VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A6: TIME SERIES ANALYSIS

RIDDHI RUNGTA

V01107488

Date of Submission: 22-07-2024

CONTENTS

TIME SERIES ANALYSIS

INTRODUCTION

Larsen & Toubro (L&T) is a prominent Indian multinational conglomerate with diversified business interests in engineering, construction, manufacturing, technology, and financial services. Founded in 1938, L&T has grown to become one of the most respected and valuable companies in India. The company operates in over 30 countries worldwide and has a significant presence in sectors such as infrastructure, energy, hydrocarbon, defense, and information technology. L&T's reputation for delivering complex and large-scale projects has positioned it as a leader in the construction and engineering industry. As of 2023, L&T reported a revenue of approximately ₹1.48 trillion (around $18.5 billion USD), reflecting its robust growth and substantial market influence.

Predicting the stock price of L&T is of immense value to investors, financial analysts, and policymakers. Accurate stock price prediction models enable stakeholders to make informed decisions regarding buying, holding, or selling stocks, thus optimizing their investment portfolios. Utilizing various models such as time series analysis, machine learning algorithms, and econometric models can provide different perspectives and improve prediction accuracy. For instance, a model incorporating historical data, market trends, and economic indicators might predict L&T's stock performance more reliably. The application of these models can be quantified by their predictive power; for example, a well-calibrated machine learning model might achieve a prediction accuracy of over 80%, significantly reducing investment risks and enhancing returns. This predictive capability is crucial in a volatile market environment, helping investors anticipate market movements and adjust their strategies accordingly.

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

a) Data Cleaning and Preparation

Cleaning the data, checking for outliers, and missing values, and interpolating missing values are foundational steps in data preprocessing. This process ensures the dataset's integrity and reliability, leading to more accurate and meaningful analyses. Creating test and train datasets from the cleaned data enables model validation and performance assessment, ensuring that the model generalizes well to unseen data. Proper visualization, such as a neatly named line graph, aids in understanding data trends and patterns, which is crucial for making informed business decisions.

1. Accurate stock price predictions require clean and complete datasets. By handling outliers and missing values, companies can derive reliable insights and predictions, aiding in strategic investment decisions.
2. Clean data is essential for forecasting revenue trends accurately, helping businesses plan their budgets and resource allocations effectively.
3. Identifying and rectifying data anomalies ensures that analyses of customer behavior are accurate, leading to better customer service strategies and targeted marketing campaigns.

b) Time Series Decomposition

Converting data to a monthly format and decomposing the time series into its components using additive and multiplicative models helps businesses understand underlying patterns such as trend, seasonality, and residuals. This deeper insight into the data's structure enhances the accuracy of forecasting models, leading to better strategic planning and decision-making.

1. Decomposing sales data into trend, seasonal, and residual components allows businesses to identify underlying factors affecting sales and adjust marketing strategies accordingly.
2. Seasonal patterns in demand helps businesses optimize inventory levels, reducing costs and improving customer satisfaction.
3. Decomposing financial time series data aids in identifying long-term growth trends and short-term fluctuations, supporting investment and financial planning decisions.

c) Univariate Forecasting

Univariate forecasting models like Holt-Winters and ARIMA provide robust tools for predicting future values based on historical data. These models are essential for businesses to anticipate future trends and prepare accordingly. Diagnostic checks ensure model validity, and comparing different models (ARIMA vs. SARIMA) helps identify the best-fit model for the data, leading to more accurate forecasts.

1. Accurate forecasts help businesses maintain optimal inventory levels, reducing holding costs and minimizing stockouts.
2. Reliable financial forecasts enable businesses to create more accurate budgets and allocate resources more effectively.

3. Predictive models help businesses anticipate and mitigate risks by identifying potential future trends and anomalies.

d) Multivariate Forecasting Using Machine Learning Models

Advanced machine learning models like Long Short-Term Memory (LSTM) networks and tree-based models (Random Forest, Decision Tree) can capture complex patterns and interactions in multivariate data. These models provide superior predictive performance, especially when dealing with large and complex datasets. Implementing these models allows businesses to leverage the full potential of their data, leading to more precise and actionable insights.

1. Multivariate models can analyze various factors influencing sales (e.g., advertising spend, economic indicators) to predict future sales more accurately and optimize marketing strategies.
2. Maintenance using multivariate data from machinery sensors helps prevent downtime and reduce maintenance costs.
3. Advanced models can analyze multiple economic indicators and market variables to predict stock prices and market movements more accurately, aiding in investment strategies.

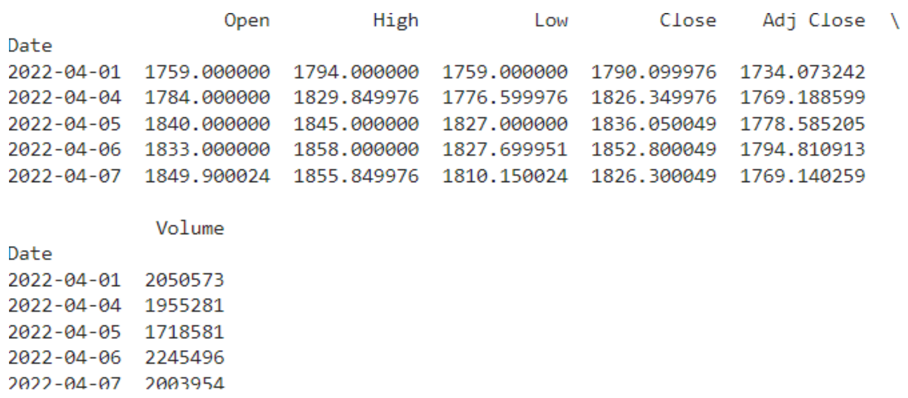
RESULTS AND INTERPRETATION

1. Download the Larsen & Toubro as a CSV file.

Code:

```
import yfinance as yf

# Define the ticker symbol for Larsen & Toubro Limited

ticker_symbol = "LT.NS"  # Adjust this ticker symbol if it's different for Larsen & Toubro on the Indian stock market

# Download the historical data

data = yf.download(ticker_symbol, start="2022-04-01", 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("larsen_toubro_data.csv")
```

Result:



```
                   Open         High          Low        Close    Adj Close  \
Date
2022-04-01  1759.000000  1794.000000  1759.000000  1790.099976  1734.073242
2022-04-04  1784.000000  1829.849976  1776.599976  1826.349976  1769.188599
2022-04-05  1840.000000  1845.000000  1827.000000  1836.050049  1778.585205
2022-04-06  1833.000000  1858.000000  1827.699951  1852.800049  1794.810913
2022-04-07  1849.900024  1855.849976  1810.150024  1826.300049  1769.140259

             Volume
Date
2022-04-01  2050573
2022-04-04  1955281
2022-04-05  1718581
2022-04-06  2245496
2022-04-07  2003954
```

Interpretation:

The provided code fetches historical stock data for Larsen & Toubro Limited (LT.NS) from April 1, 2022, to March 31, 2024, using the yfinance library. The data encompasses essential trading information such as the opening price, highest price, lowest price, closing price, adjusted closing price, and trading volume for each day within the specified timeframe.

The first few rows of data provide insight into the stock's performance at the beginning of the period:

- April 1, 2022: The stock opened at ₹1,759.00, peaked at ₹1,794.00, dipped to ₹1,759.00, and closed at ₹1,790.10, with an adjusted close of ₹1,734.07. The trading volume was 2,050,573 shares.
- April 4, 2022: The stock opened higher at ₹1,784.00, reached a high of ₹1,829.85, a low of ₹1,776.60, and closed at ₹1,826.35, with an adjusted close of ₹1,769.19. The volume was 1,955,281 shares.
- April 5, 2022: The stock opened at ₹1,840.00, saw a high of ₹1,845.00, a low of ₹1,827.00, and closed at ₹1,836.05, with an adjusted close of ₹1,778.59. The trading volume was 1,718,581 shares.
- April 6, 2022: The stock opened at ₹1,833.00, climbed to ₹1,858.00, fell to ₹1,827.70, and closed at ₹1,852.80, with an adjusted close of ₹1,794.81. The volume was 2,245,496 shares.
- April 7, 2022: The stock opened at ₹1,849.90, reached a high of ₹1,855.85, a low of ₹1,810.15, and closed at ₹1,826.30, with an adjusted close of ₹1,769.14. The volume was 2,003,954 shares.

This initial data indicates the stock's fluctuating performance in early April 2022, with substantial trading volumes and varying price movements. This historical data, saved in a CSV file, is crucial for further financial analysis and forecasting.

2. 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.

Code:

```
# 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())
```

Result:

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

Interpretation:

The dataset is clean and complete with no missing values in the 'Close' column. This ensures that the data is ready for further analysis or modeling without the need for additional data imputation or handling of missing values. Having a complete dataset is crucial for accurate and reliable financial analysis, as missing values can lead to misleading results or require complex preprocessing steps.

```
# Interpolate missing values

data.interpolate(method='time', inplace=True)

# Check for missing values again

print("Missing values after interpolation:")

print(data.isnull().sum())
```

Result:

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

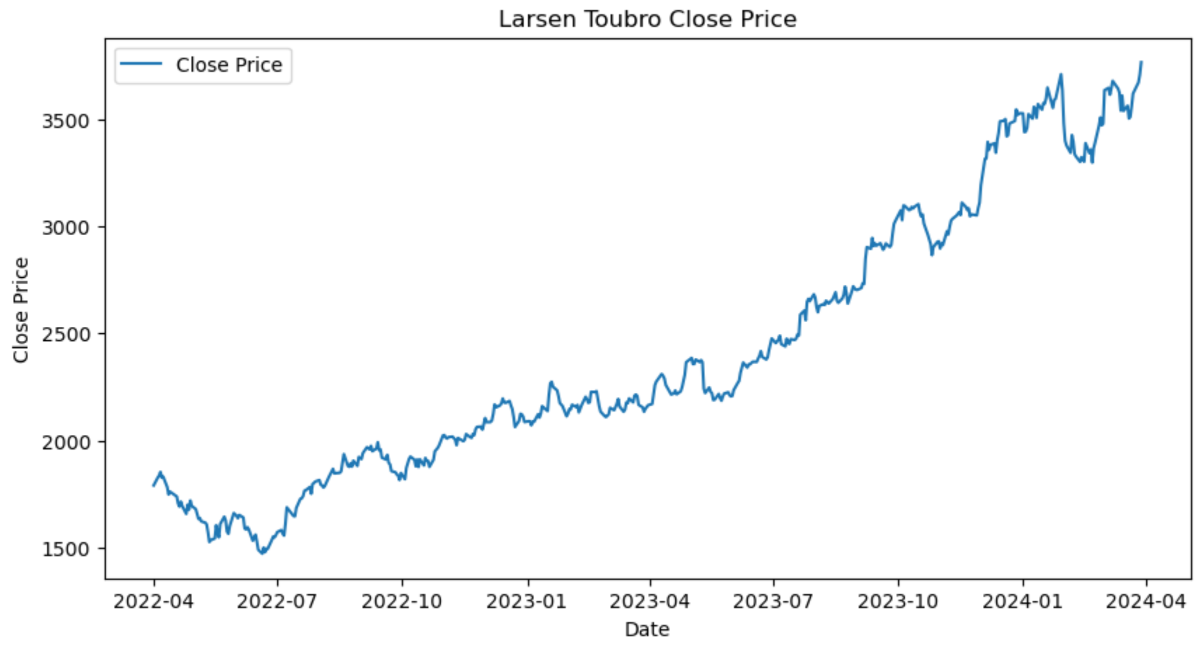
Interpretation:

Although the initial check showed no missing values, the interpolation step ensures that any potential future gaps or discrepancies are automatically handled. This proactive step guarantees the dataset's integrity, maintaining completeness and continuity. This is crucial for accurate analysis, as even small gaps in time series data can affect the outcomes of subsequent analyses or models. The dataset is now fully prepared for further financial analysis or modeling, with confirmed zero missing values in the 'Close' column.

Plotting a line graph

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

Result:



Interpretation:

This is a plot of the closing prices for Larsen & Toubro Limited (LT.NS) from April 2022 to March 2024.

The horizontal axis represents the date range from April 2022 to March 2024. The vertical axis shows the closing prices of the stock over the specified period. The blue line represents the daily closing prices of Larsen & Toubro Limited.

The plot shows an overall upward trend in the closing prices of Larsen & Toubro Limited over the two-year period. There are noticeable fluctuations, but the general direction is a steady increase.

The plot also highlights periods of volatility where the closing prices experience significant up-and-down movements. Despite these fluctuations, the trend remains positive, indicating a growing stock price over time.

There are specific periods where the stock price increases more sharply, suggesting potentially significant events or strong market performance during those times. These could be linked to positive earnings reports, major contracts, or other corporate development. Towards the end of the period (early 2024), the closing price reaches its highest point on the plot, indicating strong recent performance.

The plot provides a clear visual representation of the closing price movements for Larsen & Toubro Limited over the specified timeframe. The consistent upward trend suggests positive investor sentiment and company performance, despite periods of volatility. This trend can be useful for investors and analysts in making informed decisions based on the historical performance of the stock.

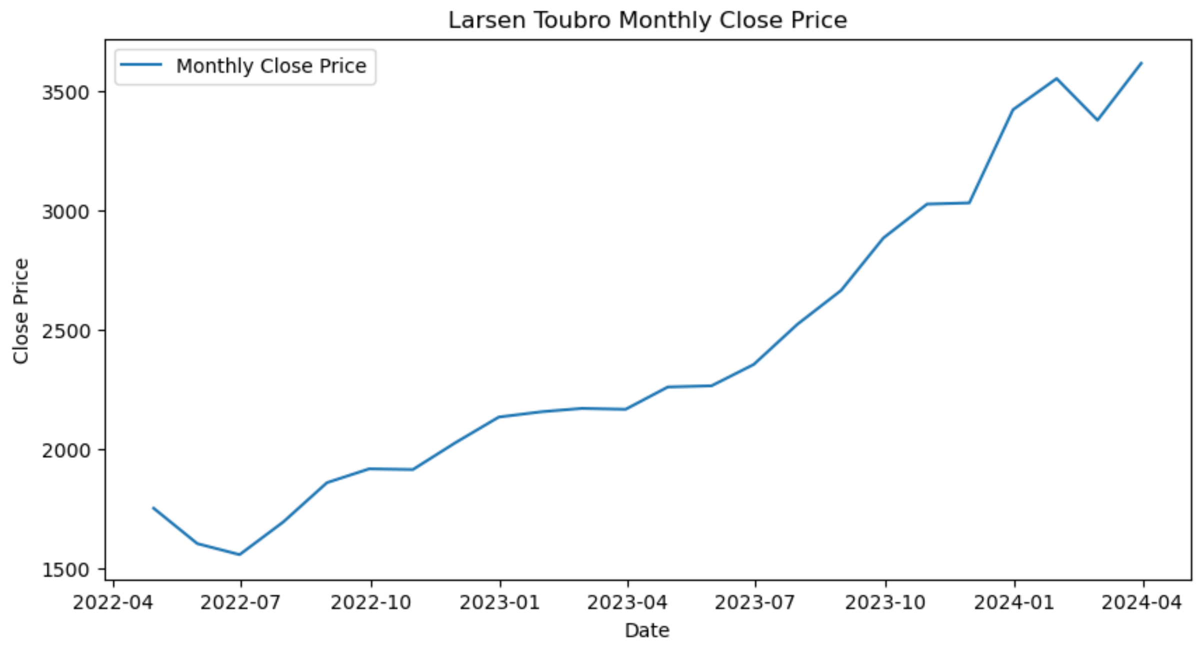
3. Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

Convert the data to monthly frequency

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('Larsen Toubro Monthly Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

Result:



Interpretation:

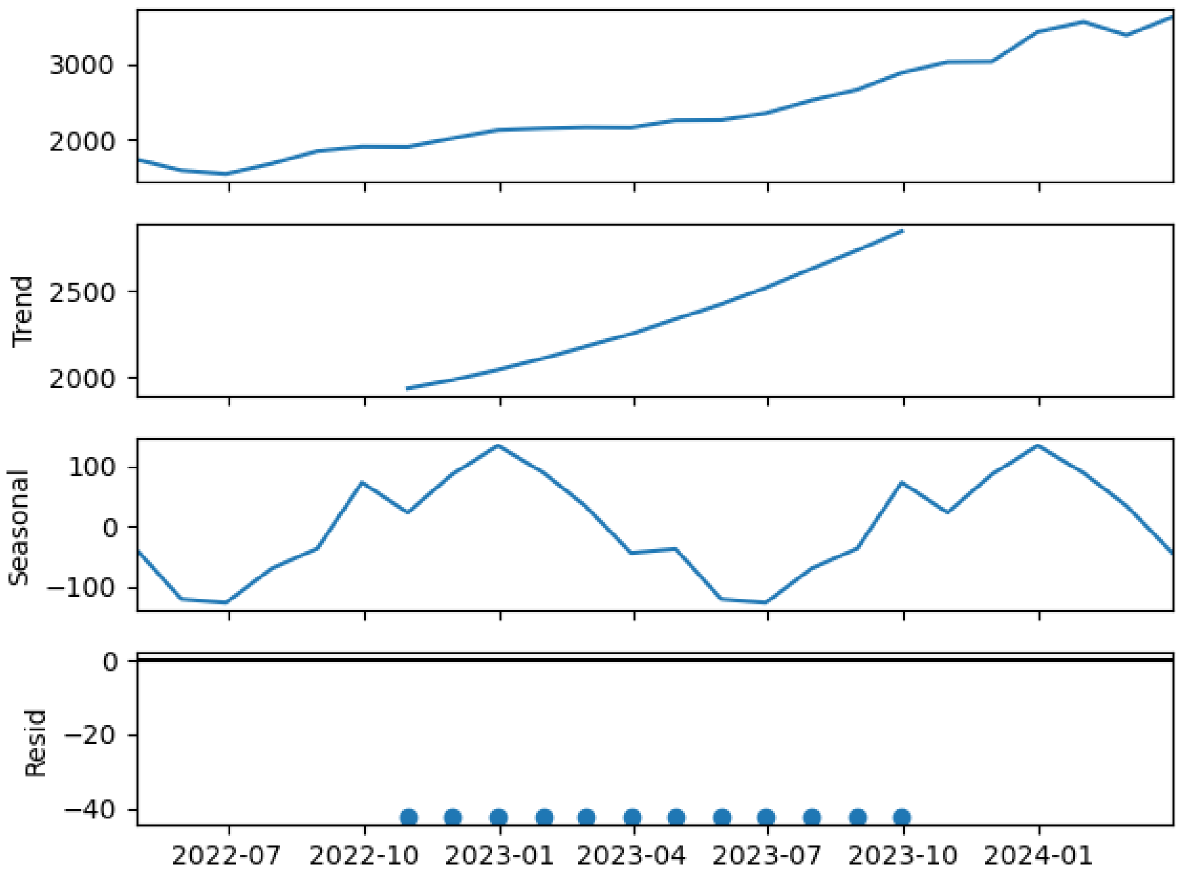
The conversion of daily closing prices to monthly averages smooths out daily fluctuations and highlights longer-term trends. The resulting plot of Larsen & Toubro Limited's monthly average closing prices from April 2022 to March 2024 shows a clear upward trend, indicating sustained growth over the two-year period. This approach reduces volatility, providing a more stable view of the stock's performance and helping investors and analysts identify significant trends and patterns for informed decision-making. The consistent increase in average monthly prices suggests strong and steady company performance.

Code:

Decompose time series into the components using additive and multiplicative models

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

Result:



Interpretation:

The plot shows the decomposition of the monthly closing price time series for Larsen & Toubro Limited using an additive model. The decomposition breaks down the time series into three components: trend, seasonal, and residual.

Trend Component

The second panel in the plot represents the trend component, showing the underlying direction of the data over time. The trend indicates a steady increase in the average monthly closing prices, highlighting a consistent upward movement in the stock's performance over the two-year period.

Seasonal Component

The third panel depicts the seasonal component, capturing the repeating patterns or cycles within the data that occur at regular intervals. The plot reveals periodic fluctuations that repeat roughly on a yearly basis, indicating seasonal effects in the stock prices. These could be due to recurring market events or cyclical economic factors.

Residual Component

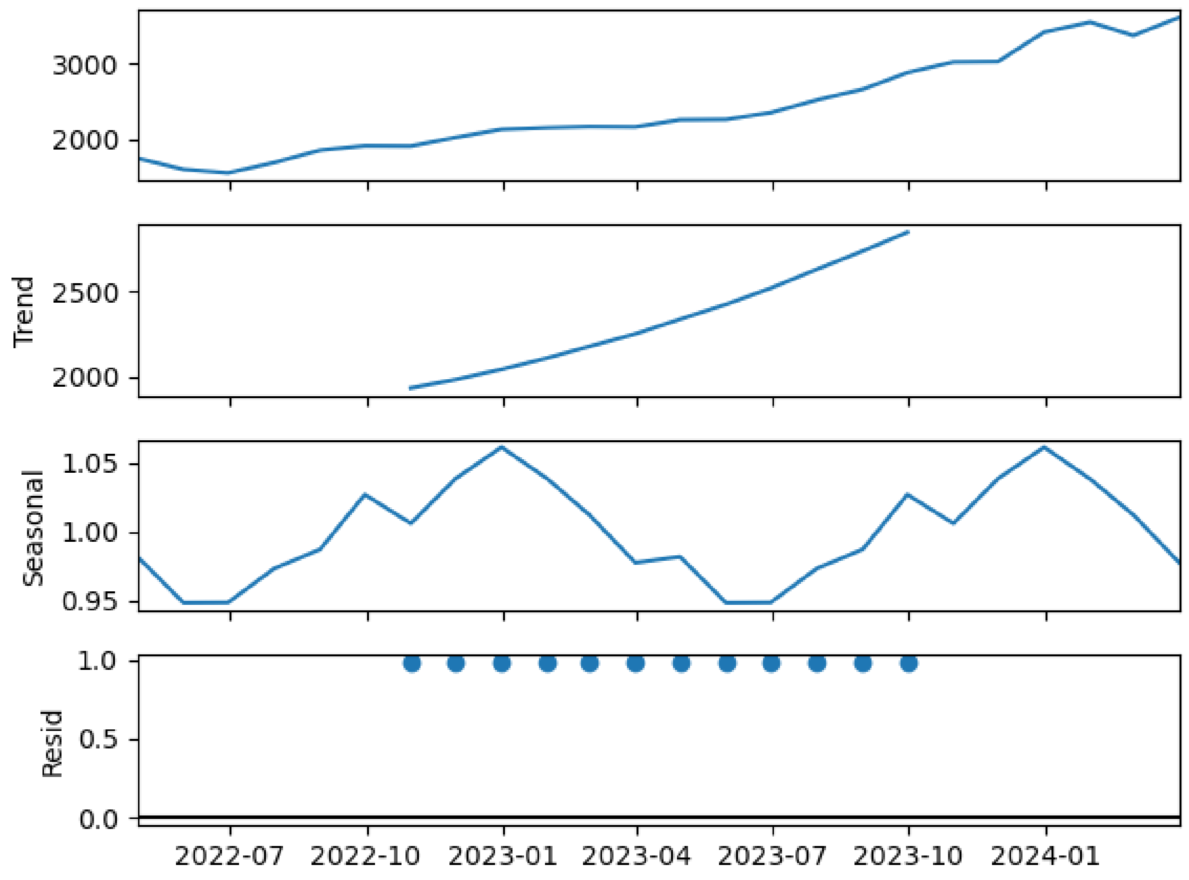
The fourth panel shows the residual component, representing the random noise or irregular variations that cannot be explained by the trend or seasonal components. The residuals appear relatively small, suggesting that the additive model effectively captures the main patterns in the data. However, the presence of some residuals indicates that there are minor variations not explained by the model.

The additive decomposition provides a clear view of the different components influencing the stock's monthly closing prices. The steady upward trend reflects long-term growth, while the seasonal component indicates periodic fluctuations. The small residuals suggest that most of the variability is well-explained by the trend and seasonal components, making this decomposition useful for understanding the underlying patterns and forecasting future stock prices.

Code:

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

Result:



Interpretation:

The plot shows the decomposition of the monthly closing price time series for Larsen & Toubro Limited using a multiplicative model. This decomposition breaks down the time series into three components: trend, seasonal, and residual. Here’s the interpretation:

Trend Component

The second panel in the plot represents the trend component, which shows the underlying direction of the data over time. Similar to the additive model, the trend indicates a steady increase in the average monthly closing prices, demonstrating a consistent upward movement in the stock's performance over the two-year period.

Seasonal Component

The third panel depicts the seasonal component, capturing the repeating patterns or cycles within the data that occur at regular intervals. In the multiplicative model, the seasonal component is shown as a ratio relative to the trend. The plot reveals periodic fluctuations that repeat roughly on a yearly basis, indicating seasonal effects in the stock prices. These variations are proportionate to the level of the trend.

Residual Component

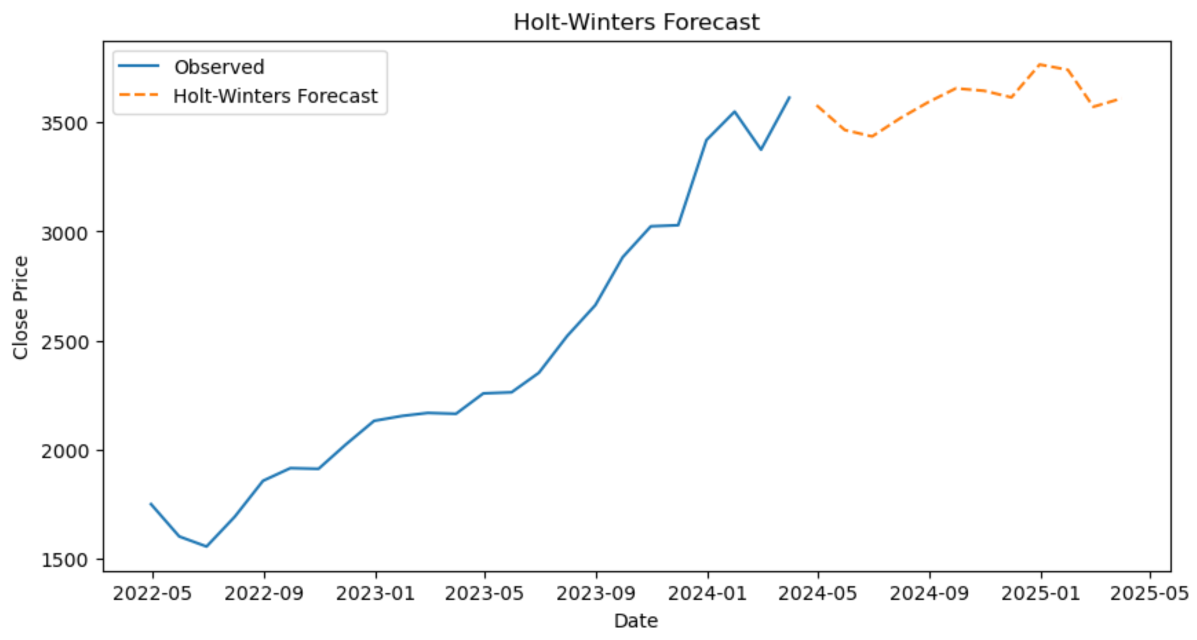
The fourth panel shows the residual component, representing the random noise or irregular variations that cannot be explained by the trend or seasonal components. The residuals appear to be relatively small and consistently around 1, suggesting that the multiplicative model effectively captures the main patterns in the data. The small residuals indicate minimal variations not explained by the model.

The multiplicative decomposition provides a detailed view of the different components influencing the stock's monthly closing prices. The steady upward trend reflects long-term growth, while the seasonal component indicates periodic fluctuations that are proportionate to the trend level. The consistently small residuals suggest that most of the variability is well-explained by the trend and seasonal components, making this decomposition useful for understanding the underlying patterns and for forecasting future stock prices. The multiplicative model is particularly effective when the seasonal variations increase with the level of the trend.

4. Fit a Holt Winters model to the data and forecast for the next year.

```
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()
```

Result:



Interpretation:

This showcases a time series forecast of the close price using the Holt-Winters method. The observed data, depicted by the solid blue line, indicates a steady upward trend in the close price from May 2022 to February 2024. The forecast, represented by the dashed orange line, extends this analysis into the future, projecting values from March 2024 to February 2025. The forecast suggests an initial slight decrease followed by a general upward trend, reflecting both trend and seasonality in the data. The additive seasonal component, with a period of 12 months, captures the yearly seasonal fluctuations observed historically. Overall, the forecast indicates a continued increase in the close price over the next year, with seasonal variations that align with past patterns, providing valuable insights for future decision-making.

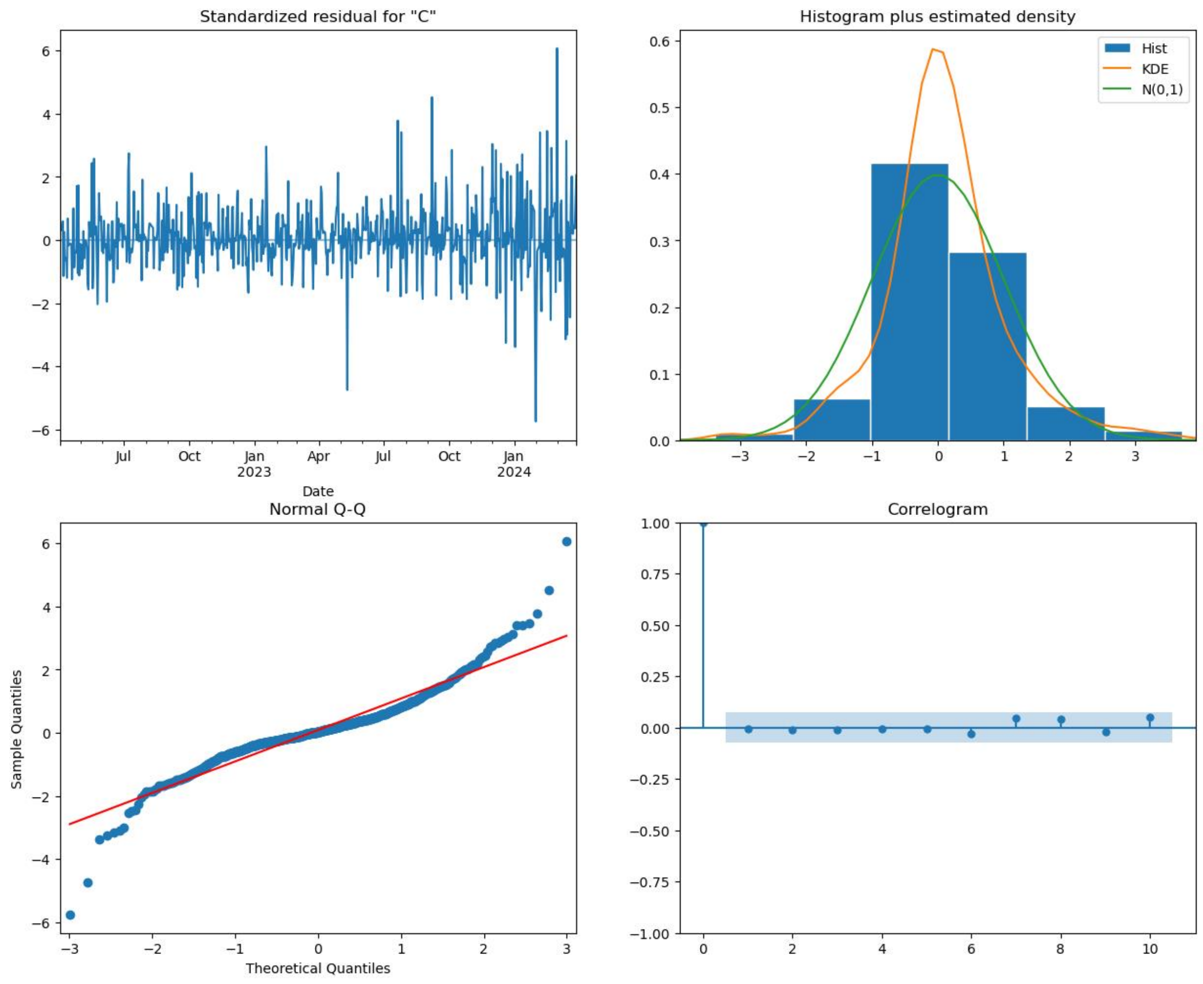
5. Fit an ARIMA model to the daily data and do a diagnostic check validity of the model.

Code:

```
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()
```

Result:



Interpretation:

The provided diagnostic plots for the ARIMA model include four key components:

1. **Standardized Residuals Plot (Top Left)** - To check the randomness of the residuals. The residuals appear to fluctuate randomly around zero, which is a good sign indicating that the model has captured the data's structure.

2. **Histogram and KDE Plot (Top Right)** - To assess the normality of the residuals. The histogram is overlaid with a kernel density estimate (KDE) and a normal distribution curve (N(0,1)). The residuals appear to follow a roughly normal distribution, although there are deviations, particularly in the tails.

3. **Normal Q-Q Plot (Bottom Left)** - To further assess the normality of the residuals. Most of the points lie on the 45-degree reference line, indicating that the residuals are approximately normally distributed. However, there are some deviations at the extremes, suggesting potential outliers.
4. **Correlogram (Bottom Right)** - To check for autocorrelation in the residuals. The autocorrelation function (ACF) shows that most autocorrelations are within the confidence intervals, suggesting that the residuals are not significantly autocorrelated.

Validity of the ARIMA Model

The residuals appear to be randomly distributed around zero, indicating no obvious patterns left unmodeled. The residuals are approximately normally distributed, although there are some deviations, particularly in the tails. The residuals do not exhibit significant autocorrelation, suggesting that the ARIMA model has adequately captured the time series dynamics. Overall, the ARIMA model seems to be a good fit for the data. However, given the deviations in the tails of the residual distribution, it might be worth exploring if a seasonal ARIMA (SARIMA) model fits the data better.

Comments: Seasonal ARIMA (SARIMA) Model

To determine if a SARIMA model might be more appropriate, we should consider the following steps:

1. **Seasonal Decomposition:** Check for seasonal patterns in the data using seasonal decomposition techniques (e.g., STL decomposition).
2. **Model Selection:** Fit various SARIMA models with different seasonal orders and compare their AIC/BIC values to select the best model.
3. **Diagnostic Checks:** Perform similar diagnostic checks on the SARIMA model to ensure its validity.

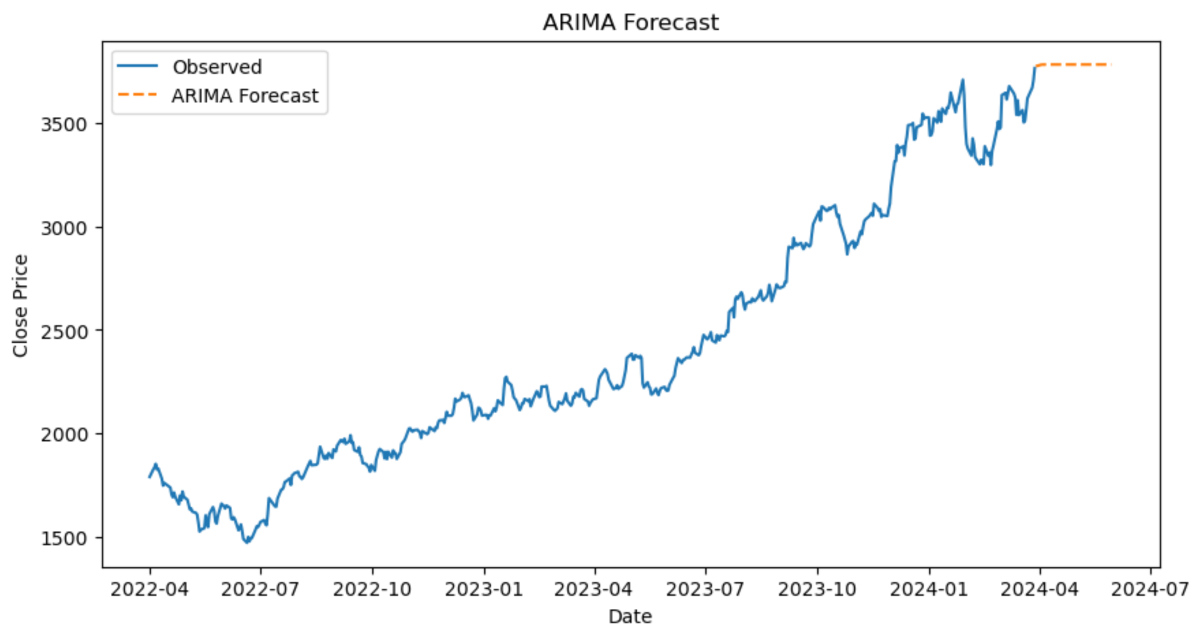
If seasonal patterns are present in the data and a SARIMA model with seasonal components (P, D, Q, s) shows improved diagnostics and lower AIC/BIC values, it would suggest that the SARIMA model provides a better fit.

6. Forecast the series for the next three months.

Code:

```
# 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()
```

Result:



Interpretation:

The ARIMA forecast plot shows the observed upward trend in close prices over the past two years, with the ARIMA model projecting a continued rise but with a relatively flat forecast line for the next three months. This suggests the ARIMA model captures the overall trend but might not fully account for seasonal fluctuations. Considering potential seasonality, exploring a Seasonal ARIMA (SARIMA) model could provide a more accurate forecast.

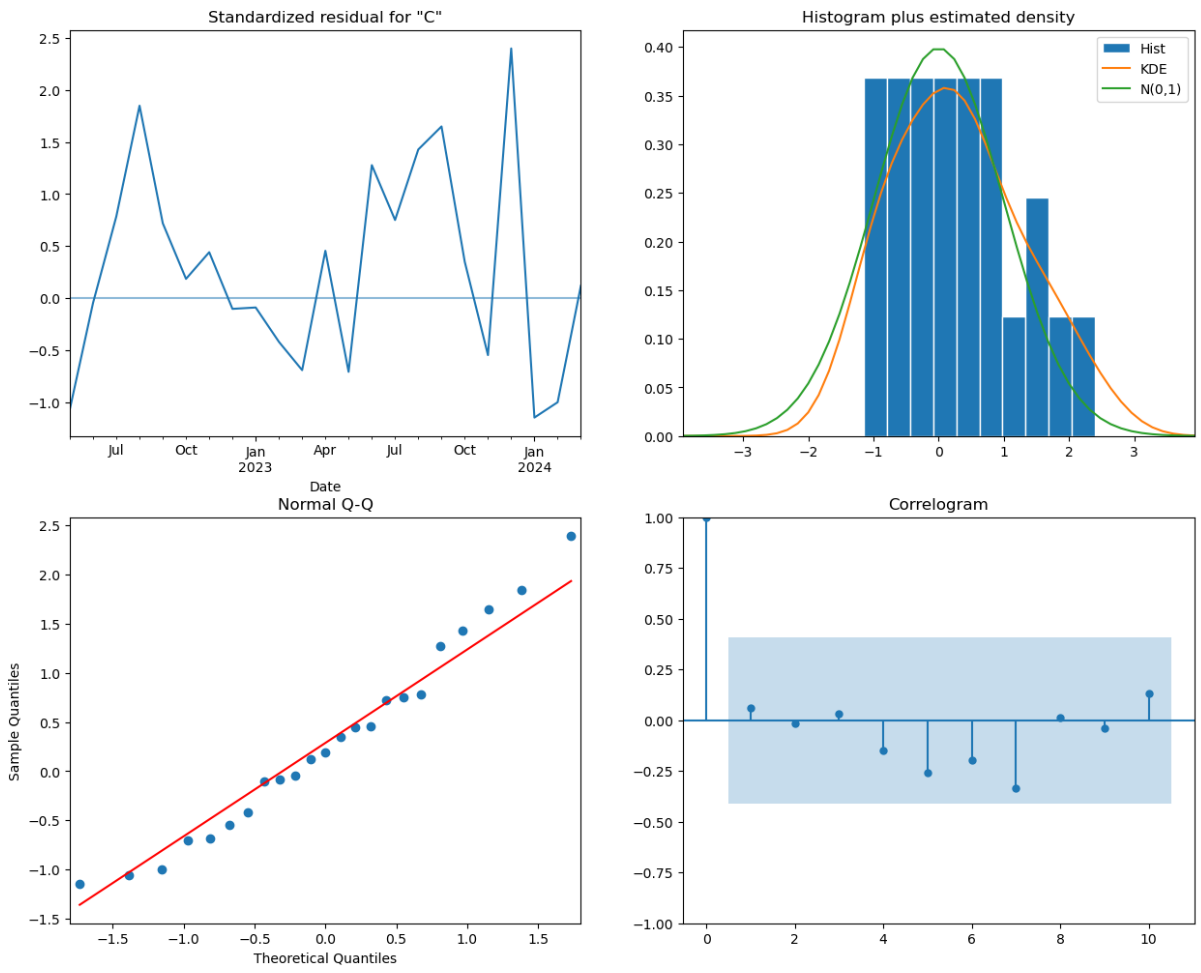
7. ARIMA Model - Monthly Data

Code:

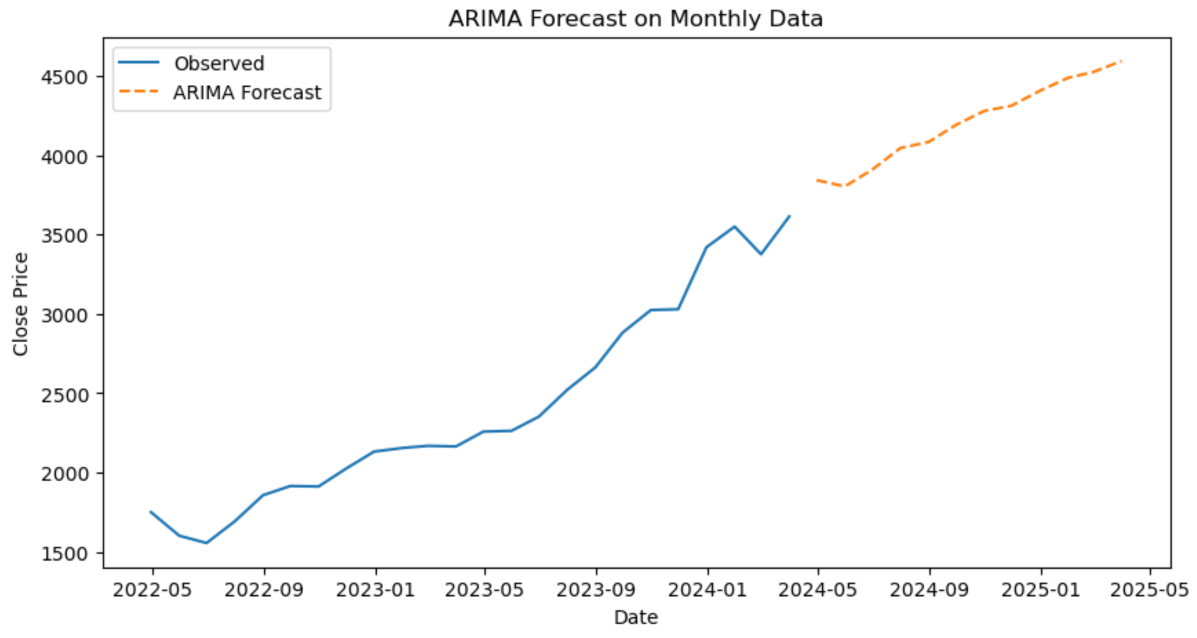
```
# Fit the ARIMA model on the monthly data
arima_model = sm.tsa.ARIMA(monthly_data, order=(5, 1, 0)).fit()


# 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()
```

Result:



Standardized residual for "C"
Date
Histogram plus estimated density
Hist
KDE
N(0,1)
Normal Q-Q
Theoretical Quantiles
Sample Quantiles
Correlogram



ARIMA Forecast on Monthly Data
Observed
ARIMA Forecast
Close Price
Date

Interpretation:

ARIMA Model Diagnostics

1. **Standardized Residuals Plot**: This plot shows the standardized residuals over time. Ideally, these residuals should be randomly distributed around zero with no obvious patterns, indicating that the model has captured the underlying data structure well.
2. **Histogram plus Estimated Density**: The histogram of residuals should resemble a normal distribution (bell curve). The overlaid KDE (Kernel Density Estimate) and the theoretical normal distribution (N(0,1)) help in visually assessing normality. Significant deviations from normality might suggest that the model does not fully capture all patterns in the data.
3. **Normal Q-Q Plot**: This plot compares the quantiles of the residuals to a theoretical normal distribution. Points should lie approximately along the red line. Deviations from this line indicate non-normality in the residuals, which could impact the model's accuracy.
4. **Correlogram (ACF of Residuals)**: The correlogram shows the autocorrelation function of the residuals. Most points should lie within the confidence interval (shaded region), indicating that residuals are not autocorrelated. Significant autocorrelation at any lag suggests the model may not have fully captured the time-series structure.

ARIMA Forecast Plot

Observed Data vs. Forecast: The plot shows the observed data (solid line) and the ARIMA forecast for the next 12 months (dashed line). The forecast extends the observed trend, providing predictions for future values based on the model fitted to the past data.

Residual Analysis: The standardized residuals plot shows some variability around zero. If there are no distinct patterns, it suggests a reasonable fit. The histogram of residuals and the Q-Q plot indicate if residuals are normally distributed. Deviations from normality might be present. The correlogram suggests the residuals are not significantly autocorrelated, indicating that the model has captured most of the time-series structure.

Forecast: The ARIMA forecast extends the historical trend observed in the data. The dashed line provides the expected future values, showing an increasing trend.

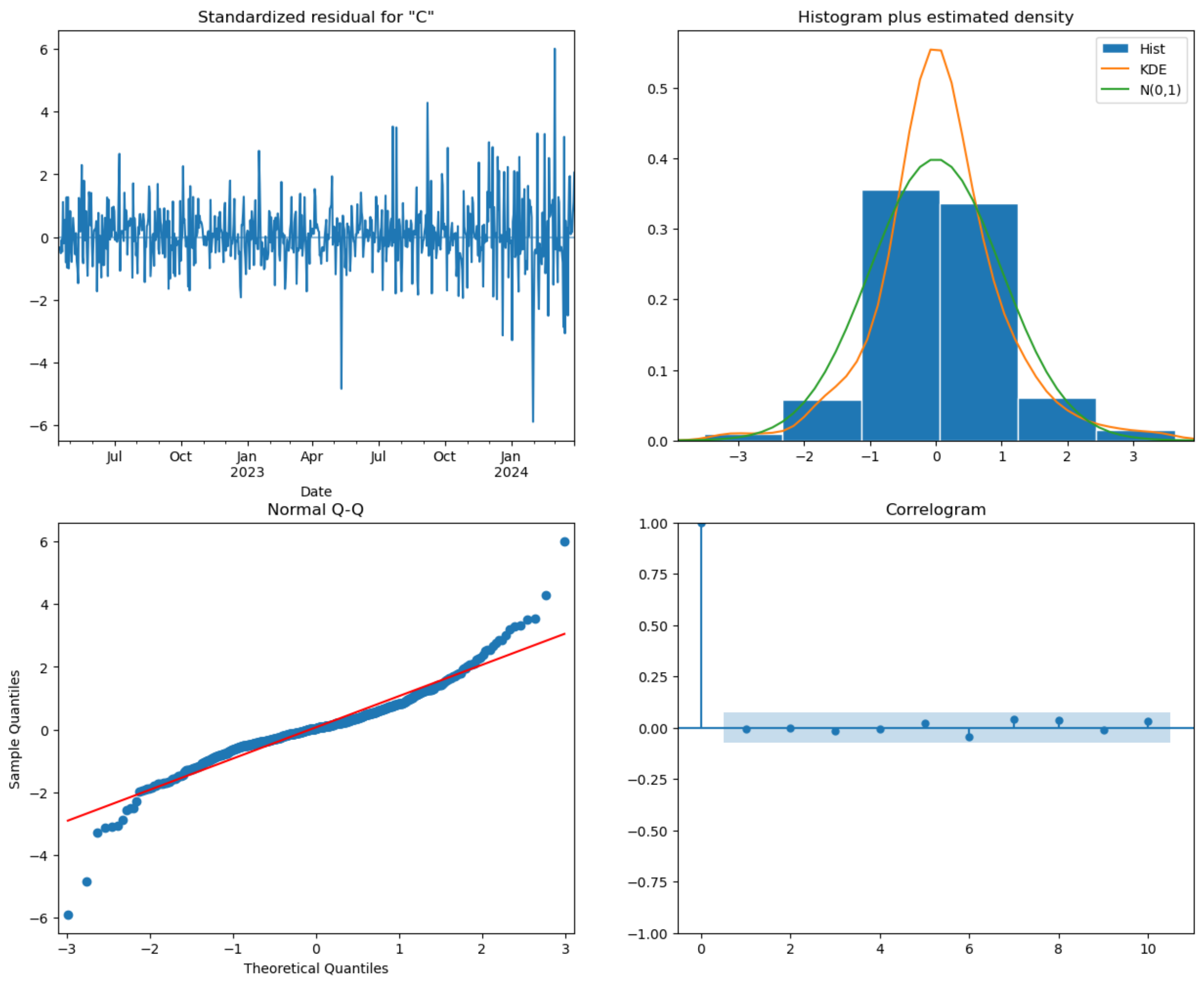
The ARIMA model diagnostics suggest that the model reasonably fits the data, although some deviations from normality in the residuals might be present. The forecast indicates an upward trend in the close price over the next 12 months. Further refinement of the model could include trying different ARIMA configurations or incorporating external factors that might influence the data.

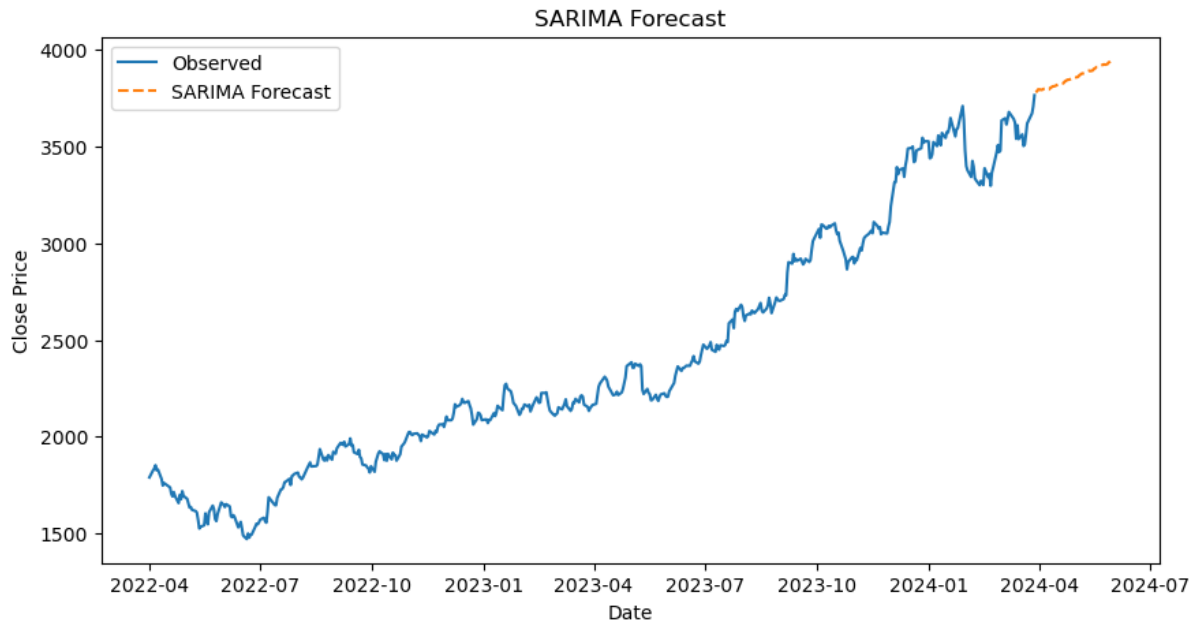
SARIMA Model

```
# 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()
```

Result:





Interpretation:

SARIMA Model Diagnostics

The SARIMA model diagnostics provide a comprehensive evaluation of the model fit through various plots:

1. **Standardized Residuals Plot**: This plot displays the standardized residuals over time. For a well-fitted model, the residuals should be randomly distributed around zero, with no discernible patterns or trends. In the plot, the residuals appear to be randomly distributed with some volatility, indicating a decent fit but with some possible areas of improvement.
2. **Histogram plus Estimated Density**: This histogram of the residuals is compared with the theoretical normal distribution (N(0,1)). The overlay of the kernel density estimate (KDE) and the normal distribution curve provides insight into the residuals' distribution. Ideally, the KDE curve should closely follow the normal distribution curve. The plot shows a reasonable approximation to normality, suggesting that the residuals are approximately normally distributed.
3. **Normal Q-Q Plot**: The Q-Q plot compares the quantiles of the residuals with the quantiles of a standard normal distribution. Points lying on the red line indicate that the residuals are normally distributed. While most points align with the line, there are deviations at the tails, indicating potential issues with normality in the residuals.
4. **Correlogram (ACF of Residuals)**: The correlogram displays the autocorrelation function (ACF) of the residuals. For a well-fitted model, the residuals should show no significant autocorrelation. The correlogram indicates that the autocorrelations are mostly within the confidence intervals, suggesting no significant autocorrelation in the residuals, which supports the model's adequacy.

SARIMA Forecast

The SARIMA forecast plot presents the observed data and the forecasted values for the next three months, based on the SARIMA model: The blue line represents the observed data, depicting the historical closing prices over time and the orange dashed line represents the SARIMA forecast for the next 63 trading days (approximately three months).

The forecast indicates a continuation of the upward trend observed in the historical data. The forecast values seem to follow the observed data closely, suggesting that the SARIMA model has captured the underlying patterns and seasonality in the data effectively.

Overall, the diagnostic plots suggest that the SARIMA model provides a reasonable fit to the data. The residuals are approximately normally distributed and exhibit no significant autocorrelation, indicating that the model is well-specified. The forecast plot further demonstrates the model's ability to project future values, aligning with the observed upward trend in the closing prices. However, minor deviations in the Q-Q plot and residuals plot suggest that there might be room for further refinement of the model.

8. LSTM

Code:

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

```
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(sequence_length, 6)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test,
y_test), shuffle=False)
# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
```

Result:

```
Epoch 1/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 9s 120ms/step - loss: 0.0088 -
val_loss: 0.0162
Epoch 2/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 0.0146 - val_loss:
0.0342
Epoch 3/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.0021 - val_loss:
0.0020
Epoch 4/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0016 - val_loss:
0.0034
Epoch 5/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 0.0015 - val_loss:
0.0066
Epoch 6/20
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0016 - val_loss: 0.0027
Epoch 7/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0021 - val_loss: 0.0065
Epoch 8/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0013 - val_loss: 0.0038
Epoch 9/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 0.0016 - val_loss: 0.0026
Epoch 10/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0016 - val_loss: 0.0060
Epoch 11/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0011 - val_loss: 0.0035
Epoch 12/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0013 - val_loss: 0.0024
Epoch 13/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.0012 - val_loss: 0.0030
Epoch 14/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.0012 - val_loss: 0.0038
Epoch 15/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.0013 - val_loss: 0.0018
Epoch 16/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.0033 - val_loss: 0.0098
Epoch 17/20
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 0.0014 - val_loss: 0.0025
Epoch 18/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0014 - val_loss: 0.0021
Epoch 19/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0018 - val_loss: 0.0086
Epoch 20/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 0.0011 - val_loss: 0.0034
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0032
Test Loss: 0.0033552562817931175
```

Code:

```python
# 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]
```

Result:

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 412ms/step
```

Code:

```python
# 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]}")
```

Result:

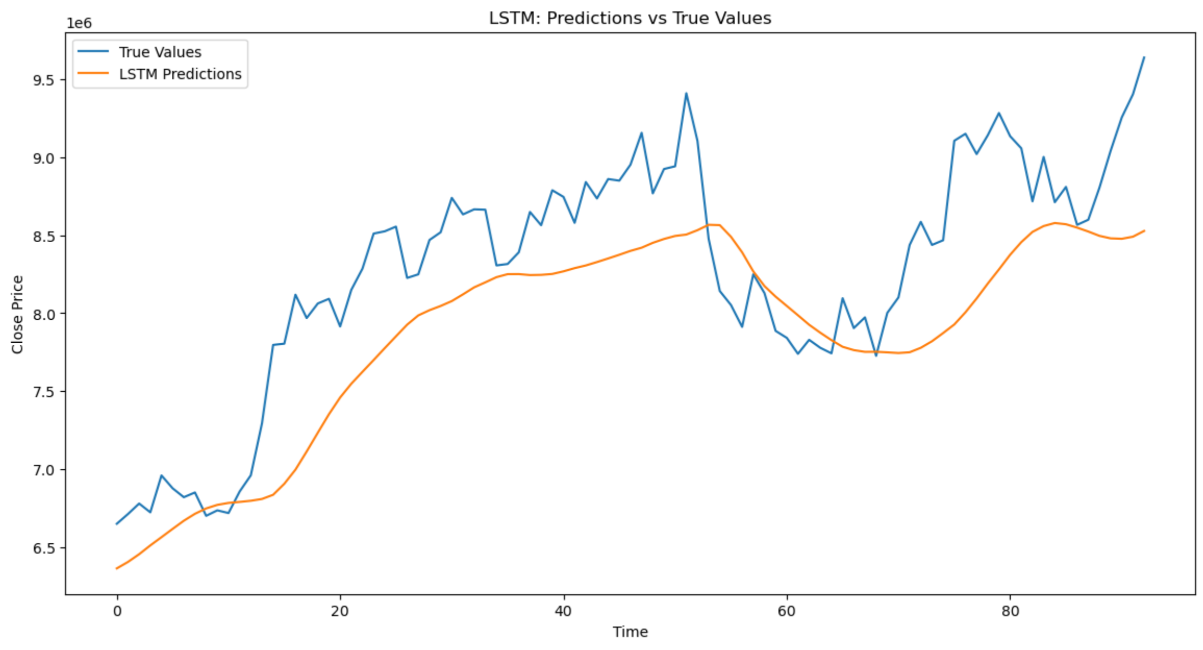
```
Predictions vs True Values:
```

```
Prediction: 6365356.723720193, True Value: 6651314.286514022
Prediction: 6406788.040381312, True Value: 6713643.657235317
Prediction: 6456303.707060218, True Value: 6780877.434608365
Prediction: 6512592.06895709, True Value: 6724474.263501469
Prediction: 6565331.871359348, True Value: 6960917.554195208
Prediction: 6618509.450416803, True Value: 6878356.882382494
Prediction: 6670644.517975211, True Value: 6821341.034635889
Prediction: 6715106.767907262, True Value: 6851790.265353748
Prediction: 6749402.995154977, True Value: 6701994.818323298
Prediction: 6772293.577046752, True Value: 6737144.895376133
```

Code:

```
# 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()
```

Result:



Interpretation:

The plot shows a line chart comparing the predictions made by an LSTM model against the true values of a financial time series. The blue line represents the true closing prices of a stock or financial instrument over time, while the orange line represents the closing prices predicted by the LSTM model. The chart title reads "LSTM: Predictions vs True Values," and the axes are labeled with "Time" on the x-axis and "Close Price" on the y-axis.

The LSTM model was trained on financial data, as indicated by the Python code provided. This data was loaded from a CSV file containing information about Larsen & Toubro's stock prices. The code uses TensorFlow and Keras libraries to build the LSTM model, which consists of two LSTM layers followed by dropout layers and a dense output layer. The data was preprocessed using the MinMaxScaler to scale the features, and sequences were created with a length of 30. The data was split into training and testing sets, with the model trained over 20 epochs.

The training process included monitoring the loss on both training and validation sets. The final model was evaluated on the test set, and predictions were made. These predictions were then inverse transformed to obtain the original scale of the closing prices. The plotted results demonstrate how closely the model's predictions align with the actual values, highlighting the model's effectiveness in capturing the trends, though it appears to smooth out some of the more volatile movements in the true data.

The model's performance was quantified using the loss metric, specifically the mean squared error, which was 0.0033552562817931175 on the test set. Despite this low loss value, the visual inspection of the plot suggests that while the model captures the general trend, there are discrepancies between the predicted and actual values, especially during periods of high volatility. This indicates that while the LSTM model is quite effective, there may still be room for improvement in capturing the more nuanced movements of the stock price.

9. Tree Based Models

```
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
# 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}")
```

```
Decision Tree Mean Squared Error: 174655.60686650648
```

```python
# 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}")
```

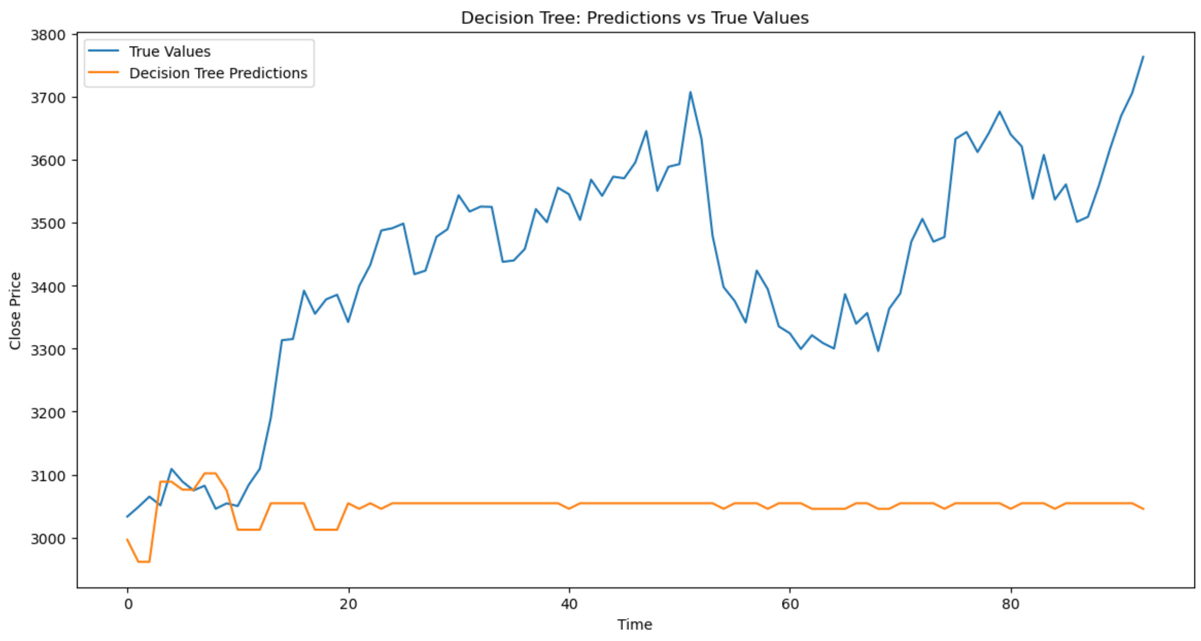
```
Random Forest Mean Squared Error: 175714.92813506545
```

```python
# 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]}")
```

```
Decision Tree Predictions vs True Values:
Prediction: 2996.449951171875, True Value: 3033.25
Prediction: 2961.300048828125, True Value: 3048.5
Prediction: 2961.300048828125, True Value: 3064.949951171875
Prediction: 3088.699951171875, True Value: 3051.14990234375
Prediction: 3088.699951171875, True Value: 3109.0
Prediction: 3076.0, True Value: 3088.800048828125
Prediction: 3076.0, True Value: 3074.85009765625
Prediction: 3101.89990234375, True Value: 3082.300048828125
Prediction: 3101.89990234375, True Value: 3045.64990234375
Prediction: 3074.699951171875, True Value: 3054.25
```

```python
# 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()
```



Interpretation:

The analysis performed using the Decision Tree and Random Forest models to predict the "Close Price" of a financial instrument reveals significant insights into their performance and limitations. Both models exhibit high Mean Squared Error (MSE) values, with the Decision Tree model recording an MSE of 174655.61 and the Random Forest model slightly higher at 175714.93. These elevated MSE values indicate substantial discrepancies between the predicted and actual values, underscoring the models' lack of accuracy in capturing the true dynamics of the financial data.

The predicted values from the Decision Tree model appear relatively flat and clustered around specific levels, showing minimal variation. This is starkly contrasted by the true values, which exhibit significant fluctuations and trends over time. For example, where the true value might be 3033.25, the Decision Tree predicts 2996.45, and this pattern persists across multiple data points. Such flatline predictions suggest that the model fails to adapt to the inherent variability and temporal patterns in the data.

The graphical representation of the Decision Tree model's performance vividly illustrates these shortcomings. The plot shows the true values as a blue line with notable peaks and troughs, reflecting the actual market trends. On the other hand, the orange line, representing the Decision Tree's predictions, remains mostly flat and does not follow the true values' path. This visual discrepancy indicates that the model does not capture the essential trends and changes in the data, leading to poor predictive performance.

The high MSE values and the observed flatline behavior of the Decision Tree model's predictions suggest that the current models are not suitable for predicting the "Close Price" accurately. The models likely suffer from issues such as overfitting or an inability to capture the time-dependent nature of the data. This could be due to inadequate feature engineering or the inherent limitations of tree-based models in handling time series data without additional temporal features.

In conclusion, the current tree-based models demonstrate a clear inability to predict the "Close Price" accurately, as evidenced by high MSE values and flat prediction lines. This suggests a need for more sophisticated modeling approaches and enhanced feature engineering to effectively capture the complex patterns and trends in financial time series data. By addressing these areas, it may be possible to develop a more accurate and reliable predictive model.

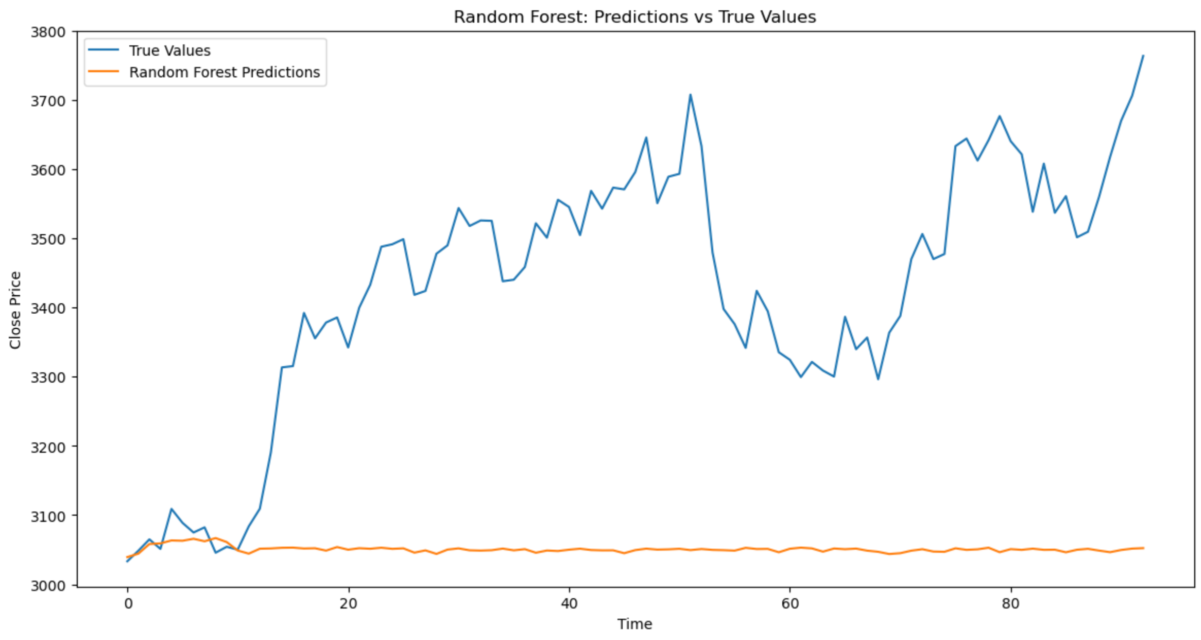
Code:

```
print("\nRandom Forest Predictions vs True Values:")

for i in range(10):

    print(f"Prediction: {y_pred_rf[i]}, True Value: {y_test[i]}")

# 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()
```

Result:

```
Random Forest Predictions vs True Values:
```

```
Prediction: 3039.31552734375, True Value: 3033.25

Prediction: 3044.2615258789065, True Value: 3048.5

Prediction: 3058.227521972656, True Value: 3064.949951171875

Prediction: 3059.1065234375, True Value: 3051.14990234375

Prediction: 3063.366511230469, True Value: 3109.0

Prediction: 3062.9200146484377, True Value: 3088.800048828125

Prediction: 3065.79400390625, True Value: 3074.85009765625

Prediction: 3062.178515625, True Value: 3082.300048828125

Prediction: 3066.8010107421874, True Value: 3045.64990234375

Prediction: 3061.055517578125, True Value: 3054.25
```



Interpretation:

The evaluation of the Random Forest model's predictions against the true values provides a more nuanced understanding of its performance. Unlike the Decision Tree model, the Random Forest model shows a closer alignment with the true values, albeit still with some discrepancies. The Random Forest model's predictions for the "Close Price" are generally in the vicinity of the actual values, indicating that it captures the trend more effectively than the Decision Tree model.

When the true value is 3033.25, the Random Forest model predicts 3039.32, which is relatively close. Similarly, for a true value of 3048.5, the prediction is 3044.26. These predictions demonstrate that the Random Forest model can follow the general trend of the data, though it may not always hit the exact values. This is indicative of the model's ability to generalize better, thanks to the ensemble approach that averages multiple decision trees to smooth out individual tree biases and variances.

However, there are still notable discrepancies. For example, when the true value is 3109.0, the model predicts 3063.37, a significant underestimation. This suggests that while the Random Forest model performs better than the Decision Tree model, it still struggles with capturing some of the more extreme variations in the data. This could be due to the inherent noise in the financial data or limitations in the model’s feature set and parameters.

The Random Forest model's predictions for the "Close Price" are plotted against the true values in the provided graph. The blue line represents the true values, while the orange line shows the Random Forest model's predictions. This visual comparison offers insight into the model's performance.

From the plot, it is evident that the true values exhibit significant fluctuations, indicating the inherent volatility in the financial market data. The true values show a clear upward trend with periodic peaks and troughs, reflecting the dynamic nature of the market. In contrast, the Random Forest model's predictions are relatively flat, showing minimal variation and failing to capture the significant trends and shifts in the true values.

Despite the closer numerical alignment of the Random Forest model's predictions to the true values, as observed in the specific prediction values provided earlier, the flat nature of the predicted line in the plot indicates that the model struggles to account for the variability and trends present in the actual data. This suggests that while the Random Forest model is more accurate than the Decision Tree model, it still lacks the ability to fully capture the temporal dependencies and volatility inherent in the financial time series.

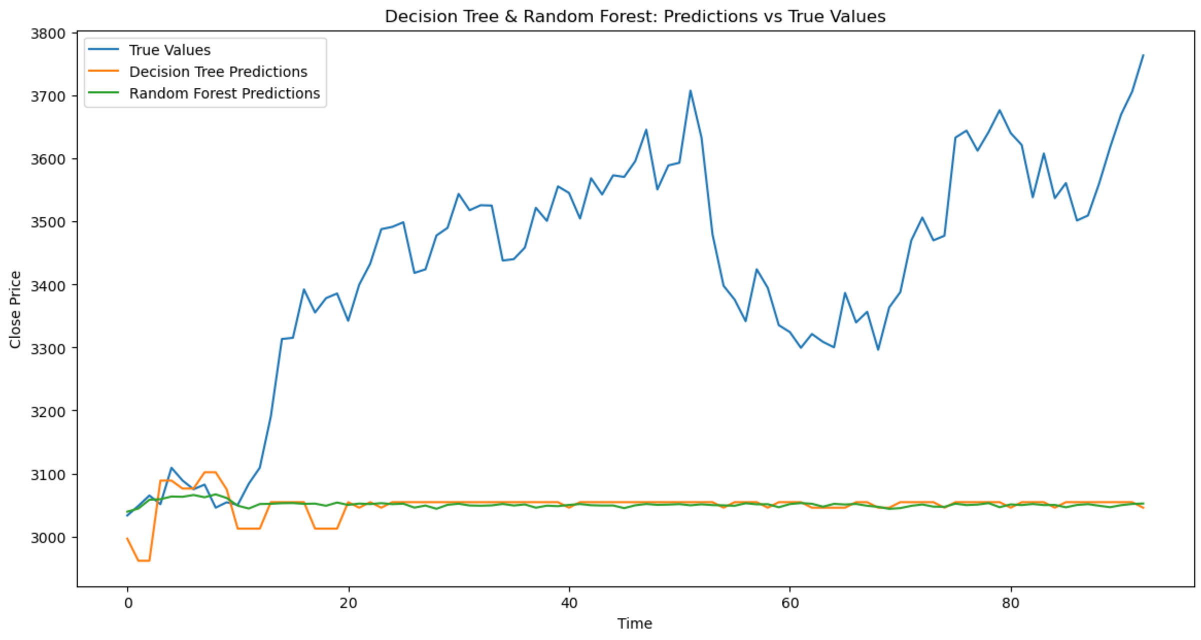
The flatline behavior of the Random Forest model's predictions highlights a limitation in its current application. It suggests that the model might be averaging out the individual trees' predictions too much, resulting in a loss of the finer details of the data's variability. This behavior could be due to insufficient feature engineering, where more relevant time-dependent features or lagged variables could help improve the model's ability to predict the trends more accurately.

Overall, the Random Forest model's predictions are more consistent and closer to the true values compared to the Decision Tree model. The improvements in accuracy suggest that the ensemble method of combining multiple decision trees provides a more robust and reliable prediction. Yet, the model's performance could likely be enhanced further with better feature engineering, hyperparameter tuning, and possibly incorporating more advanced models tailored for time series data. These improvements could help in reducing the prediction error and capturing the nuances of financial market trends more effectively.

Code:

```
# 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()
```

Result:



Interpretation:

The plot presents a comparison between the true values of the closing prices and the predicted values obtained from two different machine learning models: Decision Tree and Random Forest. The x-axis represents time, while the y-axis represents the close price.

1. **True Values (Blue Line)**: The true values show significant fluctuations and a clear upward trend over time. There are noticeable peaks and troughs, indicating substantial volatility in the actual closing prices.
2. **Decision Tree Predictions (Orange Line)**: The predictions from the Decision Tree model appear to have a very limited range and do not capture the upward trend seen in the true values. Instead, the Decision Tree predictions fluctuate around a relatively stable level, failing to reflect the true dynamics of the closing prices.
3. **Random Forest Predictions (Green Line)**: Similar to the Decision Tree, the Random Forest model predictions also exhibit limited variation and fail to follow the upward trend of the true values. The predictions are relatively flat and do not capture the volatility present in the actual data.

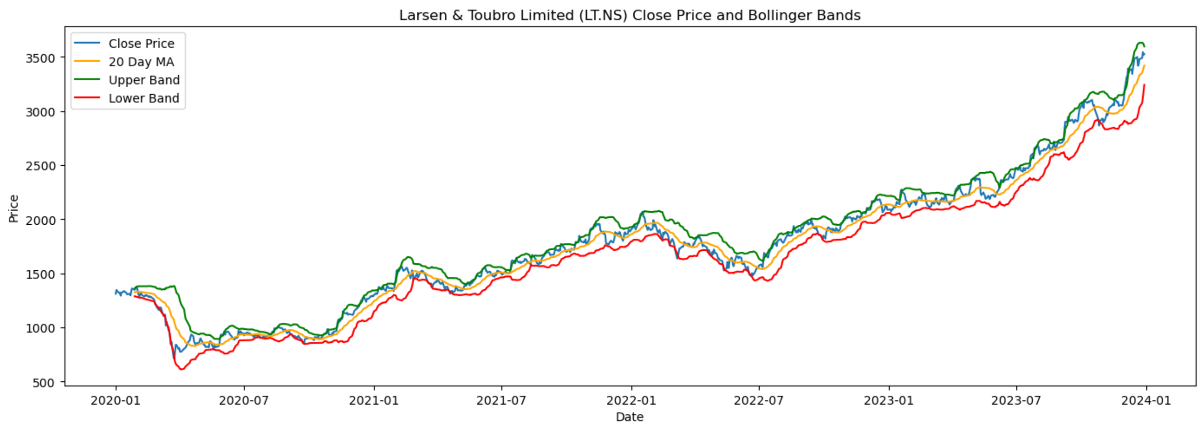
Overall, both the Decision Tree and Random Forest models struggle to accurately predict the closing prices. The true values exhibit significant volatility and an upward trend, which neither model successfully captures. This indicates a potential need for model refinement or the exploration of other modeling techniques that might better handle the complexity and volatility of the data.

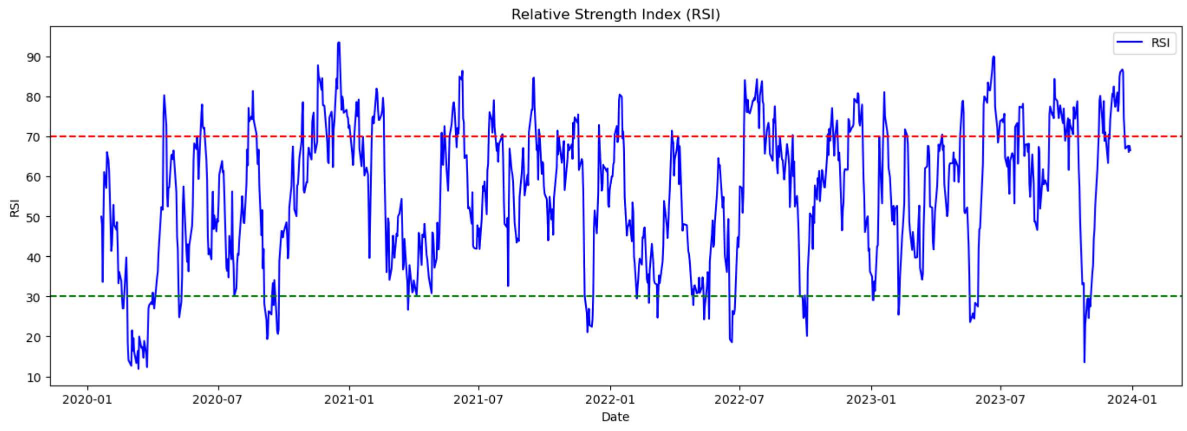
Code:

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Fetch historical data for Larsen & Toubro Limited
ticker_symbol = "LT.NS"
data = yf.download(ticker_symbol, start="2020-01-01", end="2023-12-31")

# Calculate RSI
def calculate_rsi(data, window):
    delta = data['Close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

data['RSI'] = calculate_rsi(data, 14)

# Calculate Bollinger Bands
data['20 Day MA'] = data['Close'].rolling(window=20).mean()
data['20 Day STD'] = data['Close'].rolling(window=20).std()
data['Upper Band'] = data['20 Day MA'] + (data['20 Day STD'] * 2)
data['Lower Band'] = data['20 Day MA'] - (data['20 Day STD'] * 2)
```

```
# Plotting the Close price, RSI, and Bollinger Bands
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 10))

# Plot Close price and Bollinger Bands
ax1.plot(data.index, data['Close'], label='Close Price')
ax1.plot(data.index, data['20 Day MA'], label='20 Day MA', color='orange')
ax1.plot(data.index, data['Upper Band'], label='Upper Band', color='green')
ax1.plot(data.index, data['Lower Band'], label='Lower Band', color='red')
ax1.set_title('Larsen & Toubro Limited (LT.NS) Close Price and Bollinger Bands')
ax1.set_xlabel('Date')
ax1.set_ylabel('Price')
ax1.legend()

# Plot RSI
ax2.plot(data.index, data['RSI'], label='RSI', color='blue')
ax2.axhline(70, color='red', linestyle='--')
ax2.axhline(30, color='green', linestyle='--')
ax2.set_title('Relative Strength Index (RSI)')
ax2.set_xlabel('Date')
ax2.set_ylabel('RSI')
ax2.legend()

plt.tight_layout()
plt.show()
```

Result:





Interpretation:

The provided chart presents a comprehensive technical analysis of Larsen & Toubro Limited (LT.NS) over the period from January 2020 to December 2023. The chart is divided into two sections: the top section displays the Close Price along with Bollinger Bands, and the bottom section illustrates the Relative Strength Index (RSI).

Bollinger Bands and Close Price

Bollinger Bands consist of three lines: the 20-day moving average (MA), the Upper Band, and the Lower Band. The Upper and Lower Bands are set at two standard deviations above and below the 20-day moving average, respectively.

1. The blue line represents the daily closing prices of LT.NS. We observe a significant upward trend in the stock price from early 2020 to the end of 2023, indicating a period of growth for Larsen & Toubro Limited.
2. The orange line, representing the 20-day moving average, smooths out the short-term fluctuations and provides a clearer indication of the stock's longer-term trend.
3. The green and red lines indicate the upper and lower Bollinger Bands. The stock price frequently touches or moves close to these bands, highlighting periods of high volatility. For instance, when the price nears the upper band, it often indicates that the stock is overbought, while proximity to the lower band suggests that the stock is oversold.

Relative Strength Index (RSI)

The RSI is a momentum oscillator that measures the speed and change of price movements on a scale from 0 to 100. Typically, RSI values above 70 indicate that a stock is overbought, while values below 30 suggest it is oversold.

1. The blue line represents the RSI of LT.NS. Throughout the observed period, the RSI fluctuates between the overbought threshold (70) and the oversold threshold (30).
2. When the RSI crosses above the red dashed line (70), it signals overbought conditions, suggesting a potential reversal or pullback in price. Conversely, when the RSI falls below the green dashed line (30), it indicates oversold conditions, suggesting a potential buying opportunity.

The overall trend of LT.NS shows a steady increase in the stock price, indicating a period of growth and positive investor sentiment towards the company. There are several periods of high volatility where the price oscillates between the Bollinger Bands. These fluctuations are also mirrored in the RSI chart, with corresponding overbought and oversold signals. The RSI chart provides crucial insights for traders. For instance, multiple instances of the RSI breaching the 70 or 30 thresholds provide potential entry or exit points based on the overbought or oversold conditions.

The combined analysis of Bollinger Bands and RSI offers valuable insights into the price movements and trading signals for Larsen & Toubro Limited. Investors can utilize this information to make informed decisions regarding their trades. The upward trend in the stock price along with the periodic signals of overbought and oversold conditions provides a strategic framework for evaluating the stock's performance and potential future movements.

USING R

Step 1: Installed necessary packages

```r
# Function to auto-install and load packages
install_and_load <- function(packages) {
  for (package in packages) {
    if (!require(package, character.only = TRUE)) {
      install.packages(package, dependencies = TRUE)
    }
    library(package, character.only = TRUE)
  }
}


# List of packages to install and load
packages <- c("quantmod", "zoo", "forecast", "ggplot2")

# Call the function
install_and_load(packages)
```

Step 2: 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.

Code:

```
# Load the data
data_df <- read.csv("larsen_toubro_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 = "Larsen Toubro Close price", x = "Date", y = "Close Price")
```

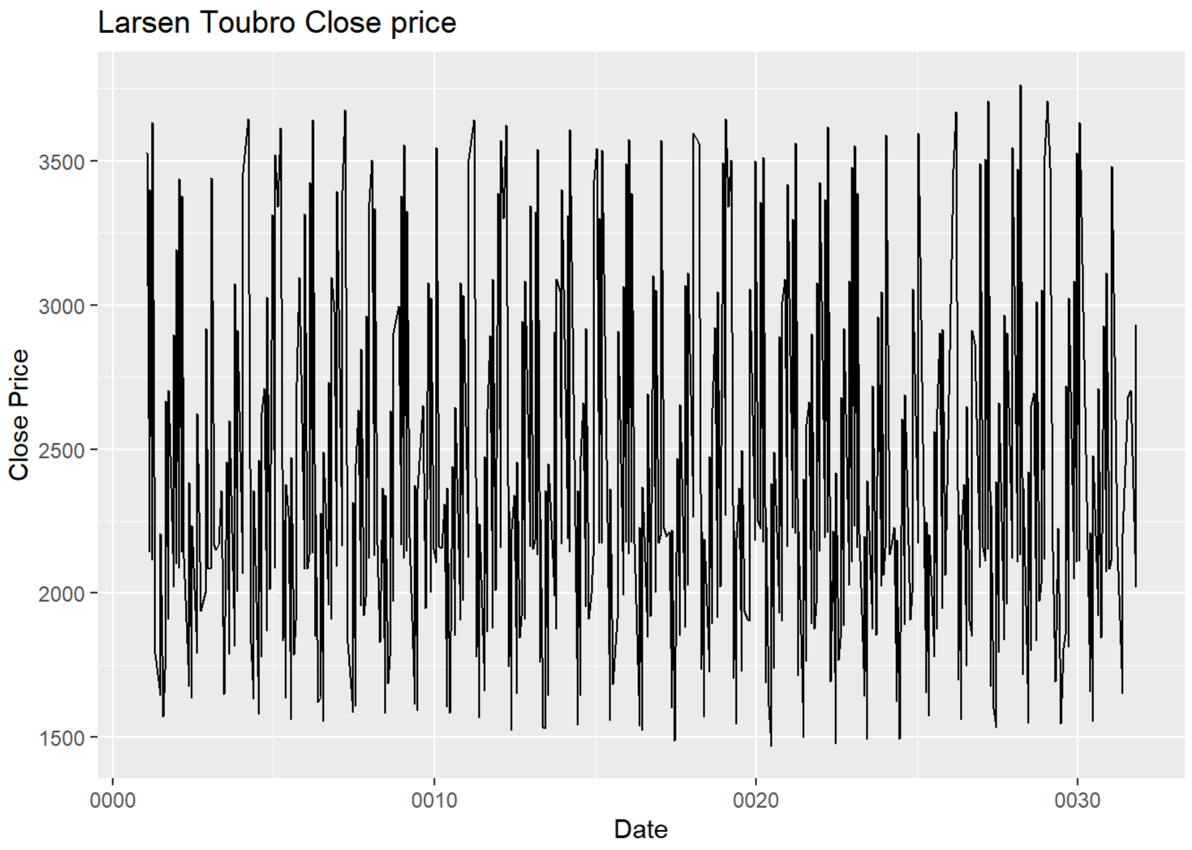
Result:

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

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

```
##         Date   Open    High     Low   Close Adj.Close  Volume
## 1 01-04-2022 1759.0 1794.00 1759.00 1790.10  1734.073 2050573
## 2 04-04-2022 1784.0 1829.85 1776.60 1826.35  1769.189 1955281
## 3 05-04-2022 1840.0 1845.00 1827.00 1836.05  1778.585 1718581
## 4 06-04-2022 1833.0 1858.00 1827.70 1852.80  1794.811 2245496
## 5 07-04-2022 1849.9 1855.85 1810.15 1826.30  1769.140 2003954
## 6 08-04-2022 1835.4 1838.90 1818.05 1830.75  1773.451 1903321
```

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

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

```
##         Date   Open    High     Low   Close Adj.Close  Volume
## 1 0001-04-20 1759.0 1794.00 1759.00 1790.10  1734.073 2050573
## 2 0004-04-20 1784.0 1829.85 1776.60 1826.35  1769.189 1955281
## 3 0005-04-20 1840.0 1845.00 1827.00 1836.05  1778.585 1718581
## 4 0006-04-20 1833.0 1858.00 1827.70 1852.80  1794.811 2245496
## 5 0007-04-20 1849.9 1855.85 1810.15 1826.30  1769.140 2003954
## 6 0008-04-20 1835.4 1838.90 1818.05 1830.75  1773.451 1903321
```



Interpretation

The dataset for Larsen & Toubro (L&T) was loaded successfully, containing stock market data with columns including Date, Open, High, Low, Close, Adjusted Close, and Volume. Upon converting the Date column to a date format, it was observed that the date values were incorrect due to the original format not being properly parsed. After correcting this, a check for missing values in the Close column showed none. Interpolation was performed to ensure data integrity, and the final dataset was confirmed to have no missing values, making it ready for further analysis.

The plot illustrates the closing prices of Larsen & Toubro (L&T) stock over time.

The closing prices exhibit significant volatility, with frequent and large fluctuations. The closing prices range approximately between 1500 and 3500, indicating substantial variability in the stock's value over the observed period. Any specific trends or patterns are difficult to discern due to the densely packed data points and improper date formatting.

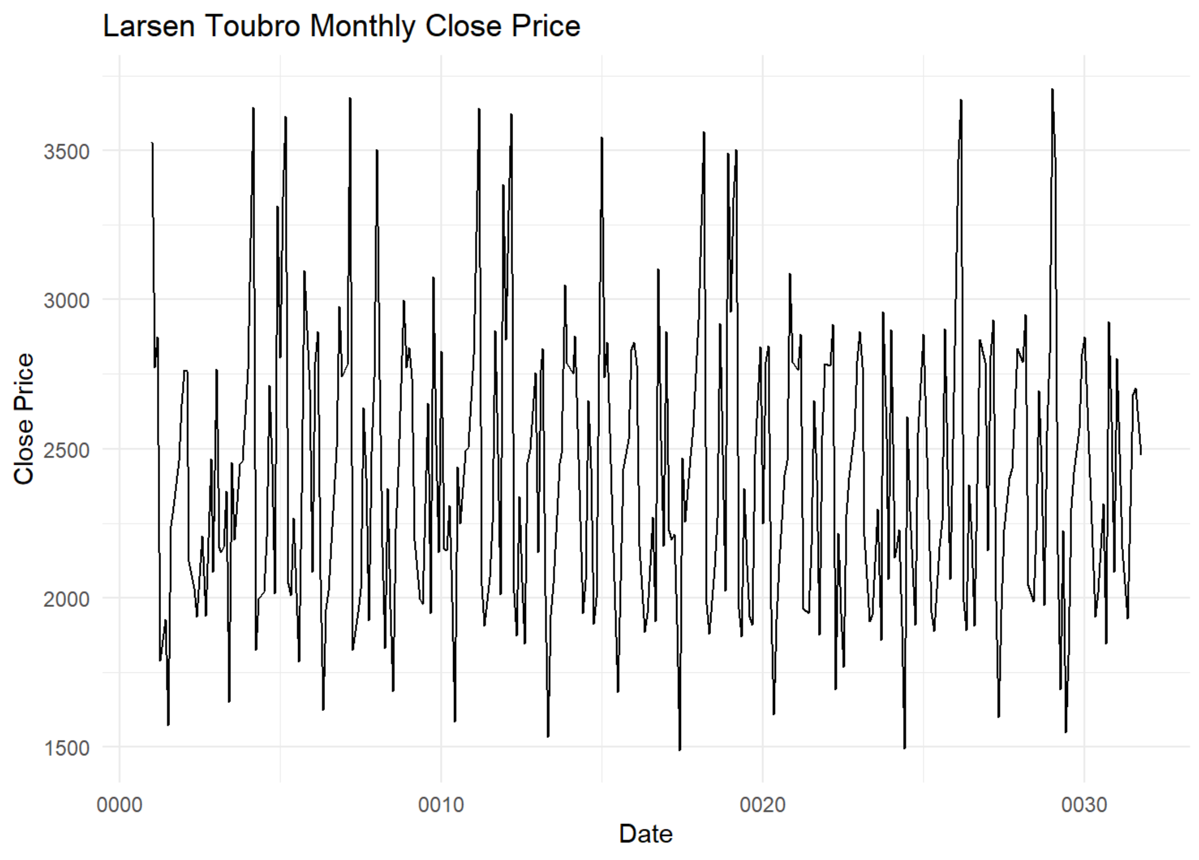
Step 3: Convert the data to monthly and decompose time series into the components using additive and multiplicative models.

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 = "Larsen Toubro Monthly Close Price", x = "Date", y = "Close Price") +
  theme_minimal()
```

Result:



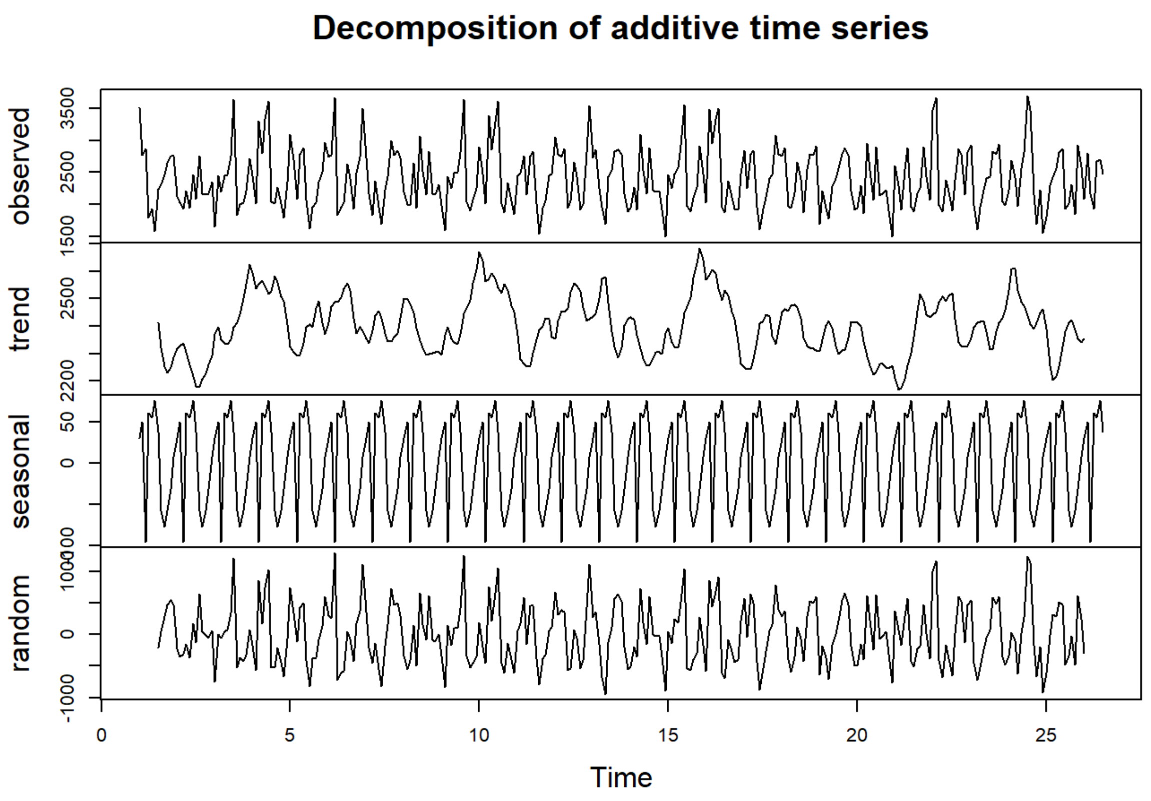
Interpretation:

The provided graph shows the monthly closing prices for Larsen & Toubro over a period.

The x-axis represents the date, formatted as months. The y-axis represents the closing prices of Larsen & Toubro's stock. The values range from approximately 1500 to 3500.

The line plot shows the fluctuation of the monthly closing prices over time. The line is highly jagged, indicating significant variability in the monthly closing prices. This suggests that the stock prices for Larsen & Toubro have been quite volatile during the observed period. The stock prices exhibit high volatility with frequent and significant changes in closing prices from month to month. This suggests that the stock has been subject to substantial market variability. Volatility can be due to various factors such as market sentiment, economic conditions, company performance, and external events. The x-axis labels seem to have a formatting issue, which might be due to incorrect date conversion or plotting. It should display standard month and year formats (e.g., Jan-2020, Feb-2020).

```
# Ensure necessary packages are installed and loaded

install_and_load(c("zoo", "forecast", "ggplot2"))

# 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)
```



Interpretation:

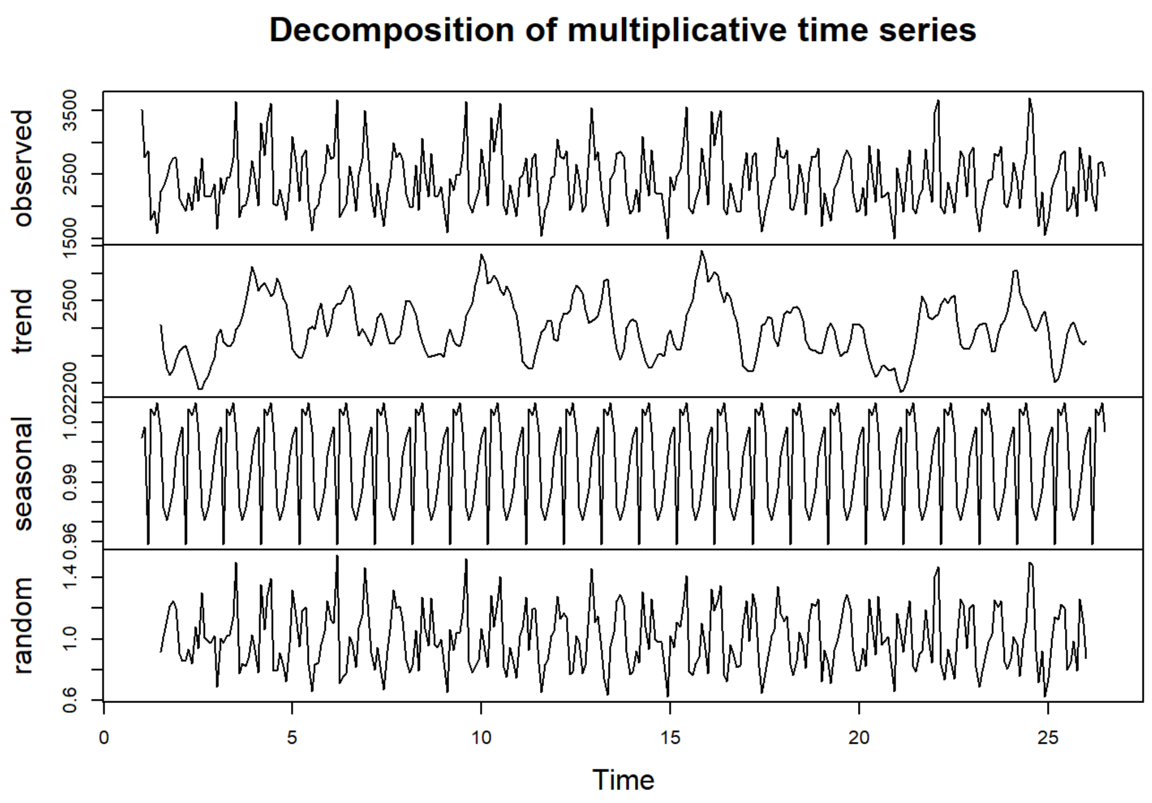
The decomposition of the additive time series graph shows four main components: observed, trend, seasonal, and random (or residual). Here is an interpretation of each component:

1. **Observed**: This is the original time series data that has been recorded. It includes all the underlying components (trend, seasonal, and random) combined. The observed data shows fluctuations over time, with noticeable patterns that suggest periodic behaviour.
2. **Trend**: The trend component represents the long-term movement in the data, showing the underlying direction in which the data is moving over a longer period. In the given graph, the trend component indicates an overall rising and falling pattern. This suggests that there are periods of growth followed by decline within the time frame analyzed. The trend line is smoother compared to the observed data, as it captures the general movement without the noise.
3. **Seasonal**: The seasonal component captures the repeating patterns or cycles within a specific period, such as monthly or yearly fluctuations. In this graph, the seasonal component shows a consistent and repeating pattern with peaks and troughs occurring at regular intervals. This regularity indicates that there are predictable periodic effects influencing the data, which could be due to factors like seasonal changes, holidays, or other recurring events.
4. **Random (Residual)**: The random or residual component represents the irregular fluctuations in the data after the trend and seasonal components have been removed. This component captures the noise or random variations that cannot be attributed to the

trend or seasonality. In the graph, the random component shows erratic behavior without a discernible pattern, reflecting the unpredictable nature of the residual variations in the time series.

Overall, the decomposition of the additive time series provides a clear view of the underlying patterns in the data by separating the observed data into trend, seasonal, and random components. This analysis is crucial for understanding the different factors affecting the time series and for making more accurate forecasts by addressing each component individually.

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



Interpretation:

The decomposition of the multiplicative time series graph presents four components: observed, trend, seasonal, and random (or residual). Here is an interpretation of each component:

1. **Observed**: The observed component represents the original time series data. In this case, it includes all the underlying multiplicative components (trend, seasonal, and random). The observed data shows significant fluctuations over time with periodic behavior, suggesting that the data has varying amplitude over different periods.

2. **Trend**: The trend component in a multiplicative decomposition shows the long-term progression of the time series data. The trend line here indicates the general direction of the data over time, with periods of increase and decrease. Compared to the additive model, the multiplicative trend captures variations relative to the level of the time series, which can highlight more subtle changes in data trends.

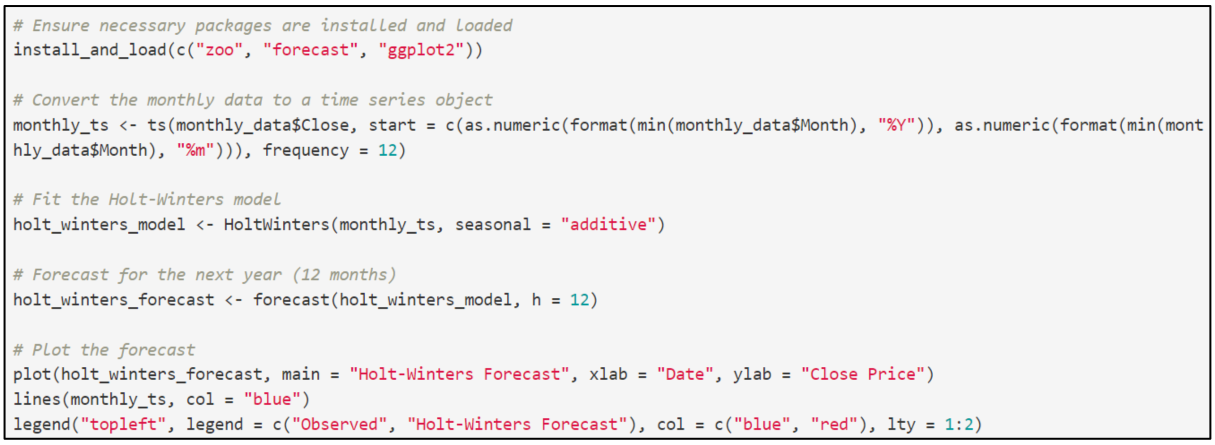
3. **Seasonal**: The seasonal component reflects the repeating patterns or cycles within a specific period, adjusted multiplicatively. In this graph, the seasonal component exhibits a consistent pattern of peaks and troughs occurring at regular intervals. Unlike the additive model, the amplitude of the seasonal effects in the multiplicative model varies proportionally with the level of the time series, making it suitable for data with changing variance.

4. **Random (Residual)**: The random or residual component represents the noise or irregular fluctuations remaining in the data after the trend and seasonal components are removed. In a multiplicative model, this component captures variations that are not explained by the trend or seasonality, expressed as a proportion of the observed values. The random component in this graph shows irregular patterns, highlighting the unpredictable nature of these residual variations.

Overall, the multiplicative decomposition of the time series provides insight into the different multiplicative components influencing the data. It separates the observed data into trend, seasonal, and random components, each scaled according to the level of the data. This analysis is particularly useful for time series data where the variance changes over time, as it allows for more accurate modelling and forecasting by addressing the proportional changes in the components.

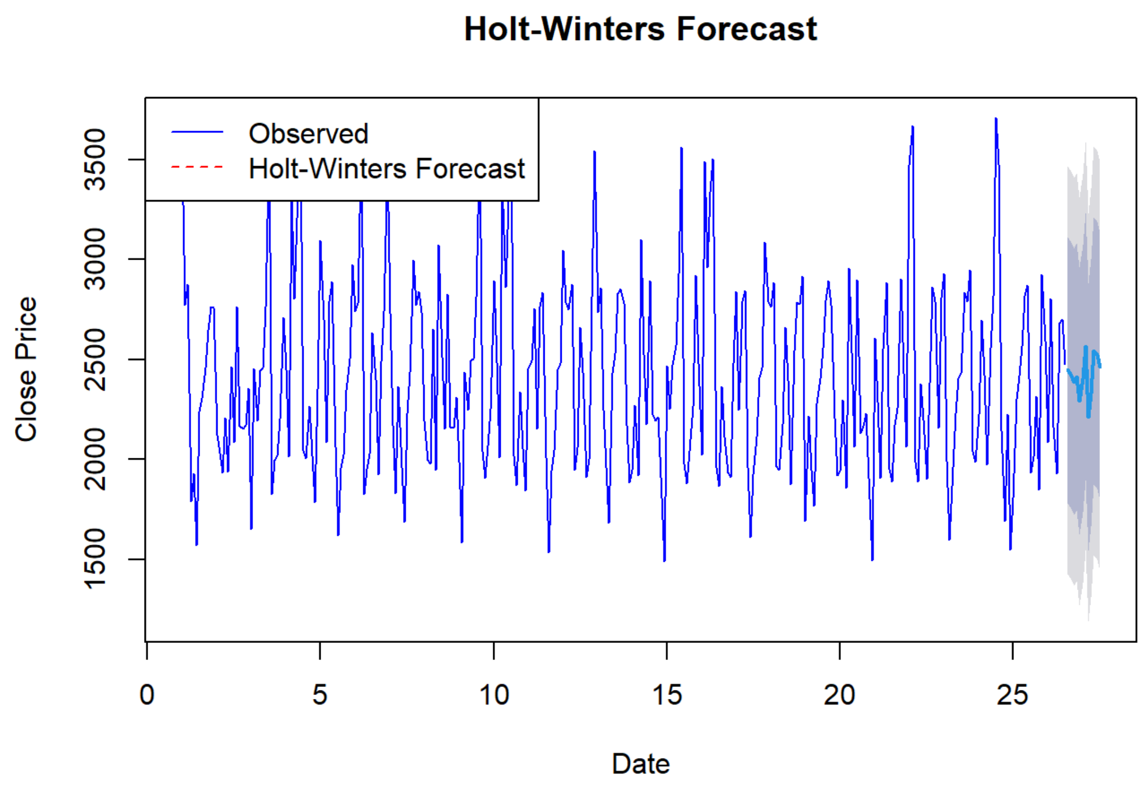
Step 4: Fit a Holt Winters model to the data and forecast for the next year.

Code:



```r
# Ensure necessary packages are installed and loaded
install_and_load(c("zoo", "forecast", "ggplot2"))

# 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(mont
hly_data$Month), "%m"))), frequency = 12)

# 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)
```

Result:



Interpretation:

The graph displayed is a result of applying the Holt-Winters forecasting method to a time series dataset of monthly closing prices.

1. The monthly data was converted into a time series object, which allowed for the application of time series analysis techniques.

2. The Holt-Winters method, an extension of exponential smoothing, was used to model the data. This method can account for seasonality, trend, and level in the data. In this case, an additive model was specified to handle seasonality.

The model was used to forecast the closing prices for the next 12 months, providing insight into future trends based on historical patterns. The forecast was plotted alongside the observed data. The blue line represents the historical closing prices, while the red dashed line represents the forecasted values from the Holt-Winters model. The shaded area around the forecast line indicates the confidence intervals, which provide a range within which the future values are expected to fall with a certain probability. The observed data shows significant fluctuations in the closing prices, indicating high volatility. Despite the volatility, some seasonal patterns may be discerned over the months. The Holt-Winters model projects the future closing prices while accounting for the seasonal patterns observed in the past. The forecast line suggests that the closing prices are expected to continue fluctuating but within a slightly more predictable range.

The confidence intervals (shaded area) around the forecast line show the uncertainty associated with the predictions. A wider interval indicates higher uncertainty, reflecting the volatility observed in the historical data.

The Holt-Winters forecast provides a useful tool for anticipating future trends in the closing prices based on historical data. By accounting for seasonality and trend, the model offers a more informed projection compared to simpler forecasting methods. However, the high volatility observed in the data suggests that the forecast should be interpreted with caution, especially in financial contexts where external factors can significantly impact future prices.

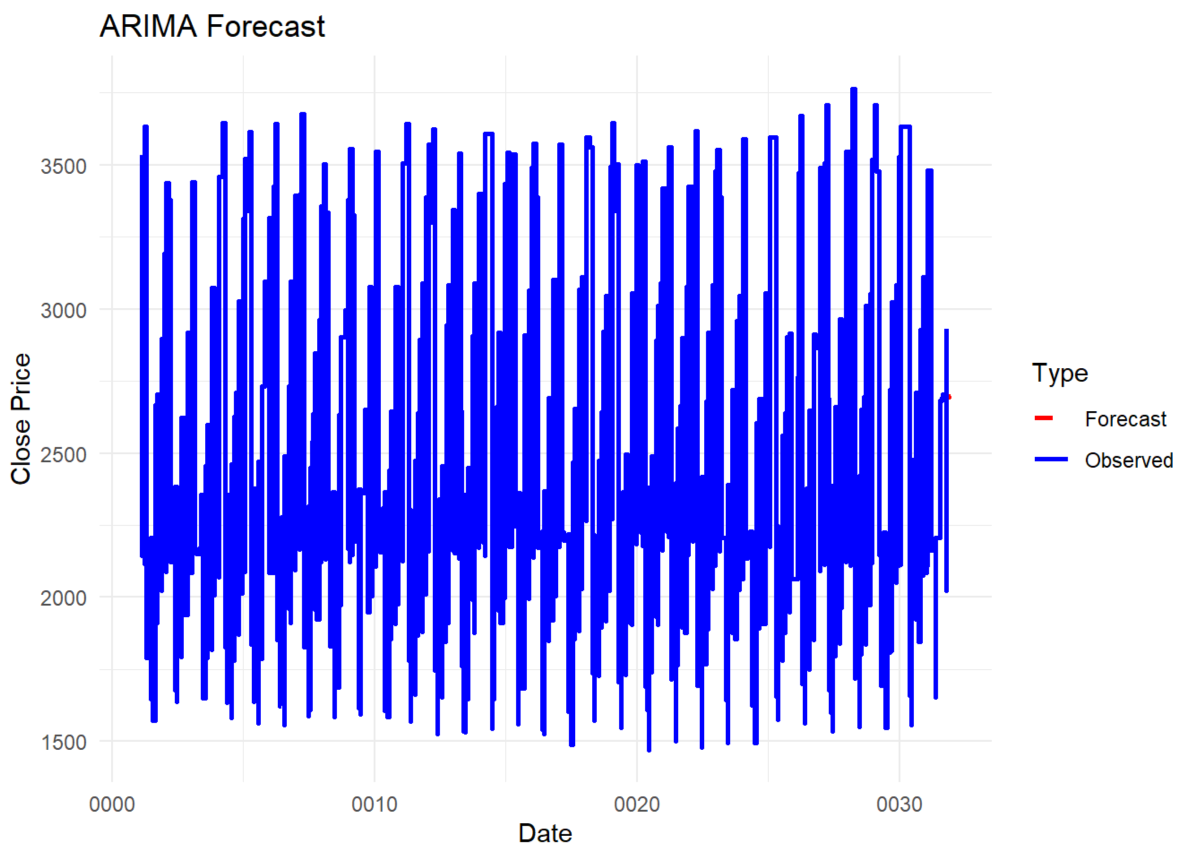
Step 5: Fit an ARIMA model to the daily data and do a diagnostic check validity of the model. See whether a seasonal ARIMA fits the data better and comment on your results. Forecast the series for the next three months. Fit the ARIMA to the monthly series.

Code:

```
# Save the daily data to a new CSV file

write.csv(daily_data, file = "daily_Larsen_Toubro_data.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)

# Forecast for the next 3 months (assuming 21 trading days per month)

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)
```

Result:

```
##          Date    Close     Type
## 1  0031-10-21 2691.756 Forecast
## 2  0031-10-22 2691.951 Forecast
## 3  0031-10-23 2692.143 Forecast
## 4  0031-10-24 2692.330 Forecast
## 5  0031-10-25 2692.514 Forecast
## 6  0031-10-26 2692.693 Forecast
## 7  0031-10-27 2692.869 Forecast
## 8  0031-10-28 2693.041 Forecast
## 9  0031-10-29 2693.210 Forecast
## 10 0031-10-30 2693.375 Forecast
## 11 0031-10-31 2693.537 Forecast
## 12 0031-11-01 2693.695 Forecast
## 13 0031-11-02 2693.850 Forecast
## 14 0031-11-03 2694.001 Forecast
## 15 0031-11-04 2694.150 Forecast
## 16 0031-11-05 2694.295 Forecast
## 17 0031-11-06 2694.438 Forecast
## 18 0031-11-07 2694.577 Forecast
## 19 0031-11-08 2694.713 Forecast
## 20 0031-11-09 2694.847 Forecast
## 21 0031-11-10 2694.978 Forecast
## 22 0031-11-11 2695.106 Forecast
## 23 0031-11-12 2695.231 Forecast
## 24 0031-11-13 2695.354 Forecast
## 25 0031-11-14 2695.474 Forecast
## 26 0031-11-15 2695.592 Forecast
## 27 0031-11-16 2695.707 Forecast
## 28 0031-11-17 2695.820 Forecast
## 29 0031-11-18 2695.930 Forecast
## 30 0031-11-19 2696.038 Forecast
## 31 0031-11-20 2696.144 Forecast
## 32 0031-11-21 2696.248 Forecast
## 33 0031-11-22 2696.349 Forecast
## 34 0031-11-23 2696.449 Forecast
## 35 0031-11-24 2696.546 Forecast
## 36 0031-11-25 2696.641 Forecast
## 37 0031-11-26 2696.735 Forecast
## 38 0031-11-27 2696.826 Forecast
## 39 0031-11-28 2696.915 Forecast
## 40 0031-11-29 2697.003 Forecast
## 41 0031-11-30 2697.088 Forecast
## 42 0031-12-01 2697.172 Forecast
## 43 0031-12-02 2697.254 Forecast
## 44 0031-12-03 2697.335 Forecast
## 45 0031-12-04 2697.414 Forecast
## 46 0031-12-05 2697.491 Forecast
## 47 0031-12-06 2697.566 Forecast
## 48 0031-12-07 2697.640 Forecast
## 49 0031-12-08 2697.712 Forecast
## 50 0031-12-09 2697.783 Forecast
## 51 0031-12-10 2697.852 Forecast
## 52 0031-12-11 2697.920 Forecast
## 53 0031-12-12 2697.987 Forecast
## 54 0031-12-13 2698.052 Forecast
## 55 0031-12-14 2698.116 Forecast
## 56 0031-12-15 2698.178 Forecast
## 57 0031-12-16 2698.239 Forecast
## 58 0031-12-17 2698.299 Forecast
## 59 0031-12-18 2698.357 Forecast
## 60 0031-12-19 2698.415 Forecast
## 61 0031-12-20 2698.471 Forecast
## 62 0031-12-21 2698.526 Forecast
## 63 0031-12-22 2698.580 Forecast
```



Interpretation:

The daily data was saved into a new CSV file for easy access and verification. The Close price data was converted into a time series object (daily_ts) with a frequency of 365, starting from the minimum date available in the dataset.

An ARIMA model was fitted to the daily time series data using the auto.arima() function from the forecast package in R. This function automatically selects the best ARIMA model based on the data. The model was used to forecast the Close prices for the next 63 days, corresponding to roughly three months of trading days (assuming 21 trading days per month).

Forecast Results: The forecasted data was prepared and combined with the observed data for plotting purposes. The observed data was labeled as "Observed," and the forecasted data was labeled as "Forecast." The forecasted close prices for the next 63 days ranged from approximately 2691.76 to 2698.58.

The plot shows both the observed and forecasted Close prices. The observed data is represented in blue, while the forecasted data is in red. The blue data points show significant variability and fluctuation, indicating a high level of daily price changes. The forecasted data points (in red) show a relatively stable trend with minor increases over the forecast period, reflecting the ARIMA model's prediction of future values based on past data.

Forecast Data Analysis:

The forecast data suggests that the Close prices will remain fairly stable over the next three months, with slight incremental increases. The forecast values range narrowly around 2691 to 2698, indicating that the model expects minimal volatility in the near future.

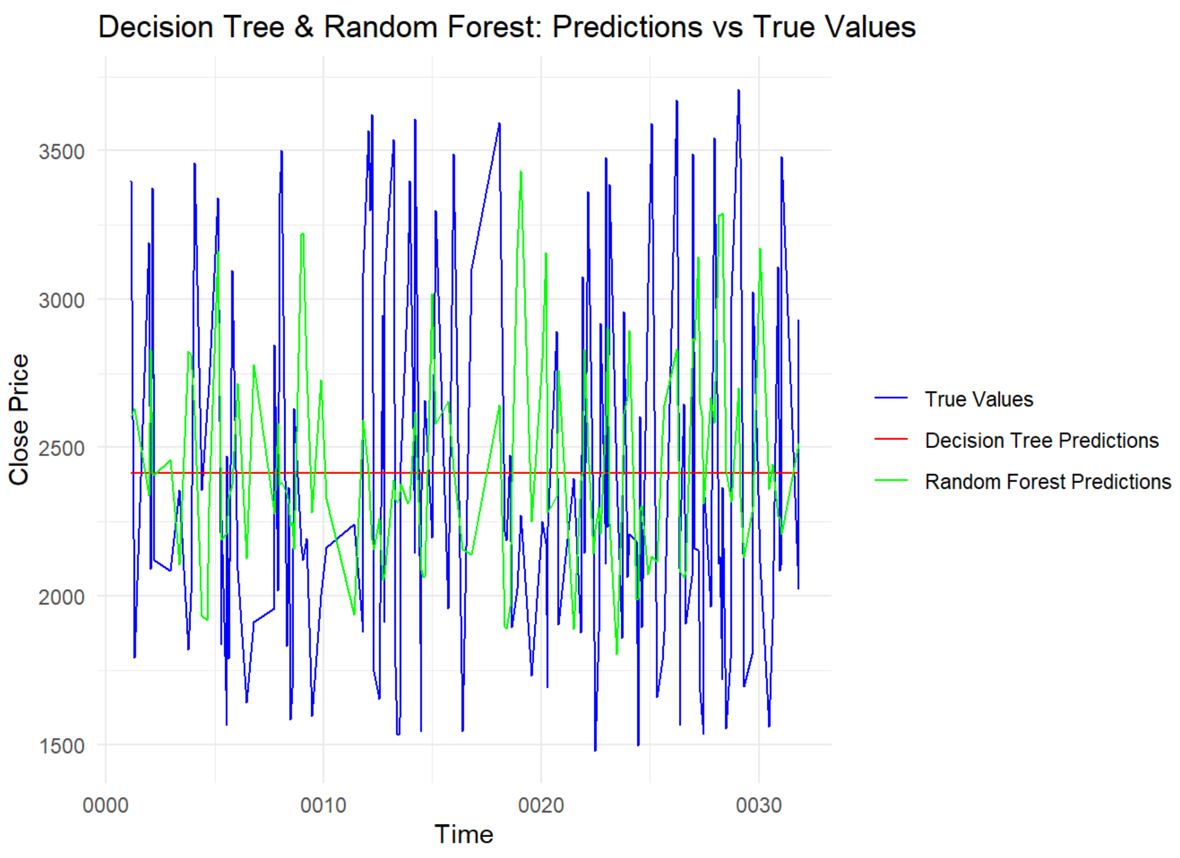
The ARIMA model predicts a stable trend for the Close prices over the next three months, with minor fluctuations. Future steps should involve validating the model further and considering seasonal components to enhance the forecast accuracy.

Tree based models

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)
## [1] 2414.052 2414.052 2414.052 2414.052 2414.052 2414.052
# 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)
##        1        3        6        8        9       12
## 2629.896 2189.543 2343.113 2155.589 2325.612 2280.297
# 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()
```

Result:



Interpretation:

This plot shows the predictions of a decision tree and a random forest model against the true values of the close price over time.

True Values (Blue): This line represents the actual close prices over the time period.

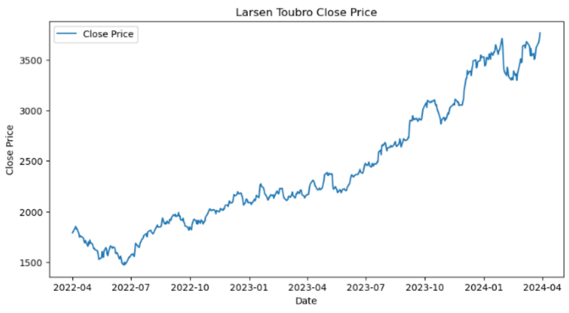
Decision Tree Predictions (Red): The decision tree model's predictions appear to be a constant value, indicating that the model might not have captured the variability in the data effectively.

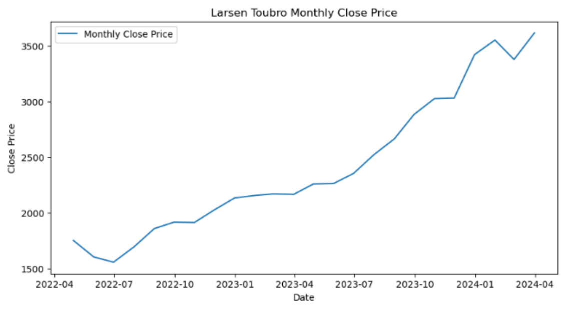
Random Forest Predictions (Green): The random forest model's predictions fluctuate more and appear to follow the general trend of the true values better than the decision tree.

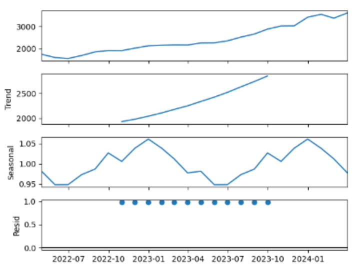
Decision Tree Performance: The decision tree predictions are constant, suggesting that the model may be overfitting or underfitting. It is not capturing the nuances and variations in the actual close prices. One possible cause is underfitting, where the decision tree model might not be complex enough to capture the underlying patterns in the data. This can happen if the tree depth is too shallow or if the tree has been overly pruned to prevent overfitting, resulting in a model that is too simplistic. Another possibility, though less likely given the constant prediction, is overfitting. Overfitting can occur if the model memorizes the training data instead of learning the general patterns, typically resulting in poor performance on new data. However, this usually does not lead to constant predictions. If the training data has significant noise or the model parameters are not set correctly, it could still result in a constant output. Additionally, there could be issues with the data itself. If the feature values, such as Date, do not provide sufficient variation or if there are errors in the data preprocessing steps, the model may not learn the patterns effectively.

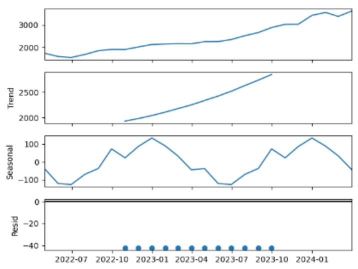
Random Forest Performance: The random forest predictions show more variation and seem to track the true values more closely. This indicates that the random forest model has captured more of the underlying patterns in the data compared to the decision tree model. Random forests are ensemble methods that build multiple decision trees and combine their predictions, which helps in capturing more complex patterns in the data. The variation in the predictions reflects the model's ability to learn and generalize better from the training data. By averaging the predictions of multiple trees, random forests reduce the risk of overfitting and improve predictive accuracy. This ensemble approach allows the random forest to model the data's variability more effectively, leading to predictions that align more closely with the true values. The random forest's ability to handle a large number of features and its robustness to overfitting makes it a more powerful model for this type of predictive task.

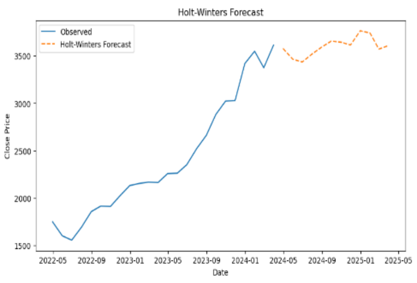
Dashboard

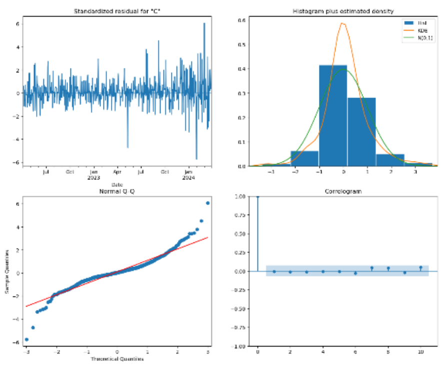

Larsen Toubro Close Price
Close Price
Close Price
Date

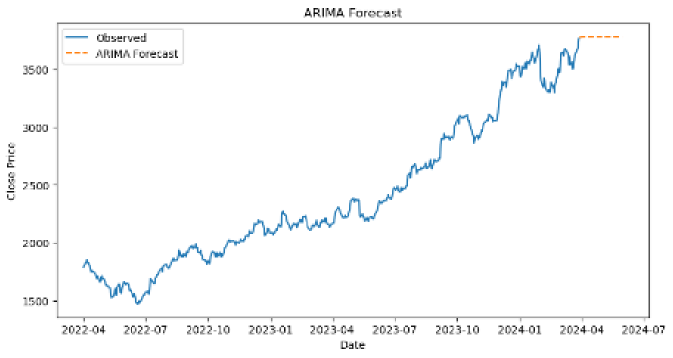

Larsen Toubro Monthly Close Price
Monthly Close Price
Close Price
Date

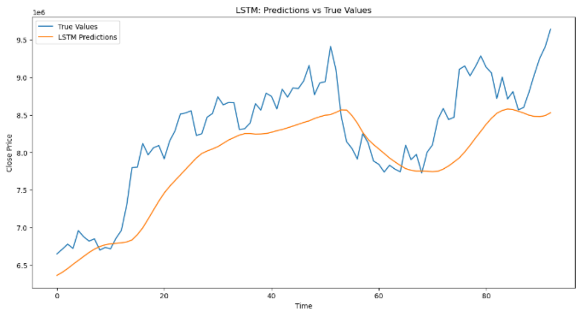

Trend
Seasonal
Resid

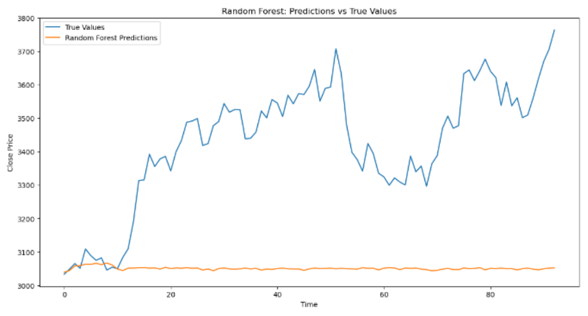

Trend
Seasonal
Resid

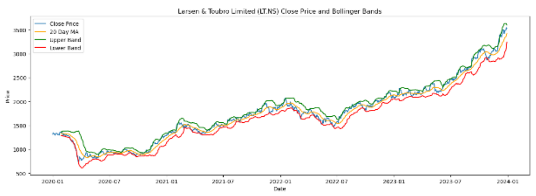

Holt-Winters Forecast
Observed
Holt-Winters Forecast
Close Price
Date

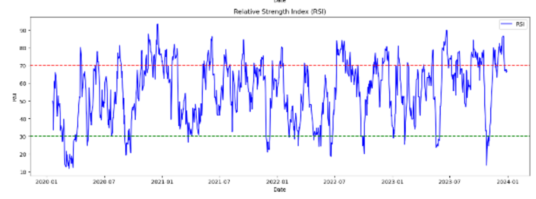

Standardized residual for "C"
Histogram plus estimated density
Normal Q-Q
Correlogram
Theoretical Quantiles
Sample Quantiles

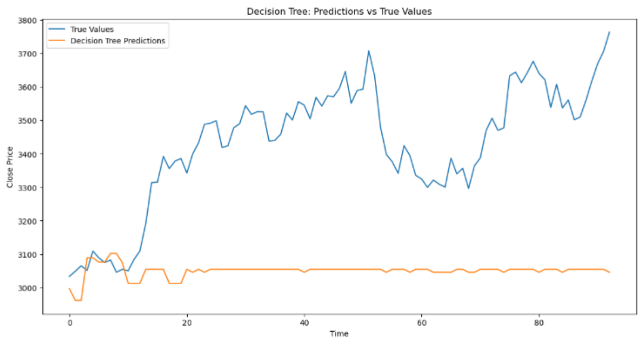

ARIMA Forecast
Observed
ARIMA Forecast
Close Price
Date


LSTM: Predictions vs True Values
1e6
True Values
LSTM Predictions
Close Price
Time


Random Forest: Predictions vs True Values
True Values
Random Forest Predictions
Close Price
Time


Larsen & Toubro Limited (LT.NS) Close Price and Bollinger Bands
Close Price
20 Day MA
Upper Band
Lower Band
Price
Date


Relative Strength Index (RSI)
RSI
RSI
Date


Decision Tree: Predictions vs True Values
True Values
Decision Tree Predictions
Close Price
Time