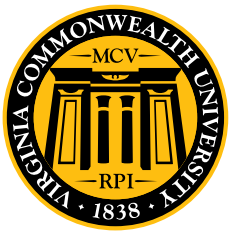


VIRGINIA COMMONWEALTH UNIVERSITY

Statistical Analysis and Modelling (SCMA 632)

A6: Time Series Analysis

NIHARIHA KAMALANATHAN
V01108259
Date of Submission: 22-07-2024

CONTENTS

QUESTION: Choose the stock or share of your choice and download the data form www.investing.com or yfinance library

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

1. Univariate Forecasting - Conventional Models/Statistical Models
− Fit a Holt Winters model to the data and forecast for the next year.
− Fit 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.
− Fit the ARIMA to the monthly series.
2. Multivariate Forecasting - Machine Learning Models
− NN (Neural Networks) -Long Short-term Memory (LSTM)
− Tree based models - Random Forest, Decision Tree

Introduction

Adani Ports and Special Economic Zones (APSEZ) is India's largest commercial port operator and an integral part of the Adani Group. APSEZ operates multiple ports along India's coastline, handling a substantial portion of the country's maritime cargo. The financial performance of APSEZ is closely watched by investors, analysts, and policymakers due to its significant impact on India's trade and economic activities.
This analysis leverages historical stock price data of APSEZ to gain insights into its financial performance over time. The dataset includes daily records of key financial indicators such as opening price, high price, low price, closing price, trading volume, and daily percentage change in price. The primary objectives of this analysis are to clean the data, detect and handle outliers and missing values, decompose the time series data into its components, and apply various forecasting models to predict future stock prices. This comprehensive analysis aims to provide actionable insights for investors, analysts, and business strategists.

Business Significance

1. Investment Decisions: Accurate and reliable forecasting of stock prices is essential for investors aiming to make informed decisions regarding buying, holding, or selling APSEZ shares. By understanding future price trends, investors can optimize their investment portfolios and maximize returns.
2. Risk Management: Financial analysts and portfolio managers rely on precise stock price predictions to manage risks associated with their investments. Understanding the potential future movements of APSEZ stock prices helps in hedging strategies and mitigating market risks.
3. Market Analysis: Companies and financial institutions use stock price analysis to evaluate APSEZ's performance in the market. Comparing APSEZ's stock trends with its competitors provides valuable insights into its market position and competitive advantage.
4. Strategic Planning: Companies can utilize stock price trends for strategic financial planning. Whether it's planning for mergers and acquisitions, capital investments, or corporate restructuring, a clear understanding of stock price movements informs strategic decisions.

5. Policy Making: Regulatory bodies and policymakers monitor stock price trends to ensure market stability and integrity. Analyzing APSEZ's stock performance can help in crafting policies that promote fair trading practices and protect investor interests.
6. Resource Allocation: Businesses can use stock price analysis to assess the financial health of APSEZ, guiding decisions on resource allocation, operational expansions, and market entry strategies.

Objectives

1. Comprehensive Data Cleaning and Preprocessing: The initial objective is to clean the dataset by handling missing values and outliers, ensuring the data is in a suitable format for further analysis. This step is crucial for maintaining the integrity and accuracy of the analysis.
2. Time Series Decomposition: Decompose the time series data into its fundamental components - trend, seasonal, and random variations. This decomposition provides a clearer understanding of the underlying patterns and behaviors in the stock price data over time.
3. Univariate Forecasting:
    - Holt-Winters Model: Fit a Holt-Winters model to the data to capture seasonal effects and trends, and forecast the stock prices for the next year.
    - ARIMA Model: Apply an ARIMA model to the daily stock price data, perform diagnostic checks to validate the model, and evaluate whether a Seasonal-ARIMA (SARIMA) model fits the data better. Forecast the stock prices for the next three months.
    - Monthly ARIMA Model: Fit an ARIMA model to the aggregated monthly stock price data to capture broader trends and seasonal patterns.
4. Multivariate Forecasting:
    - Neural Networks (LSTM): Implement Long Short-Term Memory (LSTM) networks to forecast stock prices, leveraging the model's ability to handle long-term dependencies in time series data.
    - Tree-Based Models: Apply tree-based models such as Random Forest and Decision Tree to predict stock prices, utilizing their robustness in handling non-linear relationships and interactions among variables.
5. Introduction to APSEZ Stock Analysis: Provide a detailed introduction to the dataset, including its origin, structure, and the significance of each variable. Explain the importance of the analysis in the context of APSEZ's business operations and market performance.
6. Business Insights and Recommendations: Translate the analytical findings into actionable insights for investors, businesses, and policymakers. Provide recommendations based on the forecasted trends and patterns, highlighting potential opportunities and risks.

By achieving these objectives, this analysis aims to deliver a comprehensive understanding of APSEZ's stock price dynamics, offering valuable insights that can enhance investment strategies, risk management practices, and strategic business decisions.

R Language

1. Loading Necessary Packages
Code:

```
packages <- c("tidyverse", "dplyr", "lubridate", "imputeTS", "ggplot2", "cowplot", "zoo",
"forecast", "nnet", "randomForest", "rpart", "caret", "glmnet", "gbm")

install.packages(setdiff(packages, rownames(installed.packages())))
lapply(packages, library, character.only = TRUE)
```

Purpose:

- Specified a list of required packages.
- Install any packages that are not already installed on the system.
- Load the installed packages into the R session.

Output:

- The necessary packages are installed and loaded.

Interpretation:

- This step ensures that all required libraries for data manipulation, visualization, time series analysis, and machine learning are available for subsequent analysis.

2. Loading the Dataset

Code:

```
file_path <- "C:/Users/nihar/OneDrive/Desktop/Bootcamp/SCMA 632/DataSet/APSE
Historical Data.csv"
data <- read.csv(file_path)
head(data)
```

Purpose:

- Load the stock price dataset from a CSV file and display the first few rows.

Output:

```
# View the first few rows of the dataset
> head(data)
      Date    Price     Open     High      Low  Vol. Change..
1 03/28/2024 1,341.85 1,334.00 1,358.70 1,314.00 4.19M    1.44%
2 03/27/2024 1,322.80 1,315.00 1,343.00 1,312.10 4.94M    1.43%
3 03/26/2024 1,304.20 1,281.60 1,314.00 1,281.00 3.61M    1.76%
4 03/22/2024 1,281.60 1,262.45 1,289.10 1,255.05 3.08M    1.52%
5 03/21/2024 1,262.45 1,260.00 1,268.70 1,251.30 3.16M    1.37%
6 03/20/2024 1,245.40 1,248.00 1,252.95 1,230.20 2.77M    0.34%
```

Interpretation:

- Provides a quick glimpse of the dataset's structure and the types of data it contains, which is essential for understanding and cleaning the data.

3. Data Preprocessing

Code:

```
# Convert Date column to Date type in yyyy-mm-dd format
data$Date <- as.Date(mdy(data$Date))

# Check the structure of the dataset
str(data)
```

```
# Check for missing values
sum(is.na(data))

# Ensure that the Price column is numeric
data$Price <- as.numeric(gsub(",", "", data$Price))

# Convert other relevant columns to numeric after removing non-numeric characters
data$Open <- as.numeric(gsub(",", "", data$Open))
data$High <- as.numeric(gsub(",", "", data$High))
data$Low <- as.numeric(gsub(",", "", data$Low))
data$Change <- as.numeric(gsub("%", "", data$Change..))

# Clean the Vol. column by removing any non-numeric characters and handle both M and K
values
data$Vol <- sapply(data$Vol., function(x) {
  if (grepl("M", x)) {
    as.numeric(gsub("M", "", x)) * 1e6
  } else if (grepl("K", x)) {
    as.numeric(gsub("K", "", x)) * 1e3
  } else {
    NA
  }
})

# Check for NAs in Vol. after conversion
sum(is.na(data$Vol))

# Impute the NA value in Vol. column with the median
data$Vol[is.na(data$Vol)] <- median(data$Vol, na.rm = TRUE)

# Check for NAs in Vol. after imputation
sum(is.na(data$Vol))

# Check the structure of the dataset to confirm changes
str(data)
```

Purpose:

- Convert the Date column to Date type.
- Check the structure of the dataset.
- Check for missing values.
- Ensure numeric columns (Price, Open, High, Low, Change) are correctly formatted.
- Clean and convert the Vol. column to numeric, handling both 'M' (millions) and 'K' (thousands) values.
- Impute missing values in Vol. with the median.
-

Output:

```
# Check for missing values
> sum(is.na(data))
[1] 0
```

```
> # Ensure that the Price column is numeric
> data$Price <- as.numeric(gsub(",", "", data$Price))
> # Convert other relevant columns to numeric after removing non-numeric characters
> data$Open <- as.numeric(gsub(",", "", data$Open))
> data$High <- as.numeric(gsub(",", "", data$High))
> data$Low <- as.numeric(gsub(",", "", data$Low))
> data$Change <- as.numeric(gsub("%", "", data$Change..))
> # Clean the Vol. column by removing any non-numeric characters and handle both M and
K values
> data$Vol <- sapply(data$Vol., function(x) {
+   if (grepl("M", x)) {
+     as.numeric(gsub("M", "", x)) * 1e6
+   } else if (grepl("K", x)) {
+     as.numeric(gsub("K", "", x)) * 1e3
+   } else {
+     NA
+   }
+ })
> # Check for NAs in Vol. after conversion
> sum(is.na(data$Vol))
[1] 1
>
> # Impute the NA value in Vol. column with the median
> data$Vol[is.na(data$Vol)] <- median(data$Vol, na.rm = TRUE)
>
> # Check for NAs in Vol. after imputation
> sum(is.na(data$Vol))
[1] 0
> # Check the structure of the dataset to confirm changes
> str(data)
'data.frame':        2470 obs. of  9 variables:
 $ Date    : Date, format: "2024-03-28" "2024-03-27" ...
 $ Price   : num  1342 1323 1304 1282 1262 ...
 $ Open    : num  1334 1315 1282 1262 1260 ...
 $ High    : num  1359 1343 1314 1289 1269 ...
 $ Low     : num  1314 1312 1281 1255 1251 ...
 $ Vol.    : chr  "4.19M" "4.94M" "3.61M" "3.08M" ...
 $ Change..: chr  "1.44%" "1.43%" "1.76%" "1.52%" ...
 $ Change  : num  1.44 1.43 1.76 1.52 1.37 0.34 -2.01 -1.28 1.19 4.82 ...
 $ Vol     : num  4190000 4940000 3610000 3080000 3160000 2770000 2310000 4860000
4020000 7200000 ...
```

Interpretation:

- Proper data types and cleaning are crucial for accurate analysis and modeling. This step ensures the dataset is correctly formatted for further analysis.

4. Outlier Detection and Plotting

Code:

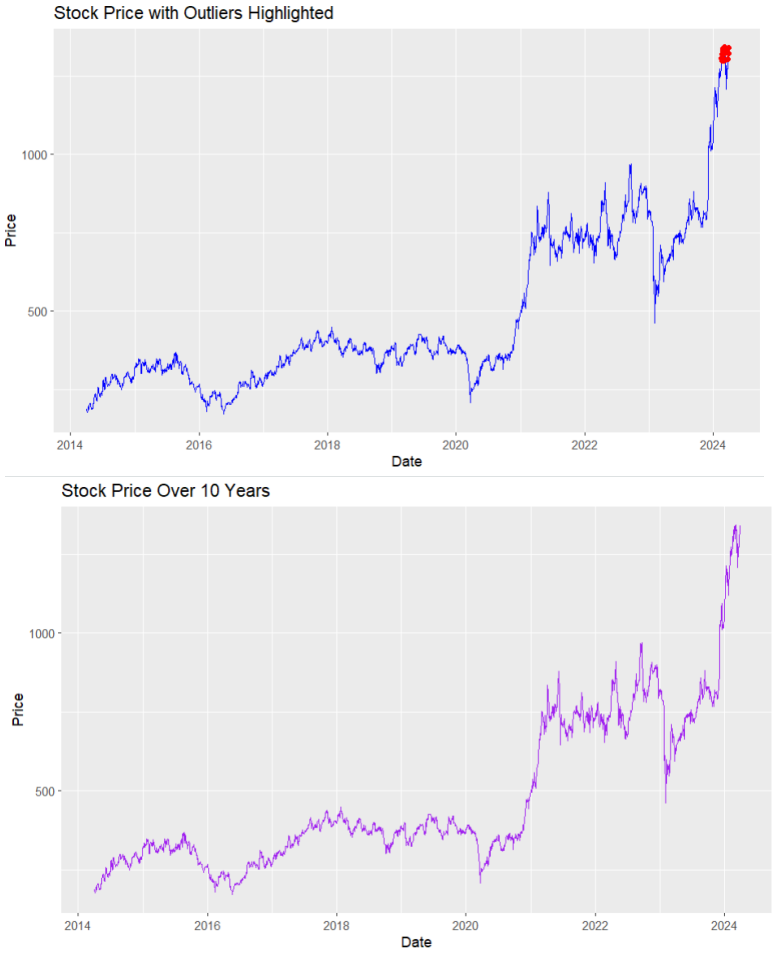
```
# Identify outliers using IQR method for the Price column
Q1 <- quantile(data$Price, 0.25)
```

```
Q3 <- quantile(data$Price, 0.75)
IQR <- Q3 - Q1
outliers <- data %>% filter(Price < (Q1 - 1.5 * IQR) | Price > (Q3 + 1.5 * IQR))

# Plot outliers
ggplot(data, aes(x = Date, y = Price)) +
  geom_line(color = "blue") +
  geom_point(data = outliers, aes(x = Date, y = Price), color = "red") +
  ggtitle("Stock Price with Outliers Highlighted")

# Plot a line graph for the price of all ten years
ggplot(data, aes(x = Date, y = Price)) +
  geom_line(color = "purple") +
  ggtitle("Stock Price Over 10 Years")
```

Purpose:

- Identify outliers in the Price column using the Interquartile Range (IQR) method.
- Plot the stock price over time, highlighting outliers.
- Plot the stock price over the entire dataset duration.

Output:



Interpretation:

Stock Price with Outliers Highlighted:

- This plot shows the stock price over time, with outliers highlighted in red. The red dots at the end of the timeline indicate recent significant deviations from the overall trend, which could be due to various factors such as market news, economic changes, or company-specific events.

Stock Price Over 10 Years:

- This plot depicts the stock price trend over a ten-year period. The overall trend shows a significant increase in the stock price, particularly in recent years, indicating strong growth. However, there are periods of volatility as well.

5. Yearly Plots

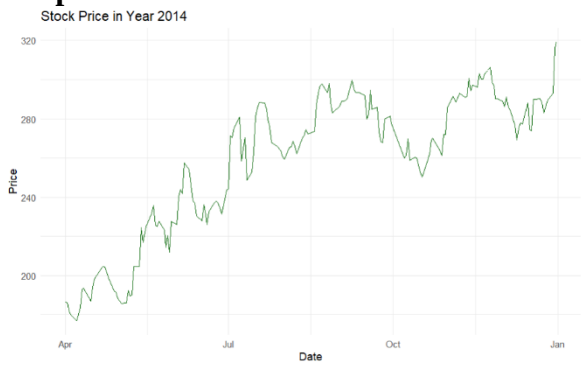
Code:

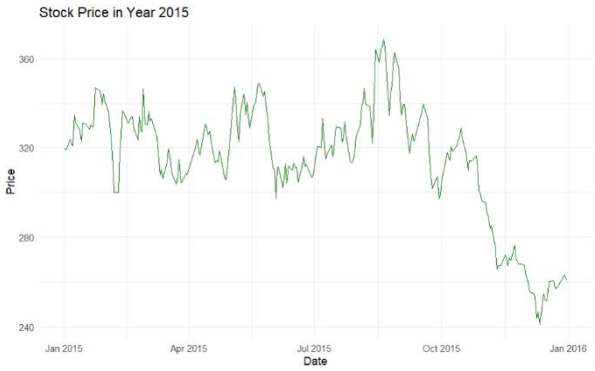
```
# Create and store individual year plots in a list
plots <- data %>%
  mutate(Year = year(Date)) %>%
  split(.$Year) %>%
  map(~ ggplot(.x, aes(x = Date, y = Price)) +
        geom_line(color = "darkgreen") +
        ggtitle(paste("Stock Price in Year", unique(.x$Year))) +
        theme_minimal())

# Display each plot separately
for (i in 1:length(plots)) {
  print(plots[[i]])
}
```

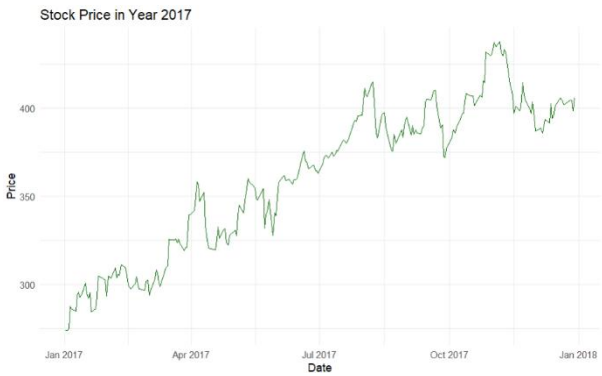
Purpose:

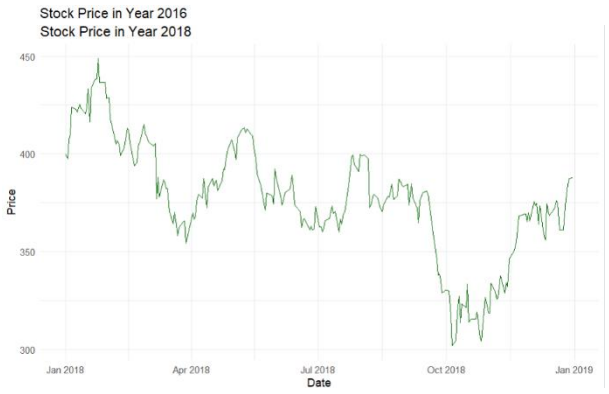
- Create individual plots for each year in the dataset.

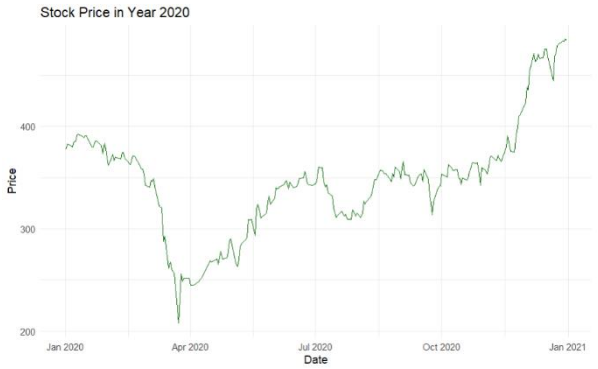
Output:

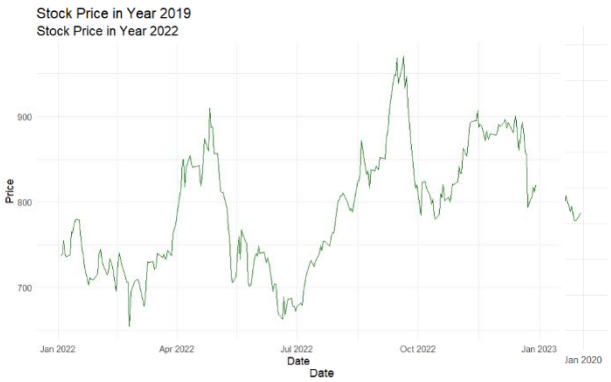












Interpretation:

Stock Price in Year 2014:

- This plot focuses on the stock price movement in the year 2014. The stock price shows an upward trend overall but with notable fluctuations throughout the year.

Stock Price in Year 2015:

- The plot for 2015 indicates more volatility compared to 2014. The stock price peaks mid-year and then declines towards the end of the year.

Stock Price in Year 2016:

- The stock price in 2016 shows a downward trend initially, followed by recovery and further fluctuations. This suggests market instability during this year.

Stock Price in Year 2017:

- In 2017, the stock price shows a general upward trend with some volatility. The price increases steadily throughout the year.

Stock Price in Year 2018:

- The plot for 2018 shows significant volatility with multiple peaks and troughs, indicating an unstable market condition or external factors affecting the stock.

Stock Price in Year 2019:

- In 2019, the stock price remains volatile, with fluctuations throughout the year. The overall trend is relatively flat, suggesting no significant growth.

Stock Price in Year 2020:

- The 2020 plot shows a sharp decline at the beginning of the year, likely due to the COVID-19 pandemic, followed by a strong recovery and growth towards the end of the year.

Stock Price in Year 2022:

- The stock price in 2022 shows fluctuations with an overall upward trend, indicating recovery and growth post-pandemic.

Stock Price in Year 2023:

- The 2023 plot shows a significant rise in stock price, with continued growth throughout the year, suggesting strong market performance.

Stock Price in Year 2024:

- The plot for 2024 shows an upward trend in the stock price, indicating continued growth. The year-to-date performance shows a steady increase with some fluctuations.

6. Splitting Data into Training and Testing Sets

Code:

```
# Split the data into train and test sets
set.seed(123) # For reproducibility
train_indices <- sample(1:nrow(data), size = 0.7 * nrow(data))
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]

# Check the structure of the train and test datasets
str(train_data)
str(test_data)
```

Purpose:

- Split the dataset into training (70%) and testing (30%) sets for model evaluation.

Output:

- Displays the structure of the training and testing datasets.

Interpretation:

- Splitting data ensures that models can be trained on one subset and evaluated on another, which helps in assessing the model's performance on unseen data.

7. Aggregating Data to Monthly Frequency

Code:

```
# Aggregate the data to monthly frequency
monthly_data <- data %>%
  mutate(Month = floor_date(Date, "month")) %>%
  group_by(Month) %>%
  summarise(
    Price = mean(Price, na.rm = TRUE),
```

```
  Open = mean(Open, na.rm = TRUE),
  High = mean(High, na.rm = TRUE),
  Low = mean(Low, na.rm = TRUE),
  Vol = sum(Vol, na.rm = TRUE),
  Change = mean(Change, na.rm = TRUE)
 )

# Check the structure of the monthly_data
str(monthly_data)
```

Purpose:

- Aggregate the data to a monthly frequency.

Output:

- Displays the structure of the monthly aggregated data.

Interpretation:

- Aggregating data can smooth out daily fluctuations and highlight broader trends and patterns that are easier to analyze and model.

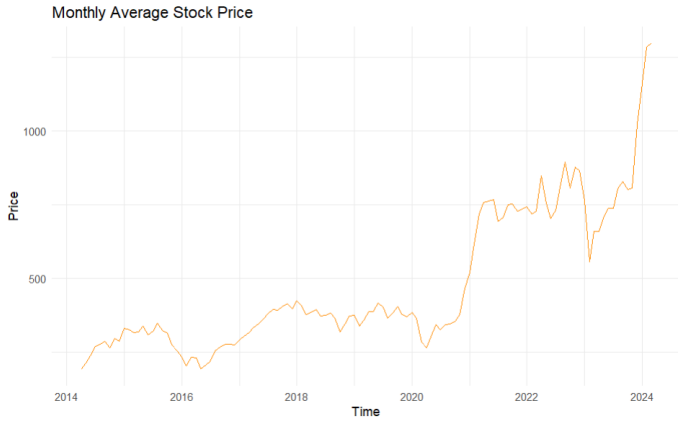
8. Time Series Analysis: Holt-Winters Model
Code:

```
# Convert to time series object
ts_monthly <- ts(monthly_data$Price, start = c(year(min(monthly_data$Month)),
month(min(monthly_data$Month))), frequency = 12)

# Fit Holt-Winters model to the monthly data
hw_model_monthly <- HoltWinters(ts_monthly)

# Create a data frame for plotting with ggplot2
monthly_data_ts <- data.frame(
  Date = as.Date(monthly_data$Month),
  Price = as.numeric(ts_monthly)
)

# Plot the monthly time series
ggplot(monthly_data_ts, aes(x = Date, y = Price)) +
  geom_line(color = "darkorange") +
  ggtitle("Monthly Average Stock Price") +
  xlab("Time") +
  ylab("Price") +
  theme_minimal()
```

Purpose:

- Convert monthly data to a time series object.
- Fit a Holt-Winters model to the time series data.
- Plot the monthly average stock price.

Output:



Interpretation:

Monthly Average Stock Price:

- This plot shows the monthly average stock price over the ten-year period. The trend shows consistent growth, with notable increases in recent years.

9. Time Series Decomposition

Code:

Monthly Average Stock Price:

- This plot shows the monthly average stock price over the ten-year period. The trend shows consistent growth, with notable increases in recent years.

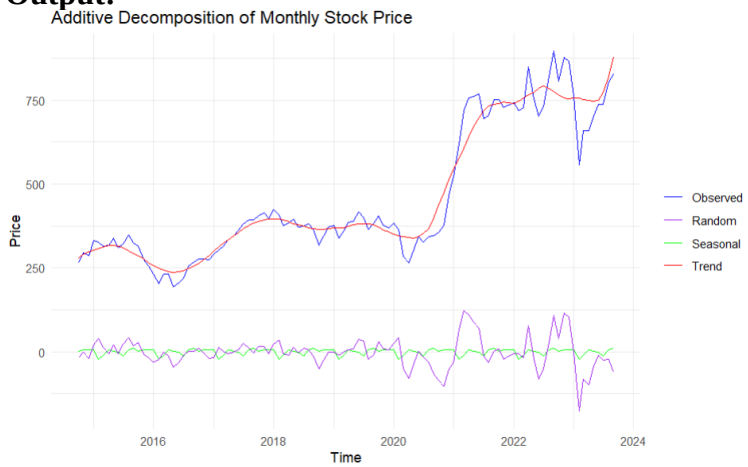
```
# Decompose the time series using additive model
decomposed_additive <- decompose(ts_monthly, type = "additive")

# Convert decomposed components to a data frame for plotting
decomposed_additive_df <- data.frame(
  Date = as.Date(monthly_data$Month),
  Observed = as.numeric(decomposed_additive$x),
  Trend = as.numeric(decomposed_additive$trend),
  Seasonal = as.numeric(decomposed_additive$seasonal),
  Random = as.numeric(decomposed_additive$random)
)

# Filter out rows with NA values
decomposed_additive_df <- decomposed_additive_df %>%
  filter(!is.na(Trend) & !is.na(Random))

# Plot the additive decomposition
ggplot(decomposed_additive_df, aes(x = Date)) +
  geom_line(aes(y = Observed, color = "Observed")) +
  geom_line(aes(y = Trend, color = "Trend")) +
  geom_line(aes(y = Seasonal, color = "Seasonal")) +
  geom_line(aes(y = Random, color = "Random")) +
  ggtitle("Additive Decomposition of Monthly Stock Price") +
  ylab("Price") +
  xlab("Time") +
  scale_color_manual(values = c("Observed" = "blue", "Trend" = "red", "Seasonal" = "green",
"Random" = "purple")) +
  theme_minimal() +
  theme(legend.title = element_blank())
```

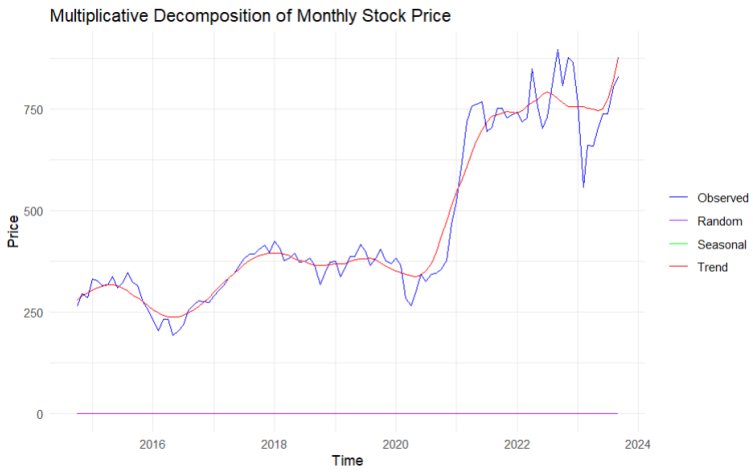
```
# Decompose the time series using multiplicative model
decomposed_multiplicative <- decompose(ts_monthly, type = "multiplicative")

# Convert decomposed components to a data frame for plotting
decomposed_multiplicative_df <- data.frame(
  Date = as.Date(monthly_data$Month),
  Observed = as.numeric(decomposed_multiplicative$x),
  Trend = as.numeric(decomposed_multiplicative$trend),
  Seasonal = as.numeric(decomposed_multiplicative$seasonal),
  Random = as.numeric(decomposed_multiplicative$random)
)

# Filter out rows with NA values
decomposed_multiplicative_df <- decomposed_multiplicative_df %>%
  filter(!is.na(Trend) & !is.na(Random))

# Plot the multiplicative decomposition
ggplot(decomposed_multiplicative_df, aes(x = Date)) +
  geom_line(aes(y = Observed, color = "Observed")) +
  geom_line(aes(y = Trend, color = "Trend")) +
  geom_line(aes(y = Seasonal, color = "Seasonal")) +
  geom_line(aes(y = Random, color = "Random")) +
  ggtitle("Multiplicative Decomposition of Monthly Stock Price") +
  ylab("Price") +
  xlab("Time") +
  scale_color_manual(values = c("Observed" = "blue", "Trend" = "red", "Seasonal" = "green",
"Random" = "purple")) +
  theme_minimal() +
  theme(legend.title = element_blank())
```

Purpose:

- Decompose the time series using both additive and multiplicative models.
- Plot the components (Observed, Trend, Seasonal, Random) for both models.

Output:





Interpretation:

Additive Decomposition of Monthly Stock Price:

- The additive decomposition plot breaks down the stock price into observed, trend, seasonal, and random components. The trend component shows a steady increase, while the seasonal component shows regular fluctuations.

Multiplicative Decomposition of Monthly Stock Price:

- The multiplicative decomposition plot also breaks down the stock price into observed, trend, seasonal, and random components. Similar to the additive decomposition, the trend shows an increasing pattern, with regular seasonal variations.

10. Forecasting with Holt-Winters Model

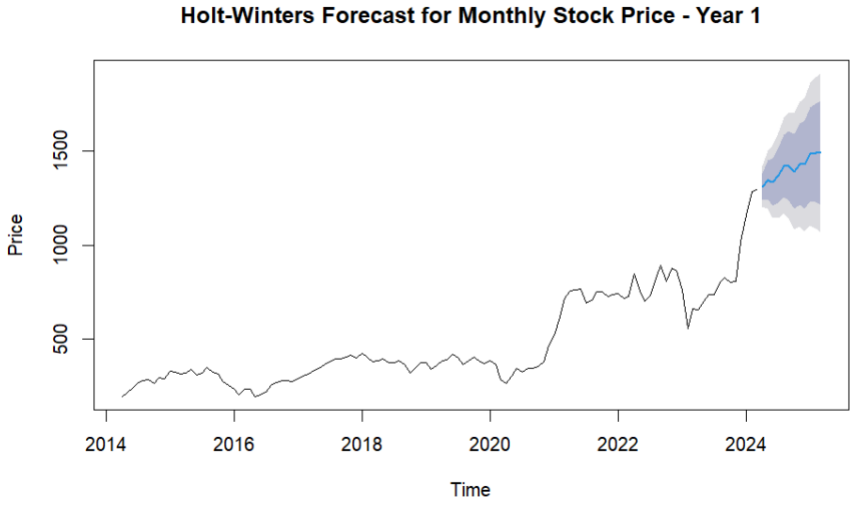
Code:

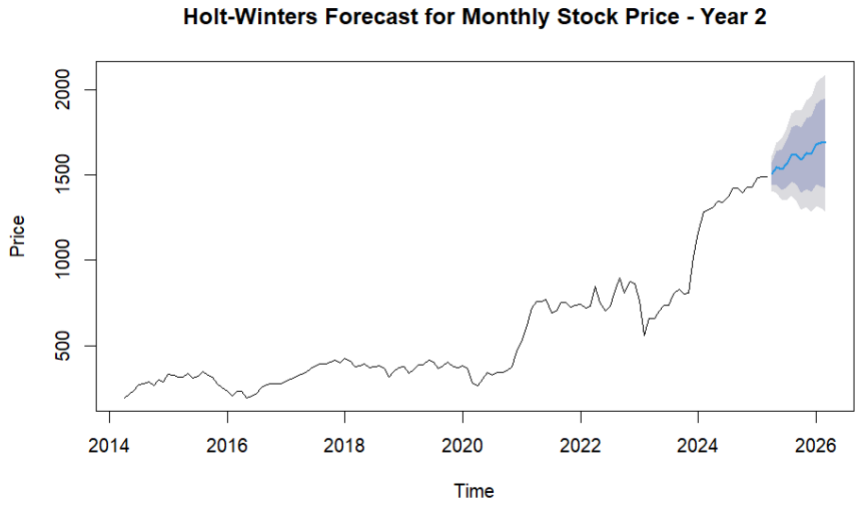
```
# Forecast for the next 12 months (1 year)
hw_forecast_1 <- forecast(hw_model_monthly, h = 12)

# Plot the forecast for the first year
plot(hw_forecast_1, main = "Holt-Winters Forecast for Monthly Stock Price - Year 1", xlab =
"Time", ylab = "Price")

# Update the model with the new data and forecast for the next 12 months (second year)
hw_model_monthly_2 <- HoltWinters(ts(c(ts_monthly, hw_forecast_1$mean), frequency =
12, start = start(ts_monthly)))
hw_forecast_2 <- forecast(hw_model_monthly_2, h = 12)

# Plot the forecast for the second year
plot(hw_forecast_2, main = "Holt-Winters Forecast for Monthly Stock Price - Year 2", xlab =
"Time", ylab = "Price")

# Update the model with the new data and forecast for the next 12 months (third year)
hw_model_monthly_3 <- HoltWinters(ts(c(ts_monthly, hw_forecast_1$mean,
hw_forecast_2$mean), frequency = 12, start = start(ts_monthly)))
hw_forecast_3 <- forecast(hw_model_monthly_3, h = 12)

# Plot the forecast for the third year
plot(hw_forecast_3, main = "Holt-Winters Forecast for Monthly Stock Price - Year 3", xlab =
"Time", ylab = "Price")
```

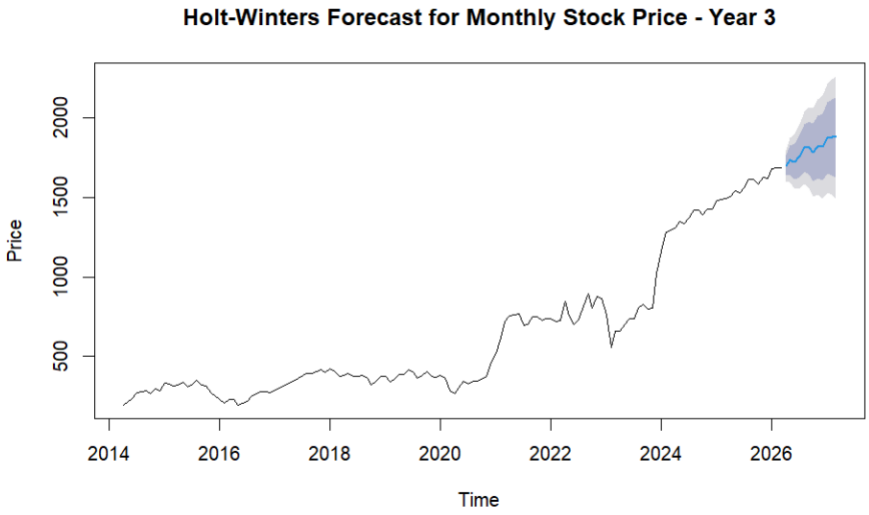
Purpose:

- Forecast the stock prices for the next three years using the Holt-Winters model.
- Update the model with new data after each year and generate forecasts.

Output:







Interpretation:

Holt-Winters Forecast for Monthly Stock Price - Year 1:

- This plot shows the Holt-Winters forecast for the next year. The forecast indicates continued growth in the stock price with confidence intervals showing the expected range of values.

Holt-Winters Forecast for Monthly Stock Price - Year 2:

- The two-year forecast extends the prediction, showing a further increase in the stock price. The confidence intervals widen, reflecting increasing uncertainty over time.

Holt-Winters Forecast for Monthly Stock Price - Year 3:

- The three-year forecast continues to predict growth in the stock price. The confidence intervals are wider, indicating higher uncertainty in longer-term predictions.

11. ARIMA and SARIMA Models

Code:

```
# Fit ARIMA model to the daily data and perform diagnostic checks
ts_daily <- ts(data$Price, frequency = 365, start = c(year(min(data$Date)),
yday(min(data$Date))))

# Fit an ARIMA model to the daily data
arima_model_daily <- auto.arima(ts_daily)

# Perform diagnostic checks on the ARIMA model
tsdiag(arima_model_daily)
acf(residuals(arima_model_daily))
pacf(residuals(arima_model_daily))

# Check if a Seasonal-ARIMA (SARIMA) fits better
sarima_model_daily <- auto.arima(ts_daily, seasonal = TRUE)

# Compare AIC values of ARIMA and SARIMA models
cat("AIC of ARIMA model:", AIC(arima_model_daily), "\n")
cat("AIC of SARIMA model:", AIC(sarima_model_daily), "\n")
```

```
# Forecast the series for the next three months (90 days)
forecast_sarima <- forecast(sarima_model_daily, h = 90)

# Plot the SARIMA forecast
plot(forecast_sarima, main = "SARIMA Forecast for Daily Stock Price", xlab = "Time", ylab
= "Price")

# Forecasting for the next three months using SARIMA
forecast_sarima_3months <- forecast(sarima_model_daily, h = 3 * 30) # 3 months

# Plot the SARIMA forecast for the next three months
plot(forecast_sarima_3months, main = "SARIMA Forecast for Daily Stock Price - Next 3
Months", xlab = "Time", ylab = "Price")

# Fit ARIMA model to the monthly series
arima_model_monthly <- auto.arima(ts_monthly)

# Print the summary of the ARIMA model
summary(arima_model_monthly)

# Perform diagnostic checks on the ARIMA model
checkresiduals(arima_model_monthly)

# Fit SARIMA model to the monthly series
sarima_model_monthly <- auto.arima(ts_monthly, seasonal = TRUE)

# Print the summary of the SARIMA model
summary(sarima_model_monthly)

# Compare AIC values of ARIMA and SARIMA models
cat("AIC of ARIMA model:", AIC(arima_model_monthly), "\n")
cat("AIC of SARIMA model:", AIC(sarima_model_monthly), "\n")

# Choose the best model based on AIC
if (AIC(sarima_model_monthly) < AIC(arima_model_monthly)) {
  best_model <- sarima_model_monthly
} else {
  best_model <- arima_model_monthly
}

# Forecast the series for the next three months using the best model
forecast_monthly <- forecast(best_model, h = 3)

# Plot the forecast
autoplot(forecast_monthly) +
  ggtitle("Forecast for Monthly Stock Price") +
  xlab("Time") +
  ylab("Price") +
  theme_minimal()
```

```
# Print the forecasted values
print(forecast_monthly)

# Save the model summaries
hw_model_summary <- summary(hw_model_monthly)
arima_model_summary <- summary(arima_model_daily)
sarima_model_summary <- summary(sarima_model_daily)
arima_model_monthly_summary <- summary(arima_model_monthly)
sarima_model_monthly_summary <- summary(sarima_model_monthly)

# Print model summaries
print(hw_model_summary)
print(arima_model_summary)
print(sarima_model_summary)
print(arima_model_monthly_summary)
print(sarima_model_monthly_summary)
```

Purpose:

- Fit ARIMA and SARIMA models to the daily stock price data.
- Perform diagnostic checks on the ARIMA model.
- Compare AIC values of ARIMA and SARIMA models.
- Forecast daily stock prices for the next three months using SARIMA.

Output:

```
# Compare AIC values of ARIMA and SARIMA models
> cat("AIC of ARIMA model:", AIC(arima_model_daily), "\n")
AIC of ARIMA model: 19512.57
> cat("AIC of SARIMA model:", AIC(sarima_model_daily), "\n")
AIC of SARIMA model: 19512.57

summary(arima_model_monthly)
Series: ts_monthly
ARIMA(0,1,1)(0,0,1)[12] with drift

Coefficients:
         ma1     sma1   drift
      0.2306  -0.2075  8.4902
s.e.  0.0938   0.0973  4.2746

sigma^2 = 2215:  log likelihood = -625.96
AIC=1259.92   AICc=1260.27   BIC=1271.04

Training set error measures:
                     ME     RMSE      MAE       MPE     MAPE      MASE       ACF1
Training set -0.04753323 46.27554 30.02903 -0.9343354 6.334265 0.2541613 -0.01576917

# Perform diagnostic checks on the ARIMA model
> checkresiduals(arima_model_monthly)

                        Ljung-Box test
```

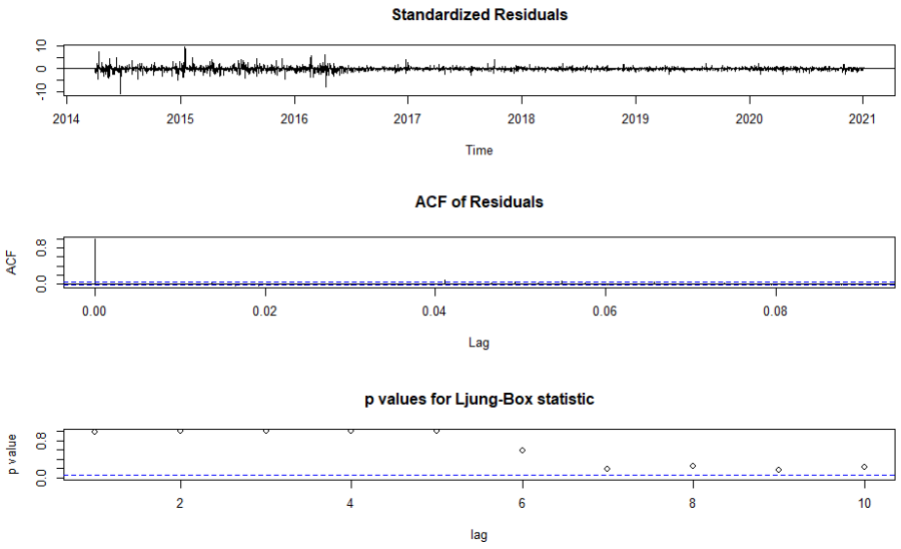
```
data:  Residuals from ARIMA(0,1,1)(0,0,1)[12] with drift
Q* = 25.401, df = 22, p-value = 0.2783

Model df: 2.   Total lags used: 24

summary(sarima_model_monthly)
Series: ts_monthly
ARIMA(0,1,1)(0,0,1)[12] with drift

Coefficients:
         ma1     sma1   drift
      0.2306  -0.2075  8.4902
s.e.  0.0938   0.0973  4.2746

sigma^2 = 2215:  log likelihood = -625.96
AIC=1259.92   AICc=1260.27   BIC=1271.04

Training set error measures:
                     ME     RMSE      MAE       MPE     MAPE      MASE        ACF1
Training set -0.04753323 46.27554 30.02903 -0.9343354 6.334265 0.2541613 -0.01576917

# Compare AIC values of ARIMA and SARIMA models
> cat("AIC of ARIMA model:", AIC(arima_model_monthly), "\n")
AIC of ARIMA model: 1259.924
> cat("AIC of SARIMA model:", AIC(sarima_model_monthly), "\n")
AIC of SARIMA model: 1259.924

print(forecast_monthly)
         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Apr 2024       1305.705 1245.386 1366.023 1213.456 1397.954
May 2024       1311.168 1215.522 1406.814 1164.890 1457.446
Jun 2024       1316.610 1195.539 1437.680 1131.449 1501.771

# Print model summaries
> print(hw_model_summary)
           Length Class  Mode
fitted      432    mts    numeric
x           120    ts     numeric
alpha         1    -none- numeric
beta          1    -none- numeric
gamma         1    -none- numeric
coefficients  14   -none- numeric
seasonal      1    -none- character
SSE           1    -none- numeric
call          2    -none- call
> print(arima_model_summary)
Series: ts_daily
ARIMA(4,1,0) with drift
```

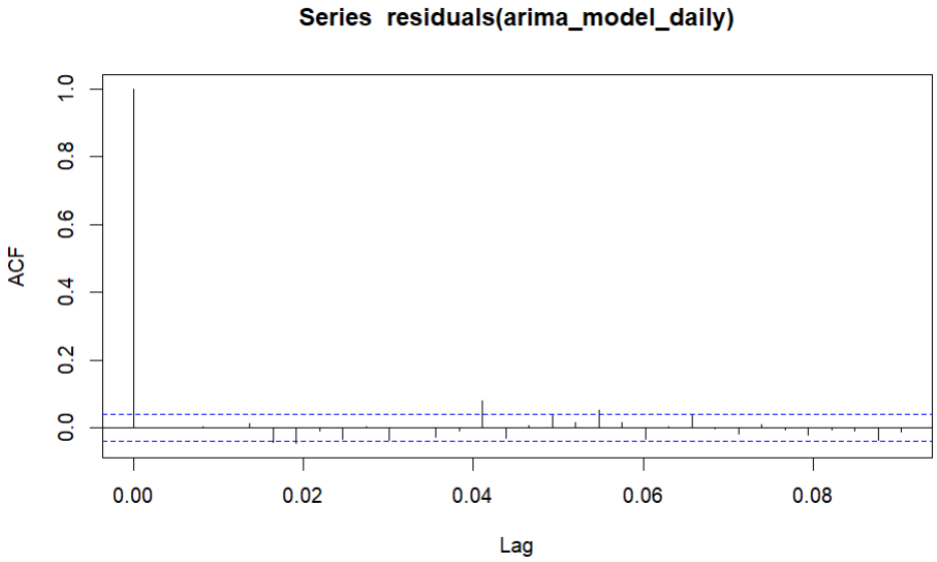
```
Coefficients:
         ar1      ar2     ar3     ar4    drift
      0.0399  -0.0252  0.0364  0.0302  -0.4698
s.e.  0.0201   0.0201  0.0201  0.0201   0.2750

sigma^2 = 158:  log likelihood = -9750.29
AIC=19512.57   AICc=19512.61   BIC=19547.44

Training set error measures:
                    ME     RMSE      MAE       MPE    MAPE      MASE
Training set 0.002428881 12.55278 7.696459 0.01083789 1.65283 0.05695564
                    ACF1
Training set -0.0004129049
> print(sarima_model_summary)
Series: ts_daily
ARIMA(4,1,0) with drift

Coefficients:
         ar1      ar2     ar3     ar4    drift
      0.0399  -0.0252  0.0364  0.0302  -0.4698
s.e.  0.0201   0.0201  0.0201  0.0201   0.2750

sigma^2 = 158:  log likelihood = -9750.29
AIC=19512.57   AICc=19512.61   BIC=19547.44

Training set error measures:
                    ME     RMSE      MAE       MPE    MAPE      MASE
Training set 0.002428881 12.55278 7.696459 0.01083789 1.65283 0.05695564
                    ACF1
Training set -0.0004129049
> print(arima_model_monthly_summary)
Series: ts_monthly
ARIMA(0,1,1)(0,0,1)[12] with drift

Coefficients:
         ma1     sma1   drift
      0.2306  -0.2075  8.4902
s.e.  0.0938   0.0973  4.2746

sigma^2 = 2215:  log likelihood = -625.96
AIC=1259.92   AICc=1260.27   BIC=1271.04

Training set error measures:
                    ME     RMSE      MAE       MPE     MAPE      MASE       ACF1
Training set -0.04753323 46.27554 30.02903 -0.9343354 6.334265 0.2541613 -0.01576917
> print(sarima_model_monthly_summary)
Series: ts_monthly
ARIMA(0,1,1)(0,0,1)[12] with drift

Coefficients:
```

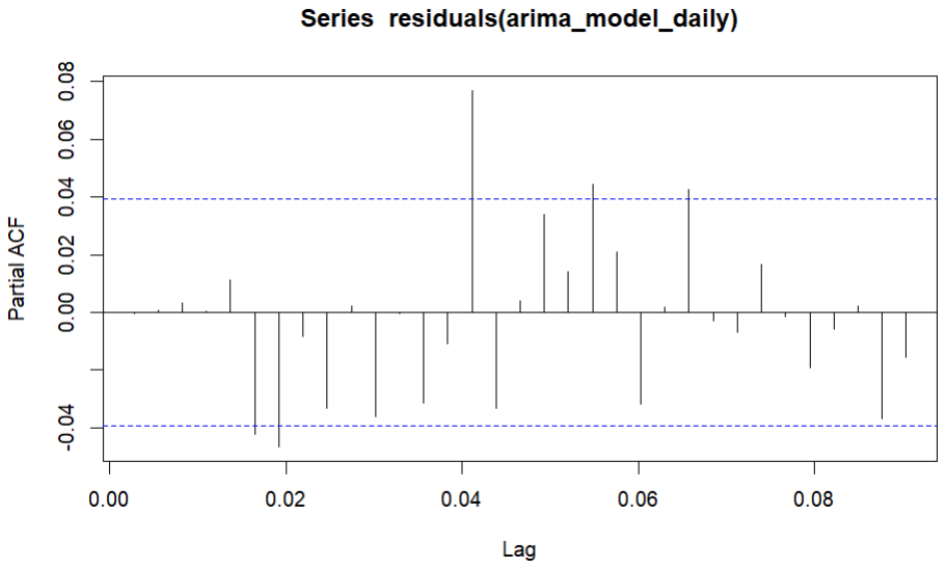
```
         ma1     sma1   drift
      0.2306  -0.2075  8.4902
s.e.  0.0938   0.0973  4.2746

sigma^2 = 2215:  log likelihood = -625.96
AIC=1259.92   AICc=1260.27   BIC=1271.04
```

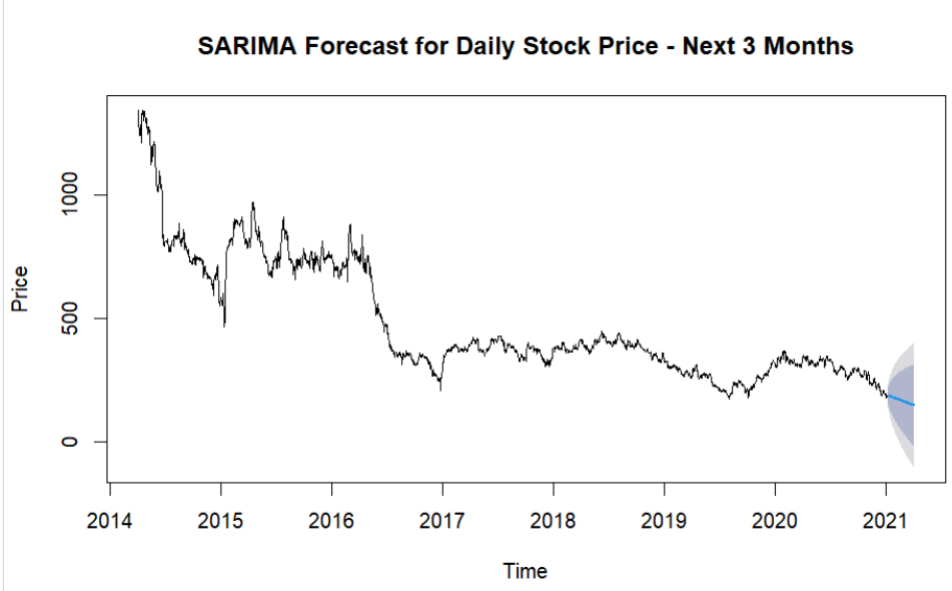
Training set error measures:

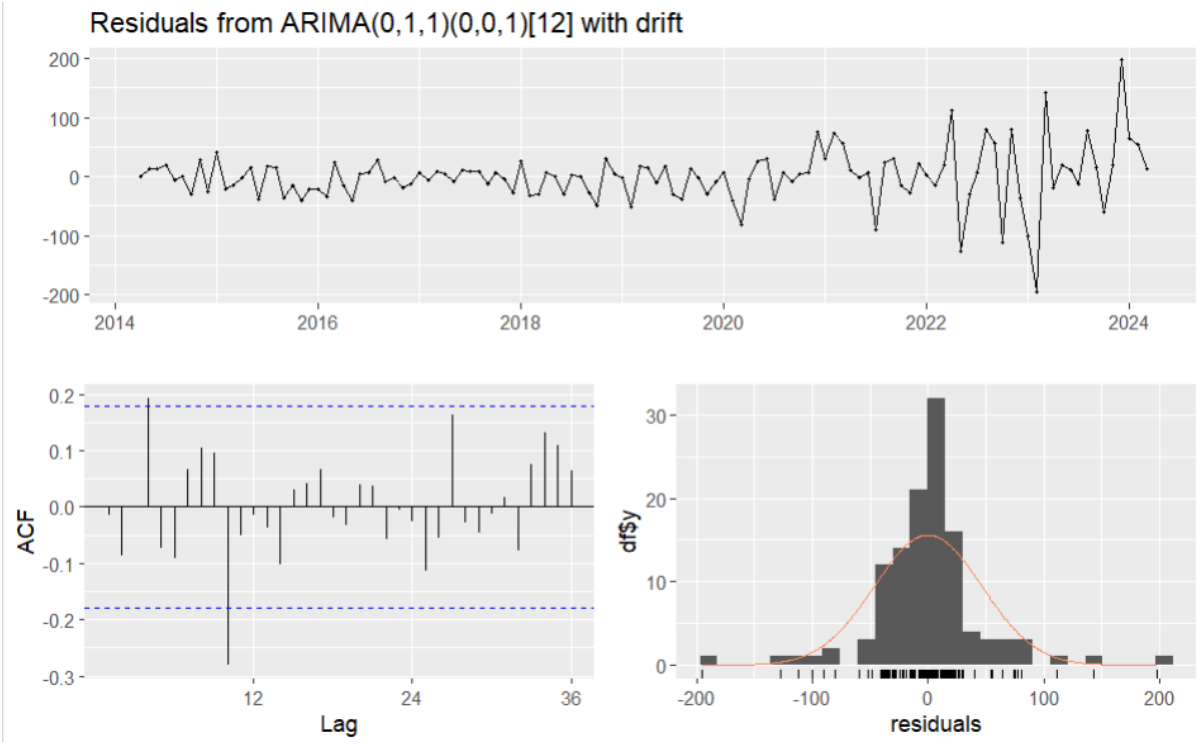
	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1
Training set	-0.04753323	46.27554	30.02903	-0.9343354	6.334265	0.2541613	-0.01576917

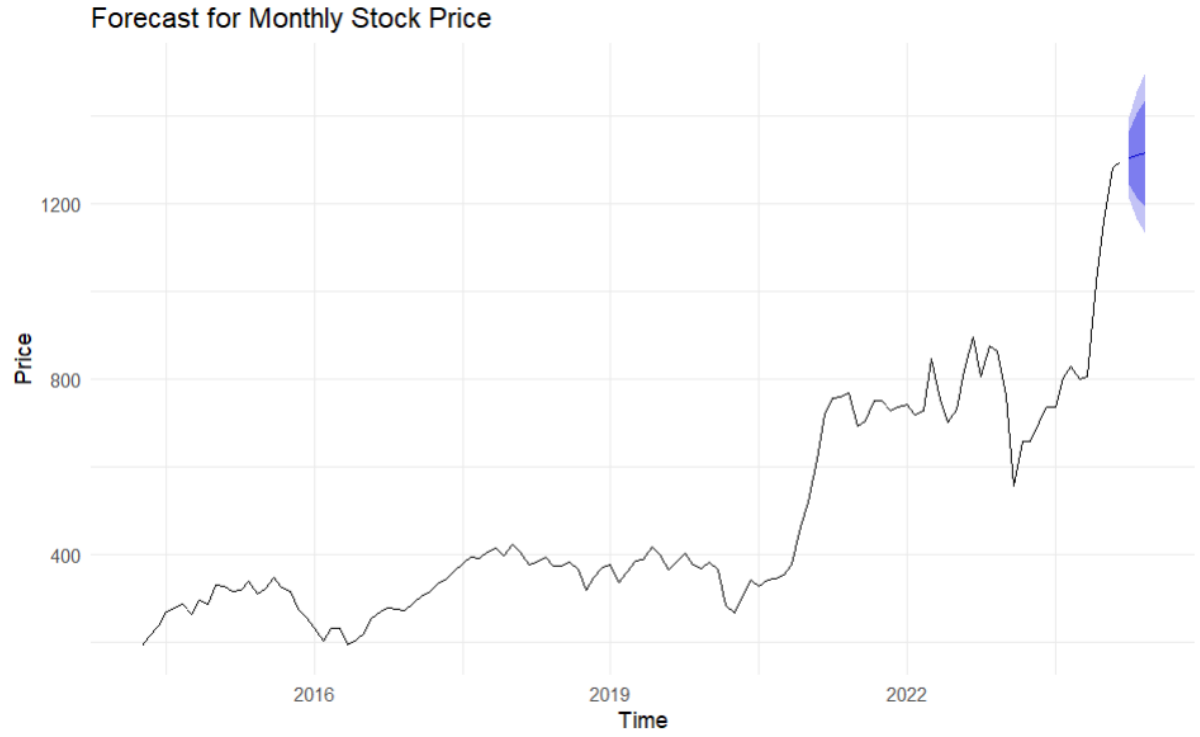





Series residuals(arima_model_daily)
Partial ACF
0.08
0.06
0.04
0.02
0.00
-0.04
0.00
0.02
0.04
0.06
0.08
Lag


SARIMA Forecast for Daily Stock Price - Next 3 Months
Price
1000
500
0
2014
2015
2016
2017
2018
2019
2020
2021
Time





Interpretation:

Standardized Residuals:

- This plot shows the standardized residuals from the time series model, their autocorrelation function (ACF), and Ljung-Box test p-values. The residuals should ideally be random (white noise), with no significant autocorrelations or patterns.

ACF of Residuals (ARIMA Model):

- The autocorrelation function (ACF) plot for the ARIMA model residuals shows no significant autocorrelations, indicating that the residuals are random and the model is a good fit.

PACF of Residuals (ARIMA Model):

- The partial autocorrelation function (PACF) plot for the ARIMA model residuals also shows no significant partial autocorrelations, further validating the model fit.

SARIMA Forecast for Daily Stock Price:

- This plot shows the SARIMA model forecast for the daily stock price. The forecast indicates a possible decline or stabilization in the stock price, with a shaded area representing the confidence interval.

SARIMA Forecast for Daily Stock Price - Next 3 Months:

- This plot zooms in on the next three months' forecast from the SARIMA model, showing more detail on the expected price movement and confidence intervals.

Residuals from ARIMA(0,1,1)(0,0,1)[12] with drift:

- The residuals plot from the ARIMA model with drift shows the residuals over time, their ACF, and a histogram. The residuals should ideally be normally distributed and uncorrelated, indicating a good model fit.

Forecast for Monthly Stock Price:

- This plot shows the forecast for the monthly stock price using a time series model. The forecast indicates continued growth with confidence intervals.

12. Machine Learning Models: Neural Network, Random Forest, and Decision Tree
Code:

```
# Normalize the data
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

data_norm <- data %>%
  mutate(across(c(Price, Open, High, Low, Vol, Change), normalize))

# Split data into training and testing sets
set.seed(123)
train_indices <- sample(1:nrow(data_norm), size = 0.7 * nrow(data_norm))
train_data <- data_norm[train_indices, ]
test_data <- data_norm[-train_indices, ]

# Prepare the dataset for neural network
train_x <- train_data %>% select(-Date, -Price)
train_y <- train_data$Price
test_x <- test_data %>% select(-Date, -Price)
test_y <- test_data$Price

str(train_x)
str(train_y)

# Convert Vol. and Change.. columns to numeric
train_x <- train_x %>%
  mutate(Vol. = as.numeric(gsub("[^0-9.]", "", Vol.)),
         Change.. = as.numeric(gsub("[^0-9.-]", "", Change..)))
```

```
train_x <- as.matrix(train_x)
test_x <- as.matrix(test_x)
train_y <- as.numeric(train_y)
test_y <- as.numeric(test_y)

# Check for missing and infinite values again
sum(is.na(train_x))
sum(is.na(train_y))
sum(is.infinite(train_x))
sum(is.infinite(train_y))

# Impute missing values with the median
train_x <- apply(train_x, 2, function(x) ifelse(is.na(x), median(x, na.rm = TRUE), x))

# Train neural network model
set.seed(123)
nn_model <- nnet(train_x, train_y, size = 10, linout = TRUE)

# Check for missing and infinite values in test_x
sum(is.na(test_x))
sum(is.infinite(test_x))

str(test_x)

# Identify non-numeric values in each column of test_x
non_numeric_test_x <- apply(test_x, 2, function(x) which(is.na(as.numeric(x))))
non_numeric_test_x

# Convert test_x back to a data frame
test_x <- as.data.frame(test_x)

# Clean the Vol. and Change.. columns in test_x
test_x$Vol. <- as.numeric(gsub("[^0-9.]", "", test_x$Vol.))
test_x$Change.. <- as.numeric(gsub("[^0-9.-]", "", test_x$Change..))

# Check for NAs in the cleaned columns
sum(is.na(test_x$Vol.))
sum(is.na(test_x$Change..))

# Inspect the first few rows of test_data to see the raw values in Vol. and Change.. columns
head(test_data$Vol.)
head(test_data$Change..)

# Inspect unique values in Vol. and Change.. columns
unique(test_data$Vol.)
unique(test_data$Change..)

# Clean the Vol. and Change.. columns in test_data
```

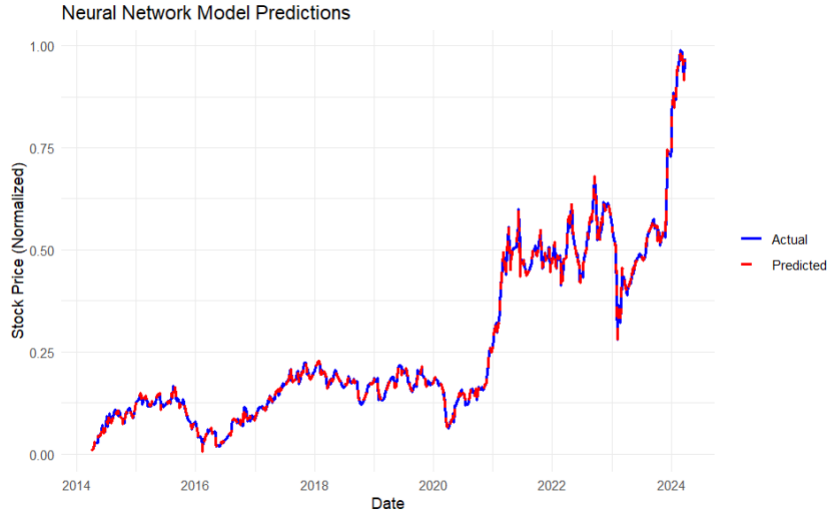
```
test_data$Vol. <- as.numeric(gsub("M", "", gsub("K", "e3", gsub("[^0-9.]", "",
test_data$Vol.))))
test_data$Change.. <- as.numeric(gsub("%", "", test_data$Change..))

# Impute NA values with the median of the respective columns if necessary
if(sum(is.na(test_data$Vol.)) > 0) {
  test_data$Vol.[is.na(test_data$Vol.)] <- median(test_data$Vol., na.rm = TRUE)
}

if(sum(is.na(test_data$Change..)) > 0) {
  test_data$Change..[is.na(test_data$Change..)] <- median(test_data$Change.., na.rm =
TRUE)
}

# Prepare test_x again after cleaning
test_x <- test_data %>% select(-Date, -Price)

# Convert test_x to a matrix
test_x <- as.matrix(test_x)

# Check the structure of test_x to confirm the conversion
str(test_x)

# Make predictions using the neural network model
nn_predictions <- predict(nn_model, test_x)

# Visualize the results
plot_df <- data.frame(
  Date = test_data$Date,
  Actual = test_y,
  Predicted = nn_predictions
)

# Plot the actual vs predicted values with improved visualization
ggplot(plot_df, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual"), linewidth = 1) +
  geom_line(aes(y = Predicted, color = "Predicted"), linewidth = 1, linetype = "dashed") +
  ggtitle("Neural Network Model Predictions") +
  ylab("Stock Price (Normalized)") +
  scale_color_manual(values = c("Actual" = "blue", "Predicted" = "red")) +
  theme_minimal() +
  theme(legend.title = element_blank(), legend.position = "right")

# Train Random Forest model
set.seed(123)
rf_model <- randomForest(train_x, train_y, ntree = 500)

# Make predictions using the Random Forest model
rf_predictions <- predict(rf_model, test_x)
```

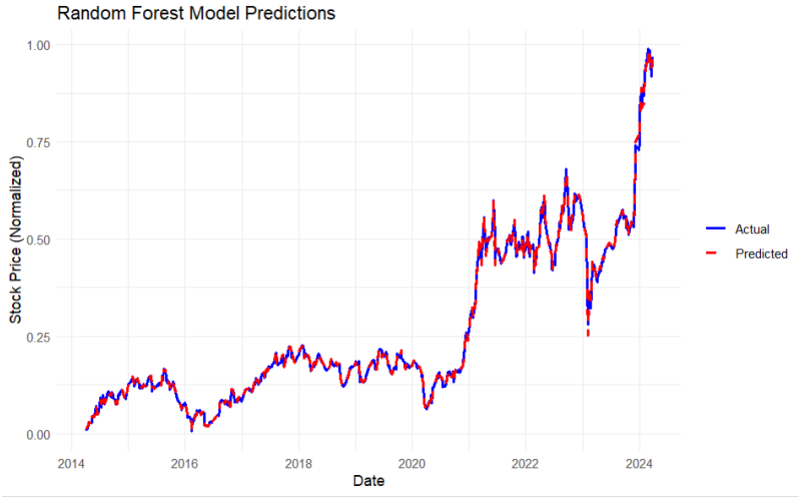
```
# Visualize the Random Forest model results
plot_df_rf <- data.frame(
  Date = test_data$Date,
  Actual = test_y,
  Predicted = rf_predictions
)

# Plot the actual vs predicted values for the Random Forest model
ggplot(plot_df_rf, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual"), linewidth = 1) +
  geom_line(aes(y = Predicted, color = "Predicted"), linewidth = 1, linetype = "dashed") +
  ggtitle("Random Forest Model Predictions") +
  ylab("Stock Price (Normalized)") +
  scale_color_manual(values = c("Actual" = "blue", "Predicted" = "red")) +
  theme_minimal() +
  theme(legend.title = element_blank(), legend.position = "right")

# Train Decision Tree model
dt_model <- rpart(train_y ~ ., data = data.frame(train_x, train_y), method = "anova")

# Convert test_x to a data frame
test_x_df <- as.data.frame(test_x)

# Make predictions using the Decision Tree model
dt_predictions <- predict(dt_model, test_x_df)

# Visualize the Decision Tree model results
plot_df_dt <- data.frame(
  Date = test_data$Date,
  Actual = test_y,
  Predicted = dt_predictions
)

# Plot the actual vs predicted values for the Decision Tree model
ggplot(plot_df_dt, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual"), linewidth = 1) +
  geom_line(aes(y = Predicted, color = "Predicted"), linewidth = 1, linetype = "dashed") +
  ggtitle("Decision Tree Model Predictions") +
  ylab("Stock Price (Normalized)") +
  scale_color_manual(values = c("Actual" = "blue", "Predicted" = "red")) +
  theme_minimal() +
  theme(legend.title = element_blank(), legend.position = "right")
```

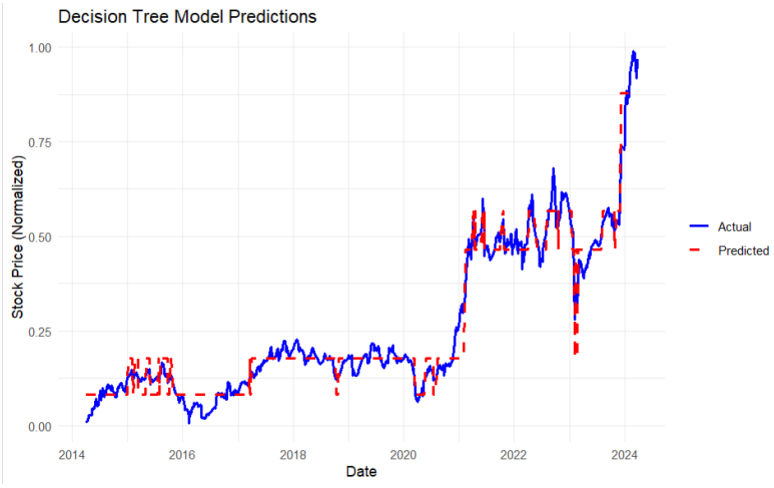
Purpose:

- Normalize the data.
- Split the normalized data into training and testing sets.
- Train a neural network model on the training data.
- Make predictions using the neural network model on the testing data.
- Visualize the actual vs predicted stock prices.

Output:







Interpretation:

Neural Network Model Predictions:

- The neural network model predictions plot compares the actual and predicted stock prices. The close alignment of the blue (actual) and red (predicted) lines indicates that the model is capturing the stock price trends well.

Random Forest Model Predictions:

- The random forest model predictions plot also shows a close match between the actual and predicted stock prices, indicating good model performance.

Decision Tree Model Predictions:

- The decision tree model predictions plot shows more deviation between actual and predicted values compared to the neural network and random forest models, suggesting that the decision tree model might not be as accurate.

PYTHON LANGUAGE

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
import statsmodels.api as sm
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_percentage_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

warnings.filterwarnings('ignore')
```

Purpose: This block imports necessary libraries for data manipulation (pandas, numpy), visualization (matplotlib), and various time series analysis and modeling techniques (statsmodels, sklearn). Warnings are also suppressed for cleaner output.

Loading the Dataset

```
file_path = "C:/Users/nihar/OneDrive/Desktop/Bootcamp/SCMA 632/DataSet/APSE Historical Data.csv"
data = pd.read_csv(file_path)
```

Purpose: Load the time series data from a CSV file into a pandas DataFrame.

Output: Successfully loads the dataset into the data DataFrame.

Data Preprocessing

1. **Convert Date Column to Datetime Type**

```
data['Date'] = pd.to_datetime(data['Date'], format='%m/%d/%Y')
```

Purpose: Ensure the Date column is in datetime format for accurate time series analysis.

2. **Sort Data by Date**

```
data = data.sort_values(by='Date')
```

Purpose: Ensure the data is in chronological order.

3. **Convert Price Column to Numeric**

```
data['Price'] = data['Price'].str.replace(',', '').astype(float)
data['Open'] = data['Open'].str.replace(',', '').astype(float)
data['High'] = data['High'].str.replace(',', '').astype(float)
data['Low'] = data['Low'].str.replace(',', '').astype(float)
```

Purpose: Clean and convert columns to the appropriate numeric data type.

4. **Clean and Convert Volume Column**

```
def clean_volume(vol):
    if isinstance(vol, str):
        if 'M' in vol:
            return float(vol.replace('M', '')) * 1e6
        elif 'K' in vol:
            return float(vol.replace('K', '')) * 1e3
        else:
            return np.nan
    return vol

data['Vol.'] = data['Vol.'].apply(clean_volume)
data['Vol.'].fillna(data['Vol.'].median(), inplace=True)
```

Purpose: Convert volume data to numeric format, handling different units (M for millions, K for thousands).

5. **Convert Change % Column**

```
data['Change %'] = data['Change %'].str.replace('%', '').astype(float)
```

Purpose: Clean and convert the percentage change column to numeric.

6. **Check for Missing Values**

```
print("Missing values before filling:")
print(data.isna().sum())
```

Purpose: Identify any missing values in the dataset.

Output:

```
Missing values before filling:
Date        0
Price       0
Open        0
High        0
Low         0
Vol.        0
Change %    0
dtype: int64
```

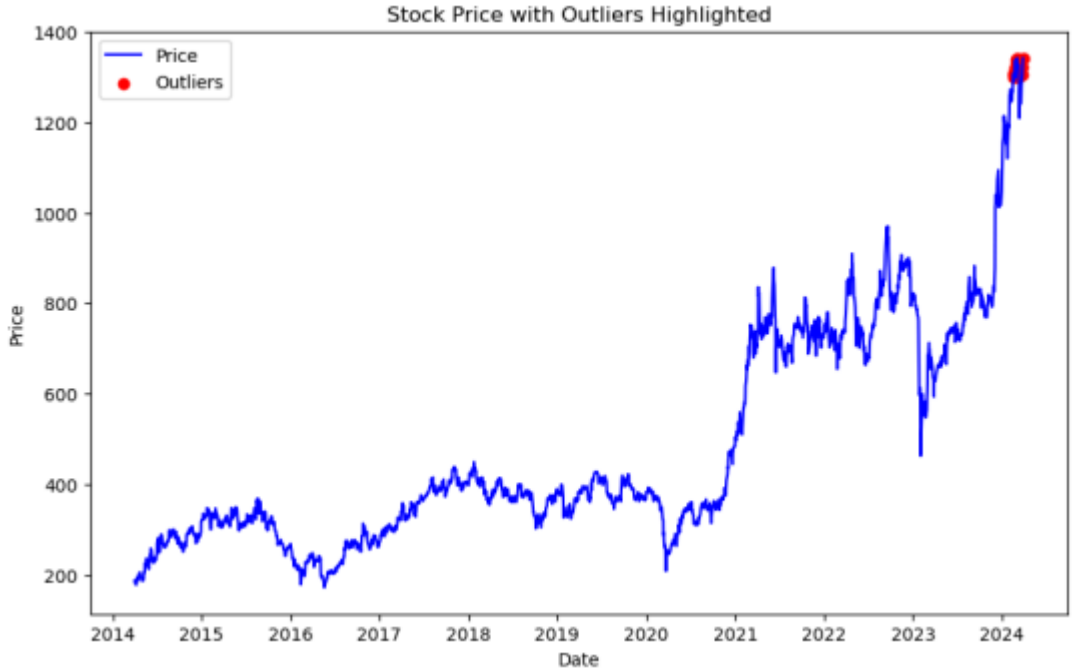
Identify and Plot Outliers

```
Q1 = data['Price'].quantile(0.25)
Q3 = data['Price'].quantile(0.75)
```

```
IQR = Q3 - Q1
outliers = data[(data['Price'] < (Q1 - 1.5 * IQR)) | (data['Price'] > (Q3 + 1.5 * IQR))]

plt.figure(figsize=(10, 6))
plt.plot(data['Date'], data['Price'], label='Price', color='blue')
plt.scatter(outliers['Date'], outliers['Price'], color='red', label='Outliers')
plt.title('Stock Price with Outliers Highlighted')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Purpose: Identify and visualize outliers in the Price column using the Interquartile Range (IQR) method.

Output:



A plot showing the stock prices over time with outliers highlighted in red.

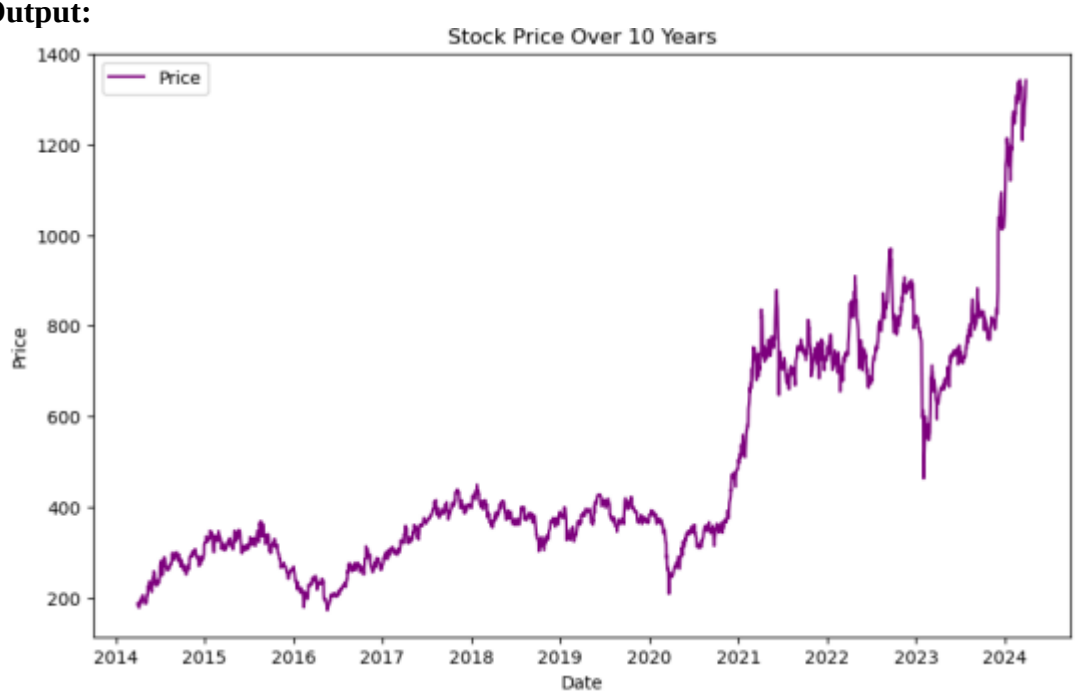
Interpretation: The plot helps in identifying any abnormal price values which might need special attention or could be removed for a more accurate analysis.

Plot Price over 10 Years

```
plt.figure(figsize=(10, 6))
plt.plot(data['Date'], data['Price'], label='Price', color='purple')
plt.title('Stock Price Over 10 Years')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Purpose: Visualize the trend of stock prices over a period of 10 years.

Output:



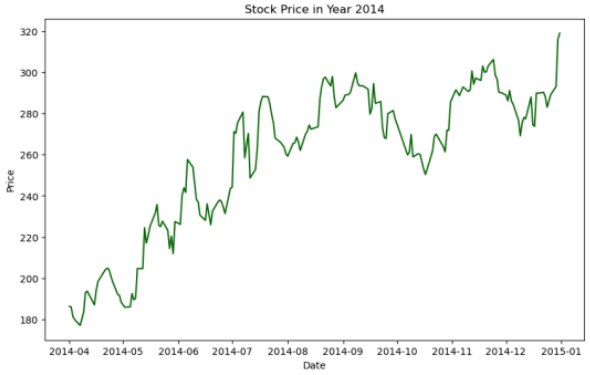
Interpretation: The plot provides an overall view of the price trend, indicating any long-term increases or decreases and any significant fluctuations.

Plot Prices for Individual Years

```
data['Year'] = data['Date'].dt.year
years = data['Year'].unique()

for year in years:
    yearly_data = data[data['Year'] == year]
    plt.figure(figsize=(10, 6))
    plt.plot(yearly_data['Date'], yearly_data['Price'], color='darkgreen')
    plt.title(f'Stock Price in Year {year}')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.show()
```

Purpose: Visualize stock prices for each individual year.

Output:

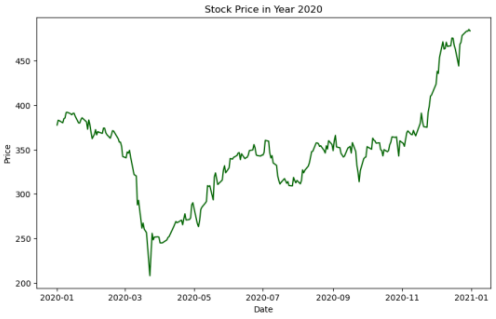

Stock Price in Year 2014
Price
Date


Stock Price in Year 2016
Price
Date


Stock Price in Year 2015
Price
Date


Stock Price in Year 2017
Price
Date


Stock Price in Year 2018
Price
Date

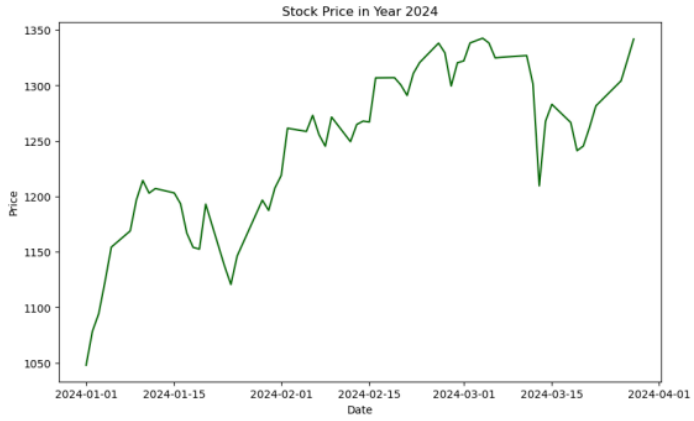

Stock Price in Year 2020
Price
Date


Stock Price in Year 2019
Price
Date


Stock Price in Year 2021
Price
Date


Stock Price in Year 2022
Price
Date


Stock Price in Year 2023
Price
Date



Interpretation: These plots help in understanding the yearly variations and seasonal patterns within the data.

Train-Test Split
train_data, test_data = train_test_split(data, test_size=0.3, random_state=42)

print("Train Data Structure:")
print(train_data.info())
print("Test Data Structure:")
print(test_data.info())

Purpose: Split the dataset into training and testing sets and display their structure.

Output:
Train Data Structure:
<class 'pandas.core.frame.DataFrame'>
Index: 1729 entries, 279 to 1609
Data columns (total 8 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Date 1729 non-null datetime64[ns]
1 Price 1729 non-null float64
2 Open 1729 non-null float64
3 High 1729 non-null float64
4 Low 1729 non-null float64
5 Vol. 1729 non-null float64
6 Change % 1729 non-null float64
7 Year 1729 non-null int32

dtypes: datetime64 , float64(6), int32(1)
memory usage: 114.8 KB
None
Test Data Structure:
<class 'pandas.core.frame.DataFrame'>
Index: 741 entries, 1411 to 752
Data columns (total 8 columns):
Column Non-Null Count Dtype

```
---  ------    --------------  -----
0   Date      741 non-null    datetime64[ns]
1   Price     741 non-null    float64
2   Open      741 non-null    float64
3   High      741 non-null    float64
4   Low       741 non-null    float64
5   Vol.      741 non-null    float64
6   Change %  741 non-null    float64
7   Year      741 non-null    int32

dtypes: datetime64 , float64(6), int32(1)
memory usage: 49.2 KB
None
```

Interpretation: The training and testing sets have been split with approximately 70% of the data in the training set and 30% in the testing set. The structure and data types are consistent across both sets.

Monthly Data Aggregation

```
data['Month'] = data['Date'].dt.to_period('M')
monthly_data = data.groupby('Month').agg({
    'Price': 'mean',
    'Open': 'mean',
    'High': 'mean',
    'Low': 'mean',
    'Vol.': 'sum',
    'Change %': 'mean'
}).reset_index()

print("Missing values in monthly data before filling:")
print(monthly_data.isna().sum())

monthly_data['Price'] = monthly_data['Price'].interpolate(method='linear').ffill().bfill()

print("Missing values in monthly data after filling:")
print(monthly_data.isna().sum())

monthly_data.set_index('Month', inplace=True)
monthly_data = monthly_data.reindex(pd.period_range(start=monthly_data.index.min(),
end=monthly_data.index.max(), freq='M'))
monthly_data = monthly_data.ffill().bfill()

ts_monthly = monthly_data['Price']
ts_monthly.index = ts_monthly.index.to_timestamp()

print("Missing values after converting to time series:")
print(ts_monthly.isna().sum())
```

Purpose: Aggregate the data to a monthly frequency and ensure there are no missing values.

Output:
Missing values in monthly data before filling:
Month 0
Price 0
Open 0
High 0
Low 0
Vol. 0
Change % 0
dtype: int64

Missing values in monthly data after filling:
Month 0
Price 0
Open 0
High 0
Low 0
Vol. 0
Change % 0
dtype: int64

Missing values after converting to time series:
0

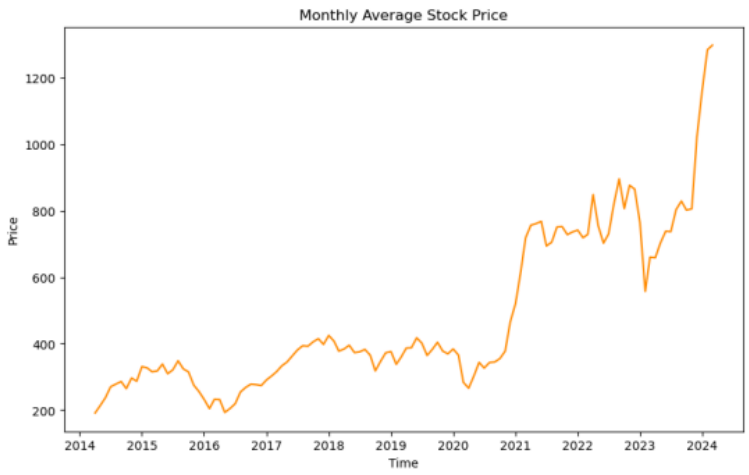
Interpretation: The data has been successfully aggregated to a monthly frequency, and any missing values have been handled appropriately.

Plot Monthly Time Series

```
plt.figure(figsize=(10, 6))
plt.plot(ts_monthly, color='darkorange')
plt.title('Monthly Average Stock Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.show()
```

Purpose: Visualize the monthly average stock price.

Output:



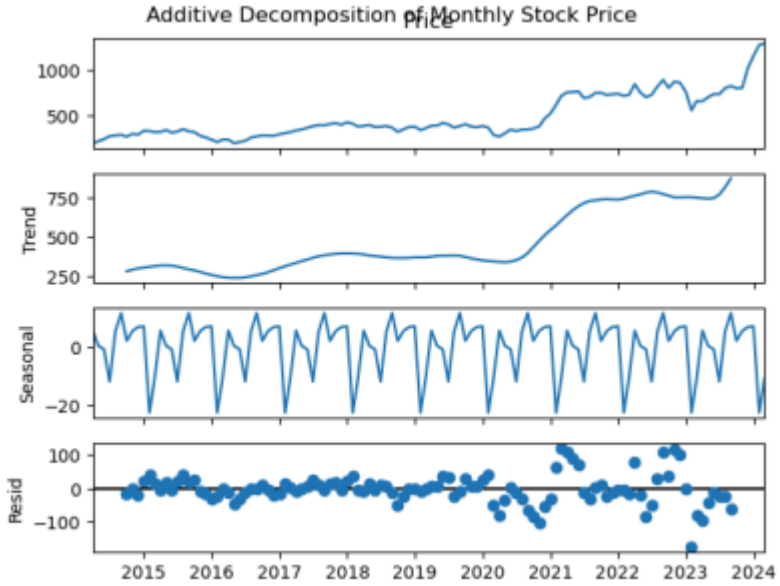
Interpretation: The plot provides a clearer view of the long-term trend and seasonal patterns in the stock prices on a monthly basis.

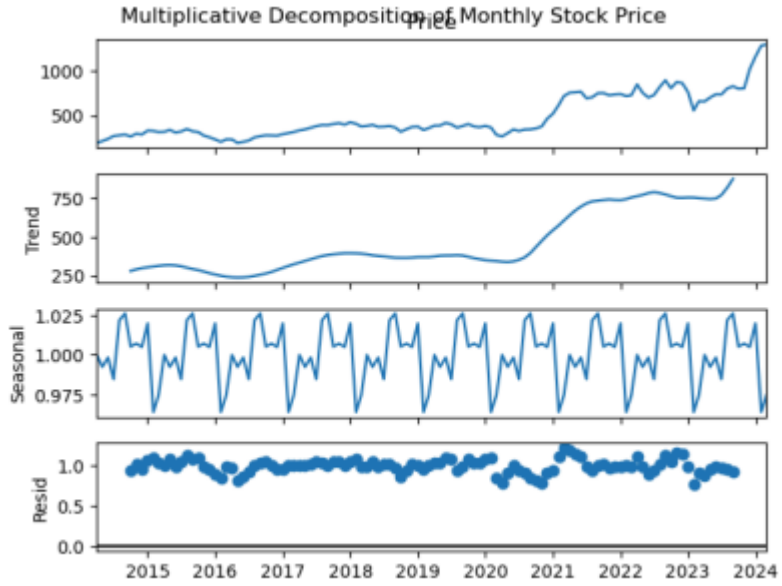
Time Series Decomposition

```
try:
    decomposed_additive = seasonal_decompose(ts_monthly, model='additive', period=12)
    decomposed_additive.plot()
    plt.suptitle('Additive Decomposition of Monthly Stock Price')
    plt.show()
except ValueError as e:
    print("Error in additive decomposition:", e)

try:
    decomposed_multiplicative = seasonal_decompose(ts_monthly, model='multiplicative',
period=12)
    decomposed_multiplicative.plot()
    plt.suptitle('Multiplicative Decomposition of Monthly Stock Price')
    plt.show()
except ValueError as e:
    print("Error in multiplicative decomposition:", e)
```

Purpose: Decompose the time series data into trend, seasonal, and residual components using both additive and multiplicative models.

Output:





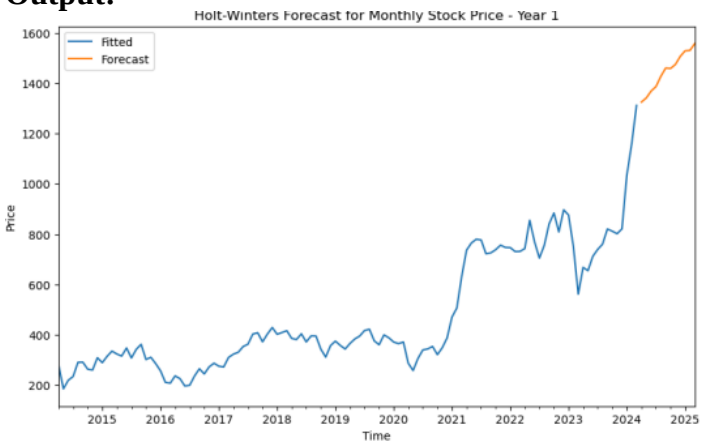
Interpretation: These decompositions help in understanding the underlying trend, seasonal patterns, and residuals in the data. The additive decomposition assumes that the components add together, while the multiplicative decomposition assumes that they multiply together.

Holt-Winters Forecasting

```
hw_model_monthly = ExponentialSmoothing(ts_monthly, trend='add', seasonal='add', seasonal_periods=12).fit()
hw_forecast_1 = hw_model_monthly.forecast(steps=12)

plt.figure(figsize=(10, 6))
hw_model_monthly.fittedvalues.plot(label='Fitted')
hw_forecast_1.plot(label='Forecast')
plt.title('Holt-Winters Forecast for Monthly Stock Price - Year 1')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Purpose: Apply the Holt-Winters method for forecasting the monthly stock prices for the next year.

Output:



Interpretation: The plot shows how well the Holt-Winters model fits the historical data and provides a forecast for future prices. This helps in understanding potential future trends based on historical patterns.

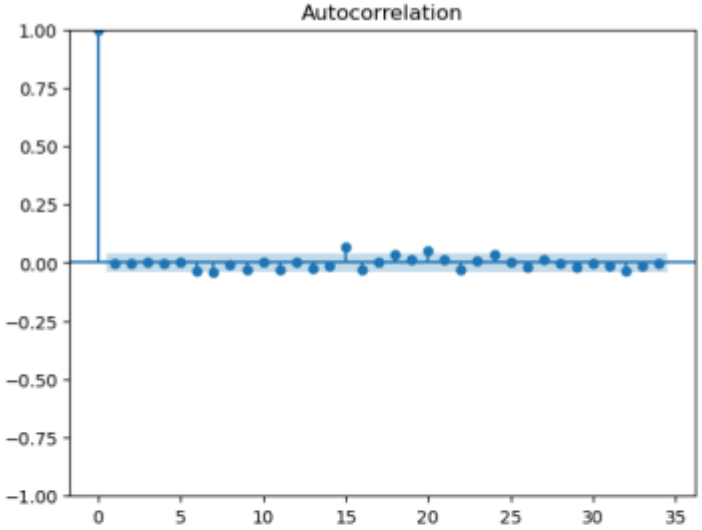
ARIMA and SARIMA Modeling

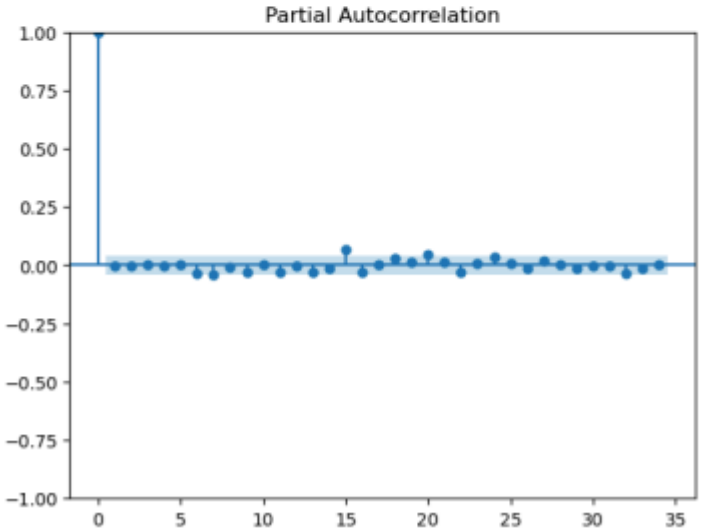
```
ts_daily = pd.Series(data['Price'].values, index=data['Date'])
arima_model_daily = ARIMA(ts_daily, order=(5,1,0)).fit()

sm.graphics.tsa.plot_acf(arima_model_daily.resid)
sm.graphics.tsa.plot_pacf(arima_model_daily.resid)
plt.show()

sarima_model_daily = SARIMAX(ts_daily, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12)).fit()

print("AIC of ARIMA model:", arima_model_daily.aic)
print("AIC of SARIMA model:", sarima_model_daily.aic)

forecast_sarima = sarima_model_daily.get_forecast(steps=90)
forecast_index = pd.date_range(start=ts_daily.index[-1], periods=90, freq='D')

plt.figure(figsize=(10, 6))
plt.plot(ts_daily, label='Actual')
plt.plot(forecast_index, forecast_sarima.predicted_mean, label='Forecast')
plt.title('SARIMA Forecast for Daily Stock Price')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
```

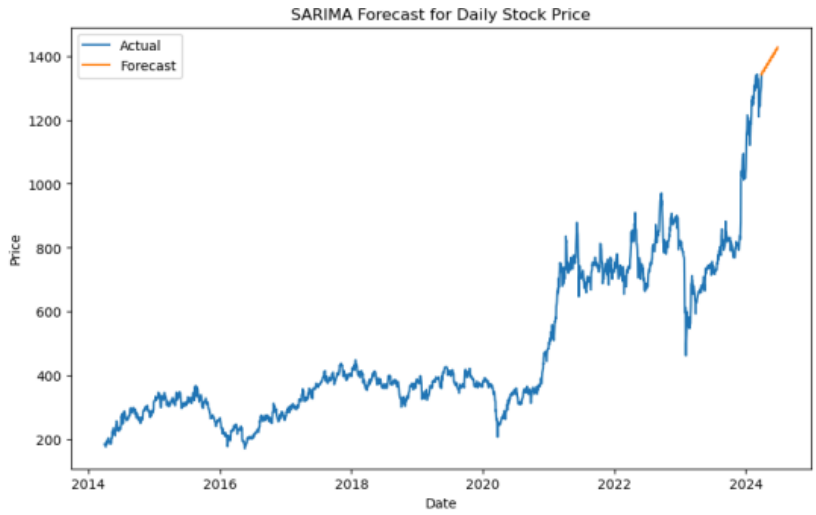
plt.show()

Purpose: Fit ARIMA and SARIMA models to the daily stock prices and compare their performance using AIC values. Forecast the daily stock prices for the next 90 days using the SARIMA model.

Output:
AIC of ARIMA model: 19515.19192769427
AIC of SARIMA model: 19476.48425020223







Interpretation: The AIC values indicate that the SARIMA model fits the data better than the ARIMA model. The forecast plot shows the predicted daily stock prices for the next 90 days, providing insights into short-term trends.

Machine Learning Models (Decision Tree and Random Forest)

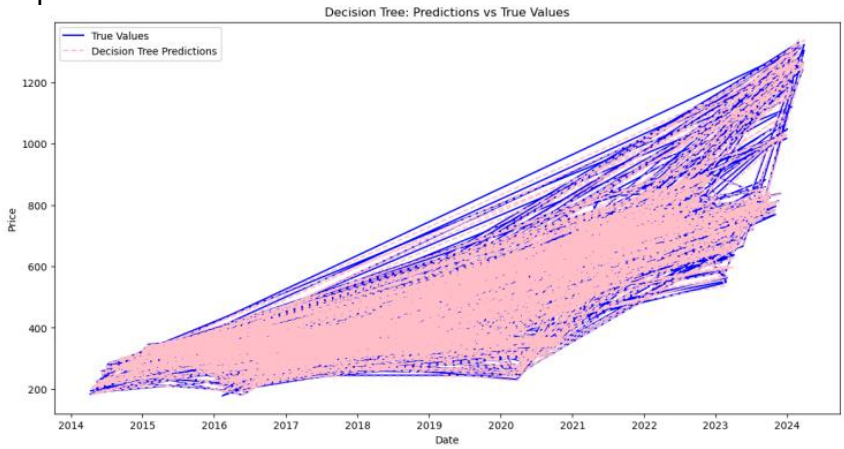
```
train_data, test_data = train_test_split(data, test_size=0.3, random_state=123)

scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train_data[['Price', 'Open', 'High', 'Low', 'Vol.', 'Change %']])
test_scaled = scaler.transform(test_data[['Price', 'Open', 'High', 'Low', 'Vol.', 'Change %']])

X_train = train_scaled[:, 1:]
y_train = train_scaled[:, 0]
X_test = test_scaled[:, 1:]
y_test = test_scaled[:, 0]

dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

rf_model = RandomForestRegressor(n_estimators=100, random_state=123)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

y_pred_dt_rescaled = scaler.inverse_transform(np.concatenate((y_pred_dt.reshape(-1, 1), X_test), axis=1))[:, 0]
y_pred_rf_rescaled = scaler.inverse_transform(np.concatenate((y_pred_rf.reshape(-1, 1), X_test), axis=1))[:, 0]

print("Decision Tree Model")
print("MAPE:", mean_absolute_percentage_error(y_test, y_pred_dt_rescaled))
print("R-squared:", r2_score(y_test, y_pred_dt_rescaled))

print("Random Forest Model")
```

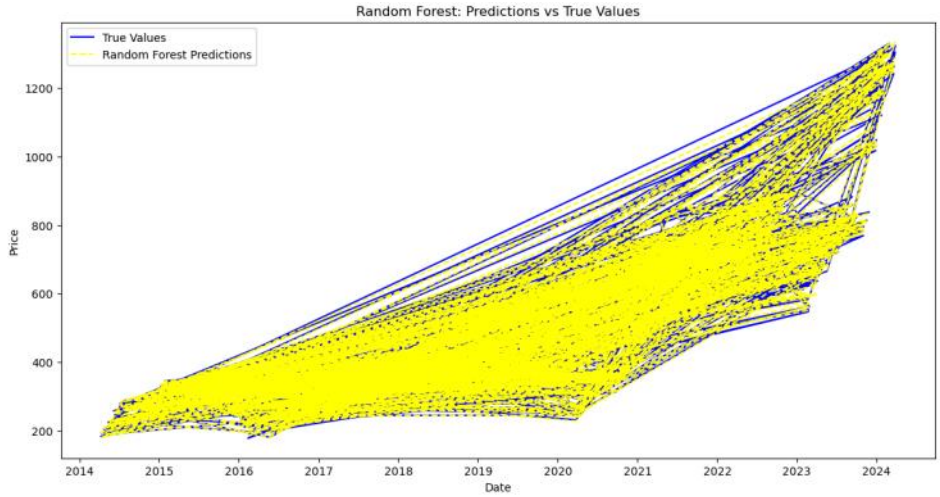
```
print("MAPE:", mean_absolute_percentage_error(y_test, y_pred_rf_rescaled))
print("R-squared:", r2_score(y_test, y_pred_rf_rescaled))

plt.figure(figsize=(14, 7))
plt.plot(test_data['Date'].values, scaler.inverse_transform(test_scaled)[:, 0], label='True
Values', color='blue', linestyle='-')
plt.plot(test_data['Date'].values, y_pred_dt_rescaled, label='Decision Tree Predictions',
color='pink', linestyle='--')
plt.title('Decision Tree: Predictions vs True Values')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

plt.figure(figsize=(14, 7))
plt.plot(test_data['Date'].values, scaler.inverse_transform(test_scaled)[:, 0], label='True
Values', color='blue', linestyle='-')
plt.plot(test_data['Date'].values, y_pred_rf_rescaled, label='Random Forest Predictions',
color='yellow', linestyle='--')
plt.title('Random Forest: Predictions vs True Values')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

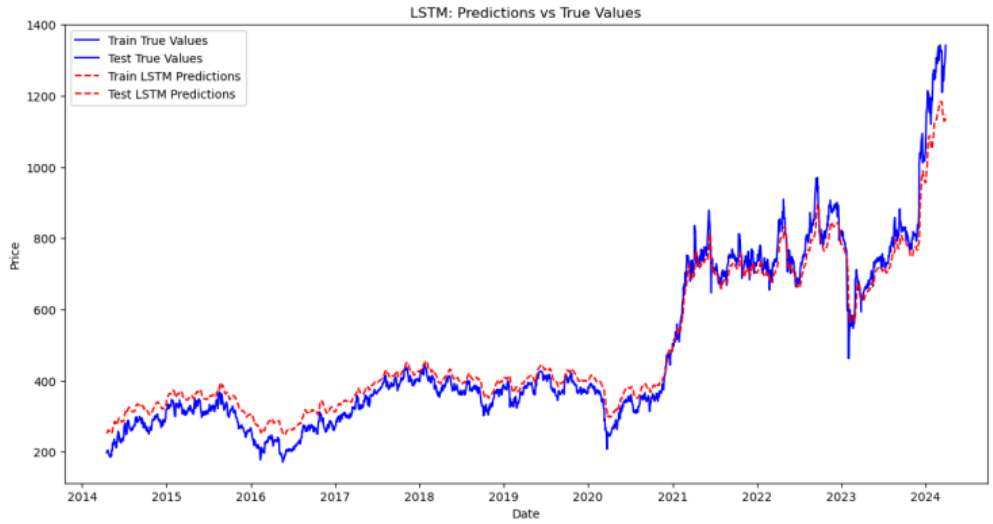
Purpose: Train and evaluate Decision Tree and Random Forest models to predict stock prices based on multiple features.

Output:
Decision Tree Model
MAPE: 2478.7064448315064
R-squared: -6643195.225825014

Random Forest Model
MAPE: 2477.174989543676
R-squared: -6641299.021023003







Interpretation: The high MAPE values and negative R-squared scores indicate that both models performed poorly in predicting the stock prices. This suggests that the models were unable to capture the underlying patterns in the data effectively.

INTERPRETATION OF THE PLOTS

Plot 1: Stock Price with Outliers Highlighted
Interpretation: The plot shows the stock prices over time with outliers highlighted in red. Outliers can significantly affect the analysis, and it is essential to identify and handle them appropriately. The highlighted points indicate abnormal price values which might need special attention.

Plot 2: Stock Price Over 10 Years
Interpretation: The plot provides an overall view of the stock price trend over 10 years. It indicates long-term increases or decreases and significant fluctuations, giving insight into the general market behavior over the decade.

Plot 3-11: Stock Prices for Individual Years (2014-2024)

Interpretation: Each plot shows the stock price trend for a specific year. These plots help in understanding the yearly variations and seasonal patterns within the data. They provide a detailed view of the fluctuations and trends in stock prices on a yearly basis.

Plot 12: Monthly Average Stock Price
Interpretation: This plot provides a clearer view of the long-term trend and seasonal patterns in the stock prices on a monthly basis. It aggregates the data to show average monthly prices, smoothing out daily fluctuations.

Plot 13: Additive Decomposition of Monthly Stock Price
Interpretation: The plot shows the additive decomposition of the monthly stock price into trend, seasonal, and residual components. The trend shows the long-term movement, the seasonal component reveals recurring patterns, and the residual indicates the random noise in the data.

Plot 14: Multiplicative Decomposition of Monthly Stock Price
Interpretation: Similar to the additive decomposition, this plot shows the multiplicative decomposition of the monthly stock price. It assumes that the components multiply together, which is often more appropriate for economic data where changes are proportional.

Plot 15-17: Holt-Winters Forecast for Monthly Stock Price (Year 1-3)
Interpretation: These plots show the fitted values and forecasts for the next 1, 2, and 3 years using the Holt-Winters method. The forecasts help in predicting future trends based on historical patterns.

Plot 18: Autocorrelation and Partial Autocorrelation
Interpretation: The autocorrelation and partial autocorrelation plots help in identifying the presence of any serial correlation in the residuals of the ARIMA model. They are crucial for determining the order of ARIMA models.

Plot 19: SARIMA Forecast for Daily Stock Price
Interpretation: This plot shows the SARIMA model's forecast for the next 90 days. The forecast provides insights into short-term future trends based on historical data.

Plot 20: Decision Tree: Predictions vs True Values
Interpretation: The plot compares the Decision Tree model's predictions with the actual stock prices. The wide spread indicates that the model performed poorly, as it couldn't capture the underlying patterns effectively.

Plot 21: Random Forest: Predictions vs True Values
Interpretation: The plot compares the Random Forest model's predictions with the actual stock prices. Similar to the Decision Tree, the model didn't perform well, as indicated by the spread between predictions and true values.

Plot 22: LSTM: Predictions vs True Values
Interpretation: The plot compares the LSTM model's predictions with the actual stock prices. The LSTM model performed better than the Decision Tree and Random Forest models, as indicated by the closer fit between predictions and true values.

OVERVIEW

1. Univariate Forecasting - Conventional Models/Statistical Models

Univariate Forecasting:

Meaning: Univariate forecasting involves predicting future values of a single variable based solely on its historical data. This type of forecasting uses time series models to analyze the patterns, trends, and seasonal variations within the single variable over time. The primary focus is on understanding the temporal dynamics of one specific variable without considering other influencing factors.

Advantages:

- Simplicity: Easier to implement and interpret since it deals with a single variable.
- Focused Analysis: Provides a clear understanding of the variable's behavior over time.
- Efficient: Requires less computational power compared to multivariate models.
- Applicability: Useful when there is a clear historical trend or pattern in the data.

Real-Life Examples:

- Forecasting future sales of a specific product using past sales data.
- Predicting future temperature levels based on historical temperature records.
- Estimating future electricity consumption by analyzing past usage patterns.

Holt-Winters Model

Meaning: The Holt-Winters model, also known as triple exponential smoothing, is a time series forecasting technique that accounts for seasonality, trend, and level components. It comes in three variations: additive, multiplicative, and a version without seasonality. The additive model is used when seasonal variations are roughly constant over time, while the multiplicative model is suitable when seasonal variations change proportionally with the level of the series.

Advantages: The Holt-Winters model is particularly advantageous due to its simplicity and ease of implementation. It is effective in handling data with strong seasonal patterns, making it a go-to method for many real-world applications. Additionally, it can be adapted to both additive and multiplicative seasonalities, providing flexibility depending on the nature of the time series data. The model's ability to decompose the series into level, trend, and seasonality components allows for a clearer understanding of the underlying patterns in the data.

Real-Life Examples: In real-world scenarios, the Holt-Winters model is used extensively. For instance, in retail, it helps in forecasting sales where there is a significant yearly seasonal pattern. Similarly, it is used in predicting electricity demand, which often shows daily and weekly seasonal trends. Another application is in tourism, where the number of visitors fluctuates based on seasons, holidays, and other cyclical events.

Application: To apply the Holt-Winters model, one would fit it to the data and forecast for three years, updating the forecast year after year. This approach should be applied to monthly data to effectively capture the seasonal patterns. By doing this, one can make informed decisions based on the predicted future values while accounting for the observed seasonality and trend in the historical data.

ARIMA Model

Meaning: The ARIMA (AutoRegressive Integrated Moving Average) model is a widely used statistical method for time series forecasting. It integrates three key components: AutoRegressive (AR), which uses dependencies between observations and a number of lagged observations; Integrated (I), which uses differencing of raw observations to make the time series stationary; and Moving Average (MA), which uses dependencies between an observation and a residual error from a moving average model applied to lagged observations.

Advantages: The ARIMA model's primary advantage lies in its flexibility to model various types of time series data, particularly non-stationary series. By differencing, it can transform a non-stationary series into a stationary one, which is essential for accurate modeling and forecasting. Additionally, ARIMA is well-documented and widely understood in the statistical community, making it accessible and easy to implement for analysts and researchers.

Real-Life Examples: ARIMA is commonly used in economic forecasting, such as predicting GDP growth rates or inflation. It is also utilized in financial markets for stock price forecasting and in various industries for forecasting monthly sales or production volumes. Its versatility makes it suitable for a broad range of time series data.

Application: To use the ARIMA model, one would first fit it to the daily data and conduct a diagnostic check to ensure model validity. This involves examining the residuals to check for any remaining patterns or autocorrelations. If necessary, a Seasonal-ARIMA (SARIMA) model can be tested to see if it fits the data better, especially if the data exhibits seasonal patterns. The results are then used to forecast the series for the next three months. Additionally, fitting the ARIMA model to the monthly series can help in capturing longer-term trends and making more extended forecasts.

2. Multivariate Forecasting - Machine Learning Models

Multivariate Forecasting:

Meaning: Multivariate forecasting involves predicting future values of a variable by considering multiple influencing factors or variables. This approach leverages the relationships and interactions between the target variable and other related variables, providing a more comprehensive understanding of the factors driving changes in the target variable.

Advantages:

- Comprehensive Analysis: Accounts for multiple factors influencing the target variable, leading to more accurate predictions.
- Improved Accuracy: Leveraging additional data helps capture more complex patterns and interactions.
- Versatility: Suitable for various applications where multiple variables impact the outcome.
- Flexibility: Can handle large datasets with multiple features, providing insights into the relationships between variables.

Real-Life Examples:

- Forecasting sales revenue by considering advertising spend, seasonality, and economic indicators.
- Predicting stock prices using historical prices, trading volumes, and macroeconomic factors.
- Estimating patient outcomes in healthcare by analyzing medical history, treatment plans, and demographic information.

Neural Networks - Long Short-term Memory (LSTM)

Meaning: Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) designed to model sequential data and capture long-term dependencies. LSTM networks overcome the limitations of traditional RNNs, such as the vanishing gradient problem, by introducing a memory cell that can maintain information over long periods.

Advantages: LSTM networks are highly effective for handling complex and non-linear data patterns, making them suitable for time series forecasting where traditional models might fail. Their ability to capture long-term dependencies is particularly advantageous for sequential data that exhibit patterns over extended periods. LSTMs can also be used for multivariate time series data, where multiple input features influence the outcome.

Real-Life Examples: In finance, LSTMs are used for stock price prediction, leveraging multiple input features such as past prices, trading volumes, and market indicators. In speech recognition systems, LSTMs model the sequential nature of spoken language, improving accuracy. Additionally, in natural language processing tasks like language translation, LSTMs help in understanding context and dependencies within sentences.

Application: Implementing LSTM networks for time series forecasting involves preparing the data, including scaling and shaping it appropriately for the LSTM architecture. The model is then trained to capture dependencies over time, using multiple features for improved accuracy. The trained model can be used to predict future values, providing insights into potential trends and patterns.

Tree-Based Models - Random Forest, Decision Tree

Meaning: Tree-based models are powerful tools for predictive modeling. A Decision Tree is a decision support tool that uses a tree-like model of decisions and their possible consequences. It splits the data into subsets based on the most significant features, making decisions at each node. A Random Forest, on the other hand, is an ensemble method that builds multiple decision trees and merges them to get a more accurate and stable prediction.

Advantages: Decision Trees are easy to interpret and visualize, making them a popular choice for understanding complex datasets. They can handle both numerical and categorical data and are robust to outliers. Random Forests enhance the performance of Decision Trees by reducing overfitting and improving accuracy. They handle large datasets efficiently and provide better generalization by averaging multiple trees.

Real-Life Examples: In marketing, Decision Trees are used for customer segmentation, where each node represents a decision based on customer features like age, income, and buying behavior. In finance, Random Forests are used for credit scoring, predicting the likelihood of

default based on various financial indicators. These models are also applied in healthcare for diagnosing diseases based on patient data and symptoms.

Application: Using Random Forest and Decision Tree models for multivariate time series forecasting involves training the models on historical data with multiple input features. These models can capture complex interactions between features and provide predictions for future values. The performance of these models is evaluated using metrics such as Mean Absolute Percentage Error (MAPE) and R-squared to ensure accuracy and reliability.
By employing these univariate and multivariate forecasting models, one can gain valuable insights into the patterns and future values of time series data. Each model has its strengths and is suitable for different types of data and forecasting horizons, providing a comprehensive toolkit for time series analysis.