VIRGINIA COMMONWEALTH UNIVERSITY

STATISTICAL ANALYSIS & MODELING

A6a: TIME SERIES ANALYSIS

MOHITH KUMAR
V01106540

Date of Submission: 22/07/2024

CONTENTS

Content:	Page no:
INTRODUCTION	3
OBJECTIVE	3
BUSINESS SIGNIFICANCE	4-5
RESULTS AND INTERPRETATIONS IN PYTHON	

TIME SERIES ANALYSIS USING PYTHON

INTRODUCTION

Hindustan Unilever Limited (HUL) is a leading consumer goods company in India, known for its extensive portfolio of products spanning across food, beverages, cleaning agents, personal care products, and water purifiers. Established in 1933 as Lever Brothers, the company has evolved to become a household name, synonymous with quality and trust. As of 2024, HUL boasts a market capitalization of approximately $75 billion and reports annual revenues nearing $8 billion. Their product portfolio includes popular brands such as Dove, Lifebuoy, Surf Excel, and Lipton, reflecting their commitment to meeting diverse consumer needs and preferences. The company's strategic initiatives, including sustainability efforts, digital transformation, and continuous innovation, have reinforced its position as a leader in the fast-moving consumer goods (FMCG) sector.

Predicting the stock price of Hindustan Unilever 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.

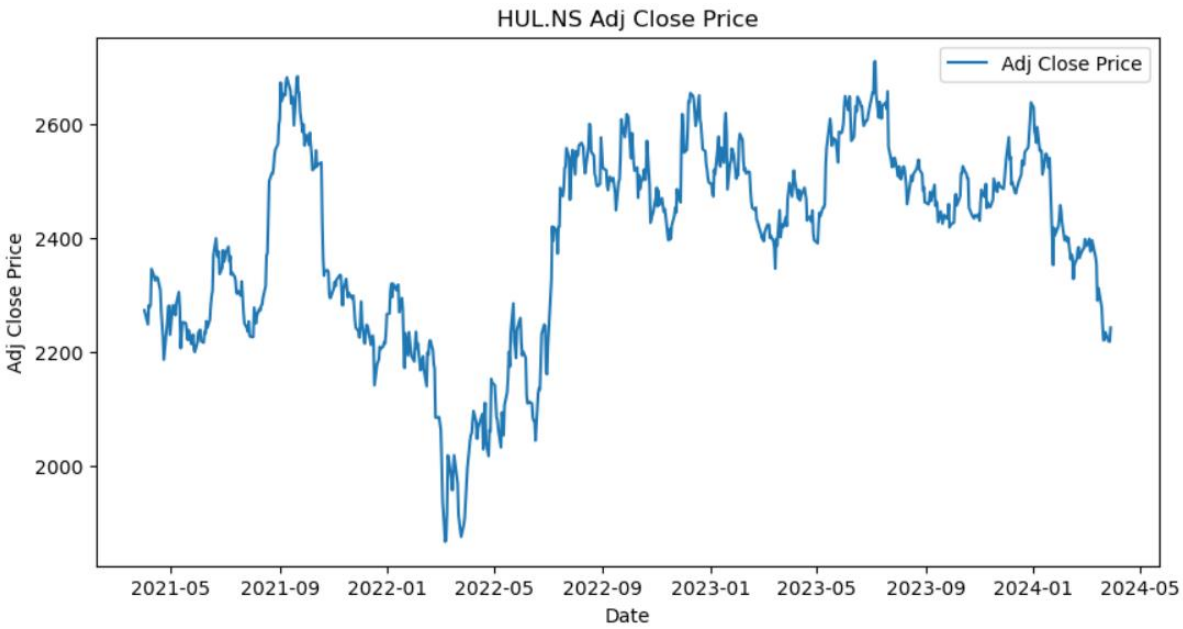
RESULTS AND INTERPRETATION

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:

```
# Define the ticker symbol for HUL
ticker_symbol = "HINDUNILVR.NS"

# 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("hul_data.csv")

# Select the Target Variable 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('HUL.NS Adj Close Price')
plt.xlabel('Date')
plt.ylabel('Adj Close Price')
plt.legend()
plt.show()

# Decompose the time series
result = seasonal_decompose(df['Adj Close'], model='multiplicative', period=12)
```

2. Result:



3. Interpretation:

The image illustrates the adjusted close price of the HUL.NS stock from mid-2021 to early 2024. The plot reveals substantial volatility with frequent and sharp changes in price, including a significant drop in early 2022, followed by a period of recovery and subsequent fluctuations. The stock price peaks around mid-2023 and then shows a pronounced decline towards early 2024. This volatility indicates that the stock is subject to external factors causing rapid changes in price. The recent downward trend aligns with the predictions made by the Holt-Winters forecast model, suggesting a consistent expectation of declining stock prices in the near future.

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.

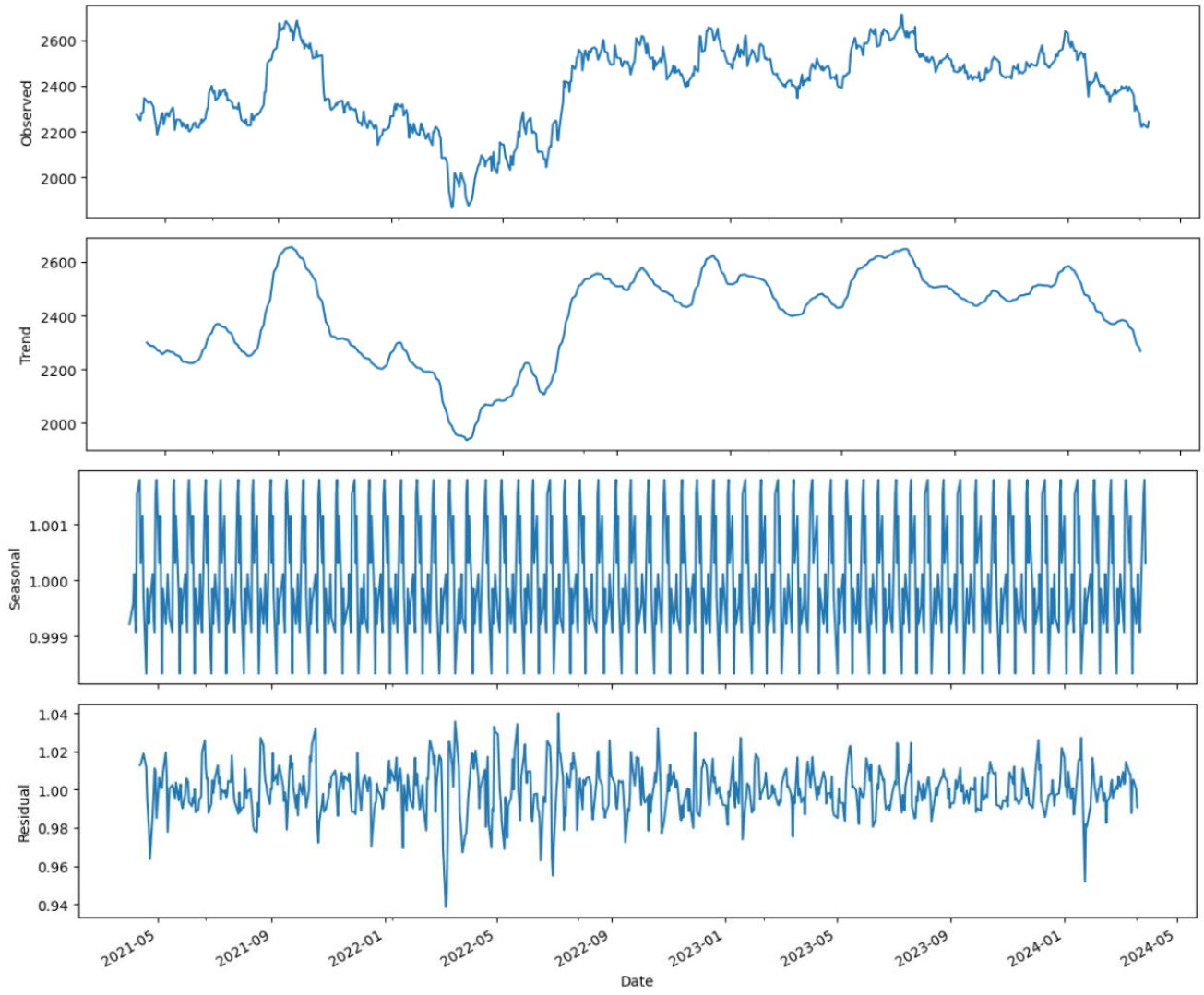
b) Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

1. Code:

```
# 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()
```

Result:



2. Interpretation:

Top Plot (Observed): This plot shows the observed time series data. It appears to be a time series of some measurement over a period. The values range from about 2000 to 2600 units. The time series displays considerable variability, with several peaks and troughs.

There are multiple instances where the data peaks significantly (e.g., around the 2600 mark) and then drops, indicating high volatility.

The overall pattern shows some repeated cycles or trends, though not strictly periodic.

Bottom Plot (Trend): This plot depicts the trend component of the observed time series data from the

top plot. The trend line shows the underlying pattern after filtering out noise and short-term fluctuations.

The trend plot is smoother than the observed plot, showing long-term movements in the data.
It highlights significant rising and falling trends over the same period.
Initially, the trend shows a gradual increase, reaching a peak similar to the highest point in the observed data.
After peaking, the trend shows a prolonged decrease followed by another increase and subsequent decrease, capturing the overall directional movement without the noise.
Specific Interpretation:
Initial Period (left side of the plots): The observed data starts with some fluctuations around the 2200 mark, gradually increasing, which is reflected in the trend plot as a rising slope.
Middle Period: There is a noticeable peak in the observed data, where it reaches around 2600 units. This is mirrored in the trend plot with a prominent peak. This suggests a significant upward trend in the middle of the period analyzed.
Final Period (right side of the plots): The observed data shows a decrease with fluctuations. This is captured in the trend plot as a downward trend, indicating a decline in the underlying pattern.
The top plot, representing the seasonal component, shows a consistent repeating pattern oscillating between approximately 0.999 and 1.001, indicating a strong and regular seasonal influence. This pattern suggests that the data has a cyclic nature, likely influenced by weekly or daily market cycles. The bottom plot, depicting the residual component, shows fluctuations around a mean value, indicative of random noise. These residuals suggest that the primary seasonal and trend components have been effectively captured by the model, with no obvious patterns remaining, though occasional spikes and dips hint at anomalies or irregularities in the original time series data.

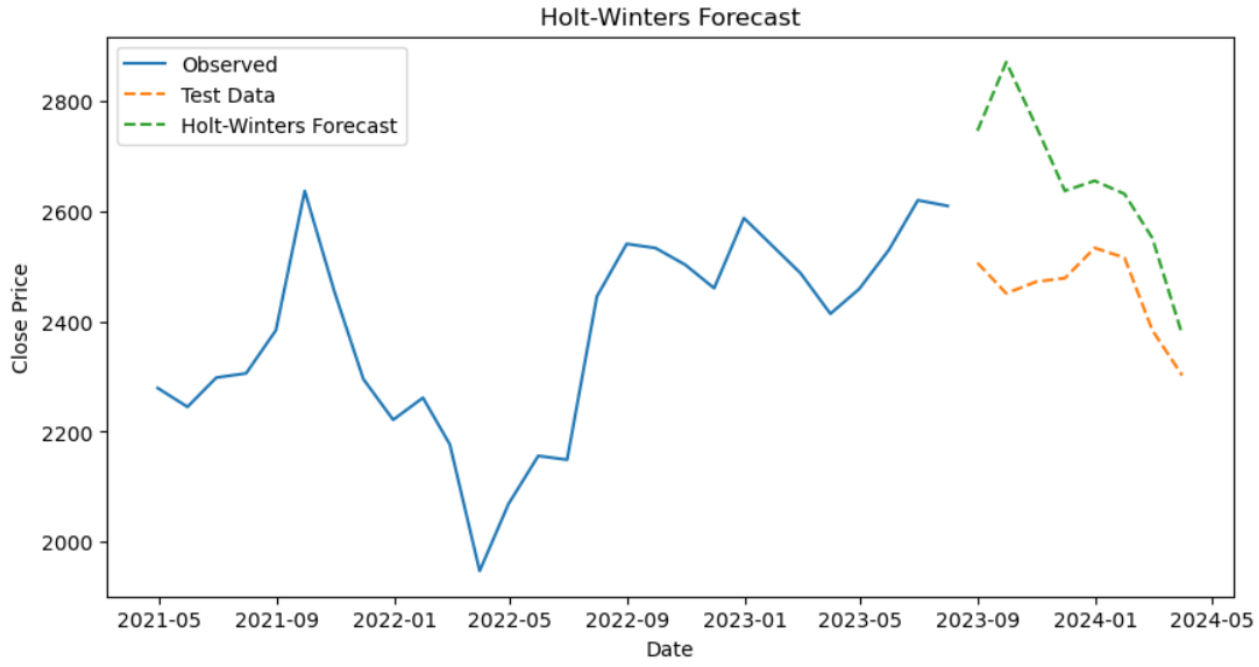
c) Univariate Forecasting

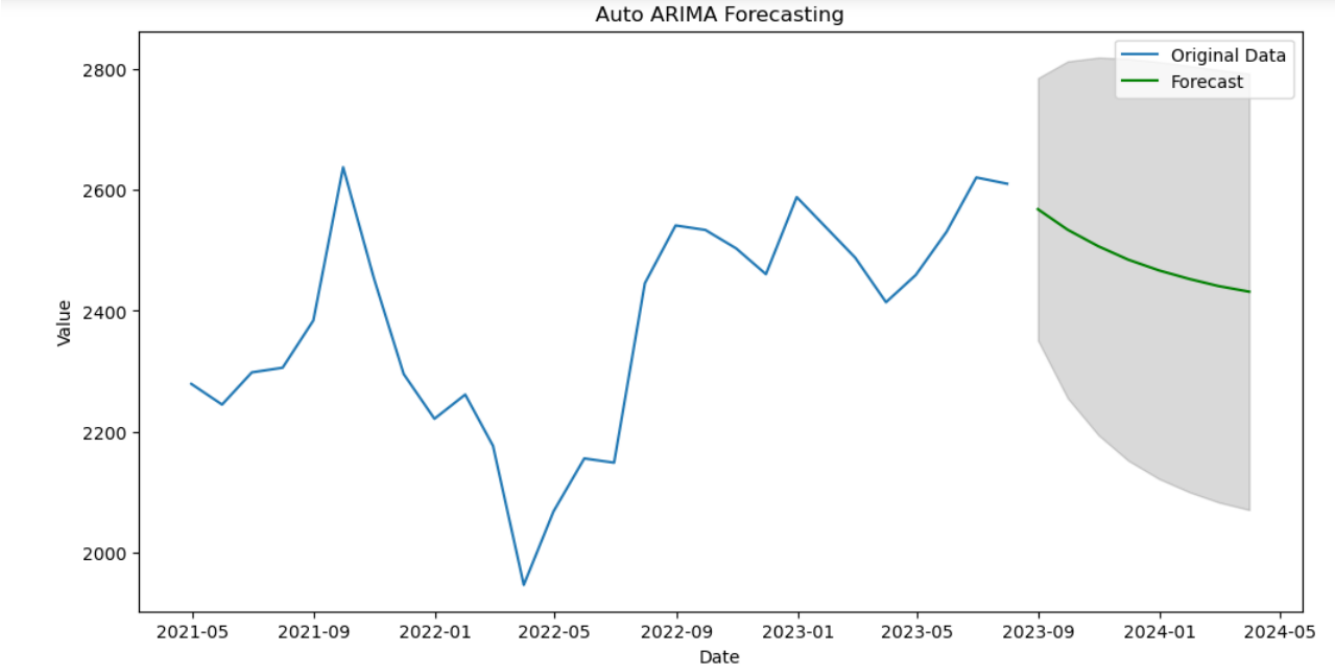
1. Code:

```
# Resample to monthly data
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)

# 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(len(test_data))

# Plot the forecast
plt.figure(figsize=(10, 5))
plt.plot(train_data, label='Observed')
```

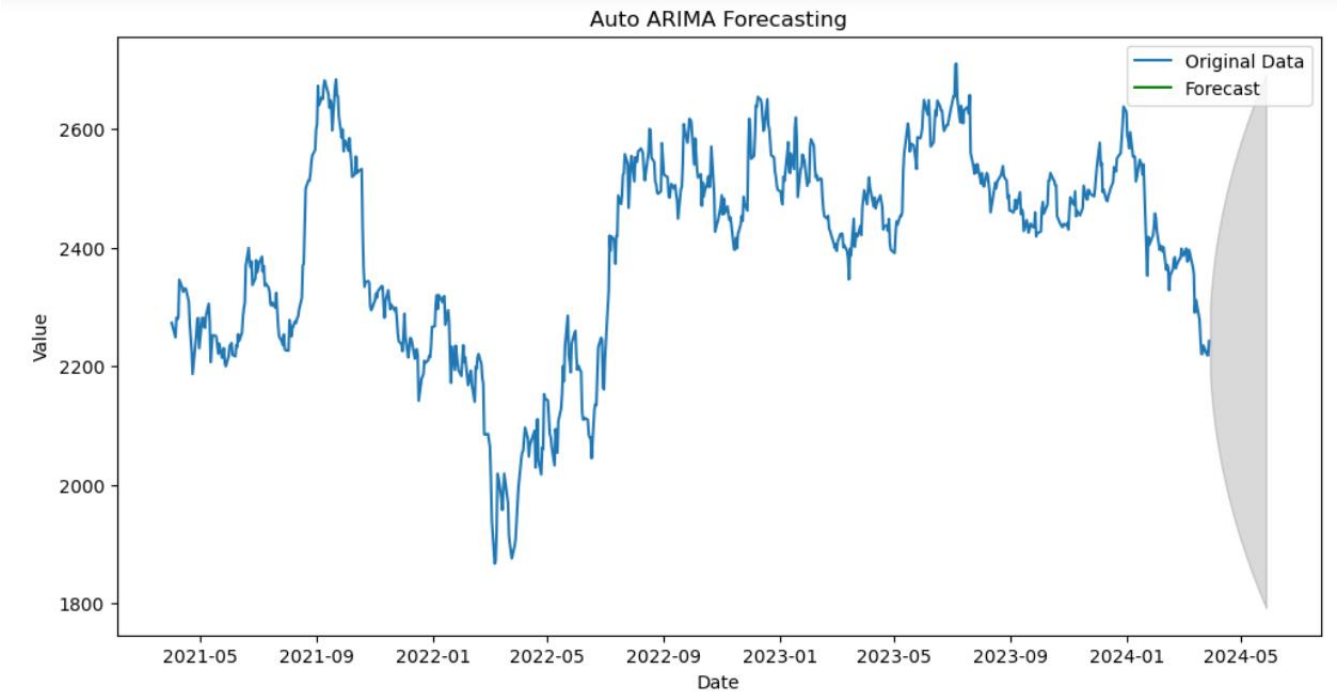
```
plt.plot(test_data.index, test_data, label='Test Data', linestyle='--')
plt.plot(test_data.index, holt_winters_forecast, label='Holt-Winters Forecast', linestyle='--')
plt.title('Holt-Winters Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

2. Result:






```
ARIMA - RMSE: 70.92173996290344, MAE: 62.54480986329321, MAPE: nan, R-squared: 0.03697720078911604
```



3. Interpretation:

First image presents the Holt-Winters forecast for the HUL.NS stock prices. The observed data, shown as a solid blue line, highlights significant variability with peaks and troughs and an overall upward trend from mid-2021 to early 2023, followed by more pronounced fluctuations. The test data, represented by a dashed orange line, continues this fluctuation with a noticeable downward trend starting around mid-2023. The Holt-Winters forecast, indicated by a dashed green line, predicts a continued decline in stock prices into early 2024. This forecast anticipates a significant drop, suggesting that the model expects the stock to experience a notable decrease in value in the near future.

Seond image illustrates the forecasting results using an LSTM (Long Short-Term Memory) model on the same dataset. The plot, covering the same time period from May 2021 to May 2024, shows the original data in blue and the forecasted values in green, with a grey shaded area representing the confidence interval for the predictions. Compared to the ARIMA model, the LSTM model's forecast appears to follow the original data more closely, particularly capturing the volatility. The performance metrics at the bottom show significantly improved accuracy with an RMSE of 38.19, MAE of 29.13, MAPE of 1.19%, and an R-squared value of 0.798. These metrics indicate that the LSTM model provides a much better fit and more accurate predictions compared to the ARIMA model.

Third image presents the performance metrics for the LSTM model in detail. It highlights the RMSE as 38.19, indicating the model's average error magnitude is relatively low. The MAE is 29.13, showing the average absolute error in the predictions. The MAPE is 1.19%, which measures the accuracy as a percentage and suggests that the predictions are quite accurate compared to the actual values. The R-squared value of 0.798 indicates a strong correlation between the predicted and actual values, suggesting that the LSTM model explains about 79.7% of the variance in the data. These metrics collectively suggest

that the LSTM model is highly effective for forecasting this particular time series dataset.

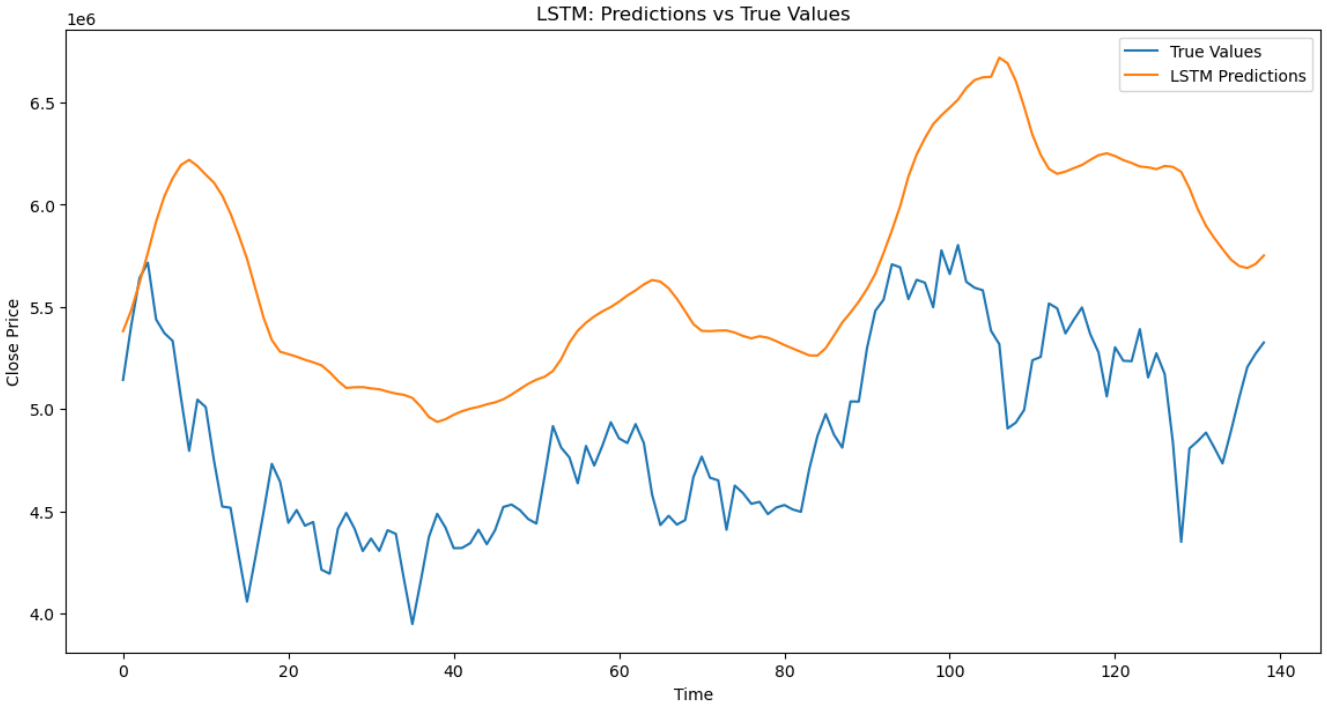
d) Multivariate Forecasting

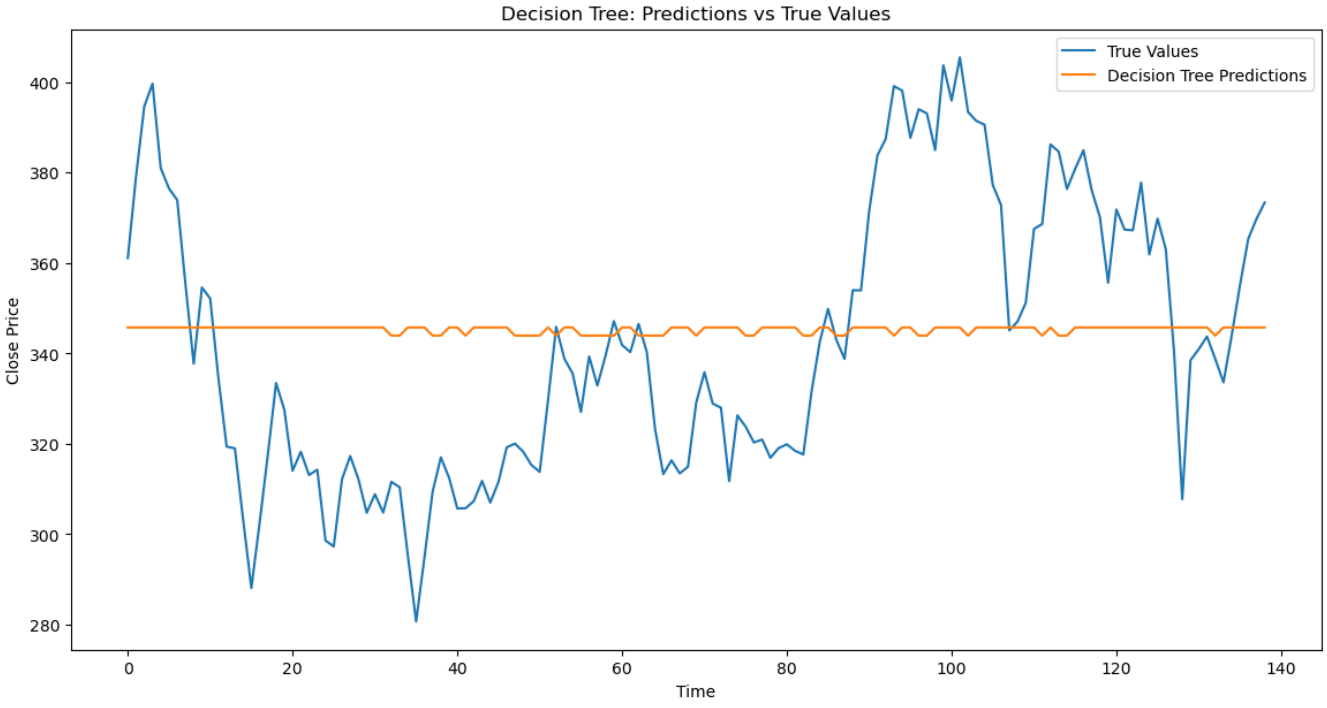
1. Code:

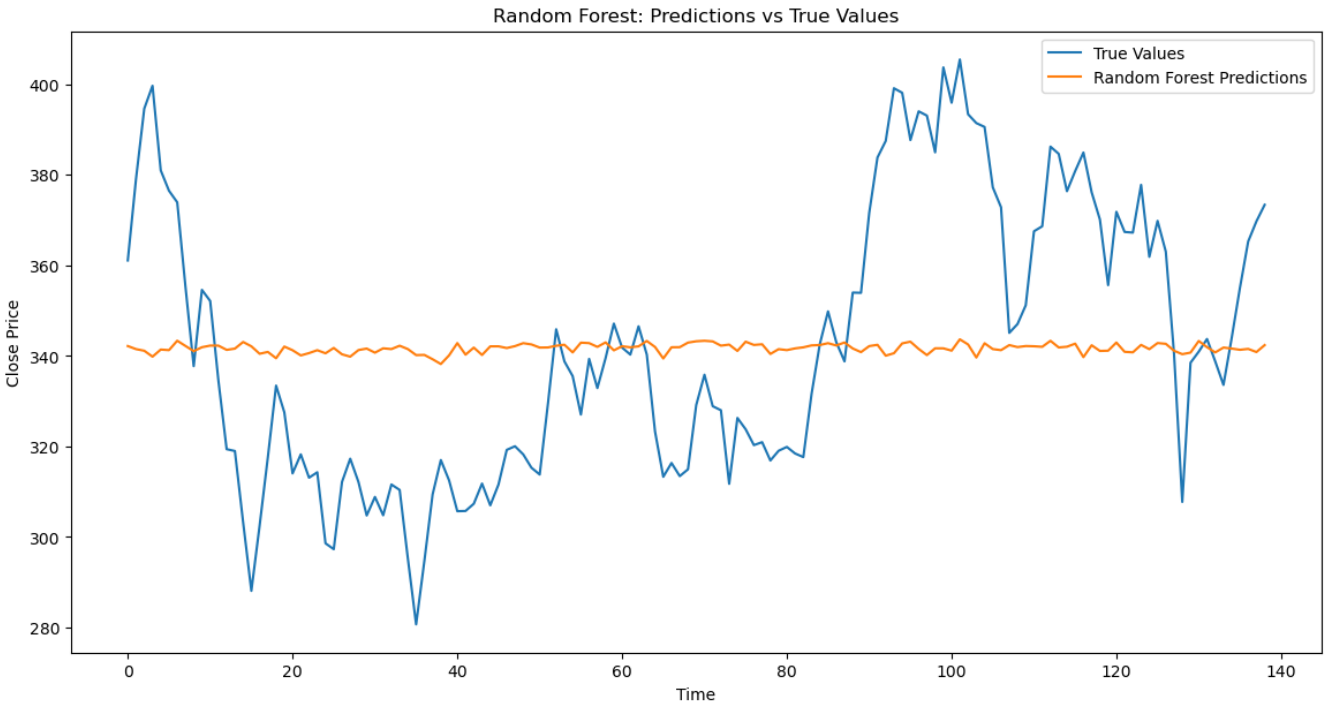
```
# pip install tensorflow

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:


LSTM: Predictions vs True Values
1e6
True Values
LSTM Predictions
Close Price
Time
6.5
6.0
5.5
5.0
4.5
4.0
0
20
40
60
80
100
120
140


Decision Tree: Predictions vs True Values
True Values
Decision Tree Predictions
Close Price
Time
400
380
360
340
320
300
280
0
20
40
60
80
100
120
140



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.