VIRGINIA COMMONWEALTH UNIVERSITY

# Statistical analysis and modelling (SCMA 632)

## A6a : Time Series Analysis

SAYA SANTHOSH

V01101901

Date of Submission: 22-07-2024

# CONTENTS

## Introduction

The stock market analysis centers on anticipating future stock prices of Amazon Inc. (AMZN) through the application of diverse time series forecasting methodologies. This analysis employs historical data of the Adjusted Close price from April 1, 2021 to March 31, 2024, to construct and assess various forecasting models, such as Holt-Winters, ARIMA, LSTM, Random Forest, and Decision Tree models. Every method provides distinct perspectives on the fluctuations of stock prices and their prospective patterns.

The dataset comprises the Adjusted Close price of AMZN obtained from Yahoo Finance. The methodologies employed in this analysis encompass Holt-Winters forecasting, ARIMA, LSTM, Random Forest, and Decision Tree models. The analysis components include time series decomposition, which involves separating the trend, seasonal, and residual 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.

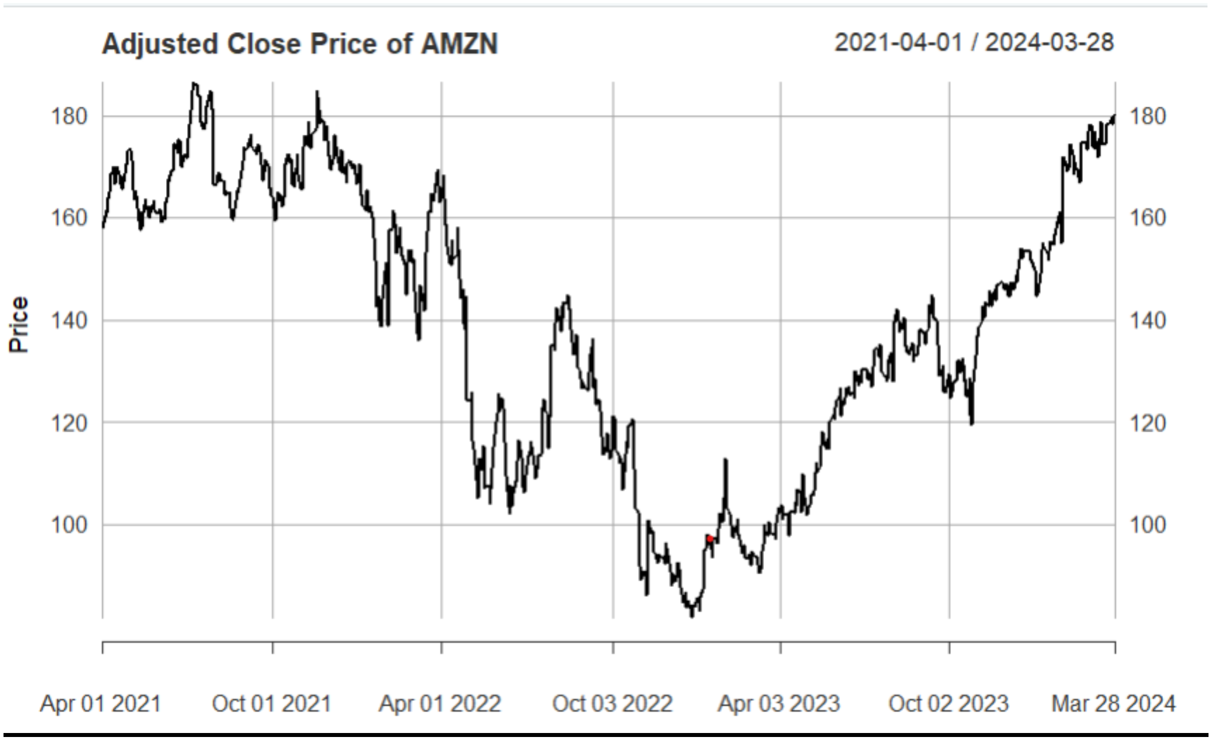
## Business Significance :

The business importance of studying and predicting Amazon stock prices rests in the capacity to make well-informed investment decisions and strategic planning. Investors may enhance their portfolios, reduce risks, and take advantage of prospective growth opportunities by comprehending the past patterns and projected trends of Amazon's stock. Precise predictive models empower investors to anticipate market fluctuations, augmenting their capacity to execute prompt purchase or sale judgments. This analysis also enables financial analysts and counselors to offer more accurate recommendations to their clients, so promoting a more secure and lucrative investment climate.

Understanding the dynamics of stock prices is essential for Amazon to make strategic business decisions, such as allocating resources, expanding into new markets, and positioning itself competitively. The knowledge acquired from the analysis of time series decomposition and forecasting models can provide valuable information for the company's financial strategies, investor relations, and market communication plans. Furthermore, through the utilization of machine learning models for predicting multiple variables, Amazon can acquire a more profound comprehension of the diverse components that impact its stock performance. This empowers the corporation to take proactive measures to resolve possible problems and uphold the trust of investors. In summary, this thorough examination of Amazon's stock prices offers vital insight, fostering sustainable corporate expansion and stability in a competitive market.

## Results and Interpretation using R

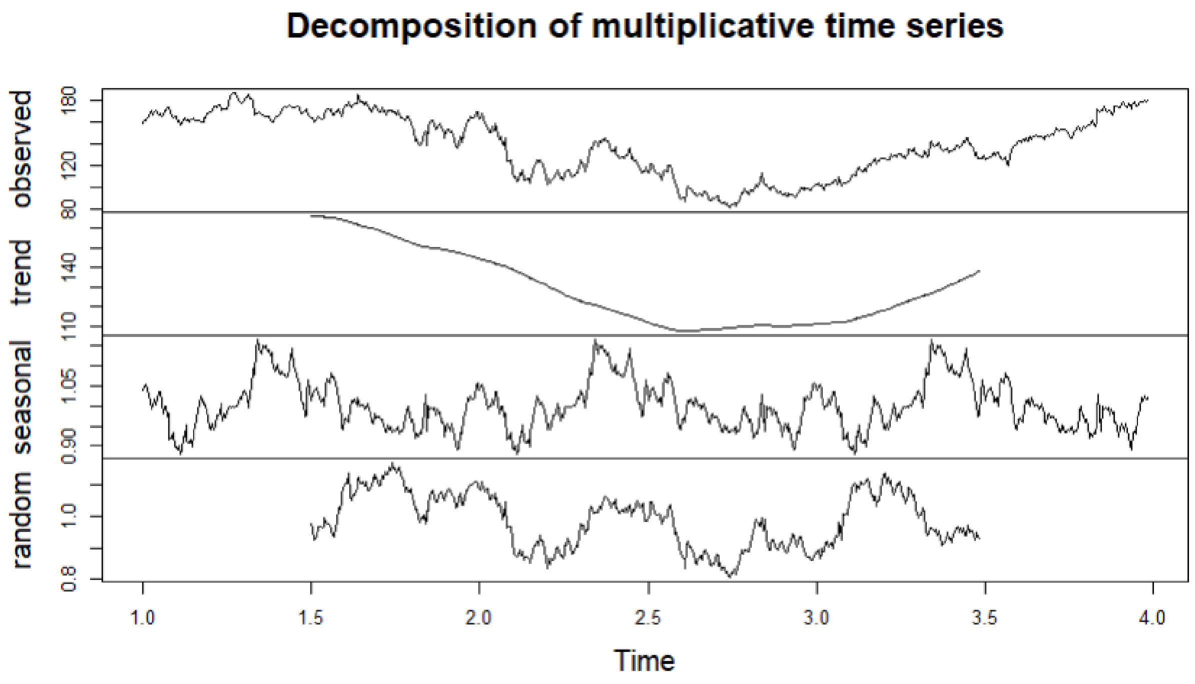
```
> # Plot the data
> plot(adj_close, main = "Adjusted Close Price of AMZN", ylab = "Price", x
lab = "Date")
```



## Interpretation:

The graph displays the adjusted closing price of Amazon (AMZN) from April 1, 2021, to March 28, 2024. The price experienced significant fluctuations over this period. Initially, it reached a peak above $180 around mid-2021 before declining sharply to below $100 by mid-2022. After hitting this low, the stock price demonstrated a gradual recovery, with occasional dips, and steadily increased throughout 2023 and into early 2024, ultimately returning to around $180. This indicates a volatile yet resilient performance, reflecting broader market conditions and company-specific factors influencing Amazon's stock price during this timeframe.

```
> # Plot the decomposed components
> plot(decomposed)
```

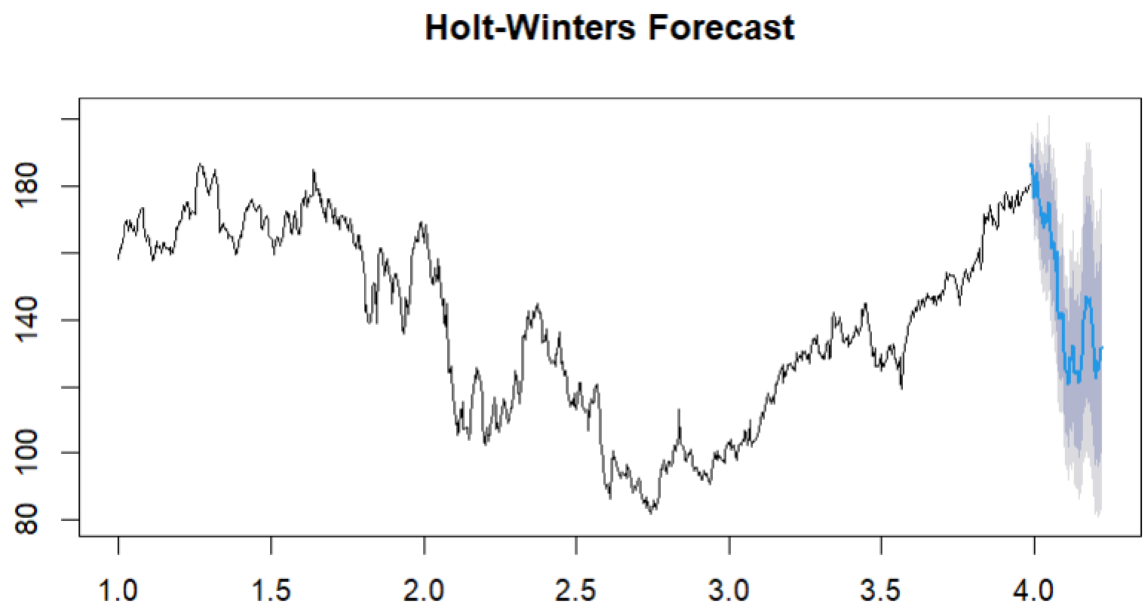


**Interpretation:**

It shows the decomposition of a multiplicative time series into its observed, trend, seasonal, and random components.

1. **Observed:** The top panel displays the original time series data, which shows significant fluctuations and an overall upward trend toward the end.
2. **Trend:** The second panel isolates the trend component, revealing a long-term decline followed by a recovery, mirroring the observed data's general movement.
3. **Seasonal:** The third panel highlights the seasonal component, showing repeating patterns that suggest periodic variations within the time series.
4. **Random:** The bottom panel represents the random or residual component, which captures the irregular fluctuations that are not explained by the trend or seasonal components.
5. This decomposition helps to understand the underlying patterns and the contributions of different factors to the observed time series, facilitating better analysis and forecasting.

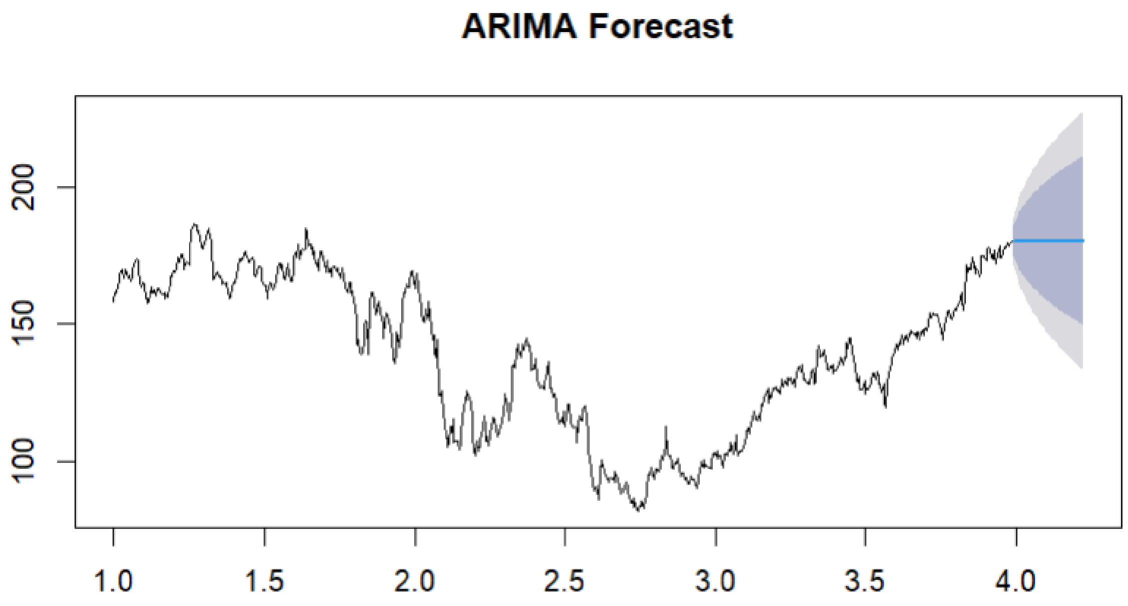
```
> # Plot the Holt-Winters forecast
> plot(hw_forecast, main = "Holt-Winters Forecast")
```



**Interpretation:**

The graph illustrates a Holt-Winters forecast applied to a time series data set. The observed data is shown in black, while the forecasted values are depicted in blue, accompanied by a shaded region representing the confidence intervals. The Holt-Winters method, which accounts for seasonality, trend, and level, projects a continuation of the observed upward trend, although with noticeable fluctuations. The confidence intervals widen as the forecast extends further into the future, indicating increasing uncertainty. The forecast suggests that the values will likely stay within a specific range, reflecting the model's predictions based on historical data patterns.

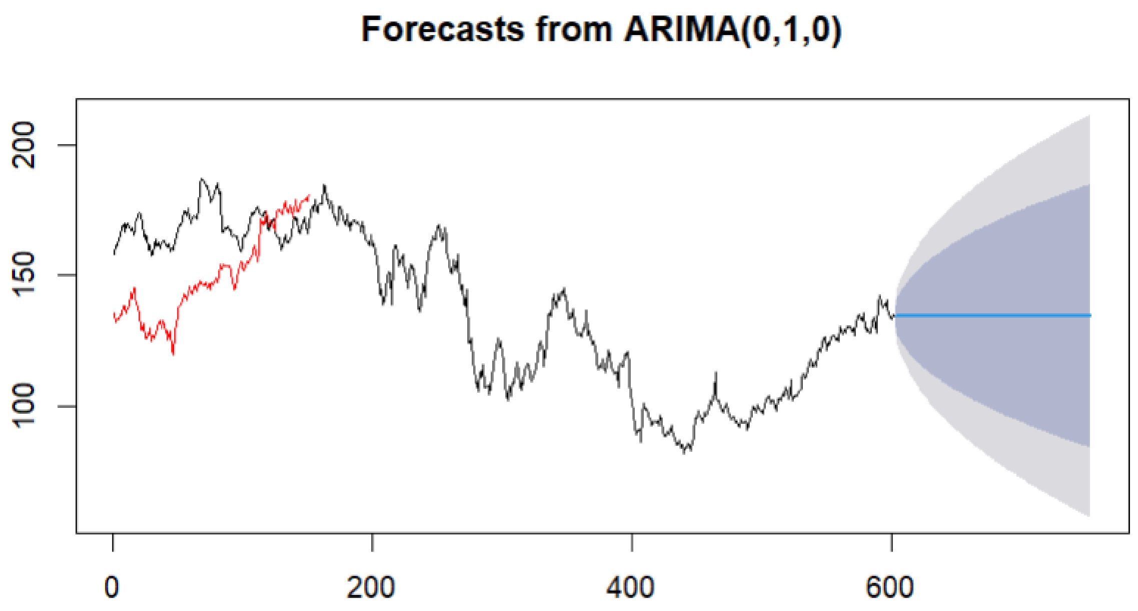
```
> # Plot the ARIMA forecast
> plot(arima_forecast, main = "ARIMA Forecast")
```



**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:**

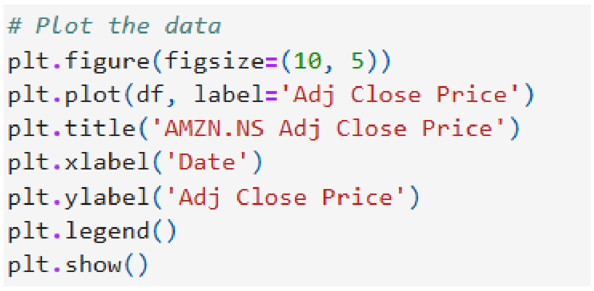
The performance metrics for the Random Forest model indicate a highly accurate and well-fitting model. The Root Mean Square Error (RMSE) of 0.13 suggests that the model's predictions deviate very minimally from the actual values. The Mean Absolute Error (MAE) of 0.09 further supports this, showing that 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: 0.92452669908218"
```

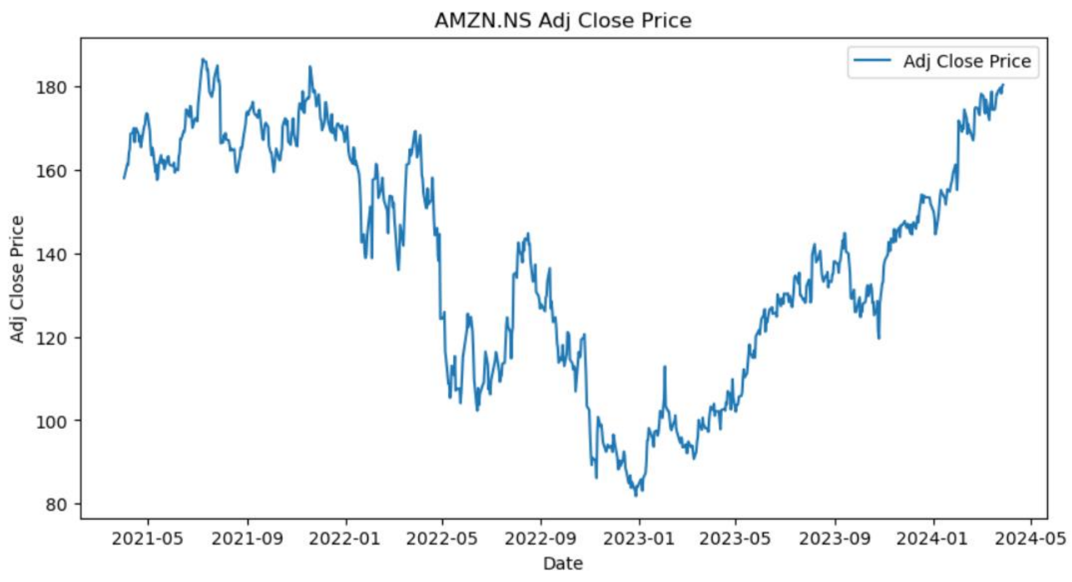
**Interpretation:**

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

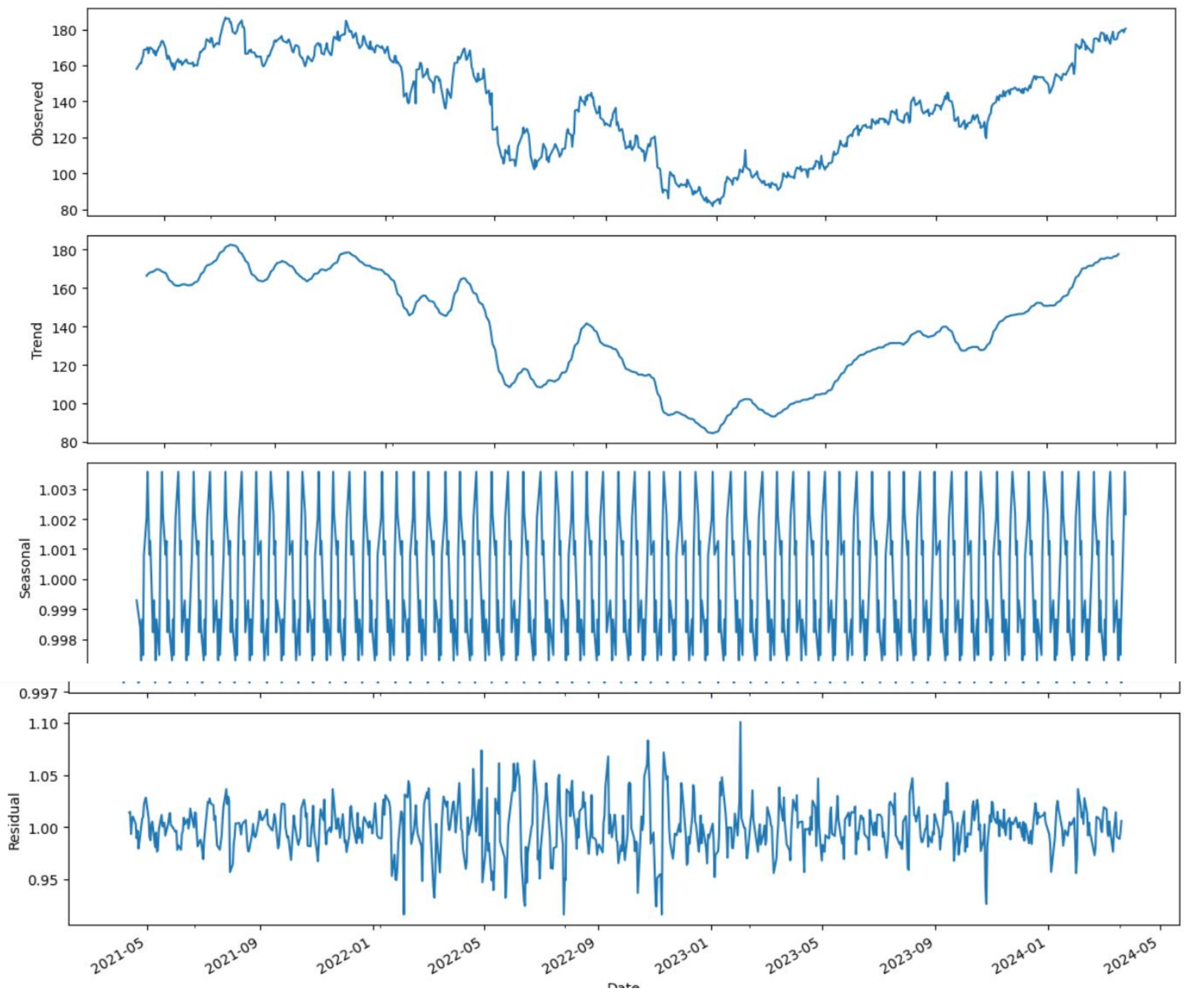


```python
# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(df, label='Adj Close Price')
plt.title('AMZN.NS Adj Close Price')
plt.xlabel('Date')
plt.ylabel('Adj Close Price')
plt.legend()
plt.show()
```



**Interpretation:**

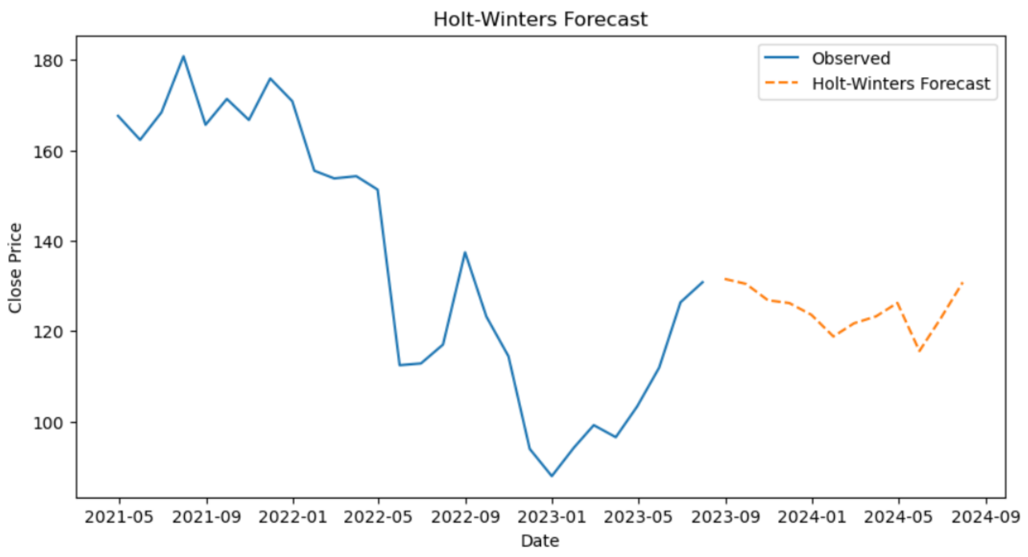
The plot shows the adjusted closing price of AMZN.NS (Amazon stock) over the period from May 2021 to May 2024. The stock price exhibits significant fluctuations throughout this period. Initially, the price is around 160-180, but it experiences a decline starting in mid-2021 and continues into early 2022, 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:

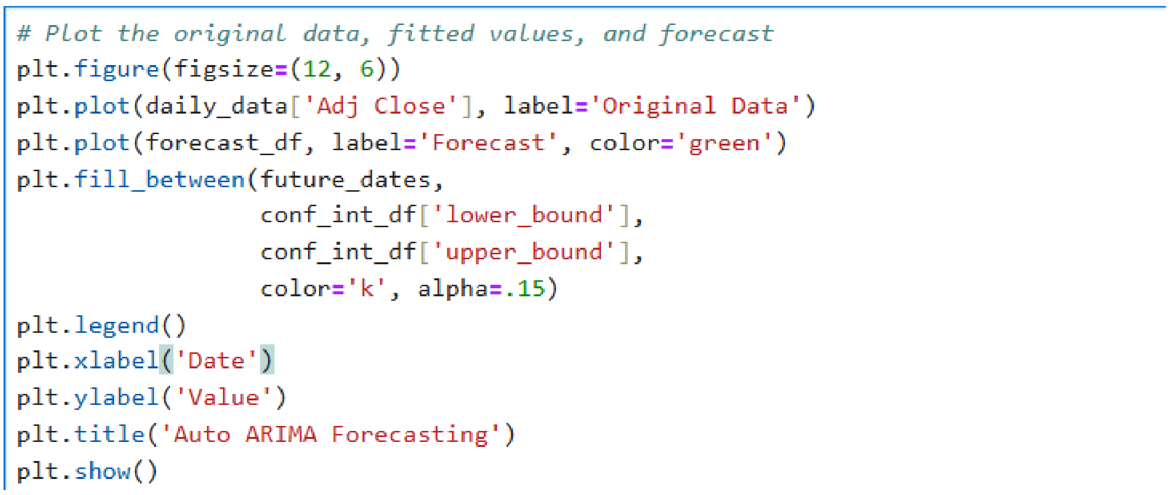
It shows the decomposition of a time series data into its observed, trend, seasonal, and residual components. The observed panel at the top represents the raw time series data. The trend component, displayed in the second panel, highlights the underlying direction of the data, showing a significant decline 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.

```
# 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()
```

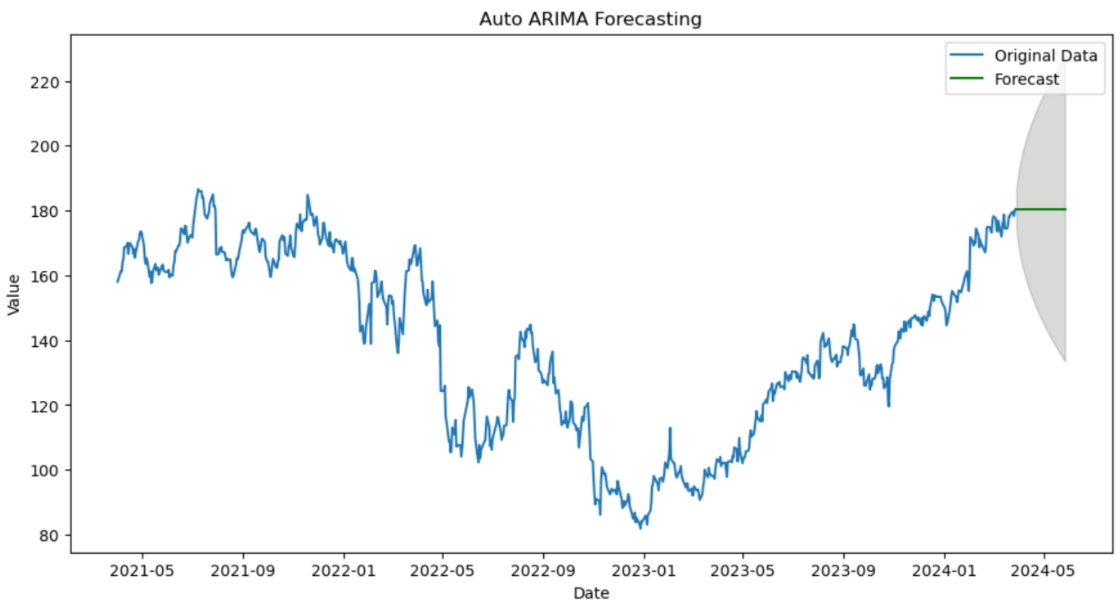


**Interpretation:**

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.



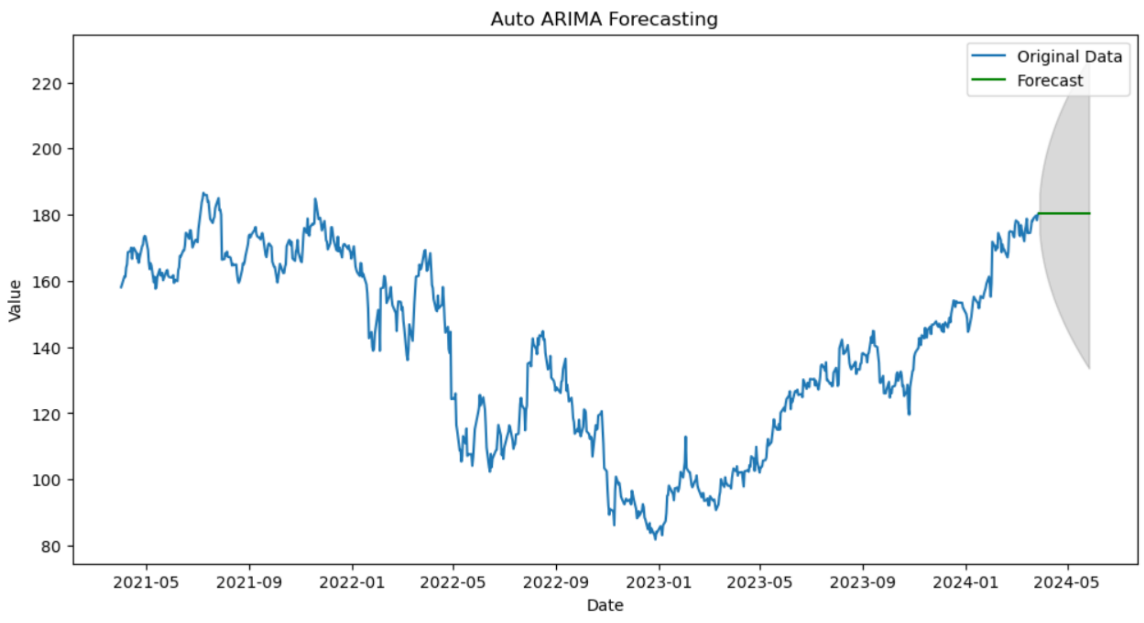
```python
# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'], label='Original Data')
plt.plot(forecast_df, label='Forecast', color='green')
plt.fill_between(future_dates,
                 conf_int_df['lower_bound'],
                 conf_int_df['upper_bound'],
                 color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()
```



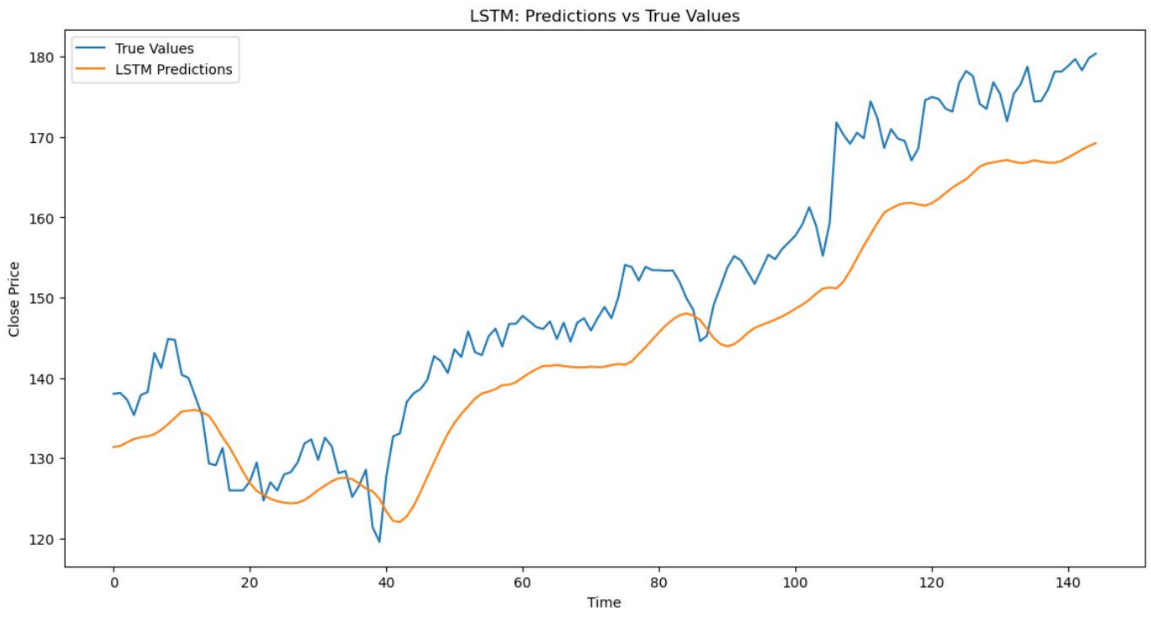
**Interpretation:**

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.

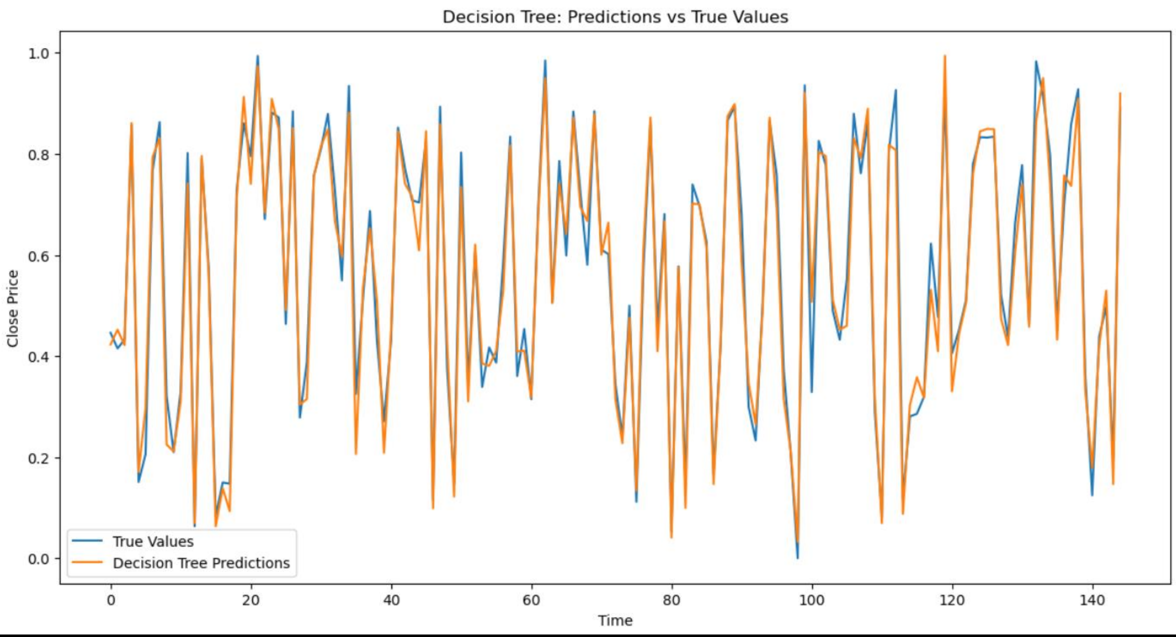
```python
# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'], label='Original Data')
plt.plot(forecast_df, label='Forecast', color='green')
plt.fill_between(future_dates,
                 conf_int_df['lower_bound'],
                 conf_int_df['upper_bound'],
                 color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()
```



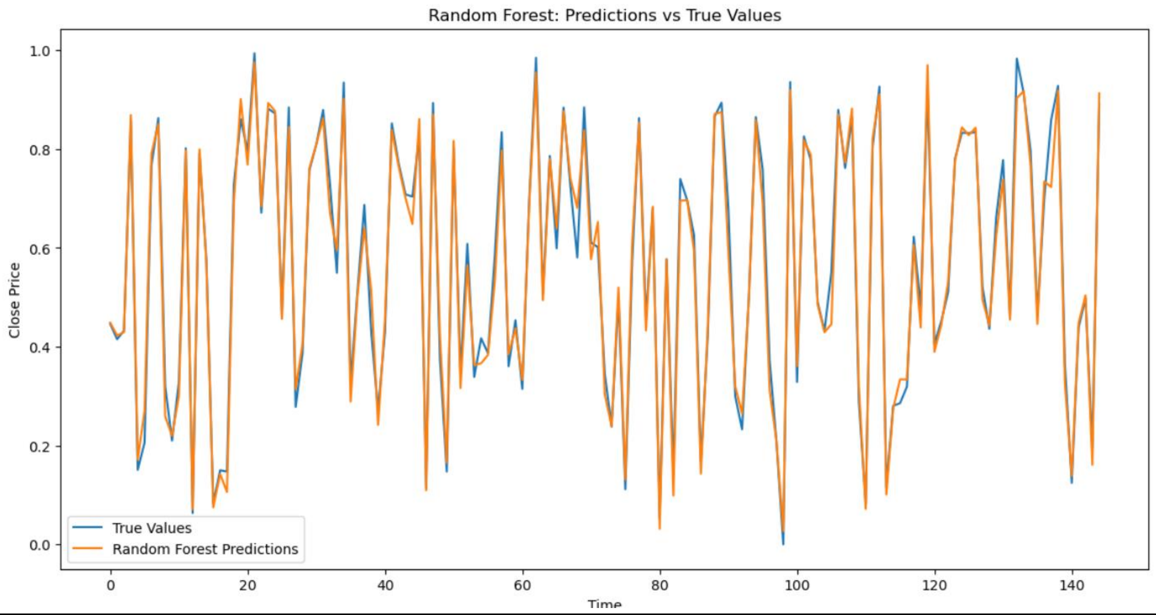
```python
# Plot the predictions vs true values
plt.figure(figsize=(14, 7))
plt.plot(y_test_scaled, label='True Values')
plt.plot(y_pred_scaled, label='LSTM Predictions')
plt.title('LSTM: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



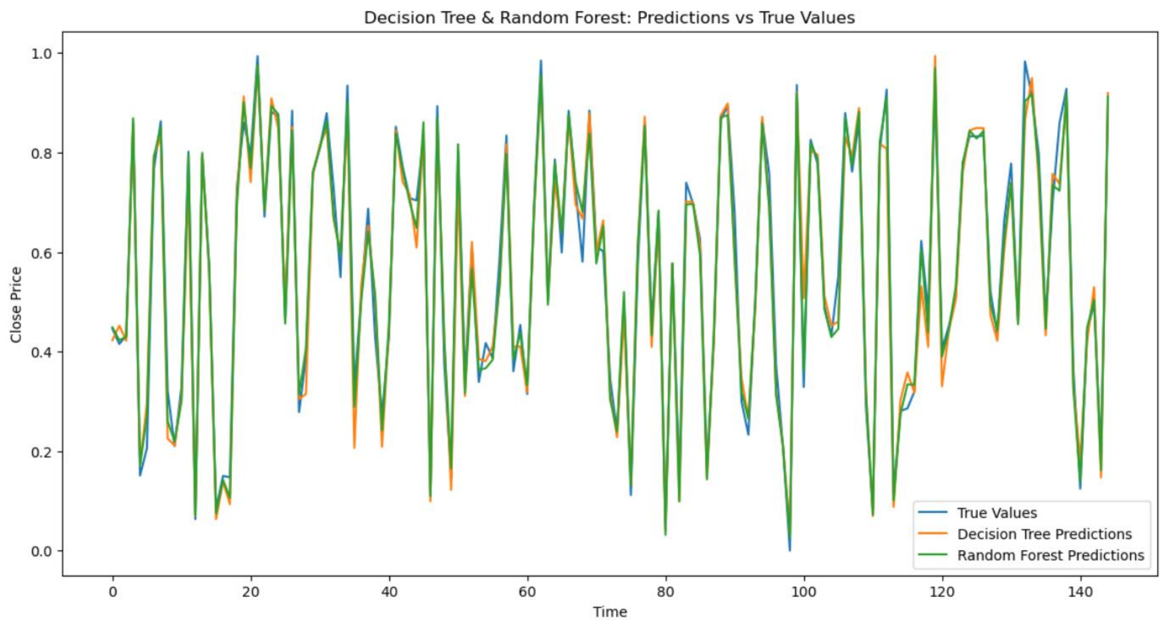
```
# Plot the predictions vs true values for Decision Tree
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.title('Decision Tree: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



```python
# 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()
```



## <u>**Recommendations**</u>

To make better investment decisions, it is important to focus on long-term trends and be cautious of short-term fluctuations. Understanding the underlying factors affecting stock prices requires attention to both trend and seasonal components. This approach is useful for accounting seasonality in stock prices and planning market entry or exit points. For short-term predictions, ARIMA can be employed, but it should be complemented with other analyses due to its moderate R-squared value. For accurate long-term predictions, LSTM is recommended, with regular updates incorporating new data. Additionally, using Random

Forest is preferable over Decision Tree for more accurate and reliable stock price predictions. Combining different models to leverage their strengths can further improve prediction accuracy.

## 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 = tra
in_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 - mea
n(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 - me
an(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 stats
models.tsa.holtwinters import ExponentialSmoothing

# Fit the Holt-Winters model
holt_winters_model = ExponentialSmoothing(train_data, seasonal='mul', seasonal_periods=1
2).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_t
est), 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.res
hape(-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()
```

## References

1. www.github.com