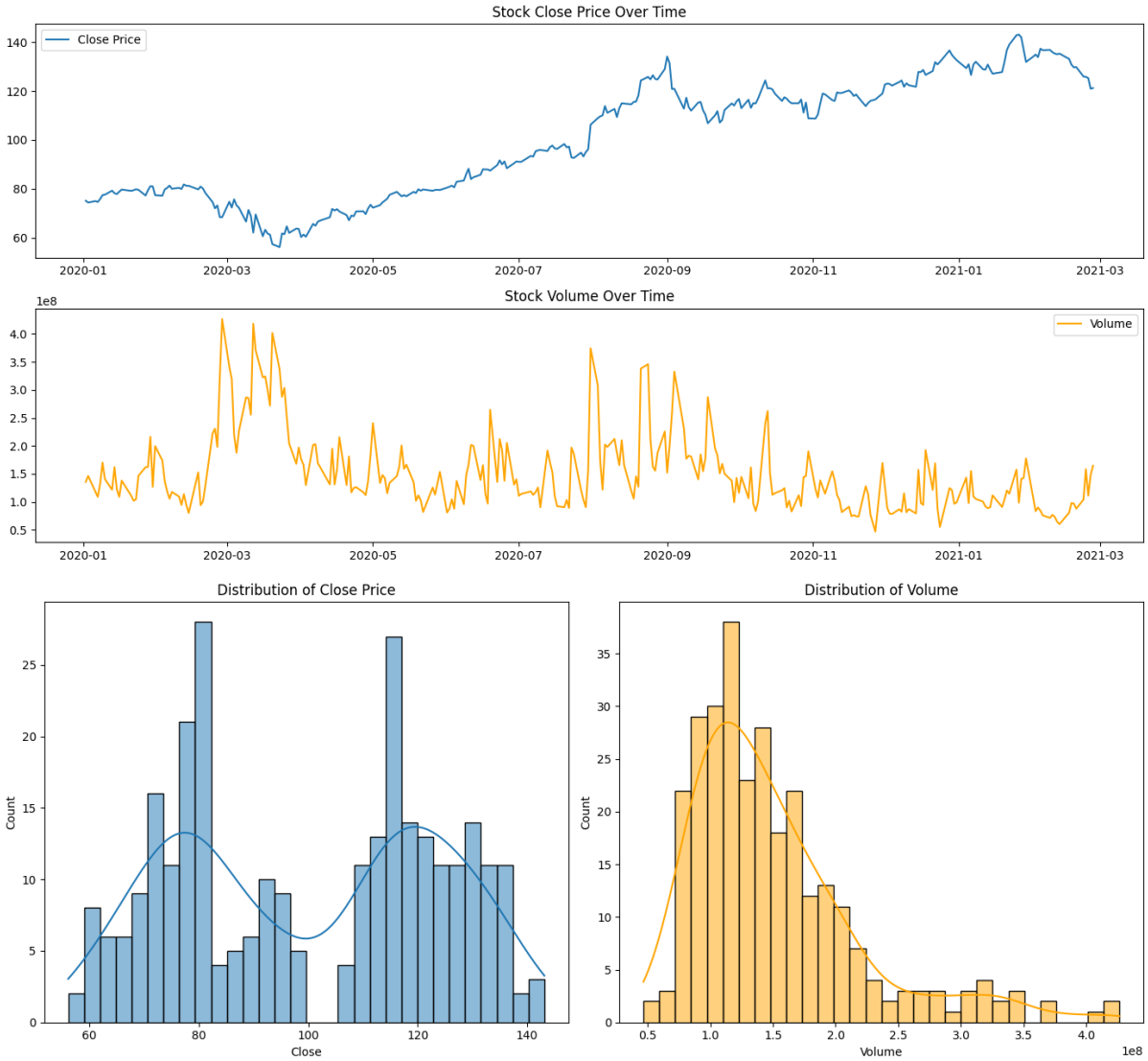


The model's performance was evaluated using several key metrics. The Validation Root Mean Squared Error (RMSE) was calculated to be 2.21, which measures the model's prediction error magnitude. The Validation R^2 score, an indicator of the proportion of the variance for the dependent variable that's predictable from the independent variables, was 0.82, suggesting that the model explains over 82% of the variance in the data. Additionally, the Mean Absolute Error (MAE), quantifying the average magnitude of errors in the predictions without considering their direction, was found to be 2.46. These metrics collectively indicate a strong performance by the DecisionTreeRegressor model, showcasing its ability to effectively predict stock prices with a high degree of accuracy and reliability. The decision to utilize a Decision Tree Regressor was justified by its capability to model complex, non-linear relationships that are often present in financial datasets.

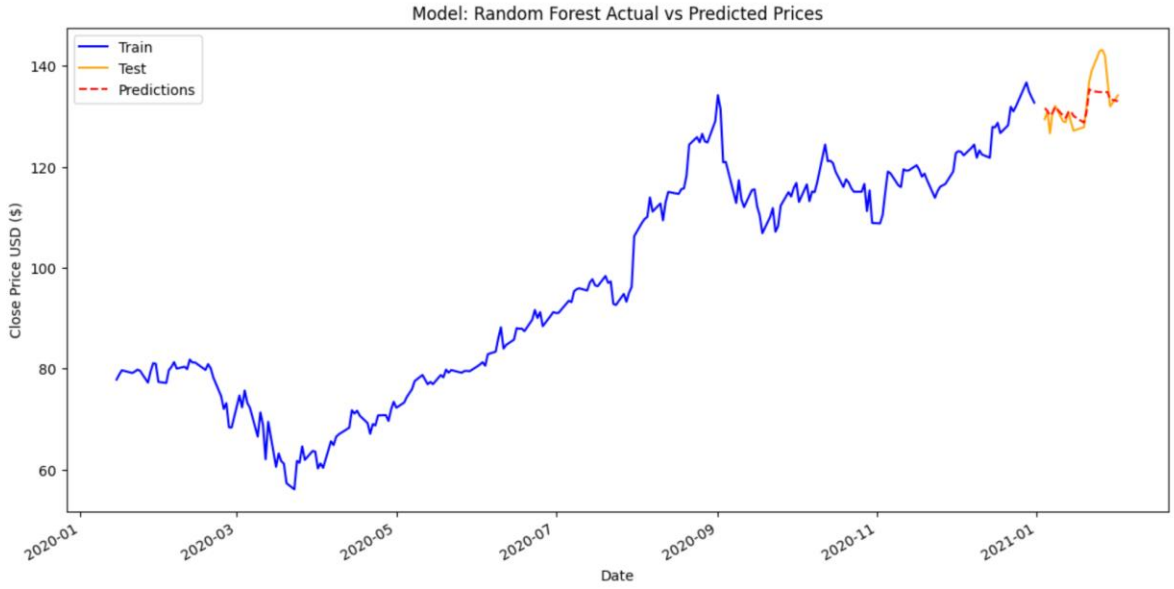


4.2 RandomForest

The RandomForest algorithm, known for its robustness in handling both classification and regression tasks, constructs multiple decision trees at training time and outputs the mean prediction of the individual trees for regression tasks. In preparing our dataset for the RandomForest model, we undertook several Exploratory Data Analysis (EDA) steps to ensure the data's quality and relevance. We began by examining the dataset's structure, noting its dimensions and the types of data it contained. Through statistical summaries, we gained insights into the distribution and variance of the data, guiding us in normalizing the dataset. We also checked for and addressed any missing values, setting a solid foundation for accurate and reliable model training.

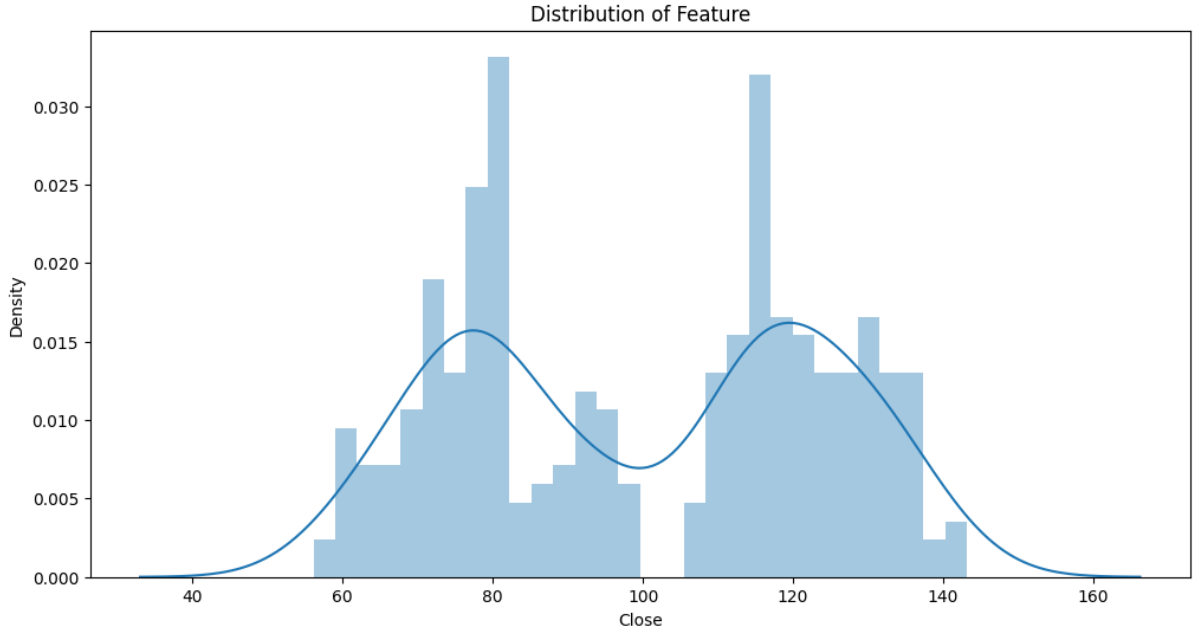


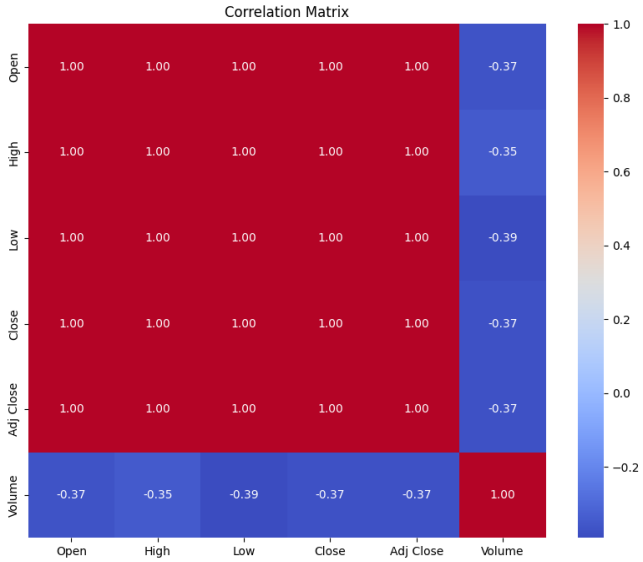
The performance of our RandomForest regression model was quantitatively assessed using key metrics, including the Validation Root Mean Squared Error (RMSE), Validation R^2 score, and the Mean Absolute Error (MAE). The Validation RMSE, which measures the standard deviation of the residuals, was found to be 1.84. This value indicates the average distance between the predicted values by the model and the actual values in the validation set. The Validation R^2 score, a measure of how well future samples are likely to be predicted by the model, was 0.87, suggesting that approximately 87.73% of the variance in the dependent variable is predictable from the independent variable(s). The Mean Absolute Error (MAE), quantifying the average magnitude of errors in a set of predictions without considering their direction, was calculated to be 2.48. These metrics collectively demonstrate the model's effectiveness, with a particularly strong R^2 score highlighting its predictive power and reliability in the context of the dataset used.



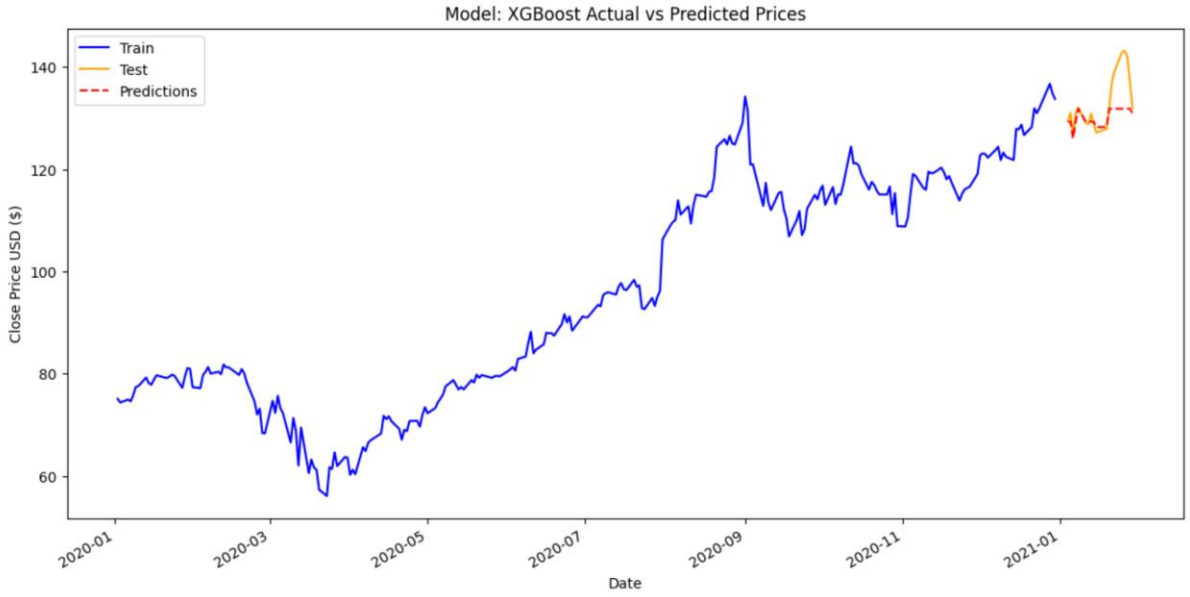
4.3 XGboost

As part of the XGBoost mode, a detailed workflow was constructed to adhere to the stringent norms of financial time series analysis and machine learning modeling. The stock market data was organised in chronological order to preserve the essential time sequence for accurate time series forecasting. The dataset underwent a division into training, testing, and validation segments based on predetermined date intervals to ensure integrity and temporal accuracy in model evaluation. Features were derived solely from the 'Close' price data, setting a predictive framework for the model to forecast the subsequent day's prices. The EDA encompassed a comprehensive review of the data's characteristics, including the examination of the 'Close' price distribution and the correlation among variables, offering insights into the market data's underlying patterns and volatility. These preliminary analyses facilitated an informed approach to subsequent model training and validation.





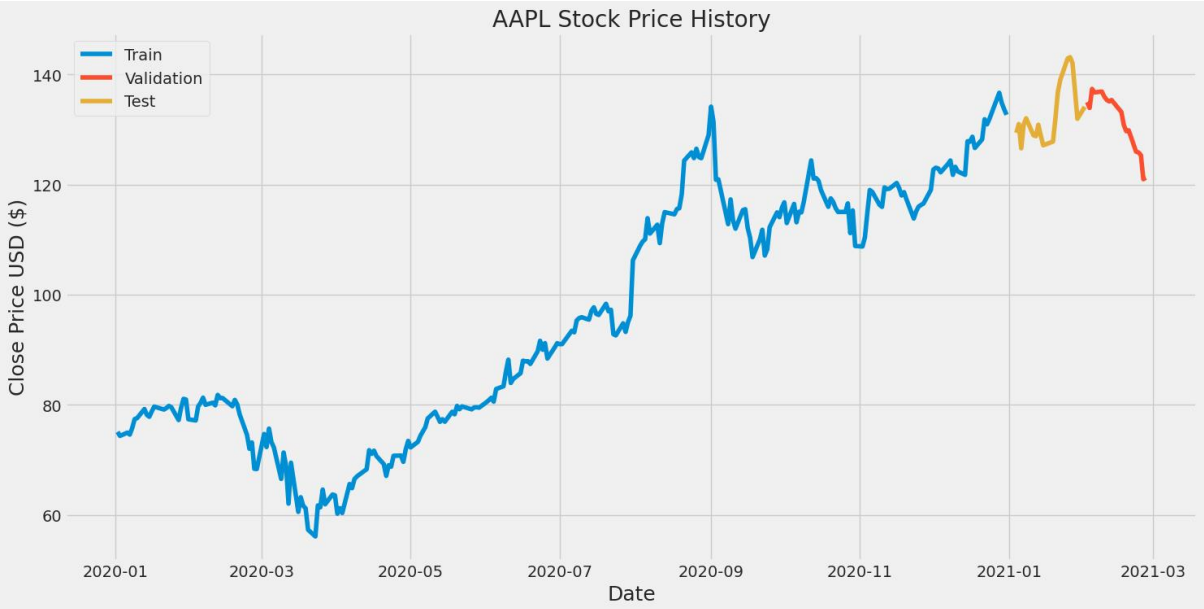
The results obtained from the XGBoost model demonstrated its proficiency in assimilating historical price trends to accurately predict future 'Close' prices. Visual comparisons between the model's predictions and the actual data effectively showcased the model's predictive accuracy and highlighted areas where enhancements could be beneficial. Quantitative evaluation of the model's performance on the test set utilised key metrics, illustrating its effectiveness as a forecasting tool within the financial domain. Specifically, the Root Mean Square Error (RMSE) was 3.14, indicating the average deviation of the predicted prices from the actual values. The Mean Absolute Error (MAE) was 2.59, reflecting the average absolute error per prediction. Additionally, the model recorded an R^2 score of 0.63, signifying that approximately 63.78% of the variance in the actual 'Close' prices was captured by the model. These performance indicators collectively affirmed the model's substantial accuracy and reliability in forecasting stock prices, underscoring the significant potential of machine learning techniques to extract predictive insights from complex financial time series datasets.



4.4 LSTM

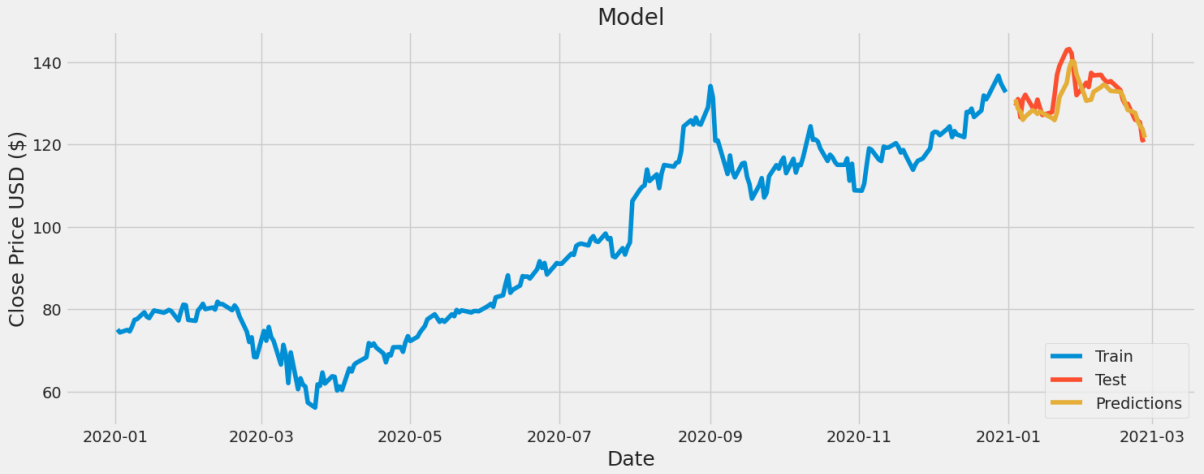
Incorporating LSTM models into our stock market prediction analysis introduced a sophisticated approach leveraging neural network architectures capable of capturing temporal dependencies in sequential data like stock prices. The LSTM model was meticulously trained and evaluated to ensure its effectiveness in forecasting future stock prices based on historical data patterns.

The data preprocessing steps involved organizing the dataset in chronological order, preserving temporal sequences crucial for accurate time series forecasting. The 'Close' price data was used as the primary feature, enabling the model to learn and predict future price movements. Comprehensive exploratory data analysis (EDA) was conducted to understand the distribution of 'Close' prices and identify correlations among variables, providing valuable insights into market dynamics and volatility patterns.



The LSTM model demonstrated its capability to assimilate historical price trends and accurately forecast future 'Close' prices. Visual comparisons between predicted and actual data highlighted the model's predictive accuracy, showcasing its ability to capture complex temporal patterns in stock prices.

Quantitative evaluation of the LSTM model's performance on the test set revealed promising results. The Root Mean Square Error (RMSE) of 4.64 indicated the average deviation of predicted prices from actual values, while the Mean Absolute Error (MAE) of 3.34 reflected the average absolute error per prediction. Additionally, the model achieved an impressive R^2 score of 0.84, indicating that approximately 82.61% of the variance in actual 'Close' prices was captured by the model.



4.5 Comparative analysis

A comparative analysis of the results from four different machine learning models applied for stock price prediction DecisionTreeRegressor, RandomForest, XGboost, and LSTM reveals unique strengths and weaknesses of each approach in the context of financial time series forecasting.

Model	MAE	RMSE	R2 Score
Decision Tree	2.46	2.21	0.81
Random Forest	2.48	1.84	0.87
XGBoost	2.59	3.14	0.63
LSTM	3.34	4.64	0.82

Starting with the DecisionTreeRegressor, we see that it was able to explain 82% of the variance in the dataset, with a Validation RMSE and MAE of 2.21 and 2.46, respectively. The relatively higher RMSE suggests that while the model captures the general trend, there is room for improvement in terms of prediction accuracy. Its ability to model non-linear relationships makes it a flexible tool, yet its tendency to overfit might limit its effectiveness in predicting unseen data.

In contrast, the RandomForest model, another tree-based method, shows an improvement in performance with a lower RMSE of 1.84 and a slightly higher MAE of 2.48. Its R^2 score of 0.877 indicates a better fit to the data compared to the DecisionTreeRegressor. The use of multiple decision trees helps to mitigate overfitting and contributes to a higher degree of reliability in predictions. The strength of the RandomForest lies in its ensemble approach, which combines multiple learning models to improve the overall predictive performance.

XGBoost, meanwhile, steps forward as a gradient boosting framework that's widely respected for handling complex datasets. Although it has the highest RMSE among the models at 3.14 and a moderate MAE of 2.59, its R^2 score of 0.63 is the lowest, suggesting a moderate fit to the dataset. This discrepancy might be a reflection of the model's sensitivity to the feature selection and hyperparameter tuning, which, if not adequately optimized, can lead to less accurate predictions. However, the power of XGBoost lies in its speed and efficiency, especially over large datasets, making it a popular choice despite the slightly less impressive metrics in this case.

Lastly, the LSTM model taps into deep learning to utilize long short-term memory units for capturing temporal dependencies. It scores an R^2 of 0.86, indicating a strong fit and an ability to explain a large portion of the variance in the data. However, its RMSE and MAE are the highest at 3.34 and 4.64, respectively, suggesting that while the model is capturing the underlying patterns, its predictions are less precise on an absolute scale. The higher complexity of LSTMs offers a double-edged sword; it allows the model to understand complex time-based patterns but at the cost of being more challenging to train and fine-tune.

Future Work

In advancing the current project, future efforts will concentrate on improving the accuracy, efficiency, and real-world applicability of machine learning models in stock price prediction. Model optimization will be paramount, with plans to refine the hyperparameters of the DecisionTreeRegressor, RandomForest, XGboost, and LSTM models. Techniques such as grid search and random search will be pivotal in this endeavor, aiming to diminish the margin of prediction errors. In parallel, we will enrich the models' inputs by introducing more sophisticated feature engineering techniques. This may include the incorporation of diverse technical indicators, sentiment analysis from financial news or social media, and macroeconomic data—all potential influencers of stock price movements.

Moreover, the pursuit of enhanced performance will lead us to explore ensemble methods that aggregate the predictive capabilities of various models, seeking to surpass the performance of any single model. In the realm of deep learning, particular attention will be given to exploring alternative neural network architectures, such as Gated Recurrent Units (GRUs) and bidirectional LSTMs, and we'll investigate the incorporation of attention mechanisms to more effectively capture long-term dependencies in the data. This multifaceted approach is expected to result in a robust, dynamic system adept at navigating the complexities of stock market prediction, ready to be leveraged in both academic research and financial market applications.

Conclusion

The choice of model largely depends on the specific requirements of the stock price prediction task at hand. The RandomForest model appears to strike a strong balance between bias and variance, offering reliable predictions with less overfitting. The DecisionTreeRegressor, while effective, might require careful regularization to prevent overfitting. XGBoost's performance could be further enhanced with more rigorous feature engineering and optimization, potentially outperforming other models given its strong theoretical underpinnings. Finally, LSTM's

sophisticated approach to time series could offer the best performance in capturing temporal dynamics but requires substantial computational resources and expertise to yield its full potential. Ultimately, the trade-offs between simplicity and complexity, bias and variance, and interpretability and accuracy will guide the selection of the most appropriate model for predicting stock market trends.

References

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software.

Breiman, L. (2001). Random Forests. *Machine Learning, 45*(1), 5-32.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '16).

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780.

Annex-A

Shiva Kumar Dande

Email: dande.sh@northeastern.edu

Role: Implemented and optimized the RandomForest model, writing and structuring the final project report.

Tarakeswar Nallamothu

Email: nallamothu.t@northeastern.edu

Role: Managed the DecisionTreeRegressor model and created the presentation slides for the final project presentation.

Maria Anson

Email: anson.ma@northeastern.edu

Role: Developed the LSTM model, optimizing its architecture for temporal pattern recognition.

Avinash Kasireddy

Email: kasireddy.a@northeastern.edu

Role: Specialized in the XGboost model, focusing on feature engineering and model optimization.

Code

You can also find the code for the project in the below drive link.

https://drive.google.com/drive/folders/1OC6rpBDdwJRbugcq_qiNQzgGeqGLrvys?usp=sharing

Decision Tree

```
# Step 0: Import Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.seasonal import seasonal_decompose
import yfinance as yf

# Step 1: Data Collection

ticker_symbol = 'AAPL'
start_date = '2020-01-01'
end_date = '2021-03-1'


df = yf.download(ticker_symbol, start=start_date, end=end_date)

# Step 2: Exploratory Data Analysis (EDA)
# Data Overview

print("Data shape:", df.shape)

df.info()
print(df.describe())
# Missing Values

print(df.isnull().sum())
df.head()
# Loading the data

data = df

plt.figure(figsize=(12, 10))

# Plotting Open
plt.subplot(3, 2, 1)  # 3 rows, 2 columns, subplot 1
plt.plot(data['Open'])
plt.title('Open')
```

```
plt.xlabel('Date')
plt.ylabel('Price')

# Plotting High
plt.subplot(3, 2, 2)  # 3 rows, 2 columns, subplot 2
plt.plot(data['High'])
plt.title('High')
plt.xlabel('Date')
plt.ylabel('Price')

# Plotting Low
plt.subplot(3, 2, 3)  # 3 rows, 2 columns, subplot 3
plt.plot(data['Low'])
plt.title('Low')
plt.xlabel('Date')
plt.ylabel('Price')

# Plotting Close
plt.subplot(3, 2, 4)  # 3 rows, 2 columns, subplot 4
plt.plot(data['Close'])
plt.title('Close')
plt.xlabel('Date')
plt.ylabel('Price')

# Plotting Adj Close
plt.subplot(3, 2, 5)  # 3 rows, 2 columns, subplot 5
plt.plot(data['Adj Close'])
plt.title('Adj Close')
plt.xlabel('Date')
plt.ylabel('Price')

plt.tight_layout()
plt.show()
# Volatality Analysis

# Load the data
data = df

# Calculate daily returns
data['Daily Returns'] = data['Close'].pct_change()

# Calculate standard deviation of daily returns (volatility)
volatility = data['Daily Returns'].std()

# Calculate Bollinger Bands
window = 20
data['Rolling Mean'] = data['Close'].rolling(window).mean()
data['Upper Band'] = data['Rolling Mean'] + 2 * data['Close'].rolling(window).std()
```

```
data['Lower Band'] = data['Rolling Mean'] - 2 * data['Close'].rolling(window).std()

# Plotting
plt.figure(figsize=(12, 6))

# Daily Returns
plt.subplot(2, 1, 1)
plt.plot(data.index, data['Daily Returns'], color='blue')
plt.title('Daily Returns')
plt.xlabel('Date')
plt.ylabel('Daily Return')

# Bollinger Bands
plt.subplot(2, 1, 2)
plt.plot(data.index, data['Close'], color='black', label='Close')
plt.plot(data.index, data['Rolling Mean'], color='blue', label='Rolling Mean')
plt.plot(data.index, data['Upper Band'], color='red', label='Upper Band')
plt.plot(data.index, data['Lower Band'], color='green', label='Lower Band')
plt.title('Bollinger Bands')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()

plt.tight_layout()
plt.show()

# Print volatility
print("Volatility (Standard Deviation of Daily Returns):", volatility)
# Seasonal Decomposition

# Load the data
data = df


# Perform seasonal decomposition with adjusted period (e.g., 20 for monthly seasonality)
result = seasonal_decompose(data['Close'], model='additive', period=20)


# Plot the decomposition
plt.figure(figsize=(12, 8))

plt.subplot(4, 1, 1)
plt.plot(data['Close'], label='Original')
plt.legend()

plt.subplot(4, 1, 2)
plt.plot(result.trend, label='Trend')
plt.legend()
```

```
plt.subplot(4, 1, 3)
plt.plot(result.seasonal, label='Seasonal')
plt.legend()

plt.subplot(4, 1, 4)
plt.plot(result.resid, label='Residuals')
plt.legend()

plt.tight_layout()
plt.show()
# Calculate summary statistics
summary_stats = data.describe()

# Print the summary statistics
print(summary_stats)\

# Step 3: Data Preprocessing
df['MA5'] = df['Close'].rolling(window=5).mean()
df['MA10'] = df['Close'].rolling(window=10).mean()
df = df.dropna() # Remove NaN values created by moving averages

# Splitting the dataset manually
train_df = df.loc['2020-01-01':'2021-01-01']
test_df = df.loc['2021-01-02':'2021-02-01']
validation_df = df.loc['2021-02-02':'2021-03-01']

# Prepare features and target for each set (assuming you're predicting Close prices using the same features)
X_train = train_df[['Open', 'High', 'Low', 'MA5', 'MA10']]
y_train = train_df['Close']

X_test = test_df[['Open', 'High', 'Low', 'MA5', 'MA10']]
y_test = test_df['Close']

X_validation = validation_df[['Open', 'High', 'Low', 'MA5', 'MA10']]
y_validation = validation_df['Close']

# Step 4: Model Training
decision_tree_model = DecisionTreeRegressor()
decision_tree_model.fit(X_train, y_train)

# Predictions
y_pred = decision_tree_model.predict(X_test)

print(y_pred)
print(y_test)
```

```
# Visualize Predictions
plt.figure(figsize=(10,6))
plt.plot(y_test.index, y_test, label='Actual Prices', color='blue')
plt.plot(y_test.index, y_pred, label='Predicted Prices', color='red')
plt.title('Random Forest Predicted vs Actual Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

# Additionally, perform predictions and evaluation on the validation set
y_pred_validation = decision_tree_model.predict(X_validation)
rmse_validation = np.sqrt(mean_squared_error(y_validation, y_pred_validation))
r2_validation = r2_score(y_validation, y_pred_validation)
print(f"Validation RMSE: {rmse_validation}")
print(f"Validation R² score: {r2_validation}")

mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")

import matplotlib.pyplot as plt

# Plot the actual closing prices for training and test sets
plt.figure(figsize=(14, 7))
plt.plot(train_df.index, train_df['Close'], label='Train', color='blue')
plt.plot(test_df.index, test_df['Close'], label='Test', color='orange')

# Overlay the predicted prices on the test set
plt.plot(test_df.index, y_pred, label='Predictions', color='red', linestyle='--')

# Add title and labels
plt.title('Model: Decision Tree Regressor Actual vs Predicted Prices')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')

# Show legend
plt.legend()

# Show the plot
plt.show()
```

RandomForest

```
# Step 0: Import Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import yfinance as yf


# Step 1: Data Collection
ticker_symbol = 'AAPL'
start_date = '2020-01-01'
end_date = '2021-03-1'


df = yf.download(ticker_symbol, start=start_date, end=end_date)


# Step 2: Exploratory Data Analysis (EDA)
# Data Overview
print("Data shape:", df.shape)

df.info()

print(df.describe())

# Missing Values
print(df.isnull().sum())

# Visualizing Stock Price Trends
plt.figure(figsize=(14, 7))
plt.subplot(2, 1, 1)
plt.plot(df['Close'], label='Close Price')
plt.title('Stock Close Price Over Time')
plt.legend()
plt.subplot(2, 1, 2)
plt.plot(df['Volume'], label='Volume', color='orange')
plt.title('Stock Volume Over Time')
plt.legend()
plt.tight_layout()
plt.show()

# Correlation Analysis
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Feature Correlation Matrix')
plt.show()

# Distribution of Features
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.histplot(df['Close'], bins=30, kde=True)
```

```
plt.title('Distribution of Close Price')
plt.subplot(1, 2, 2)
sns.histplot(df['Volume'], bins=30, color='orange', kde=True)
plt.title('Distribution of Volume')
plt.tight_layout()
plt.show()

# Step 3: Data Preprocessing
df['MA5'] = df['Close'].rolling(window=5).mean()
df['MA10'] = df['Close'].rolling(window=10).mean()
df = df.dropna() # Remove NaN values created by moving averages

# Splitting the dataset manually
train_df = df.loc['2020-01-01':'2021-01-01']
test_df = df.loc['2021-01-02':'2021-02-01']
validation_df = df.loc['2021-02-02':'2021-03-01']

# Prepare features and target for each set (assuming you're predicting Close prices using the same features)
X_train = train_df[['Open', 'High', 'Low', 'MA5', 'MA10']]
y_train = train_df['Close']

X_test = test_df[['Open', 'High', 'Low', 'MA5', 'MA10']]
y_test = test_df['Close']

X_validation = validation_df[['Open', 'High', 'Low', 'MA5', 'MA10']]
y_validation = validation_df['Close']


# Step 4: Model Training
random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest_model.fit(X_train, y_train)

# Predictions
y_pred = random_forest_model.predict(X_test)

print(y_pred)
print(y_test)

# Optional: Visualize Predictions
plt.figure(figsize=(10,6))
plt.plot(y_test.index, y_test, label='Actual Prices', color='blue')
plt.plot(y_test.index, y_pred, label='Predicted Prices', color='red')
plt.title('Random Forest Predicted vs Actual Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
# Step 5: Model Evaluation
# Additionally, perform predictions and evaluation on the validation set
y_pred_validation = random_forest_model.predict(X_validation)
rmse_validation = np.sqrt(mean_squared_error(y_validation, y_pred_validation))
r2_validation = r2_score(y_validation, y_pred_validation)
print(f"Validation RMSE: {rmse_validation}")
print(f"Validation R² score: {r2_validation}")

from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")

import matplotlib.pyplot as plt



# Plot the actual closing prices for the training set, the test set, and overlay the predictions
plt.figure(figsize=(14, 7))  # Set the size of the plot
plt.plot(train_df.index, train_df['Close'], label='Train', color='blue')  # Training data in blue
plt.plot(test_df.index, test_df['Close'], label='Test', color='orange')  # Test data in orange
plt.plot(test_df.index, y_pred, label='Predictions', color='red', linestyle='--')  # Predictions in red dashed
line

# Add title and labels
plt.title('Model: Random Forest Actual vs Predicted Prices')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')

# Show legend
plt.legend()

# Optionally, you can format the x-axis to show dates better
plt.gcf().autofmt_xdate()  # Rotate the dates for better spacing

# Show the plot
plt.show()
```

XGboost

```
## 1. Download Apple Stock Data using yfinance
import yfinance as yf

ticker_symbol = 'AAPL'
start_date = '2020-01-01'
end_date = '2021-03-1'

# Download the stock data
```

```
df = yf.download(ticker_symbol, start=start_date, end=end_date)

df.head()

import matplotlib.pyplot as plt
## Exploratory Data Analysis (EDA)
import seaborn as sns
# Plotting the distribution of features
plt.figure(figsize=(12, 6))
sns.distplot(df['Close'], bins=30)
plt.title('Distribution of Feature')
plt.show()

# Correlation matrix heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()


## 2. Perform Exploratory Data Analysis (EDA)
import matplotlib.pyplot as plt
import seaborn as sns

# Plot closing prices
plt.figure(figsize=(10, 6))
plt.plot(df['Close'])
plt.title('Apple Stock Closing Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()

# Display basic statistics
print(df.describe())

# Manual split based on the specific date ranges
train_df = df.loc['2020-01-01':'2021-01-01']
test_df = df.loc['2021-01-02':'2021-02-01']
# Validation set, in case you need it later
validation_df = df.loc['2021-02-02':'2021-03-01']

# training based on close price
X_train = train_df[['Close']]
y_train = train_df['Close'].shift(-1) # Predict next day's close for the training set
X_train = X_train[:-1] # Remove last row which has no next day's close for the training set
y_train = y_train.dropna() # Remove last NA value from the training set

X_test = test_df[['Close']]
```

```
y_test = test_df['Close'].shift(-1) # Predict next day's close for the test set
X_test = X_test[:-1] # Remove last row which has no next day's close for the test set
y_test = y_test.dropna() # Remove last NA value from the test set

# Ensure that 'df' is sorted by date if it's not already
df = df.sort_index()

# Define your date ranges for training and testing
train_start, train_end = '2020-01-01', '2021-01-01'
test_start, test_end = '2021-01-02', '2021-02-01'
validation_start, validation_end = '2021-02-02', '2021-03-01'

train_df = df.loc[train_start:train_end].copy()
test_df = df.loc[test_start:test_end].copy()
validation_df = df.loc[validation_start:validation_end].copy()

# Feature and target creation
# Using 'Close' price for today to predict 'Close' price for the next day
# Shift creates a new column where each value is the 'Close' price of the next day
train_df['Target'] = train_df['Close'].shift(-1)
test_df['Target'] = test_df['Close'].shift(-1)
validation_df['Target'] = validation_df['Close'].shift(-1)


# Drop the last row in each set because it does not have a next day 'Close' price
train_df = train_df.iloc[:-1]
test_df = test_df.iloc[:-1]
validation_df = validation_df.iloc[:-1]

# Define features and labels for training and testing
X_train, y_train = train_df[['Close']], train_df['Target']
X_test, y_test = test_df[['Close']], test_df['Target']
X_validation, y_validation = validation_df[['Close']], validation_df['Target']

## 4. Train an XGBoost Model
import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Train XGBoost model
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3, learning_rate
= 0.1,
                max_depth = 5, alpha = 10, n_estimators = 100)

xgb_model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

```
# Calculate metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"R^2: {r2}")

import matplotlib.pyplot as plt

y_pred = xgb_model.predict(X_test)

# Plot actual vs predicted prices
plt.figure(figsize=(10, 6))
plt.plot(y_test.index, y_test, label='Actual Prices', color='blue')
plt.plot(y_test.index, y_pred, label='Predicted Prices', color='red')
plt.title('XGBoost Predicted vs Actual Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

pip install xgboost

import matplotlib.pyplot as plt

# Plot the actual closing prices for the training set, the test set, and overlay the predictions
plt.figure(figsize=(14, 7))  # Set the size of the plot
plt.plot(train_df.index, train_df['Close'], label='Train', color='blue')  # Training data in blue
plt.plot(test_df.index, test_df['Close'], label='Test', color='orange')  # Test data in orange
plt.plot(test_df.index, y_pred, label='Predictions', color='red', linestyle='--')  # Predictions in red dashed
line

# Add title and labels
plt.title('Model: XGBoost Actual vs Predicted Prices')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')

# Show legend
plt.legend()

# Optionally, you can format the x-axis to show dates better
plt.gcf().autofmt_xdate()  # Rotate the
```

LSTM

```
import pandas as pd
```

```
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

from pandas_datareader.data import DataReader
import yfinance as yf
from pandas_datareader import data as pdr
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np
import yfinance as yf
from pandas_datareader import data as pdr
from sklearn import metrics

from datetime import datetime

import warnings

# Disable all warnings
warnings.filterwarnings('ignore')

stock_symbol = 'AAPL'
start_date = '2020-01-01'
end_date = '2021-03-1'

# Extracting the Stock
yf.pdr_override()
df = pdr.get_data_yahoo(stock_symbol, start_date, end_date)
df

df.isnull().sum()

# Splitting the dataset manually
train_df = df.loc['2020-01-01':'2021-01-01']
test_df = df.loc['2021-01-02':'2021-02-01']
validation_df = df.loc['2021-02-02':'2021-03-01']

# Plot the entire dataset and mark the train, val and test
plt.figure(figsize=(16,8))
plt.title('AAPL Stock Price History')
plt.plot(train_df['Close'], label='Train')
plt.plot(validation_df['Close'], label='Validation')
plt.plot(test_df['Close'], label='Test')
```

```
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.legend(loc='upper left')
plt.show()

train_len = train_df.shape[0]

dataset = df.filter(['Close'])

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

train_data = scaled_data[0:int(train_len), :]
# Split the data into x_train and y_train data sets
X_train = []
y_train = []

sequence_length = 30

for i in range(sequence_length, len(train_data)):
    X_train.append(train_data[i-sequence_length:i, 0])
    y_train.append(train_data[i, 0])

# Convert the x_train and y_train to numpy arrays
X_train, y_train = np.array(X_train), np.array(y_train)

# Reshape the data
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

test_data = scaled_data[train_len - sequence_length: , :]

# Create the data sets x_test and y_test
x_test = []
y_test = dataset[train_len:]['Close'].values

for i in range(sequence_length, len(test_data)):
    x_test.append(test_data[i-sequence_length:i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))


print(X_train.shape)
```

```
print(x_test.shape)

from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

from keras.losses import Huber

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (X_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dropout(0.02))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss=Huber())

# Train the model
model.fit(X_train, y_train, batch_size=1, epochs=3)

# Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)
print("MAE :",metrics.mean_absolute_error(y_test, predictions))
print("RMSE :",metrics.mean_squared_error(y_test, predictions, squared=False))
print("r2 Score :", metrics.r2_score(y_test, predictions))

# Plot the data
train = dataset[:train_len]
valid = dataset[train_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Test', 'Predictions'], loc='lower right')
plt.show()

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
```

```
from keras.layers import Dense, LSTM

# Assuming df is your DataFrame containing all columns including 'Open', 'High', 'Low', 'Close', 'Adj Close', and 'Volume'

sequence_length = 7
# Select relevant columns
data = df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]

# Convert dataframe to numpy array
dataset = data.values

# Get the number of rows to train the model on
training_data_len = train_len

# Separate the target variable (Close) from features
features = dataset[:, :5]  # Columns 'Open' to 'Adj Close'
target = dataset[:, 3]  # 'Close' column

# Scale the features
feature_scaler = MinMaxScaler(feature_range=(0, 1))
scaled_features = feature_scaler.fit_transform(features)

# Scale the target variable
target_scaler = MinMaxScaler(feature_range=(0, 1))
scaled_target = target_scaler.fit_transform(target.reshape(-1, 1))

# Create the training data set
train_data = scaled_features[0:int(training_data_len), :]
train_target = scaled_target[0:int(training_data_len), :]

# Split the data into x_train and y_train data sets
x_train = []
y_train = []
for i in range(sequence_length, len(train_data)):
    x_train.append(train_data[i-sequence_length:i, :])  # Include all columns for the past 60 days
    y_train.append(train_target[i, 0])  # 'Close' column as the target variable

# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 5))  # 5 is the number of features

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(x_train.shape[1], 5)))
model.add(LSTM(64, return_sequences=False))
```

```
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss=Huber())

# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=4)

# Create the testing data set
test_data = scaled_features[training_data_len - sequence_length:, :]

# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, 3]  # 'Close' column as the target variable
for i in range(sequence_length, len(test_data)):
    x_test.append(test_data[i-sequence_length:i, :])  # Include all columns for the past 60 days

# Convert the data to a numpy array
x_test = np.array(x_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 5))  # 5 is the number of features

# Get the models predicted price values
predictions = model.predict(x_test)
predictions = target_scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)
print("MAE :",metrics.mean_absolute_error(y_test, predictions))
print("RMSE :",metrics.mean_squared_error(y_test, predictions, squared=False))
print("r2 Score :", metrics.r2_score(y_test, predictions))

# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```