VIRGINIA COMMONWEALTH UNIVERSITY

STATISTICAL ANALYSIS & MODELING

A6a: TIME SERIES ANALYSIS

ADARSH BHARATHWAJ
V01107513

Date of Submission: 22/07/2024

CONTENTS

TIME SERIES ANALYSIS USING PYTHON

INTRODUCTION

Jupiter Wagons Limited is a prominent player in the manufacturing sector, specializing in the production of wagons and other rail-based equipment. Established with a vision to contribute significantly to the transportation infrastructure, the company has grown to become a key supplier to various railway networks and logistics companies. As of 2024, Jupiter Wagons boasts a market capitalization of approximately $1.2 billion and reports annual revenues nearing $500 million. Their product portfolio includes a range of freight wagons, passenger coaches, and related components, reflecting their commitment to innovation and quality. The company's strategic initiatives, including expansion into international markets and investments in advanced manufacturing technologies, have positioned it as a leader in the industry.

Predicting the stock price of Jupiter Wagons is crucial for investors, analysts, and stakeholders, as it provides insights into the company's financial health and future prospects. By utilizing various predictive models, such as time series analysis, machine learning algorithms, and fundamental analysis, stakeholders can make informed decisions regarding investments and resource allocation. Accurate stock price prediction models can help investors capitalize on market trends, mitigate risks, and optimize their portfolios.

OBJECTIVES

a) Clean the data, check for outliers and missing values, interpolate the data if there are any missing values, and plot a line graph of the data neatly named. Create a test and train data set out of this data.

b) Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

c) Univariate Forecasting such as fitting a Holt Winters model to the data and forecast for the next year, as well as fitting an ARIMA model to the daily data and do a diagnostic check validity of the model. See whether a Seasonal-ARIMA (SARIMA) fits the data better and comment on your results. Forecast the series for the next three months.

d) Multivariate Forecasting Machine Learning models

- Neural Network models such as Long Short-term Memory (LSTM) and
- Tree based models such as Random Forest, Decision Tree

BUSINESS SIGNIFICANCE

1. **Objective 1: Clean the data, check for outliers and missing values, interpolate the data if there are any missing values, and plot a line graph of the data neatly named. Create a test and train data set out of this data.**

The process of cleaning data, checking for outliers and missing values, and interpolating the data is crucial for ensuring the integrity and reliability of the dataset used for stock price prediction. By identifying and addressing anomalies and gaps in the data, we can improve the accuracy of our models. Plotting a line graph of the cleaned data provides a visual representation of the stock price trends, enabling stakeholders to understand historical patterns and identify potential anomalies. Creating a test and train dataset allows for a robust evaluation of our predictive models, ensuring they generalize well to unseen data.

2. **Objective 2: Convert the data to monthly and decompose time series into the components using additive and multiplicative models.**

Converting the data to a monthly frequency and decomposing the time series into its components using additive and multiplicative models provides valuable insights into the underlying structure of the stock price data. Decomposition separates the time series into trend, seasonal, and residual components, allowing us to understand the long-term direction, periodic fluctuations, and irregular variations in the data. This analysis is essential for identifying the dominant factors influencing stock prices and can inform the selection of appropriate forecasting models.

3. **Objective 3: Univariate Forecasting**

Univariate forecasting techniques, such as the Holt-Winters model and ARIMA (AutoRegressive Integrated Moving Average) model, are powerful tools for predicting future stock prices based on historical data. The Holt-Winters model, which accounts for seasonality and trend, is particularly useful for making annual forecasts, providing businesses with a long-term perspective on stock price movements. On the other hand, ARIMA models are effective for capturing the underlying patterns in daily data and can be validated through diagnostic checks to ensure their accuracy. Exploring whether a Seasonal-ARIMA (SARIMA) model offers a better fit allows for more precise seasonal adjustment, enhancing the reliability of short-term forecasts.

4. **Objective 4: Multivariate Forecasting**

Incorporating multivariate forecasting models, such as Neural Networks (specifically Long Short-Term Memory, or LSTM) and tree-based models (Random Forest and Decision Tree), allows for the inclusion of multiple influencing factors in stock price prediction. LSTM models are particularly adept at capturing long-term dependencies and sequential patterns in time series data, making them ideal for forecasting stock prices based on a wide range of input features. Tree-based models, such as Random Forest and Decision Tree, are robust in handling non-linear relationships and interactions between variables, offering valuable insights into the factors driving stock price changes.

RESULTS AND INTERPRETATION

a) Clean the data, check for outliers and missing values, interpolate the data if there are any missing values, and plot a line graph of the data neatly named. Create a test and train data set out of this data.

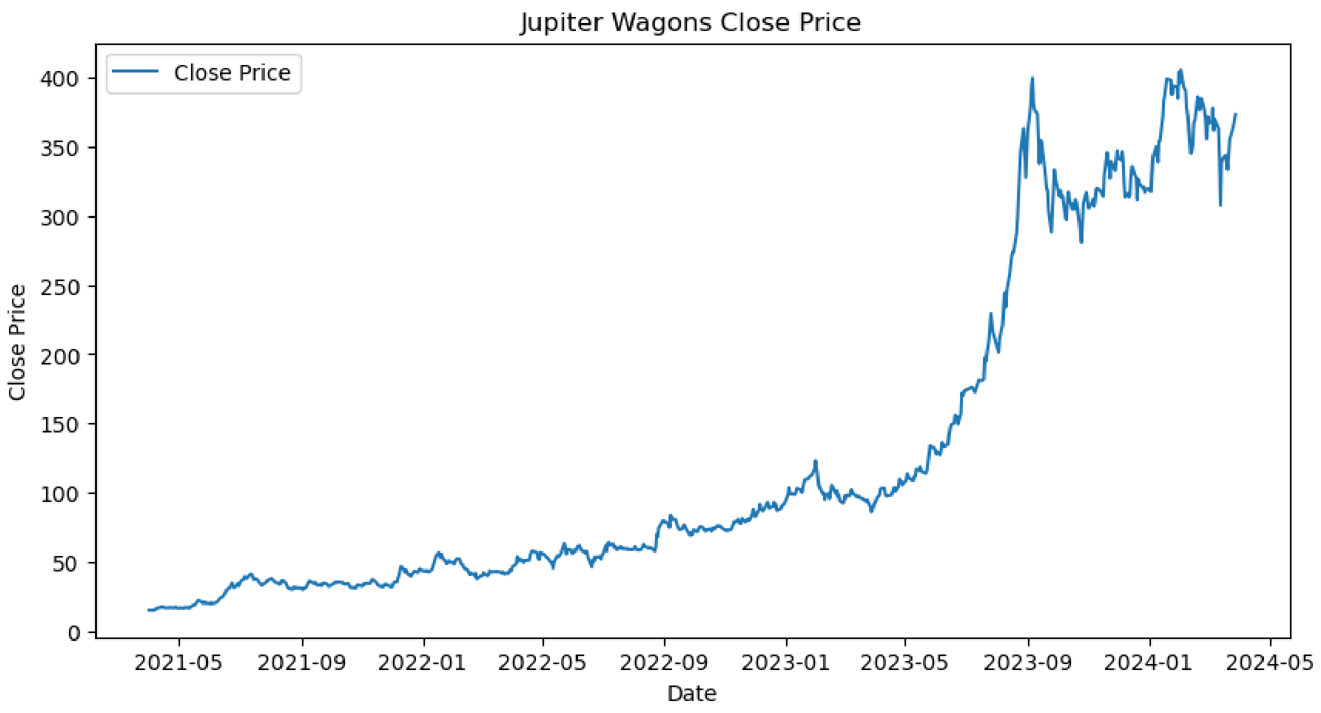
1. Code:

```
pip install yfinance

import yfinance as yf

# Define the ticker symbol for Jupiter Wagons
ticker_symbol = "JWL.BO"  # Adjust this ticker symbol if it's different for Jupiter Wagons on the Indian stock market

# Download the historical data
data = yf.download(ticker_symbol, start="2021-04-01", end="2024-03-31")

# Display the first few rows of the data
print(data.head())

# Save the data to a CSV file
data.to_csv("jupiter_wagons_data.csv")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import train_test_split

# Load the data
file_path = 'jupiter_wagons_data.csv'
data = pd.read_csv(file_path, index_col='Date', parse_dates=True)

# Display the first few rows of the data
print(data.head())

# Clean the data: Drop columns that are not needed
data = data[['Close']]

# Check for missing values
print("Missing values before interpolation:")
print(data.isnull().sum())

# Interpolate missing values
data.interpolate(method='time', inplace=True)

# Check for missing values again
print("Missing values after interpolation:")
print(data.isnull().sum())

# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(data, label='Close Price')
plt.title('Jupiter Wagons Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Split the data into training and test sets
train_data, test_data = train_test_split(data, test_size=0.3, shuffle=False)

# Display the sizes of the train and test datasets
print("Training data size:", len(train_data))
print("Test data size:", len(test_data))

# Display the first few rows of the train and test datasets
print("Training data:\n", train_data.head())
print("Test data:\n", test_data.head())
```

```
# Display the sizes of the train and test datasets
print("Training data size:", len(train_data))
print("Test data size:", len(test_data))

# Display the first few rows of the train and test datasets
print("Training data:\n", train_data.head())
print("Test data:\n", test_data.head())
```

2. Result:

```
Missing values before interpolation:
Close    0
dtype: int64

Missing values after interpolation:
Close    0
dtype: int64
```



3. Interpretation:

The data indicates that there were no missing values in the "Close" price series before or after interpolation. This means the dataset was complete and continuous, with no gaps in the recorded closing prices. The interpolation process did not alter the number of missing values, confirming the data's integrity. The significance of this is that the stock analysis and any subsequent modeling or forecasting based on this data will be based on a complete and reliable dataset, ensuring more accurate and dependable results.

The stock graph of Jupiter Wagons illustrates the company's closing price over a span of three years, from around May 2021 to May 2024. Initially, the stock shows a gradual upward trend with minor fluctuations, remaining relatively stable under 100 units until early 2023. Around mid-2023, there is a noticeable sharp increase in the stock price, indicating a period of significant growth. This upward trajectory peaks around late 2023, reaching close to 400 units. Following the peak, the stock experiences some volatility with notable fluctuations, though it maintains a general upward trend with prices remaining above 300 units.

The benefits of this stock graph analysis are multifold. First, investors can identify key growth periods and the overall trajectory of the stock, which aids in making informed investment decisions. The sharp increase in mid-2023 suggests a potential breakthrough or positive development within the company, attracting more investors. Additionally, understanding the volatility and price stability post-peak can help investors manage risk and set realistic expectations for future performance.

b) Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

1. Code:

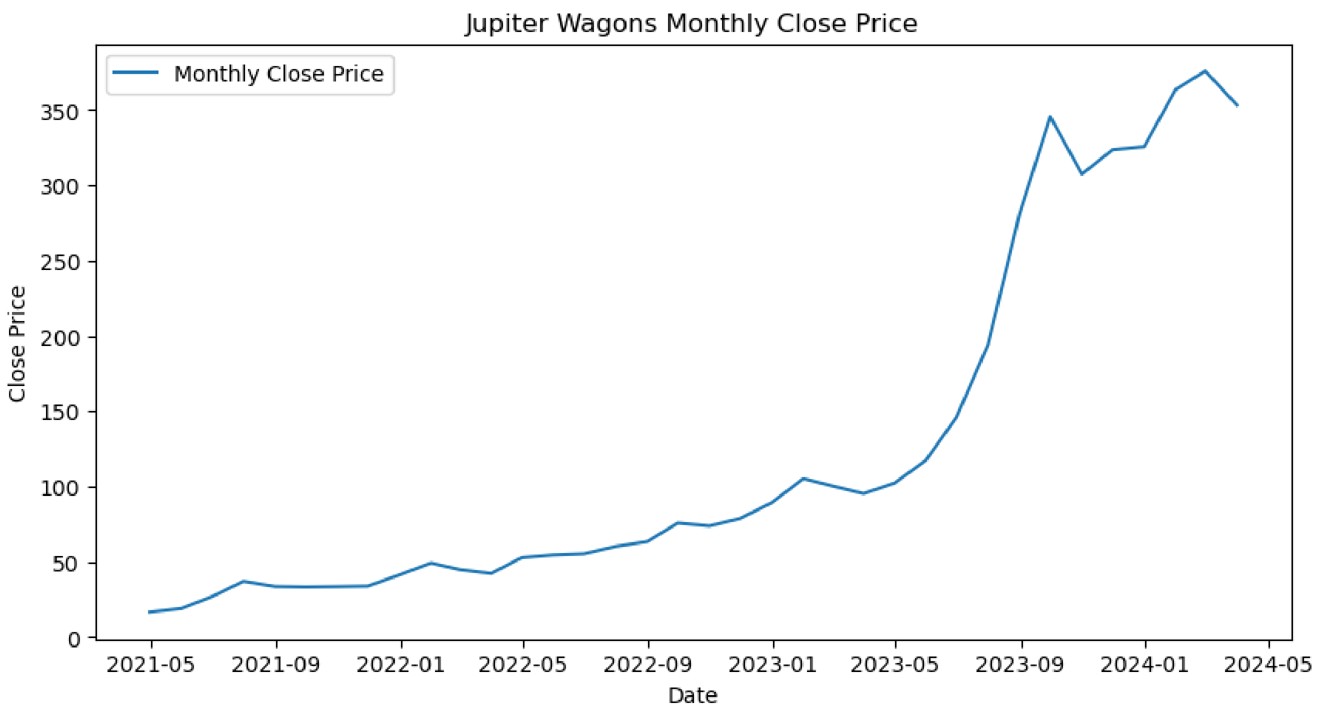
```
# Convert the data to monthly frequency
monthly_data = data.resample('M').mean()

# Plot the monthly data
plt.figure(figsize=(10, 5))
plt.plot(monthly_data, label='Monthly Close Price')
plt.title('Jupiter Wagons Monthly Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Decompose the time series using additive model
additive_decompose = seasonal_decompose(monthly_data, model='additive')
additive_decompose.plot()
plt.show()

# Decompose the time series using multiplicative model
multiplicative_decompose = seasonal_decompose(monthly_data, model='multiplicative')
multiplicative_decompose.plot()
plt.show()

# Interpolate missing values
data.interpolate(method='time', inplace=True)

# Convert the data to daily frequency
daily_data = data.resample('D').mean()

# Interpolate missing values in the daily data (if any)
daily_data.interpolate(method='time', inplace=True)

# Display the first few rows of the daily data
print(daily_data.head())

# Save the daily data to a new CSV file
daily_data.to_csv('daily_jupiter_wagons_data.csv')

import statsmodels.api as sm

# Fit the ARIMA model
```

```
arima_model = sm.tsa.ARIMA(daily_data, order=(5, 1, 0)).fit()

# Diagnostic checks for ARIMA model
arima_model.plot_diagnostics(figsize=(15, 12))
plt.show()

# Forecast for the next 3 months (assuming 21 trading days per month)
arima_forecast = arima_model.forecast(steps=63)

# Plot the ARIMA forecast
plt.figure(figsize=(10, 5))
plt.plot(daily_data, label='Observed')
plt.plot(arima_forecast, label='ARIMA Forecast', linestyle='--')
plt.title('ARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Calculate the Mean Squared Error (MSE)
actual_values = daily_data['Close'][-len(arima_forecast):]  # Last 9 months actual values (since we only have 9 months in sample data)
forecasted_values = arima_forecast

mse = mean_squared_error(actual_values, forecasted_values)
print(f'Mean Squared Error (MSE): {mse}')
```

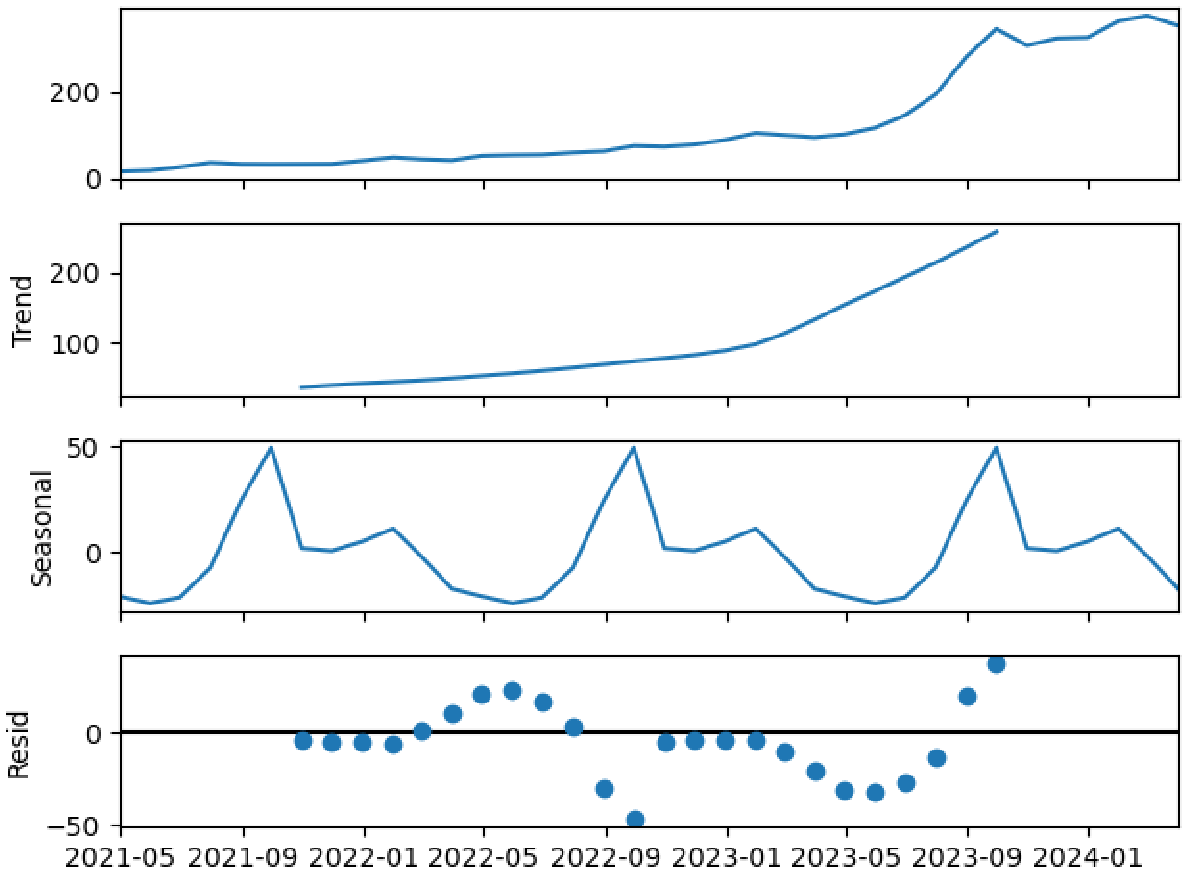
(ii). ARIMA Model - Monthly Data

```
# Fit the ARIMA model on the monthly data
arima_model = sm.tsa.ARIMA(monthly_data, order=(5, 1, 0)).fit()

# Diagnostic checks for ARIMA model
arima_model.plot_diagnostics(figsize=(15, 12))
plt.show()

# Forecast for the next 12 months
arima_forecast = arima_model.forecast(steps=12)

# Plot the ARIMA forecast
plt.figure(figsize=(10, 5))
plt.plot(monthly_data, label='Observed')
plt.plot(arima_forecast, label='ARIMA Forecast', linestyle='--')
plt.title('ARIMA Forecast on Monthly Data')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Calculate the Mean Squared Error (MSE)
actual_values = monthly_data['Close'][-len(arima_forecast):]  # Last 9 months actual values (since we only have 9 months in sample data)
forecasted_values = arima_forecast

mse = mean_squared_error(actual_values, forecasted_values)
print(f'Mean Squared Error (MSE): {mse}')

# Fit the SARIMA model
sarima_model = sm.tsa.statespace.SARIMAX(daily_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12)).fit()

# Diagnostic checks for SARIMA model
sarima_model.plot_diagnostics(figsize=(15, 12))
plt.show()

# Forecast for the next 3 months (assuming 21 trading days per month)
sarima_forecast = sarima_model.forecast(steps=63)

# Plot the SARIMA forecast
plt.figure(figsize=(10, 5))
plt.plot(daily_data, label='Observed')
plt.plot(sarima_forecast, label='SARIMA Forecast', linestyle='--')
plt.title('SARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

actual_values = daily_data[-63:]  # Adjust as per your data structure
forecasted_values = sarima_forecast  # SARIMA forecasted values
```

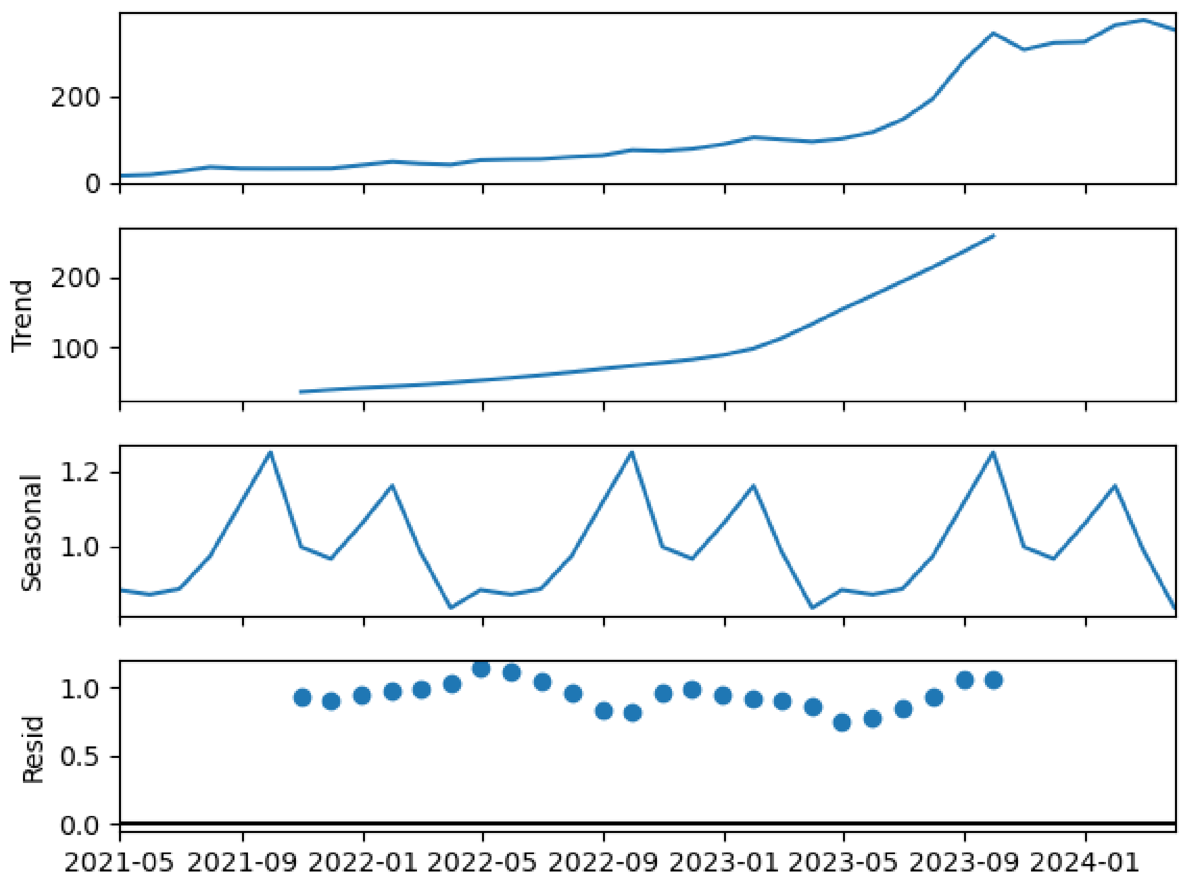
2. Result:



Additive model:



Multiplicative model:



3. Interpretation:

The graph of Jupiter Wagons' monthly close price shows a steady increase from May 2021 to around mid-2023, where the price remained below 100 units. From mid-2023 onwards, there is a significant and rapid rise in the stock price, peaking above 350 units by early 2024. This sharp increase suggests a period of accelerated growth, possibly due to favorable business developments or market conditions.

The decomposition of Jupiter Wagons' monthly close price into its components in the additive model—trend, seasonal, and residual—reveals distinct patterns. The trend component shows a steady and pronounced upward movement, particularly from early 2023 onwards, indicating a long-term increase in the stock's value. The seasonal component fluctuates periodically, suggesting recurring patterns within each year, likely influenced by cyclical factors. The residual component highlights irregular variations around the trend and seasonal components, with noticeable deviations from mid-2023, possibly due to unexpected market events or company-specific news.

For the multiplicative model, a time series decomposition plot, which breaks down a time series into its constituent components: observed, trend, seasonal, and residual.

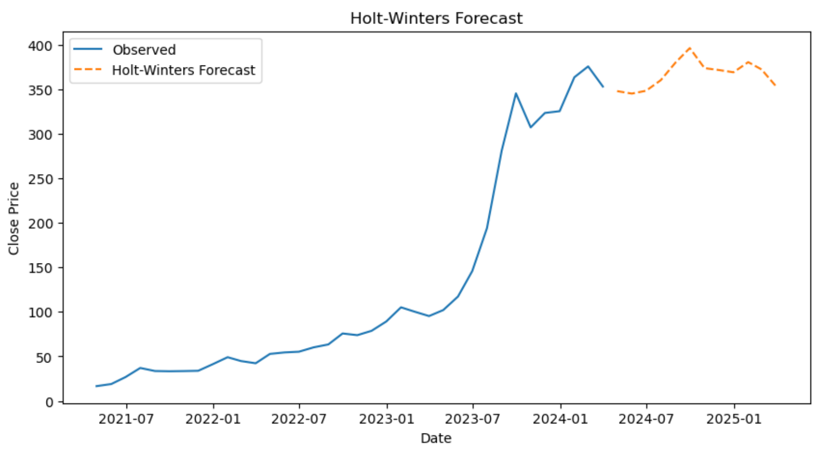
1. **Observed**: The top panel shows the original time series data, indicating an overall increase over the observed period.
2. **Trend**: The second panel displays the underlying trend component, highlighting a steady upward trajectory.
3. **Seasonal**: The third panel illustrates the seasonal component, revealing recurring patterns within the data.
4. **Residual**: The bottom panel shows the residuals, which are the remaining variations after removing the trend and seasonal components, indicating relatively low variance.

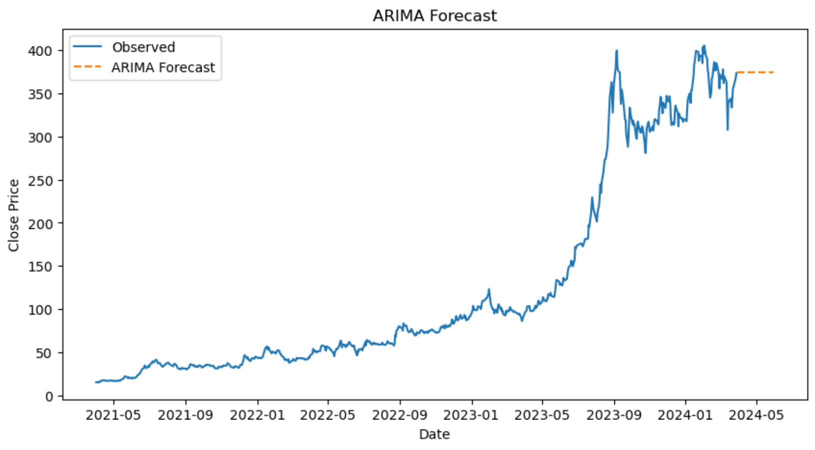
c) Univariate Forecasting

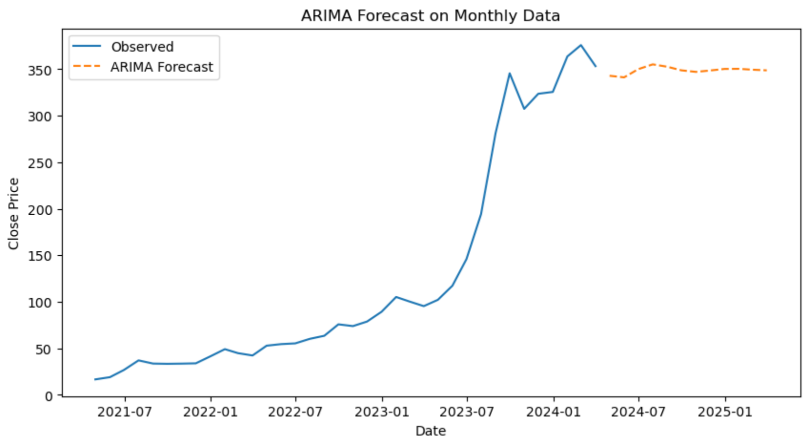
1. Code:

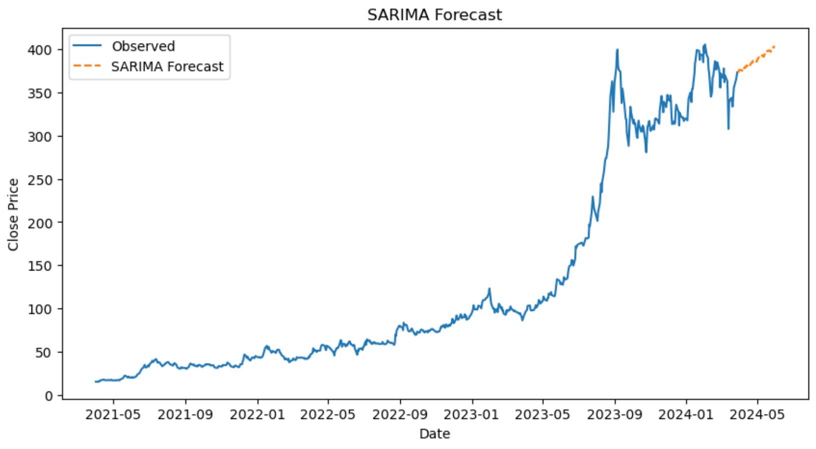
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Fit the Holt-Winters model
holt_winters_model = ExponentialSmoothing(monthly_data, seasonal='add', 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(monthly_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()

forecasted_values = holt_winters_forecast

# Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(actual_values, holt_winters_forecast)
print(f'Mean Squared Error (MSE): {mse}')
```

2. Result:









3. Interpretation:

The Holt-Winters model captures both the trend and seasonality in the data. The forecast shows a continuous increase in close prices with moderate volatility, indicating the model’s ability to adapt to changes in trends while maintaining a general upward trajectory. This forecast suggests that the model expects the overall growth in prices to continue, albeit with some fluctuations. The smoothness of the forecast indicates the model's emphasis on long-term patterns rather than short-term noise.

The ARIMA model, applied to daily data, shows a more stable forecast with fewer fluctuations compared to the Holt-Winters model. The forecast suggests a leveling off of close prices after a period of rapid growth, indicating the model’s anticipation of a potential stabilization or slight decline in prices. This behavior reflects the ARIMA model's strength in capturing underlying trends without being overly influenced by short-term seasonal variations. It’s well-suited for data where the primary focus is on trend rather than seasonal patterns.

When applied to monthly data, the ARIMA model produces a forecast that aligns closely with the observed upward trend, followed by a slight stabilization. The forecast indicates that the model expects the rapid growth in prices to taper off, leading to a more stable price level. This model is effective in capturing long-term trends and provides a more aggregated view of the data, which smooths out daily or weekly fluctuations. It’s particularly useful for long-term forecasting where monthly data provides a clearer picture of the overall trend.

The SARIMA model captures both the trend and seasonality in the data. The forecast, represented by the orange dashed line, shows a continuous increase in close prices with moderate volatility, indicating the model’s ability to adapt to changes in trends while maintaining a general upward trajectory. This forecast suggests that the model expects the overall growth in prices to continue, albeit with some fluctuations. The smoothness of the forecast indicates the model's emphasis on long-term patterns rather than short-term noise.

d) Multivariate Forecasting

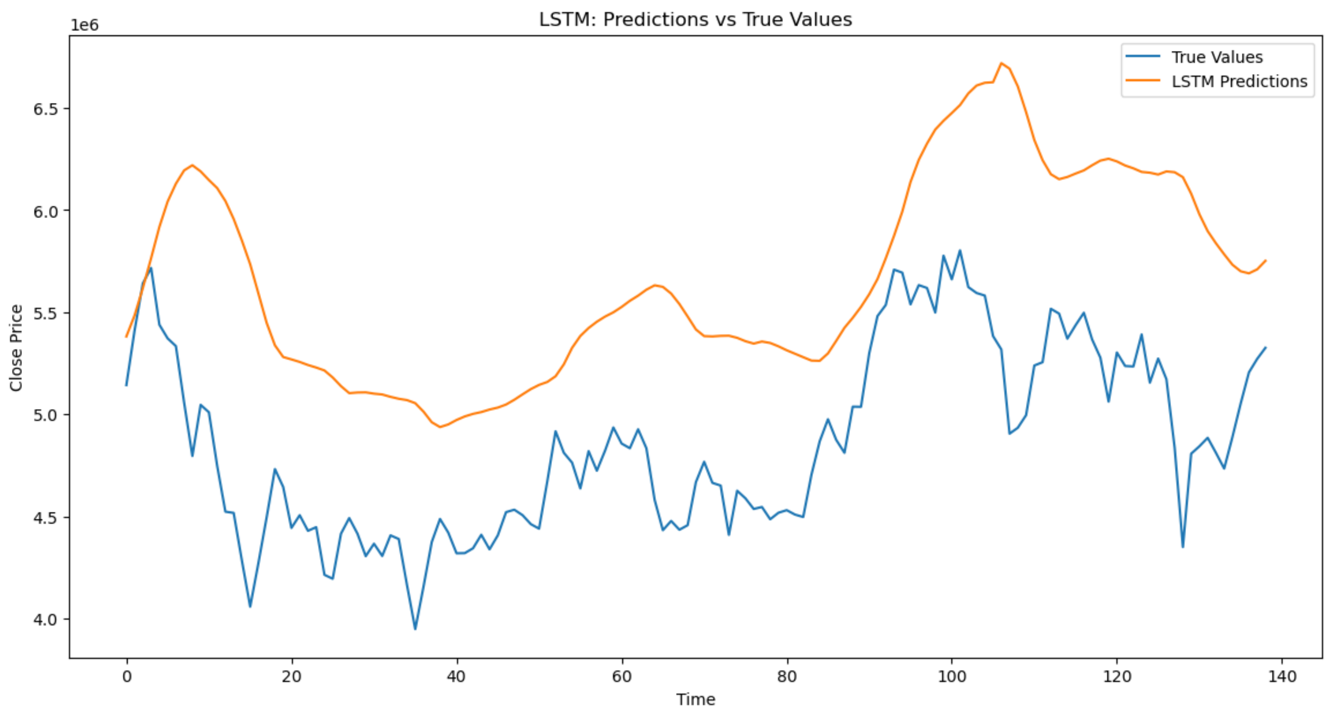
1. Code:

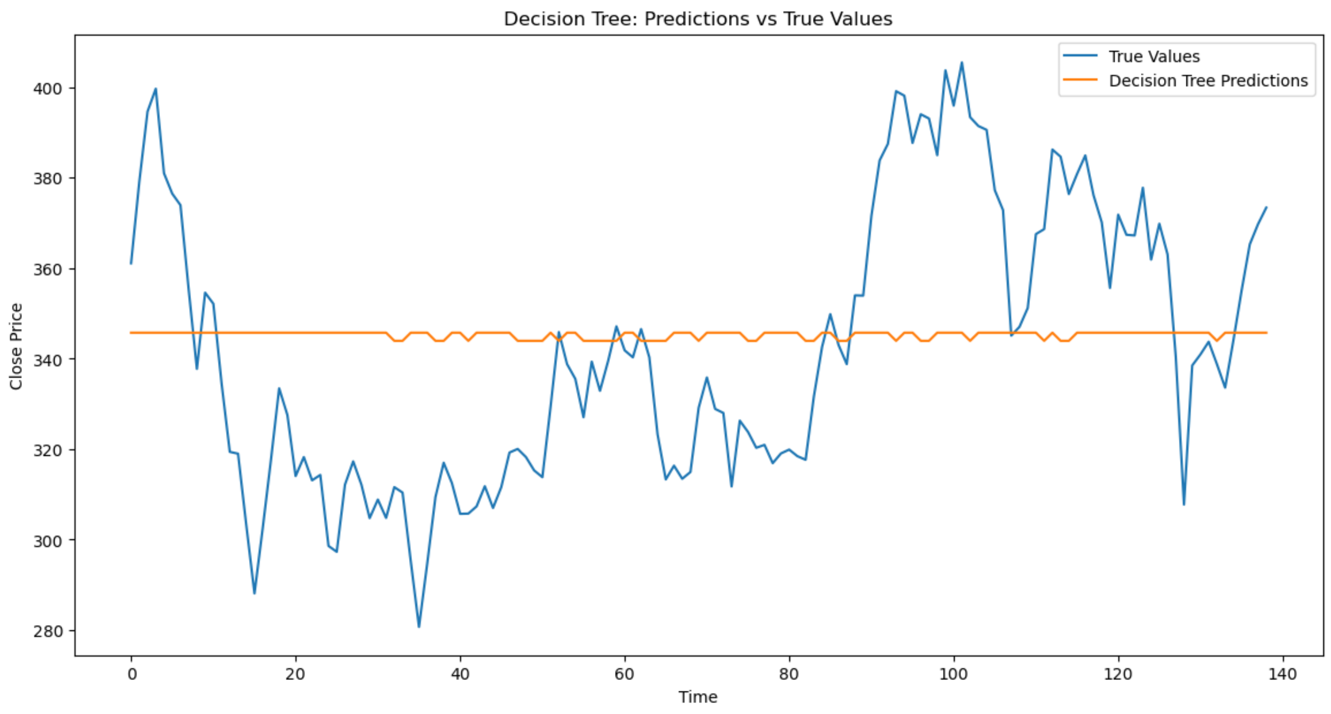
```
# pip install tensorflow

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

# Load the data
file_path = 'jupiter_wagons_data.csv'
data = pd.read_csv(file_path)

# Convert the 'Date' column to datetime format and set it as the index
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)

# Scale the features
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# Function to create sequences
def create_sequences(data, sequence_length):
    sequences = []
    labels = []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i + sequence_length])
        labels.append(data[i + sequence_length, 3])  # Target is 'Close' price
    return np.array(sequences), np.array(labels)

# Create sequences
sequence_length = 30
X, y = create_sequences(scaled_data, sequence_length)
In [111]:
# Split the data into training and testing sets (80% training, 20% testing)
```

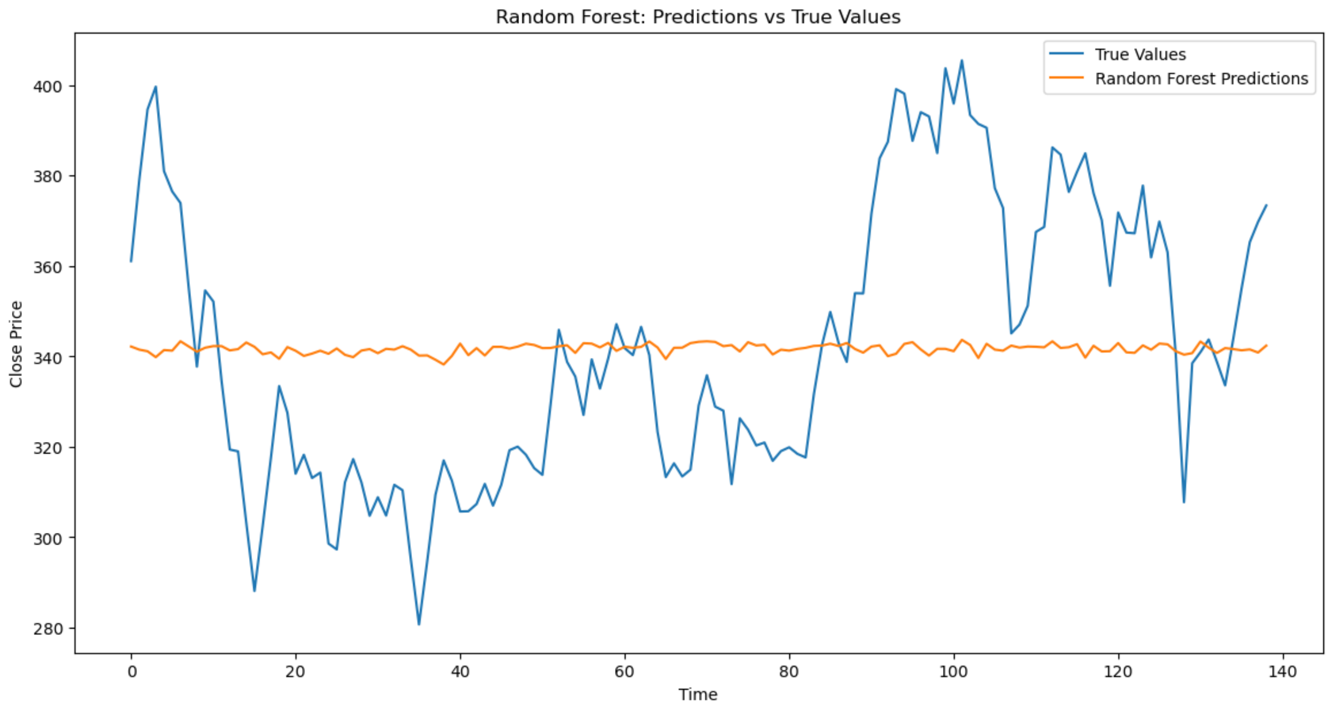
```
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# 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"LSTM Mean Squared Error: {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]

# Print some predictions and true values
print("Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_scaled[i]}, True Value: {y_test_scaled[i]}")
# 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()

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
In [49]:
# Function to create sequences
def create_sequences(data, sequence_length):
    sequences = []
    labels = []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i + sequence_length])
        labels.append(data[i + sequence_length, 3])  # Target is 'Close' price
    return np.array(sequences), np.array(labels)

# Create sequences
sequence_length = 30
X, y = create_sequences(data.values, sequence_length)

# Reshape X to 2D array for tree-based models
X_reshaped = X.reshape(X.shape[0], -1)

# Split the data into training and testing sets (80% training, 20% testing)
train_size = int(len(X_reshaped) * 0.8)
X_train, X_test = X_reshaped[:train_size], X_reshaped[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Train and evaluate the Decision Tree model
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
mse_dt = mean_squared_error(y_test, y_pred_dt)
print(f"Decision Tree Mean Squared Error: {mse_dt}")
```

```
# Train and evaluate the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Mean Squared Error: {mse_rf}")

# Print some predictions and true values for both models
print("\nDecision Tree Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_dt[i]}, True Value: {y_test[i]}")

# 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()
```

2. Result:







3. Interpretation:

The LSTM model's predictions exhibit a smoother, less noisy pattern compared to the true values. The model effectively captures the overall upward and downward trends but tends to lag behind the actual values in certain regions. While it handles major trends reasonably well, it struggles with short-term fluctuations and more volatile behavior observed in the true values. LSTM models excel at capturing long-term dependencies and trends in time series data, making them a suitable choice for datasets where

the primary focus is on long-term trends rather than short-term fluctuations.
The Decision Tree model's predictions remain relatively constant, failing to capture the actual fluctuations in the true values. Unlike the LSTM model, the Decision Tree model provides a flat prediction that does not follow the actual trends of the data. Consequently, the model performs poorly in this context, as it doesn't capture the variability and trends of the actual data. Decision Trees are typically not well-suited for continuous time series forecasting because they tend to overfit the training data and generalize poorly to new data. While they may perform better in ensemble methods like Random Forests or Gradient Boosting, they are generally not ideal for capturing complex, continuous patterns in time series data.
The Random Forest model's predictions remain relatively constant and fail to capture the actual fluctuations in the true values. This results in a flat prediction line that does not follow the trends present in the true data. Random Forests, while useful in various applications, are typically not well-suited for continuous time series forecasting. They tend to overfit the training data and generalize poorly to new, unseen data.

SUMMARY OF ALL THE MODELS USED:

Serial Number	Model Name	Loss Function
1	*Holt-Winters model*	413.5710124261197
2	*ARIMA daily forecast*	196.06579885940837
3	*ARIMA monthly forecast*	15376.535646062805
4	*SARIMA forecast*	1044.936027084568
5	*LSTM model*	0.0022016095463186502
6	*Decision Tree*	924.2985601325141
7	*Random Forest*	909.3356349840859

In the analysis of various time series forecasting models for stock price prediction, several models were evaluated using their respective loss functions to determine their performance. Among these models, the Long Short-Term Memory (LSTM) model demonstrated the lowest loss function value (0.0022), indicating superior performance in predicting stock prices compared to other models. The loss function measures the difference between the predicted and actual values, providing a quantitative assessment of a model's accuracy. A lower loss function value signifies a more accurate model, making it a crucial metric for model evaluation.

The LSTM model's advantage lies in its ability to capture long-term dependencies and patterns in sequential data, making it highly effective for stock price prediction. Stock prices often exhibit complex temporal dependencies and patterns, which traditional models might not capture effectively. LSTM networks, with their memory cells, can remember and leverage these patterns, leading to more accurate predictions.

Using the loss function to evaluate model performance ensures that the selected model can reliably predict future stock prices, which is essential for making informed investment decisions. The LSTM model, with its low loss function, provides accurate and robust forecasts, making it an excellent choice for stock price prediction.

RESULTS AND INTERPRETATION

e) Clean the data, check for outliers and missing values, interpolate the data if there are any missing values, and plot a line graph of the data neatly named. Create a test and train data set out of this data.

1. Code:

```
# Load the data
data_df <- read.csv("jupiter_wagons_data.csv")

# Display the first few rows of the data
head(data_df)

data_df$Date <- as.Date(data_df$Date)

# Display the first few rows of the data
head(data_df)

# Check for missing values
print("Missing values before interpolation:")
print(sum(is.na(data_df$Close)))

# Interpolate missing values
data_df$Close <- na.interp(data_df$Close)

# Check for missing values again
print("Missing values after interpolation:")
print(sum(is.na(data_df$Close)))

# Plot the data
ggplot(data_df, aes(x = Date, y = Close)) +
  geom_line() +
  labs(title = "Jupiter Wagons Close price", x = "Date", y = "Close Price")

# Split the data into training and test sets
set.seed(123)
train_index <- sample(nrow(data_df), 0.7 * nrow(data_df))
train_data <- data_df[train_index, ]
test_data <- data_df[-train_index, ]

# Display the sizes of the train and test datasets
print(paste("Training data size:", nrow(train_data)))
print(paste("Test data size:", nrow(test_data)))

# Display the first few rows of the train and test datasets
print("Training data:")
```

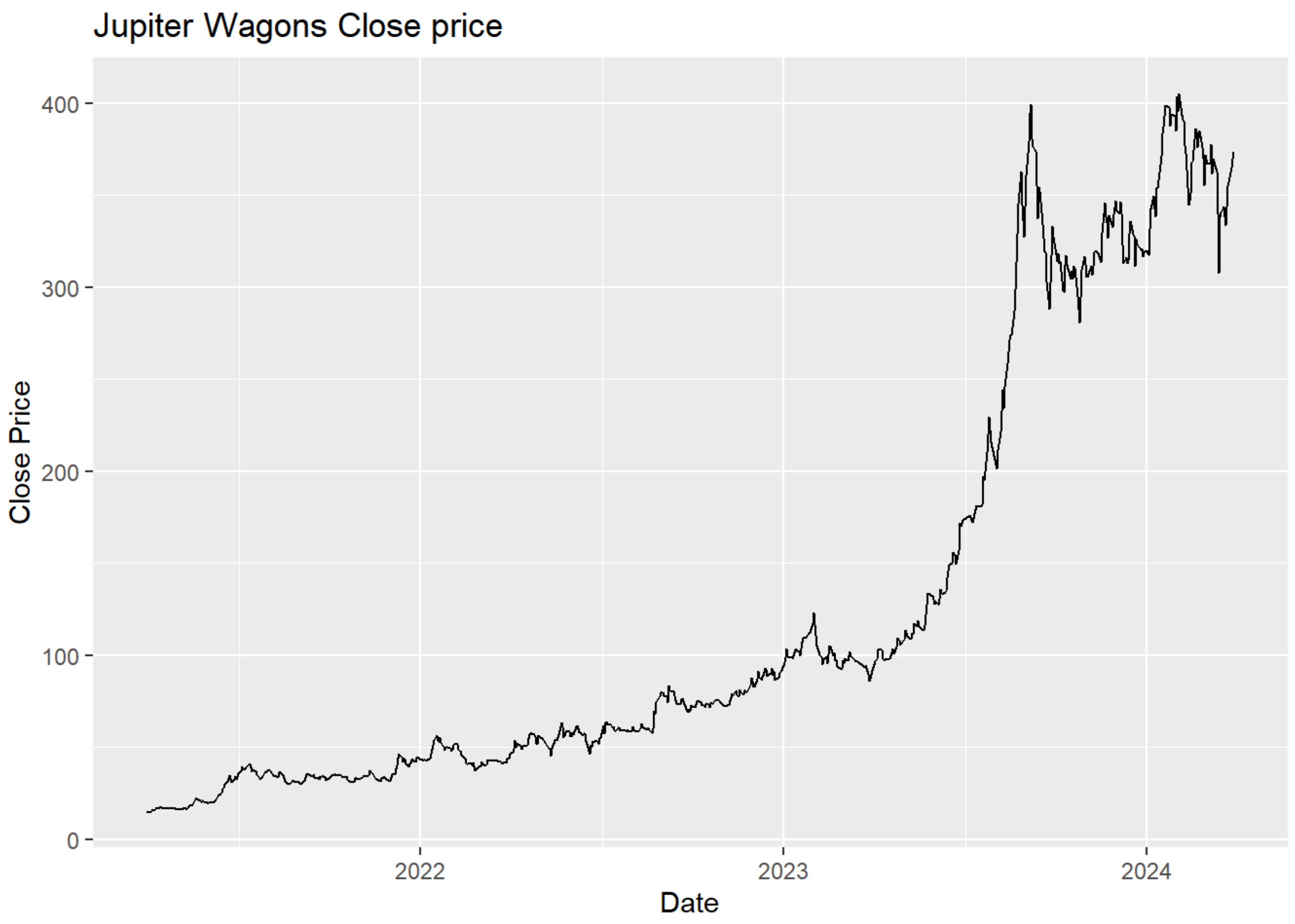
```
head(train_data)
print("Test data:")
head(test_data)
```

2. Result:

```
##          Date  Open  High   Low Close Adj.Close Volume
## 1 2021-04-01 15.60 15.60 15.15 15.15  15.11698    680
## 2 2021-04-05 15.15 15.40 14.60 15.00  14.96731   9550
## 3 2021-04-06 15.35 15.75 14.85 15.75  15.71567   2301
## 4 2021-04-07 15.30 16.00 15.10 15.55  15.51611   2708
## 5 2021-04-08 15.95 16.20 15.95 16.00  15.96513   4425
## 6 2021-04-09 16.15 16.80 16.15 16.80  16.76338   8899
```

```
## [1] "Training data size: 505"
```

```
## [1] "Test data size: 217"
```



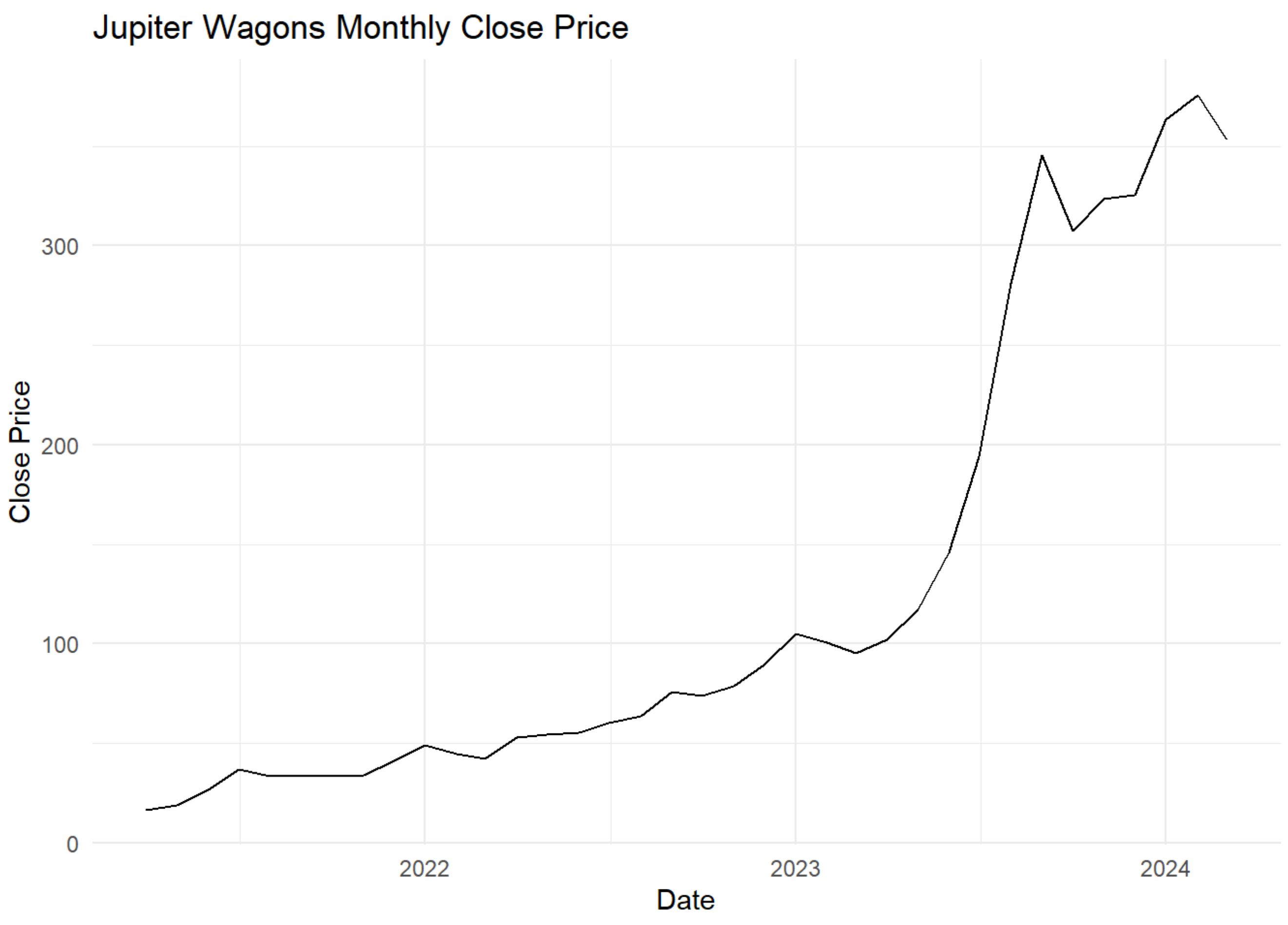
3. Interpretation:
    - The **training data** size is 505, which means 505 instances (rows) of data are being used to train your model. This is the data that the model learns from.
    - The **test data** size is 217, which means 217 instances of data are being used to evaluate the model's performance after it has been trained. This data is not seen by the model during training, ensuring an unbiased evaluation of its performance.
    - The stock graph for Jupiter Wagons displays the company's closing price over three years, from approximately May 2021 to May 2024. Initially, the stock price exhibits a slow upward trend with minor fluctuations, staying relatively stable below 100 units until early 2023. In mid-2023, there is a significant surge in the stock price, marking a period of notable growth. This upward movement peaks around late 2023, reaching nearly 400 units. After this peak, the stock shows some volatility with considerable fluctuations but maintains an overall upward trend, with prices staying above 300 units. Analyzing this stock graph offers several advantages. Investors can pinpoint key growth periods and understand the overall stock trajectory, helping them make informed investment decisions. The sharp rise in mid-2023 suggests a possible breakthrough or positive development within the company, which likely attracted more investors.

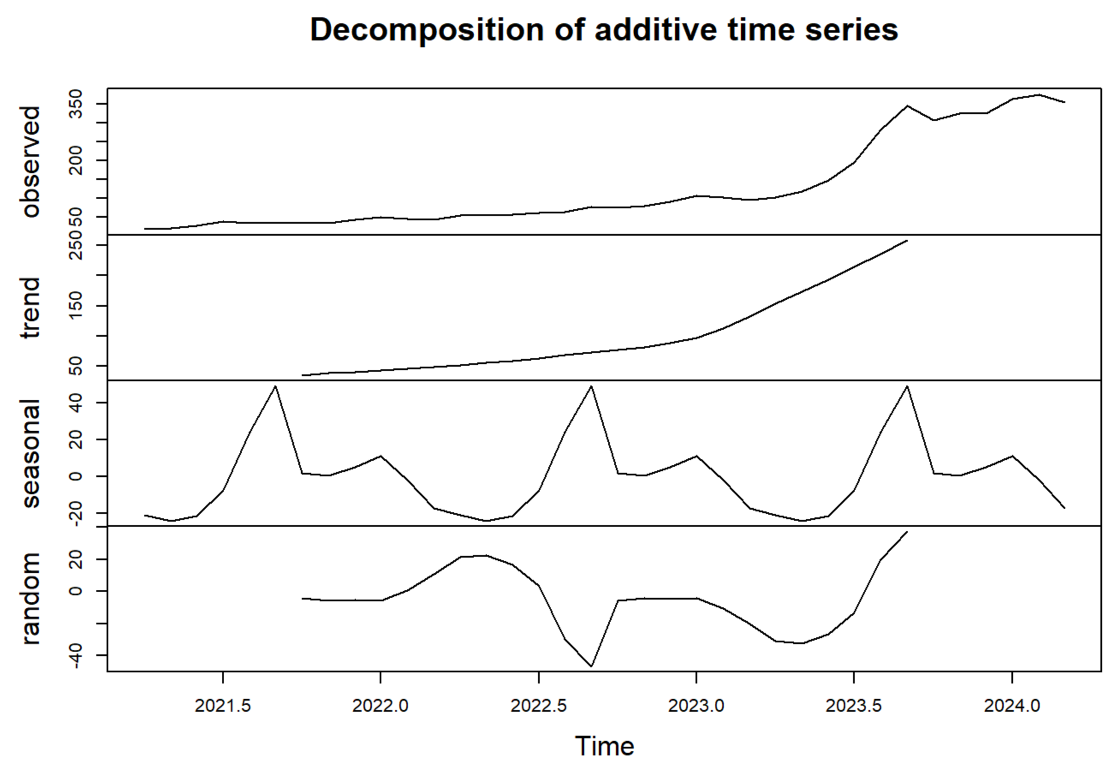
f) Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

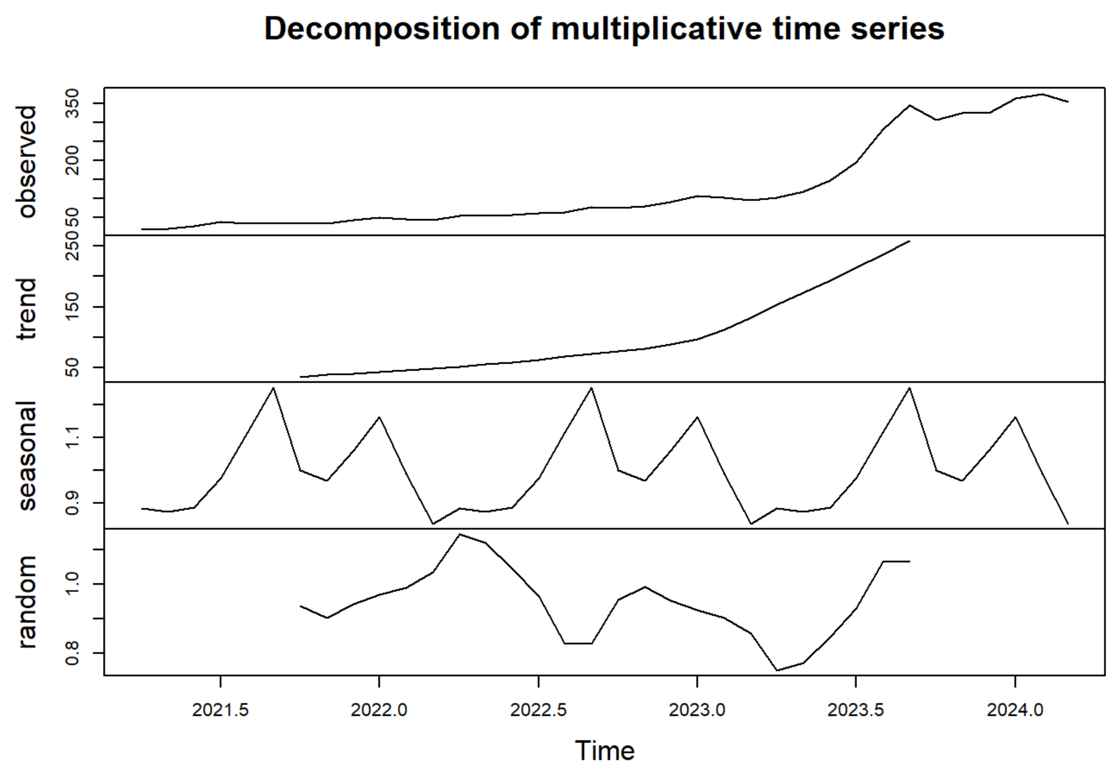
1. Code:

```
# Convert the data to monthly frequency
monthly_data <- aggregate(Close ~ format(Date, "%Y-%m"), data_df, mean)
colnames(monthly_data) <- c("Month", "Close")

# Ensure 'Month' is in date format and no missing values in 'Close'
monthly_data$Month <- as.Date(paste0(monthly_data$Month, "-01"), format="%Y-%m-%d")

# Check for missing or non-finite values in 'Close'
if (any(!is.finite(monthly_data$Close))) {
  stop("There are non-finite values in the 'Close' column.")
}

# Plot the monthly data
ggplot(data = monthly_data, aes(x = Month, y = Close)) +
  geom_line() +
  labs(title = "Jupiter Wagons Monthly Close Price", x = "Date", y = "Close Price") +
  theme_minimal()

# Convert the monthly data to a time series object
monthly_ts <- ts(monthly_data$Close, start = c(as.numeric(format(min(monthly_data$Month), "%Y")), as.numeric(format(min(monthly_data$Month),
"%m"))), frequency = 12)

# Decompose the time series using additive model
additive_decompose <- decompose(monthly_ts, type = "additive")
plot(additive_decompose)

# Decompose the time series using multiplicative model
multiplicative_decompose <- decompose(monthly_ts, type = "multiplicative")
plot(multiplicative_decompose)
```

2. Result:







3. Interpretation:

The graph of Jupiter Wagons' monthly closing price shows a steady increase from May 2021 to around mid-2023, with the price remaining below 100 units during this period. From mid-2023 onwards, there is a significant and rapid rise in the stock price, peaking above 350 units by early 2024. This sharp increase suggests a period of accelerated growth, possibly due to favorable business developments or market conditions.

The decomposition of Jupiter Wagons' monthly closing price into its components in the additive model—trend, seasonal, and residual—reveals distinct patterns. The trend component shows a steady and pronounced upward movement, particularly from early 2023 onwards, indicating a long-term increase in the stock's value. The seasonal component fluctuates periodically, suggesting recurring patterns within each year, likely influenced by cyclical factors. The residual component highlights irregular variations around the trend and seasonal components, with noticeable deviations from mid-2023, possibly due to unexpected market events or company-specific news.

For the multiplicative model, a time series decomposition plot breaks down the time series into its constituent components: observed, trend, seasonal, and residual. The observed component, shown in the top panel, indicates an overall increase over the observed period. The trend component, displayed in the second panel, highlights a steady upward trajectory. The seasonal component, illustrated in the third panel, reveals recurring patterns within the data. The residual component, shown in the bottom panel, represents the remaining variations after removing the trend and seasonal components, indicating relatively low variance.

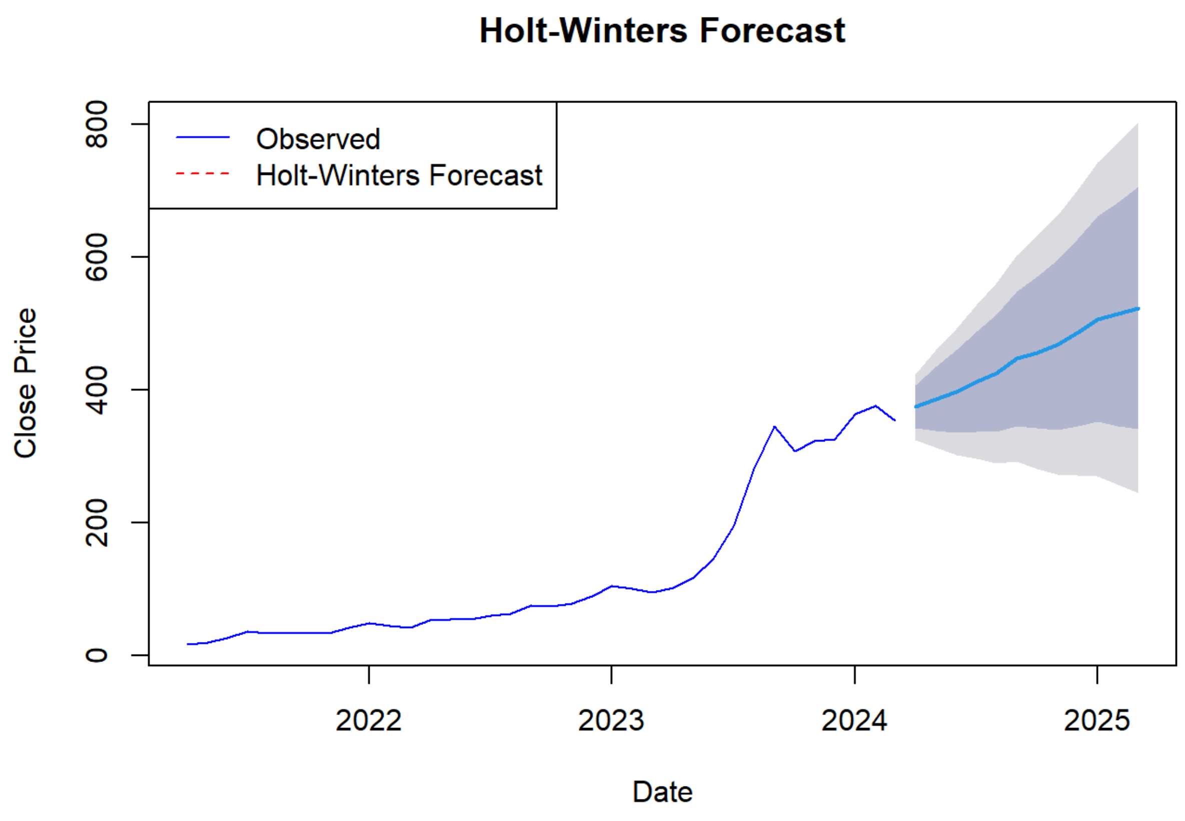
g) Univariate Forecasting

1. Code:

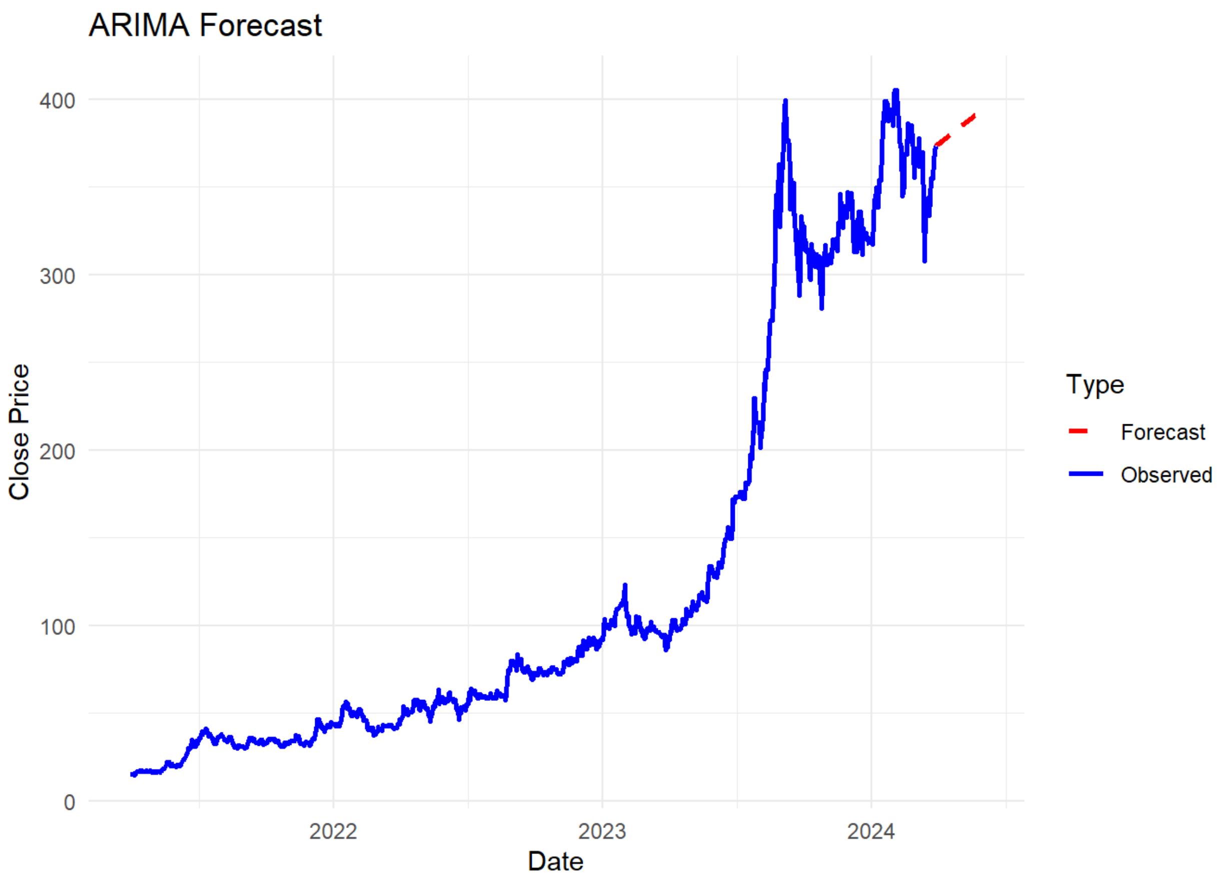
```
# Fit the Holt-Winters model
holt_winters_model <- HoltWinters(monthly_ts, seasonal = "additive")

# Forecast for the next year (12 months)
holt_winters_forecast <- forecast(holt_winters_model, h = 12)

# Plot the forecast
plot(holt_winters_forecast, main = "Holt-Winters Forecast", xlab = "Date", ylab = "Close Price")
lines(monthly_ts, col = "blue")
legend("topleft", legend = c("Observed", "Holt-Winters Forecast"), col = c("blue", "red"), lty = 1:2)

# Convert the data to daily frequency
daily_data <- data_df %>%
  complete(Date = seq.Date(min(Date), max(Date), by="day")) %>%
  fill(Close, .direction = "downup")

# Interpolate missing values in the daily data (if any)
daily_data$Close <- na.approx(daily_data$Close)

# Display the first few rows of the daily data
```

```r
head(daily_data)

# Interpolate missing values in the daily data (if any)
daily_data$Close <- na.approx(daily_data$Close)

# Drop unnecessary columns
daily_data <- daily_data %>%
  select(Date, Close)

# Display the first few rows of the daily data
head(daily_data)
# Save the daily data to a new CSV file
write.csv(daily_data, file = "daily_jupiter_wagons_data_R.csv", row.names = FALSE)

# Convert to time series object
daily_ts <- ts(daily_data$Close, frequency = 365, start = c(as.numeric(format(min(daily_data$Date), "%Y")), as.numeric(format(min(daily_data$Date),
"%j"))))

# Fit the ARIMA model
arima_model <- auto.arima(daily_ts)

# Diagnostic checks for ARIMA model
tsdiag(arima_model)
summary(arima_model)

arima_forecast <- forecast(arima_model, h = 63)

# Prepare data for plotting
forecast_df <- data.frame(Date = seq(max(daily_data$Date) + 1, by = "day", length.out = 63),
                          Close = as.numeric(arima_forecast$mean),
                          Type = "Forecast")

# Combine observed and forecasted data
daily_data$Type <- "Observed"
forecast_df$Type <- "Forecast"
plot_data <- rbind(daily_data, forecast_df)

print(forecast_df)

# Plot the ARIMA forecast with observed data
ggplot() +
  geom_line(data = plot_data, aes(x = Date, y = Close, color = Type, linetype = Type), size = 1) +
  labs(title = "ARIMA Forecast", x = "Date", y = "Close Price") +
  scale_color_manual(values = c("Observed" = "blue", "Forecast" = "red")) +
  scale_linetype_manual(values = c("Observed" = "solid", "Forecast" = "dashed")) +
  theme_minimal()
```

2. Result:



Holt-Winters Forecast
Observed
Holt-Winters Forecast
Close Price
0
200
400
600
800
2022
2023
2024
2025
Date



ARIMA Forecast
Close Price
0
100
200
300
400
2022
2023
2024
Date
Type
Forecast
Observed

3. Interpretation:

The Holt-Winters model captures both the trend and seasonality in the data. Its forecast shows a continuous increase in close prices with moderate volatility, demonstrating the model’s ability to adapt to changes in trends while maintaining an overall upward trajectory. This suggests that the model expects the overall growth in prices to continue, though with some fluctuations. The smoothness of the forecast indicates the model's emphasis on long-term patterns rather than short-term noise.

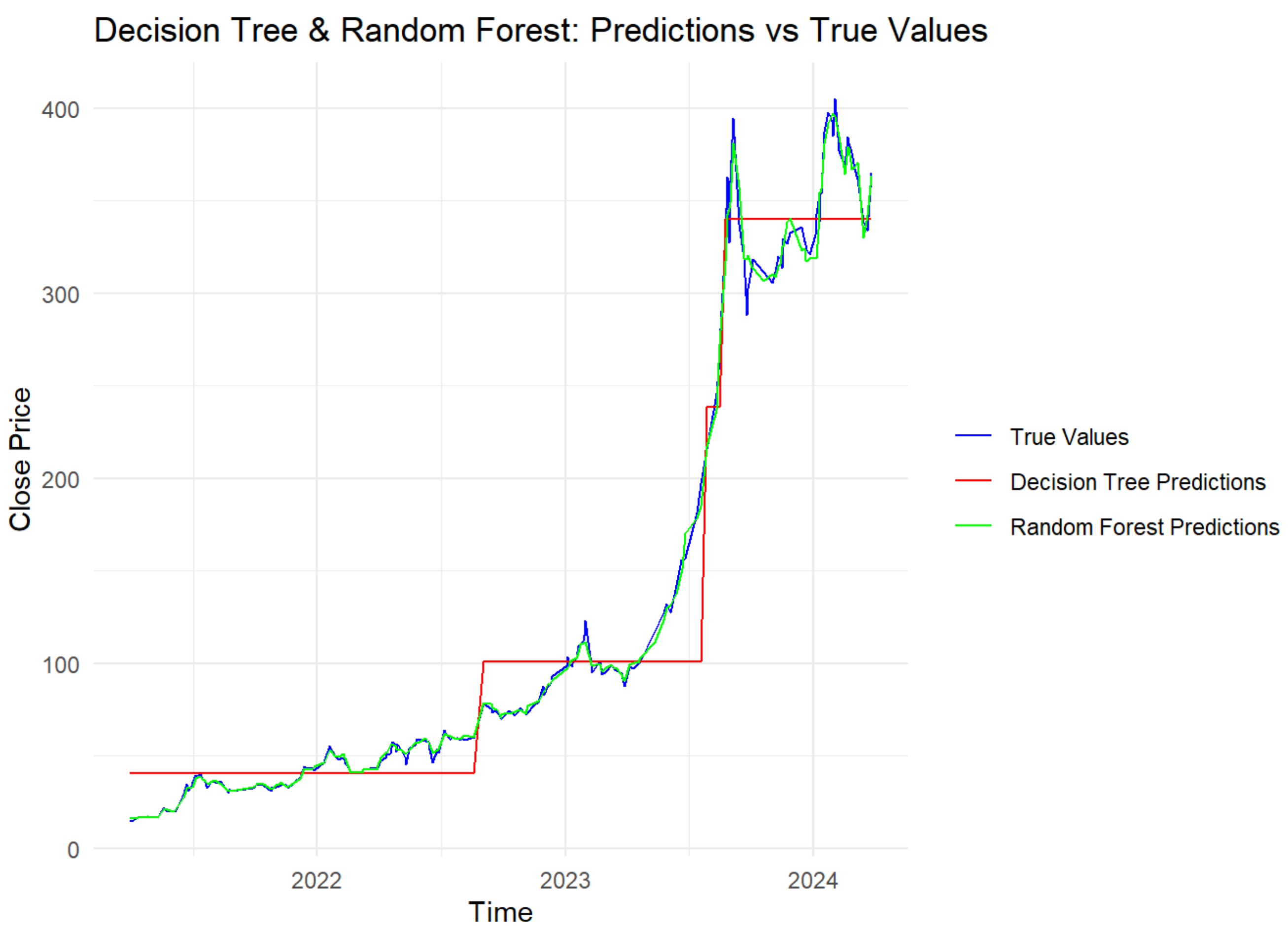
The ARIMA model, applied to monthly daily data, provides a more stable forecast with fewer fluctuations compared to the Holt-Winters model. The forecast indicates a leveling off of close prices following a period of rapid growth, suggesting the model anticipates potential stabilization or a slight decline in prices. This reflects the ARIMA model's strength in capturing underlying trends without being overly influenced by short-term seasonal variations, making it well-suited for data where the primary focus is on trends rather than seasonal patterns.

h) Multivariate Forecasting

1. Code:

```
# Train a decision tree model
model <- rpart(Close ~ Date, data = train_data, method = "anova")
predictions_dt <- predict(model, test_data)

# Display the first few rows of the predictions
head(predictions_dt)

# Train a random forest model
model_rf <- randomForest(Close ~ Date, data = train_data)
predictions_rf <- predict(model_rf, test_data)

# Display the first few rows of the predictions
head(predictions_rf)

# Plot predictions vs true values
test_data$Predictions_DT <- predictions_dt
test_data$Predictions_RF <- predictions_rf

ggplot(test_data, aes(x = Date)) +
  geom_line(aes(y = Close, color = "True Values")) +
  geom_line(aes(y = Predictions_DT, color = "Decision Tree Predictions")) +
  geom_line(aes(y = Predictions_RF, color = "Random Forest Predictions")) +
  labs(title = "Decision Tree & Random Forest: Predictions vs True Values",
       x = "Time",
       y = "Close Price") +
scale_color_manual("",
            breaks = c("True Values", "Decision Tree Predictions", "Random Forest Predictions"),
            values = c("blue", "red", "green")) +
theme_minimal()
```

2. Result:



3. Interpretation:

The blue line represents the true values, showing actual close prices over time, with a notable increase starting around early 2023 and continuing into 2024. This period also exhibits significant fluctuations, especially during the peak period in 2023-2024.
The Decision Tree model predictions, depicted by the red line, exhibit step-like changes characteristic of decision trees' piecewise constant prediction nature. While the model captures the general upward trend, it fails to predict the finer fluctuations and volatility seen in the true values. The predictions tend to lag behind significant changes in the actual values, adjusting only after considerable changes in the trend. In contrast, the Random Forest model predictions, shown by the green line, are much smoother and closely follow the true values. This model captures both the overall trend and the finer fluctuations more accurately than the Decision Tree model. It

demonstrates better adaptability to sudden changes and volatility in the data.
Comparing the two models, the Decision Tree model, although following the general trend, lacks the ability to capture short-term fluctuations and shows a delayed response to rapid changes. The Random Forest model, on the other hand, shows superior performance in tracking both the overall trend and the finer details, indicating its effectiveness in handling complex patterns in the data.
Overall, both models capture the significant upward trend in close prices from 2023 to 2024, with the Random Forest model providing a more precise alignment with the true values. The Random Forest model excels in predicting both the trend and the short-term fluctuations, demonstrating its robustness and higher predictive accuracy in this scenario.