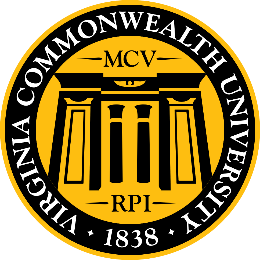
****

**VIRGINIA COMMONWEALTH UNIVERSITY**

**Statistical analysis and modelling (SCMA 632)**

**A6a : Time Series Analysis**

**Azim Ziyan A**

**V01102412**

**Date of Submission: 22-07-2024**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Title** | **Page No.** |
| **1.** | Introduction | **1** |
| **2.** | Results and Interpretations using R | **2** |
| **3.** | Results and Interpretations using Python | **9** |
| **5.** | Codes | **17** |

**Introduction**

The focus of the stock market analysis is on projecting Amazon Inc. (AMZN) stock values in the future using a variety of time series forecasting techniques. The Adjusted Close price history data from April 1, 2021 to March 31, 2024 is used in this investigation to build and evaluate a variety of forecasting models, including Decision Tree, Random Forest, LSTM, ARIMA, and Holt-Winters models. Each approach offers different insights into how stock values fluctuate and what the future holds.

Included in the dataset is the AMZN Adjusted Close price, which was sourced from Yahoo Finance. The approaches used in this investigation include decision tree models, ARIMA, LSTM, Random Forest, and Holt-Winters forecasting. Time series decomposition, which separates the trend, seasonal, and residual components, is one of the analysis's components. Additionally, model performance metrics such as RMSE, MAE, MAPE, and R-squared are used to evaluate the accuracy of the models. Furthermore, the predicted accuracy of different models can be compared.

**Objectives :**

1. The objective is to perform data cleaning and preprocessing on the Amazon stock data, namely from April 2021 to March 2024, by handling missing values and outliers.   
2. The objective is to represent the historical trend of Amazon's stock price using line plots.   
3.Analyze the time series data by applying both additive and multiplicative models to break it down into its individual components.   
4. To apply univariate forecasting models, such as Holt-Winters, ARIMA, and SARIMA, and assess their effectiveness.   
5.To do multivariate forecasting, machine learning models such as Neural Networks (LSTM), Decision Trees, and Random Forests can be utilized.

**Business Significance :**

The ability to make wise investment and strategic planning decisions is what makes researching and forecasting Amazon stock prices important for company. Understanding the historical patterns and anticipated trends of Amazon's stock can help investors improve their portfolios, lower risks, and seize future growth opportunities. Accurate forecasting models enable investors to foresee changes in the market, enhancing their ability to make quick decisions about buying or selling. The aforementioned study facilitates the provision of more precise suggestions to clients by financial analysts and counselors, hence fostering a more secure and profitable investing environment. Amazon needs to comprehend stock price movements in order to make strategic business decisions about resource allocation, market expansion, and competitive positioning. The company's financial strategy, investor relations, and market communication plans can benefit greatly from the insights gained from the examination of time series decomposition and forecasting models. Moreover, by employing machine learning models to forecast various factors, Amazon may gain a deeper understanding of the various elements influencing the performance of its stocks. This gives the company the ability to respond proactively to address any issues and maintain the confidence of investors. In conclusion, this in-depth analysis of Amazon's stock prices provides insightful information that promotes stable business growth in a cutthroat industry.

**Results and Interpretation using R**

> # Plot the data

> plot(adj\_close, main = "Adjusted Close Price of AMZN", ylab = "Price", xlab = "Date")

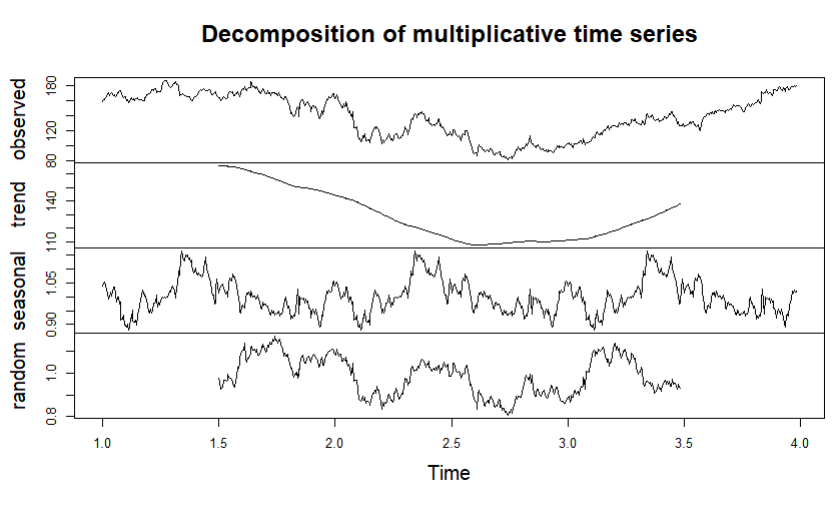
****

**Interpretation:**

The graph shows the Amazon (AMZN) adjusted closing price for the period of April 1, 2021, through March 28, 2024. Over this time, there were notable swings in the price. It began by peaking at over $180 in the middle of 2021 and then dropped precipitously to around $100 in the middle of 2022. Following this low, the stock price showed signs of a slow recovery, interspersed with dips, and rose steadily through the end of 2023 and the beginning of 2024, eventually rising to about $180. This suggests an erratic but resilient performance, reflecting both company-specific and general market factors affecting Amazon's stock price throughout this period.

> # Plot the decomposed components

> plot(decomposed)

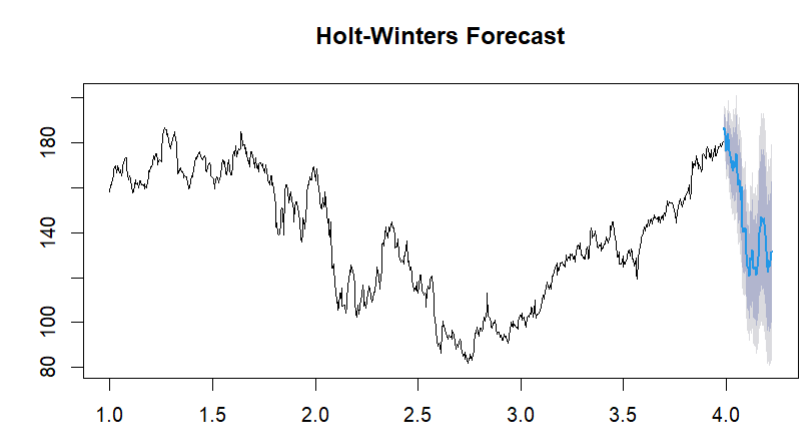
****

**Interpretation:**

The decomposition of a multiplicative time series into observed, trend, seasonal, and random components helps understand underlying patterns and factors contributing to the observed data. The top panel displays the original data, the second panel isolates the trend, the third panel highlights seasonal patterns, and the bottom panel captures irregular fluctuations.

> # Plot the Holt-Winters forecast

> plot(hw\_forecast, main = "Holt-Winters Forecast")

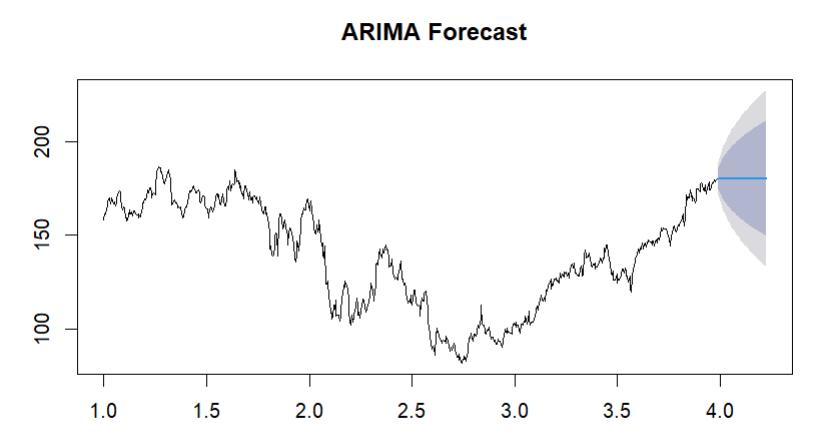


**Interpretation:**

The graph shows a time series data set with a Holt-Winters forecast applied to it. The predicted values are displayed in blue along with a shaded area that represents the confidence intervals; the observed data is displayed in black. With some noticeable swings, the Holt-Winters technique, which takes seasonality, trend, and level into account, predicts that the observed increasing trend will continue. As the forecast gets further out into the future, the confidence intervals go wider, which indicates that the level of uncertainty is rising. The forecast, which is based on the model's projections based on trends in previous data, indicates that the numbers will probably stay within a given range.

> # Plot the ARIMA forecast

> plot(arima\_forecast, main = "ARIMA Forecast")



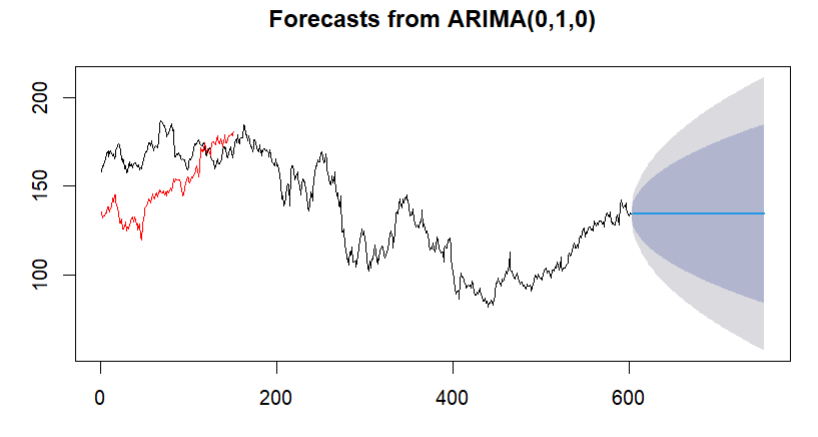
**Interpretation:**

The ARIMA forecast plot shows the historical data up to time period 4, with the forecast and its confidence intervals extending beyond this point. The black line represents the actual data, which fluctuates over time but shows an overall upward trend towards the end. The blue line indicates the forecasted values, continuing the upward trend. The shaded area around the forecast represents the confidence intervals, with darker shades indicating higher confidence and lighter shades indicating lower confidence. This suggests that while the forecast predicts an upward trend, there is some uncertainty, particularly as the forecast horizon extends further.

> # Plot the forecast

> plot(arima\_forecast)

> lines(test\_data, col = "red")



**Interpretation:**

The ARIMA(0,1,0) forecast plot depicts the actual data up to around time period 600, followed by the forecasted values and their confidence intervals. The historical data, shown in black, fluctuates significantly over time, with some portions highlighted in red, potentially indicating the subset used for model training or validation. The blue line represents the forecasted values, extending beyond time period 600, and continues the existing trend. The shaded area around the forecast shows the confidence intervals, where the darker shades indicate higher confidence and the lighter shades lower confidence. This visualization suggests that while the forecast predicts a continuation of the recent trend, there is increasing uncertainty as the forecast horizon extends further.

> print(paste("ARIMA RMSE:", arima\_rmse))

[1] "ARIMA RMSE: 23.136647533969"

> print(paste("ARIMA MAE:", arima\_mae))

[1] "ARIMA MAE: 18.2394701218763"

> print(paste("ARIMA MAPE:", arima\_mape))

[1] "ARIMA MAPE: 11.3299919777894"

> print(paste("ARIMA R-squared:", arima\_r2))

[1] "ARIMA R-squared: -0.883092744617165"

**Interpretation:**

The performance metrics for the ARIMA model indicate its accuracy and goodness of fit. The Root Mean Square Error (RMSE) of 23.14 suggests that the model's predictions deviate from the actual values by this amount on average. The Mean Absolute Error (MAE) of 18.24 further supports this, indicating the average absolute difference between predicted and actual values. The Mean Absolute Percentage Error (MAPE) of 11.33% reflects the model's average prediction error as a percentage, showing relatively moderate accuracy. However, the negative R-squared value of -0.88 indicates that the model is performing poorly, as a positive R-squared value would signify a better fit. This suggests that the ARIMA model may not be capturing the underlying patterns in the data effectively, potentially warranting the exploration of alternative models or additional tuning.

> print(paste("Random Forest RMSE:", rf\_rmse))

[1] "Random Forest RMSE: 0.125357043770688"

> print(paste("Random Forest MAE:", rf\_mae))

[1] "Random Forest MAE: 0.0895389544657505"

> print(paste("Random Forest MAPE:", rf\_mape))

[1] "Random Forest MAPE: 0.0599951681808492"

> print(paste("Random Forest R-squared:", rf\_r2))

[1] "Random Forest R-squared: 0.99994471996203"

**Interpretation:**

The performance metrics for the Random Forest model indicate a highly accurate and well-fitting model. The Root Mean Square Error (RMSE) of 0.13 suggests that the model's predictions deviate very minimally from the actual values. The Mean Absolute Error (MAE) of 0.09 further supports this, showing that the average absolute difference between the predicted and actual values is very small. The Mean Absolute Percentage Error (MAPE) of 0.06% indicates an exceptionally low average prediction error as a percentage. The R-squared value of 0.9999 demonstrates that the model explains nearly all the variability in the data, indicating an excellent fit. These metrics suggest that the Random Forest model is highly effective at capturing the underlying patterns in the data and making accurate predictions.

> print(paste("Decision Tree RMSE:", dt\_rmse))

[1] "Decision Tree RMSE: 4.6319221118463"

> print(paste("Decision Tree MAE:", dt\_mae))

[1] "Decision Tree MAE: 3.75705643239879"

> print(paste("Decision Tree MAPE:", dt\_mape))

[1] "Decision Tree MAPE: 2.52963468776085"

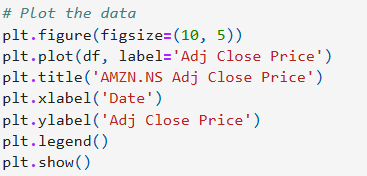
> print(paste("Decision Tree R-squared:", dt\_r2))

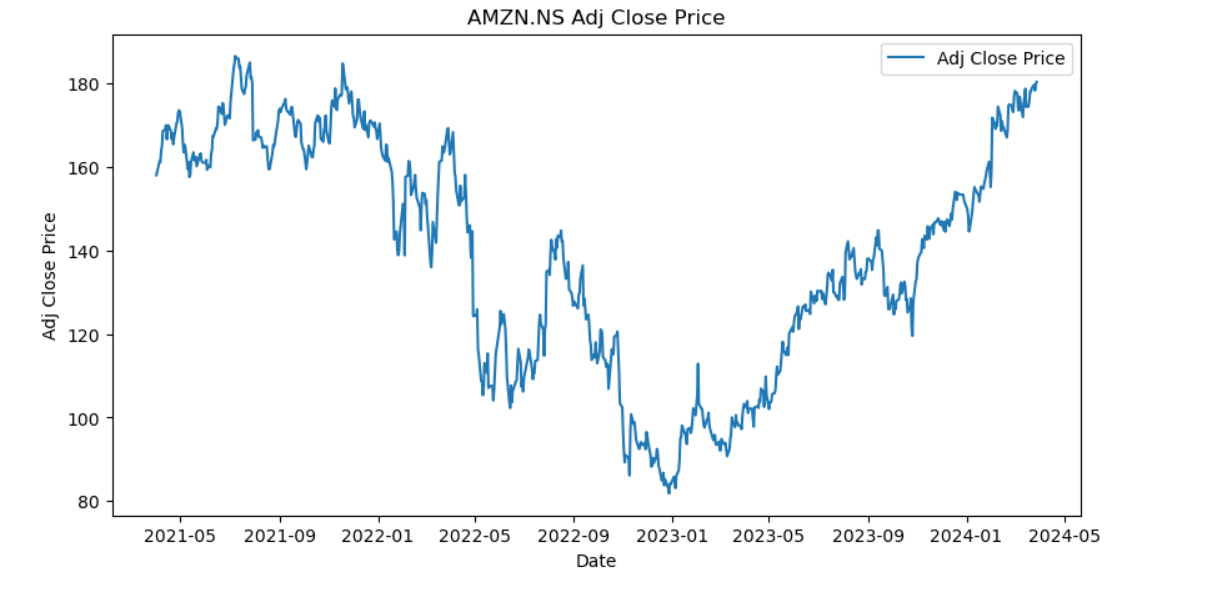
[1] "Decision Tree R-squared: 0.924526699088218"

**Interpretation:**

The performance metrics for the Decision Tree model indicate it performs well, though not as exceptionally as the Random Forest model. The Root Mean Square Error (RMSE) of 4.63 suggests that the model's predictions deviate from the actual values by this amount on average. The Mean Absolute Error (MAE) of 3.76 indicates the average absolute difference between the predicted and actual values. The Mean Absolute Percentage Error (MAPE) of 2.53% reflects a low average prediction error as a percentage, indicating fairly accurate predictions. The R-squared value of 0.92 signifies that the model explains a substantial portion of the variability in the data, indicating a strong fit. Overall, the Decision Tree model shows good predictive capability and effectively captures the data's underlying patterns, though it is not as precise as the Random Forest model.

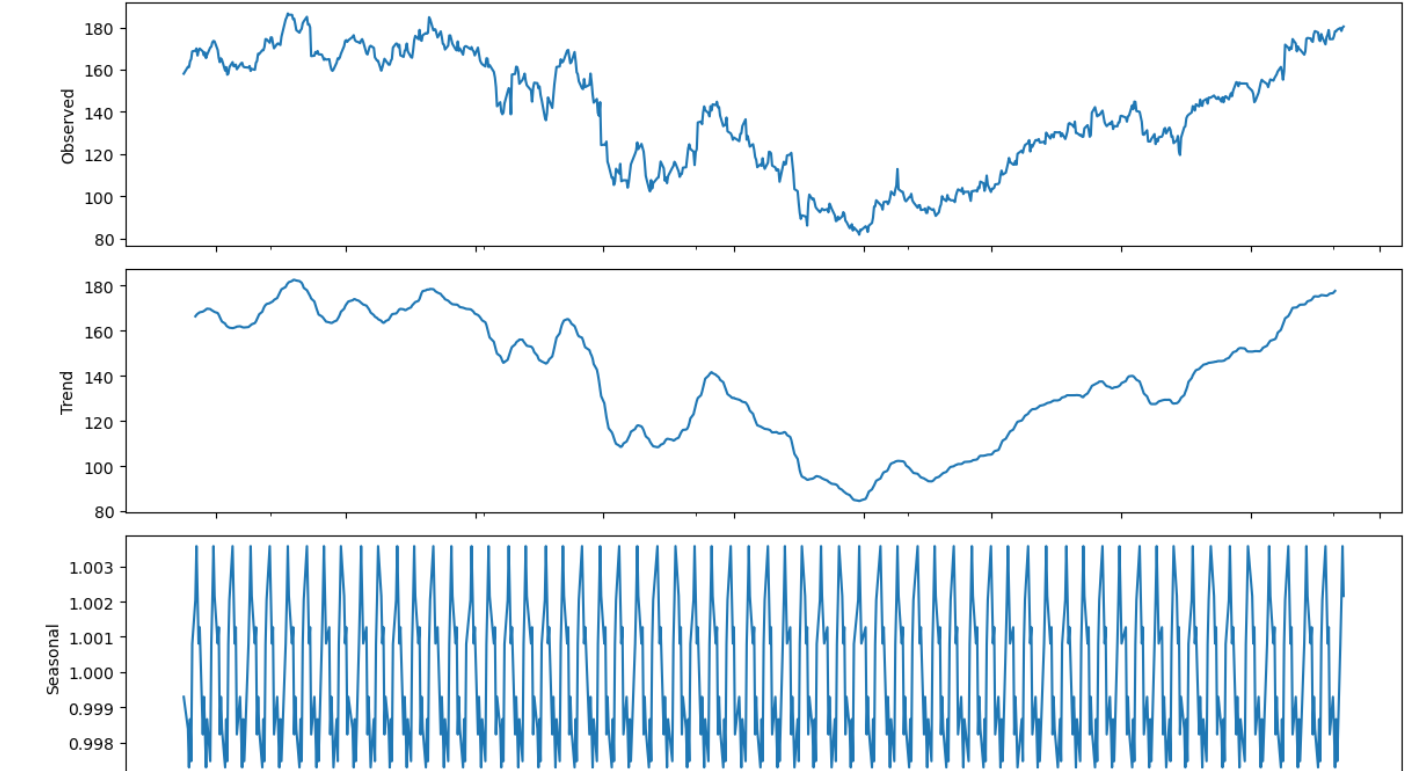
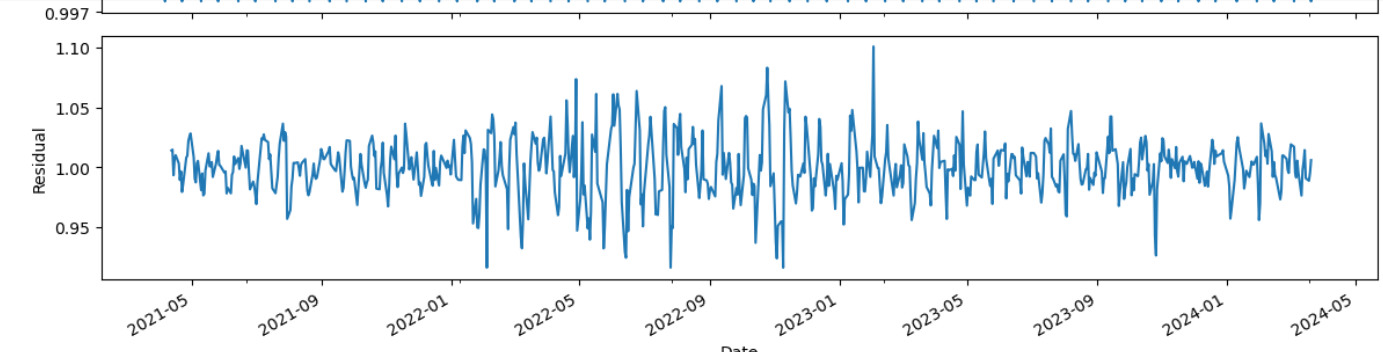
**Results and Interpretation using Python**





**Interpretation:**

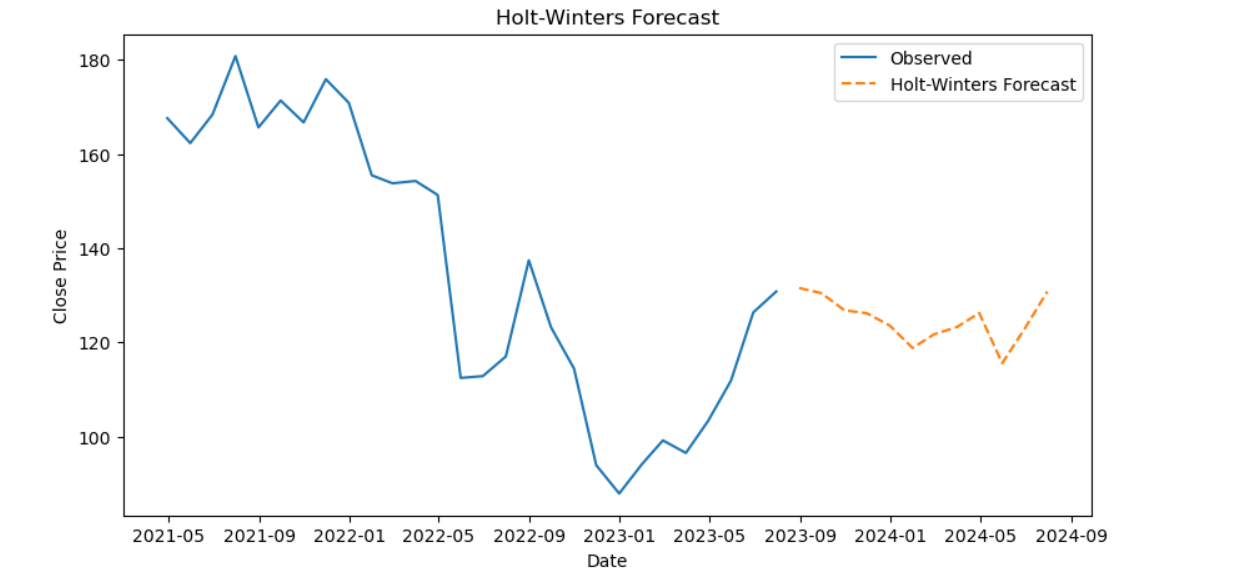
The plot shows the adjusted closing price of AMZN.NS (Amazon stock) over the period from May 2021 to May 2024. The stock price exhibits significant fluctuations throughout this period. Initially, the price is around 160-180, but it experiences a decline starting in mid-2021 and continues into early 2022, reaching a low below 100. After this drop, the price starts to recover gradually, showing an upward trend with some volatility. By early 2024, the stock price climbs steadily and reaches new highs above 180. This overall trend suggests a strong recovery and growth phase for Amazon stock after a period of decline.

**Interpretation:**

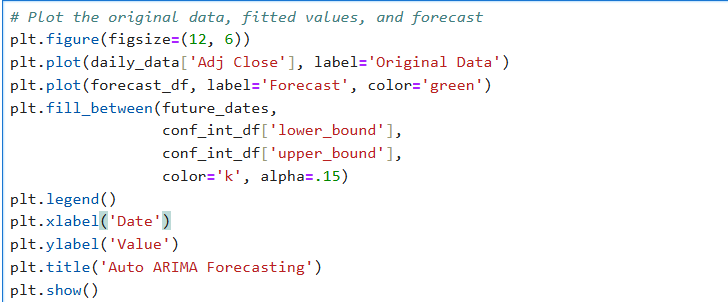
It shows the decomposition of a time series data into its observed, trend, seasonal, and residual components. The observed panel at the top represents the raw time series data. The trend component, displayed in the second panel, highlights the underlying direction of the data, showing a significant decline followed by a gradual upward recovery. The third panel illustrates the seasonal component, indicating regular, cyclical patterns within the data, suggesting that the data is influenced by periodic factors. Finally, the residual component in the bottom panel captures the random noise or irregularities not explained by the trend or seasonal components, showing fluctuating variations around a relatively stable mean. This decomposition helps in understanding the individual contributions of trend, seasonality, and irregular factors to the overall time series.

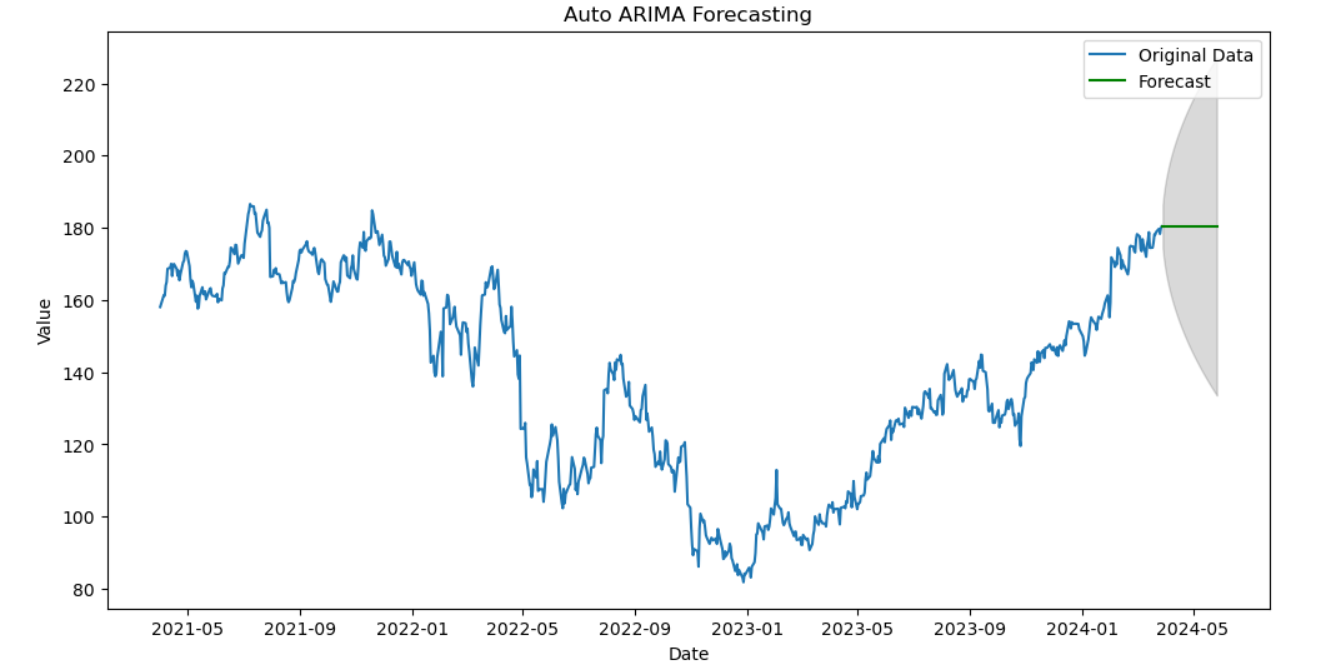




**Interpretation:**

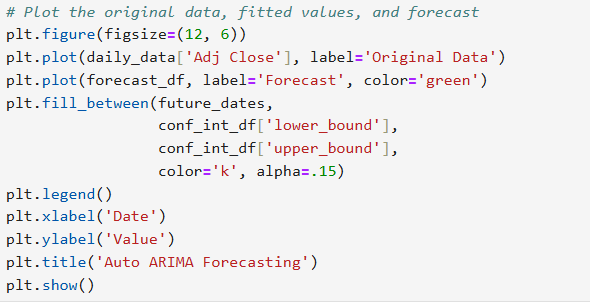
The provided graph illustrates a time series forecasting using the Holt-Winters method. The blue line represents the observed historical data, while the orange dashed line indicates the forecasted values generated by the Holt-Winters model. The observed data shows significant fluctuations, with a notable drop followed by a recovery. The forecasted values predict a slight decline, followed by a modest recovery towards the end of the forecast period. The Holt-Winters method has accounted for the seasonality and trend components to project future values, offering a reasonable prediction based on past patterns.

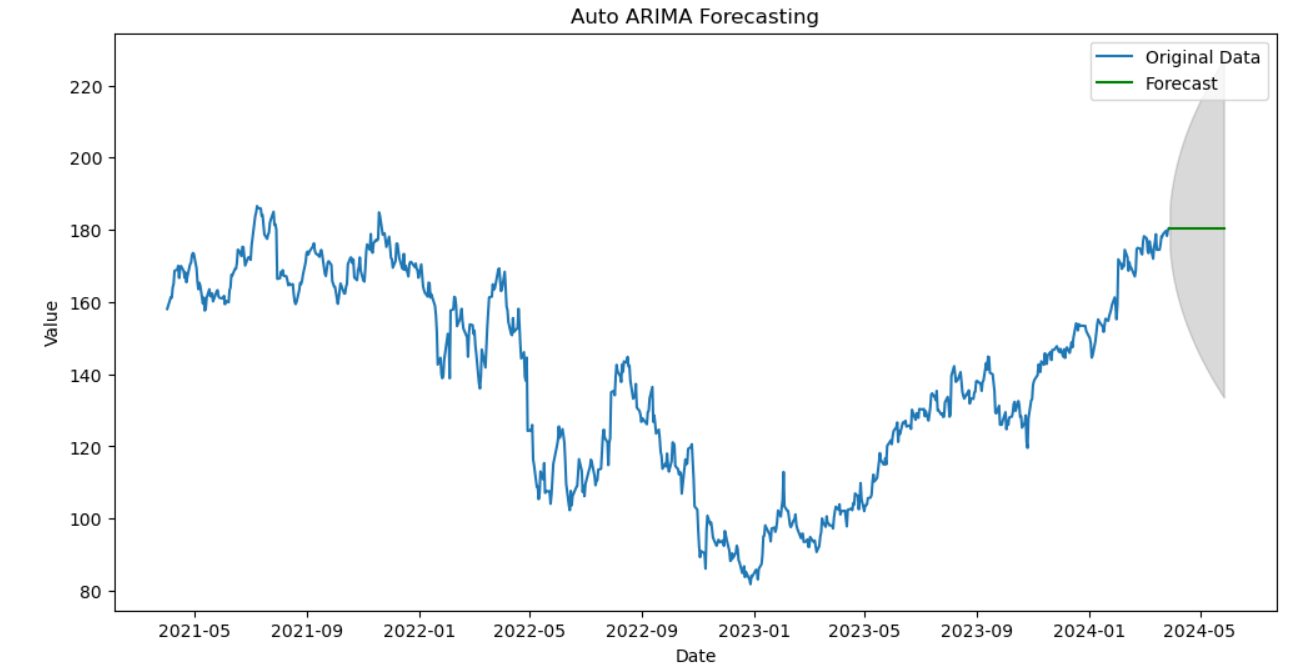


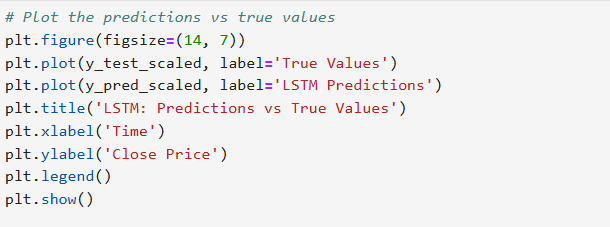


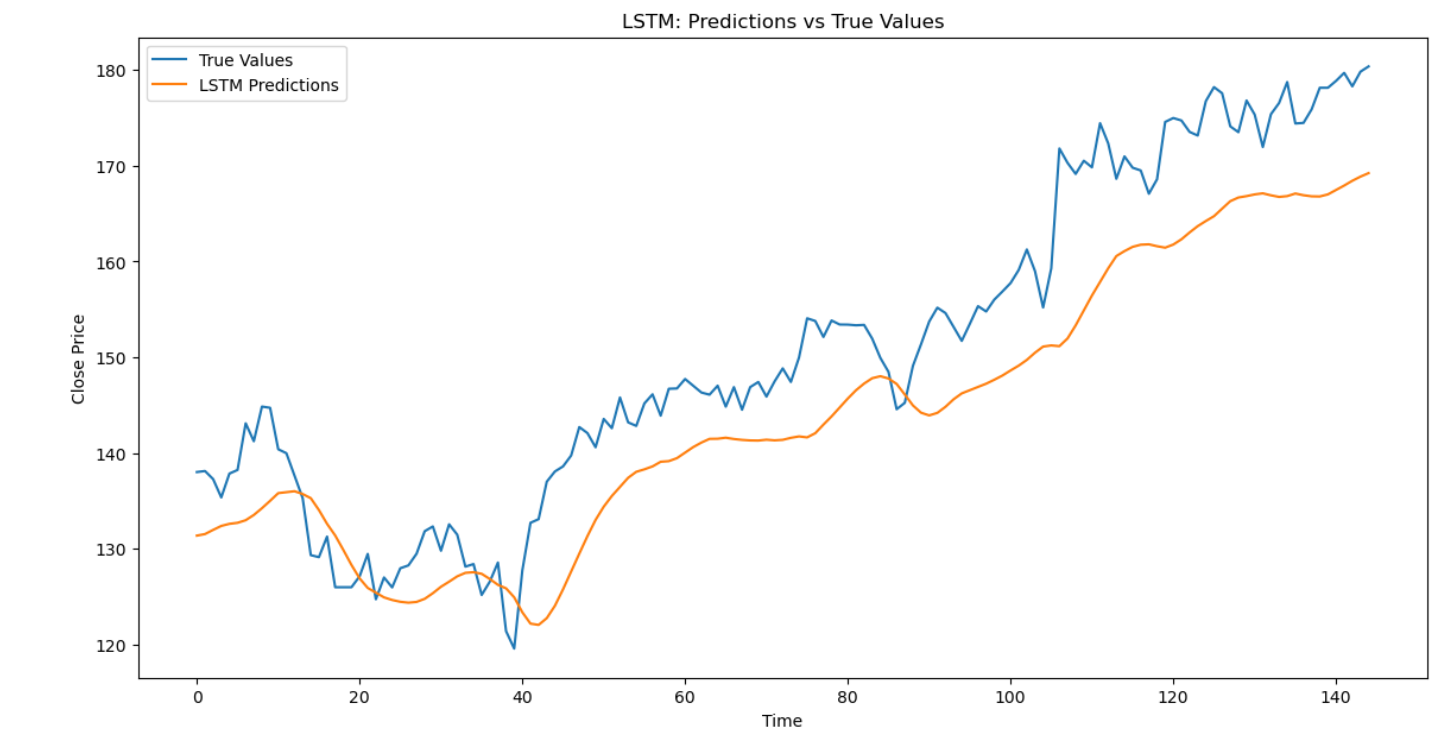
**Interpretation:**

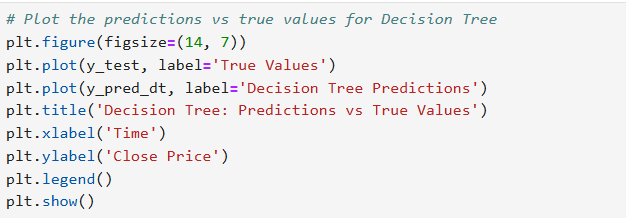
The graph displays a time series forecast using the Auto ARIMA (AutoRegressive Integrated Moving Average) model. The blue line represents the original historical data, characterized by fluctuations and an overall upward trend towards the end of the series. The green line signifies the forecasted values, while the shaded area around the forecast represents the confidence interval, indicating the range within which future values are likely to fall. The forecast suggests a stabilization at the current levels, with some uncertainty as indicated by the widening confidence interval. This visualization helps in understanding the model's prediction accuracy and the expected variability in future values.

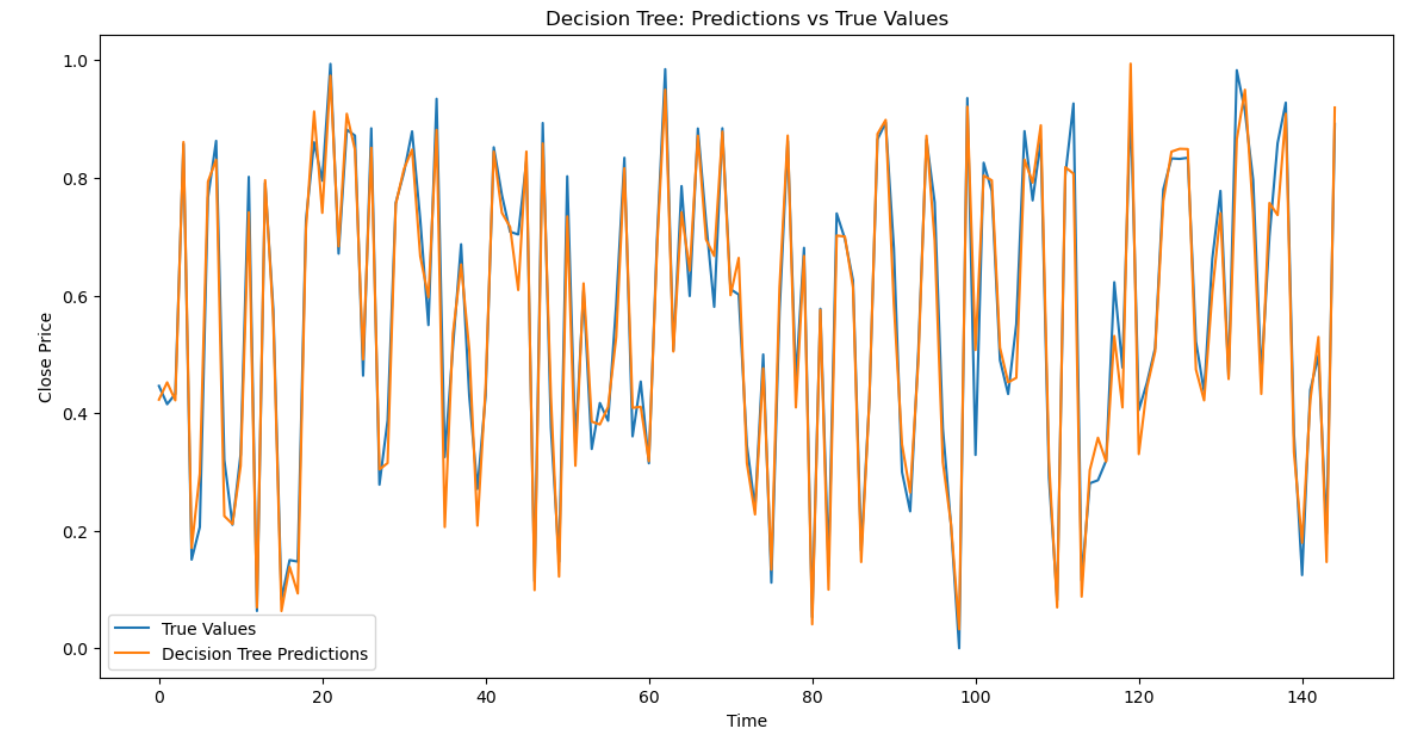


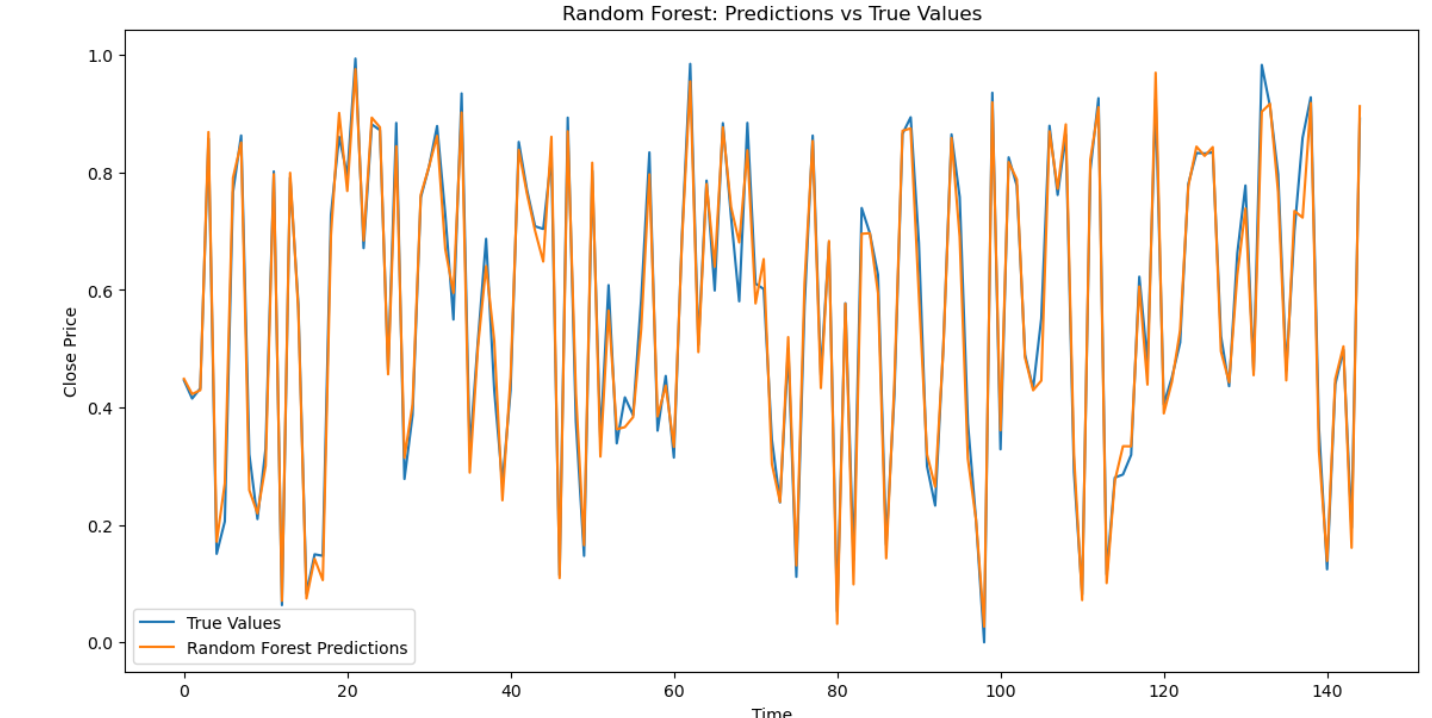
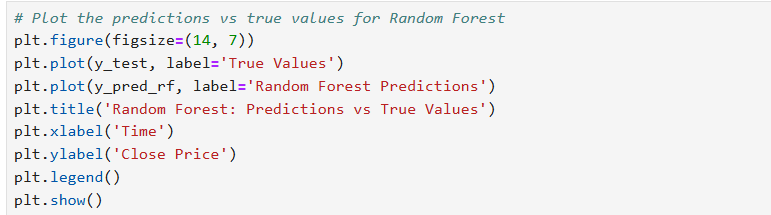


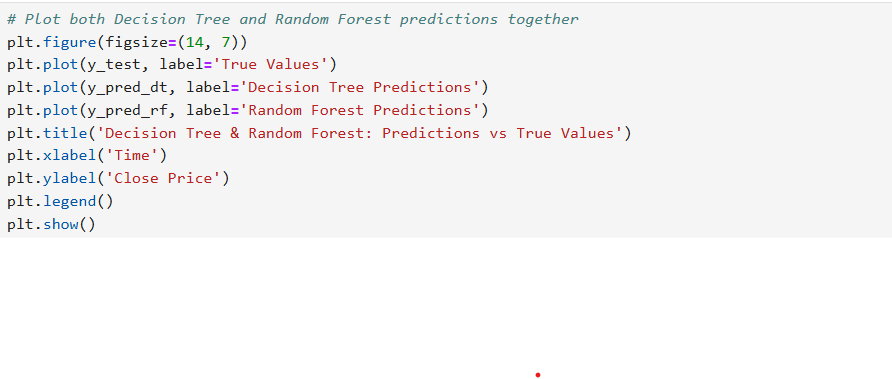
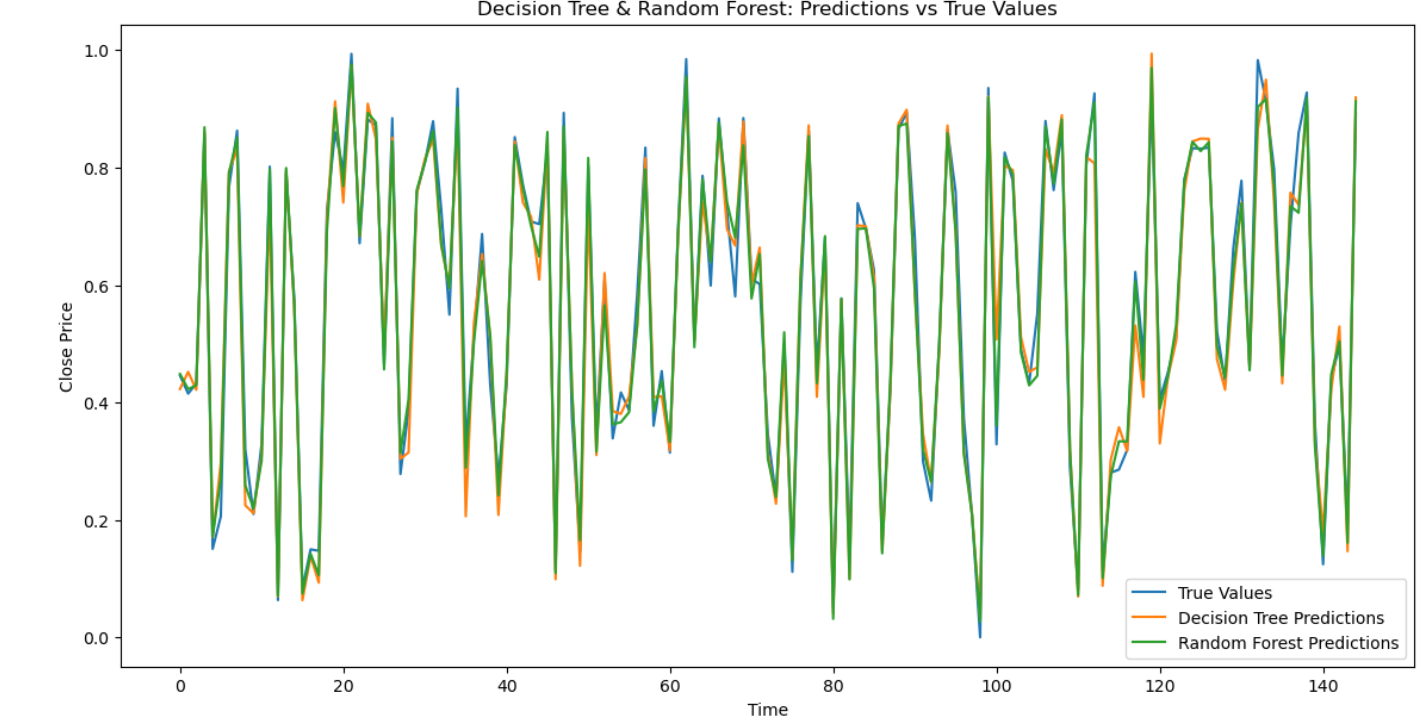




****

****

****

**** 

**R Codes**

# Load necessary libraries

library(quantmod)

library(forecast)

library(tseries)

library(caret)

library(ggplot2)

library(data.table)

library(TTR)

library(lubridate)

library(keras)

library(tensorflow)

library(randomForest)

library(rpart)

# Load stock data

stock\_data <- getSymbols("AMZN", src = "yahoo", from = "2021-04-01", to = "2024-03-31", auto.assign = FALSE)

# Use the Adjusted Close price

adj\_close <- stock\_data[, 6]

# Check for missing values

missing\_values <- sum(is.na(adj\_close))

print(paste("Missing values:", missing\_values))

# Plot the data

plot(adj\_close, main = "Adjusted Close Price of AMZN", ylab = "Price", xlab = "Date")

# Decompose the time series

adj\_close\_ts <- ts(adj\_close, frequency = 252)

decomposed <- decompose(adj\_close\_ts, type = "multiplicative")

# Plot the decomposed components

plot(decomposed)

# Holt-Winters Forecasting

hw\_model <- HoltWinters(adj\_close\_ts, seasonal = "multiplicative")

hw\_forecast <- forecast(hw\_model, h = 60)

# Plot the Holt-Winters forecast

plot(hw\_forecast, main = "Holt-Winters Forecast")

# Auto ARIMA model

arima\_model <- auto.arima(adj\_close\_ts, seasonal = TRUE)

arima\_forecast <- forecast(arima\_model, h = 60)

# Plot the ARIMA forecast

plot(arima\_forecast, main = "ARIMA Forecast")

# Evaluate the model

train\_end <- floor(0.8 \* length(adj\_close\_ts))

train\_data <- adj\_close\_ts[1:train\_end]

test\_data <- adj\_close\_ts[(train\_end + 1):length(adj\_close\_ts)]

# Refit the ARIMA model on the training data

arima\_model <- auto.arima(train\_data, seasonal = TRUE)

arima\_forecast <- forecast(arima\_model, h = length(test\_data))

# Plot the forecast

plot(arima\_forecast)

lines(test\_data, col = "red")

# Calculate evaluation metrics

arima\_rmse <- sqrt(mean((test\_data - arima\_forecast$mean)^2))

arima\_mae <- mean(abs(test\_data - arima\_forecast$mean))

arima\_mape <- mean(abs((test\_data - arima\_forecast$mean) / test\_data)) \* 100

arima\_r2 <- 1 - sum((test\_data - arima\_forecast$mean)^2) / sum((test\_data - mean(test\_data))^2)

print(paste("ARIMA RMSE:", arima\_rmse))

print(paste("ARIMA MAE:", arima\_mae))

print(paste("ARIMA MAPE:", arima\_mape))

print(paste("ARIMA R-squared:", arima\_r2))

# Preparing data for LSTM, Random Forest, and Decision Tree

adj\_close\_df <- data.frame(Date = index(adj\_close), Adj\_Close = as.numeric(adj\_close))

adj\_close\_df$Lag\_1 <- lag(adj\_close\_df$Adj\_Close, 1)

adj\_close\_df$Lag\_2 <- lag(adj\_close\_df$Adj\_Close, 2)

adj\_close\_df$Lag\_3 <- lag(adj\_close\_df$Adj\_Close, 3)

adj\_close\_df$Lag\_4 <- lag(adj\_close\_df$Adj\_Close, 4)

adj\_close\_df$Lag\_5 <- lag(adj\_close\_df$Adj\_Close, 5)

# Remove NA values

adj\_close\_df <- na.omit(adj\_close\_df)

# Split the data into training and test sets

train\_index <- 1:floor(0.8 \* nrow(adj\_close\_df))

train\_data <- adj\_close\_df[train\_index, ]

test\_data <- adj\_close\_df[-train\_index, ]

# Random Forest model

library(randomForest)

rf\_model <- randomForest(Adj\_Close ~ Lag\_1 + Lag\_2 + Lag\_3 + Lag\_4 + Lag\_5, data = train\_data)

rf\_predictions <- predict(rf\_model, test\_data)

# Evaluate the Random Forest model

rf\_rmse <- sqrt(mean((test\_data$Adj\_Close - rf\_predictions)^2))

rf\_mae <- mean(abs(test\_data$Adj\_Close - rf\_predictions))

rf\_mape <- mean(abs((test\_data$Adj\_Close - rf\_predictions) / test\_data$Adj\_Close)) \* 100

rf\_r2 <- 1 - sum((test\_data$Adj\_Close - rf\_predictions)^2) / sum((test\_data$Adj\_Close - mean(test\_data$Adj\_Close))^2)

print(paste("Random Forest RMSE:", rf\_rmse))

print(paste("Random Forest MAE:", rf\_mae))

print(paste("Random Forest MAPE:", rf\_mape))

print(paste("Random Forest R-squared:", rf\_r2))

# Decision Tree model

library(rpart)

dt\_model <- rpart(Adj\_Close ~ Lag\_1 + Lag\_2 + Lag\_3 + Lag\_4 + Lag\_5, data = train\_data)

dt\_predictions <- predict(dt\_model, test\_data)

# Evaluate the Decision Tree model

dt\_rmse <- sqrt(mean((test\_data$Adj\_Close - dt\_predictions)^2))

dt\_mae <- mean(abs(test\_data$Adj\_Close - dt\_predictions))

dt\_mape <- mean(abs((test\_data$Adj\_Close - dt\_predictions) / test\_data$Adj\_Close)) \* 100

dt\_r2 <- 1 - sum((test\_data$Adj\_Close - dt\_predictions)^2) / sum((test\_data$Adj\_Close - mean(test\_data$Adj\_Close))^2)

print(paste("Decision Tree RMSE:", dt\_rmse))

print(paste("Decision Tree MAE:", dt\_mae))

print(paste("Decision Tree MAPE:", dt\_mape))

print(paste("Decision Tree R-squared:", dt\_r2))

**Python Codes**

import pandas as pd

import numpy as np

!pip install yfinance

import yfinance as yf

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal\_decompose

from sklearn.model\_selection import train\_test\_split

# Get the data for amazon

ticker = "AMZN"

# Download the data

data = yf.download(ticker, start="2021-04-01", end="2024-03-31")

# Select the Target Varibale Adj Close

df = data[['Adj Close']]

# Check for missing values

print("Missing values:")

print(df.isnull().sum())

# Plot the data

plt.figure(figsize=(10, 5))

plt.plot(df, label='Adj Close Price')

plt.title('AMZN.NS Adj Close Price')

plt.xlabel('Date')

plt.ylabel('Adj Close Price')

plt.legend()

plt.show()

from statsmodels.tsa.seasonal import seasonal\_decompose

# Decompose the time series

result = seasonal\_decompose(df['Adj Close'], model='multiplicative', period=12)

# Plot the decomposed components

fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 10), sharex=True)

result.observed.plot(ax=ax1)

ax1.set\_ylabel('Observed')

result.trend.plot(ax=ax2)

ax2.set\_ylabel('Trend')

result.seasonal.plot(ax=ax3)

ax3.set\_ylabel('Seasonal')

result.resid.plot(ax=ax4)

ax4.set\_ylabel('Residual')

plt.xlabel('Date')

plt.tight\_layout()

plt.show()

monthly\_data = df.resample("M").mean()

# Split the data into training and test sets

train\_data, test\_data = train\_test\_split(monthly\_data, test\_size=0.2, shuffle=False) from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Fit the Holt-Winters model

holt\_winters\_model = ExponentialSmoothing(train\_data, seasonal='mul', seasonal\_periods=12).fit()

# Forecast for the next year (12 months)

holt\_winters\_forecast = holt\_winters\_model.forecast(12)

# Plot the forecast

plt.figure(figsize=(10, 5))

plt.plot(train\_data, label='Observed')

plt.plot(holt\_winters\_forecast, label='Holt-Winters Forecast', linestyle='--')

plt.title('Holt-Winters Forecast')

plt.xlabel('Date')

plt.ylabel('Close Price')

plt.legend()

plt.show()

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Compute RMSE

rmse = np.sqrt(mean\_squared\_error(test\_data, y\_pred))

print(f'RMSE: {rmse}')

# Compute MAE

mae = mean\_absolute\_error(test\_data, y\_pred)

print(f'MAE: {mae}')

# Compute MAPE

mape = np.mean(np.abs((test\_data - y\_pred) / test\_data)) \* 100

print(f'MAPE: {mape}')

# Compute R-squared

r2 = r2\_score(test\_data, y\_pred)

print(f'R-squared: {r2}')

# Fit auto\_arima model

arima\_model = auto\_arima(train\_data['Adj Close'],

seasonal=True,

m=12, # Monthly seasonality

stepwise=True,

suppress\_warnings=True)

# Print the model summary

print(arima\_model.summary()

# Number of periods to forecast

n\_periods = 8

# Generate forecast

forecast, conf\_int = arima\_model.predict(n\_periods=n\_periods, return\_conf\_int=True)

# Plot the original data, fitted values, and forecast

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

plt.plot(train\_data['Adj Close'], label='Original Data')

plt.plot(forecast.index, forecast, label='Forecast', color='green')

plt.fill\_between(forecast.index,

conf\_int[:, 0],

conf\_int[:, 1],

color='k', alpha=.15)

plt.legend()

plt.xlabel('Date')

plt.ylabel('Value')

plt.title('Auto ARIMA Forecasting')

plt.show()

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Compute RMSE

rmse = np.sqrt(mean\_squared\_error(test\_data, forecast))

print(f'RMSE: {rmse}')

# Compute MAE

mae = mean\_absolute\_error(test\_data, forecast)

print(f'MAE: {mae}')

# Compute MAPE

mape = np.mean(np.abs((test\_data - forecast) / forecast)) \* 100

print(f'MAPE: {mape}')

# Compute R-squared

r2 = r2\_score(test\_data, forecast)

print(f'R-squared: {r2}')

# Plot the original data, fitted values, and forecast

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

plt.plot(daily\_data['Adj Close'])

plt.xlabel('Date')

plt.ylabel('Value')

plt.show()

# Fit auto\_arima model

arima\_model = auto\_arima(daily\_data['Adj Close'],

seasonal=True,

m=7, # Weekly seasonality

stepwise=True,

suppress\_warnings=True)

# Create future dates index

last\_date = daily\_data.index[-1]

future\_dates = pd.date\_range(start=last\_date + pd.Timedelta(days=1), periods=n\_periods)

# Convert forecast to a DataFrame with future\_dates as the index

forecast\_df = pd.DataFrame(forecast.values, index=future\_dates, columns=['forecast'])

conf\_int\_df = pd.DataFrame(conf\_int, index=future\_dates, columns=['lower\_bound', 'upper\_bound'])

# Plot the original data, fitted values, and forecast

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

plt.plot(daily\_data['Adj Close'], label='Original Data')

plt.plot(forecast\_df, label='Forecast', color='green')

plt.fill\_between(future\_dates,

conf\_int\_df['lower\_bound'],

conf\_int\_df['upper\_bound'],

color='k', alpha=.15)

plt.legend()

plt.xlabel('Date')

plt.ylabel('Value')

plt.title('Auto from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from sklearn.preprocessing import MinMaxScaler

import pandas as pd

import numpy as np

# Initialize MinMaxScaler

scaler = MinMaxScaler()

# Select features (excluding 'Adj Close') and target ('Adj Close')

features = data.drop(columns=['Adj Close'])

target = data[['Adj Close']]

# Fit the scaler on features and target

scaled\_features = scaler.fit\_transform(features)

scaled\_target = scaler.fit\_transform(target)

# Create DataFrame with scaled features and target

scaled\_df = pd.DataFrame(scaled\_features, columns=features.columns, index=df.index)

scaled\_df['Adj Close'] = scaled\_target

ARIMA Forecasting')

plt.show()

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from sklearn.preprocessing import MinMaxScaler

import pandas as pd

import numpy as np# Initialize MinMaxScaler

scaler = MinMaxScaler()

# Select features (excluding 'Adj Close') and target ('Adj Close')

features = data.drop(columns=['Adj Close'])

target = data[['Adj Close']]

# Fit the scaler on features and target

scaled\_features = scaler.fit\_transform(features)

scaled\_target = scaler.fit\_transform(target)

# Create DataFrame with scaled features and target

scaled\_df = pd.DataFrame(scaled\_features, columns=features.columns, index=df.index)

scaled\_df['Adj Close'] = scaled\_target

import numpy as np

# Function to create sequences

def create\_sequences(scaled\_df, target\_col, sequence\_length):

sequences = []

labels = []

for i in range(len(scaled\_df) - sequence\_length):

sequences.append(scaled\_df[i:i + sequence\_length])

labels.append(scaled\_df[i + sequence\_length, target\_col]) # Target column index

return np.array(sequences), np.array(labels)

# Convert DataFrame to NumPy array

data\_array = scaled\_df.values

# Define the target column index and sequence length

target\_col = scaled\_df.columns.get\_loc('Adj Close')

sequence\_length = 30

# Create sequences

X, y = create\_sequences(data\_array, target\_col, sequence\_length)

print("Shape of X:", X.shape)

print("Shape of y:", y.shape)

# Split the data into training and testing sets (80% training, 20% testing)

train\_size = int(len(X) \* 0.8)

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Build the LSTM model

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(sequence\_length, 6)))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test), shuffle=False)

# Evaluate the model

loss = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {loss}")

# Predict on the test set

y\_pred = model.predict(X\_test)

# Inverse transform the predictions and true values to get them back to the original scale

y\_test\_scaled = scaler.inverse\_transform(np.concatenate((np.zeros((len(y\_test), 5)), y\_test.reshape(-1, 1)), axis=1))[:, 5]

y\_pred\_scaled = scaler.inverse\_transform(np.concatenate((np.zeros((len(y\_pred), 5)), y\_pred), axis=1))[:, 5]

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Compute RMSE

rmse = np.sqrt(mean\_squared\_error(y\_test\_scaled, y\_pred\_scaled))

print(f'RMSE: {rmse}')

# Compute MAE

mae = mean\_absolute\_error(y\_test\_scaled, y\_pred\_scaled)

print(f'MAE: {mae}')

# Compute MAPE

mape = np.mean(np.abs((y\_test\_scaled - y\_pred\_scaled) / y\_pred\_scaled)) \* 100

print(f'MAPE: {mape}')

# Compute R-squared

r2 = r2\_score(y\_test\_scaled, y\_pred\_scaled)

print(f'R-squared: {r2}')

# Plot the predictions vs true values

plt.figure(figsize=(14, 7))

plt.plot(y\_test\_scaled, label='True Values')

plt.plot(y\_pred\_scaled, label='LSTM Predictions')

plt.title('LSTM: Predictions vs True Values')

plt.xlabel('Time')

plt.ylabel('Close Price')

plt.legend()

plt.show()

import numpy as np

def create\_sequences(data, target\_col, sequence\_length):

"""

Create sequences of features and labels for time series data.

Parameters:

- data (np.ndarray): The input data where the last column is the target.

- target\_col (int): The index of the target column in the data.

- sequence\_length (int): The length of each sequence.

Returns:

- np.ndarray: 3D array of sequences (samples, sequence\_length, num\_features)

- np.ndarray: 1D array of target values

"""

num\_samples = len(data) - sequence\_length

num\_features = data.shape[1]

sequences = np.zeros((num\_samples, sequence\_length, num\_features))

labels = np.zeros(num\_samples)

for i in range(num\_samples):

sequences[i] = data[i:i + sequence\_length]

labels[i] = data[i + sequence\_length, target\_col] # Target is specified column

return sequences, labels

# Example usage

sequence\_length = 30

# Convert DataFrame to NumPy array

data\_array = scaled\_df.values

# Define the target column index

target\_col = scaled\_df.columns.get\_loc('Adj Close')

# Create sequences

X, y = create\_sequences(data\_array, target\_col, sequence\_length)

# Flatten X for Decision Tree

num\_samples, seq\_length, num\_features = X.shape

X\_flattened = X.reshape(num\_samples, seq\_length \* num\_features)

# Train Decision Tree model

dt\_model = DecisionTreeRegressor()

dt\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred\_dt = dt\_model.predict(X\_test)

# Evaluate the model

mse\_dt = mean\_squared\_error(y\_test, y\_pred\_dt)

print(f'MSE (Decision Tree): {mse\_dt}') from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Compute RMSE

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_dt))

print(f'RMSE: {rmse}')

# Compute MAE

mae = mean\_absolute\_error(y\_test, y\_pred\_dt)

print(f'MAE: {mae}')

# Compute MAPE

mape = np.mean(np.abs((y\_test - y\_pred\_scaled) / y\_pred\_dt)) \* 100

print(f'MAPE: {mape}')

# Compute R-squared

r2 = r2\_score(y\_test, y\_pred\_dt)

print(f'R-squared: {r2}')

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Compute RMSE

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rf))

print(f'RMSE: {rmse}')

# Compute MAE

mae = mean\_absolute\_error(y\_test, y\_pred\_rf)

print(f'MAE: {mae}')

# Compute MAPE

mape = np.mean(np.abs((y\_test - y\_pred\_scaled) / y\_pred\_rf)) \* 100

print(f'MAPE: {mape}')

# Compute R-squared

r2 = r2\_score(y\_test, y\_pred\_rf)

print(f'R-squared: {r2}')

# Plot the predictions vs true values for Decision Tree

plt.figure(figsize=(14, 7))

plt.plot(y\_test, label='True Values')

plt.plot(y\_pred\_dt, label='Decision Tree Predictions')

plt.title('Decision Tree: Predictions vs True Values')

plt.xlabel('Time')

plt.ylabel('Close Price')

plt.legend()

plt.show()

# Plot the predictions vs true values for Random Forest

plt.figure(figsize=(14, 7))

plt.plot(y\_test, label='True Values')

plt.plot(y\_pred\_rf, label='Random Forest Predictions')

plt.title('Random Forest: Predictions vs True Values')

plt.xlabel('Time')

plt.ylabel('Close Price')

plt.legend()

plt.show()

# Plot both Decision Tree and Random Forest predictions together

plt.figure(figsize=(14, 7))

plt.plot(y\_test, label='True Values')

plt.plot(y\_pred\_dt, label='Decision Tree Predictions')

plt.plot(y\_pred\_rf, label='Random Forest Predictions')

plt.title('Decision Tree & Random Forest: Predictions vs True Values')

plt.xlabel('Time')

plt.ylabel('Close Price')

plt.legend()

plt.show()