Stock Price Analysis and Forecasting

Rashi Chandel

2024-08-24

1. Introduction

This dissertation, therefore, is dedicated to the analysis and prediction of stock prices using some sophisticated time series analysis methods. Stock prices generally have a complicated nature-essentially volatile, trended, and seasonal. Thus, modeling in this work lies in the implementation of models like ARIMA and Prophet, developed by Facebook, when it comes to forecasting stock prices. Additionally, the analysis of seasonality using STL will be performed in order to identify the underlying patterns in the stock data.

2. Data Loading and Preprocessing

We load the dataset and do some preprocessing to remove missing values, transform date variables, and add more features-lag and moving averages-that will be helpful in model development.

Loading and Preprocessing Data

```r
# Load the dataset
data <- read.csv("C:/Users/RASHI/Downloads/dessertation rashi/World-Stock-Prices-Dataset.csv")

# Clean the dataset and add features
data <- data %>%
  na.omit() %>%
  mutate(Date = ymd_hms(Date)) %>%
  arrange(Date) %>%
  drop_na() %>%
  mutate(
    Lag1 = lag(Close, 1),           # 1-day lag of Close price
    Lag2 = lag(Close, 2),           # 2-day lag of Close price
    MA7 = rollmean(Close, 7, fill = NA),   # 7-day moving average
    MA30 = rollmean(Close, 30, fill = NA)  # 30-day moving average
  ) %>%
  drop_na()  # Drop remaining NA values after feature engineering
```

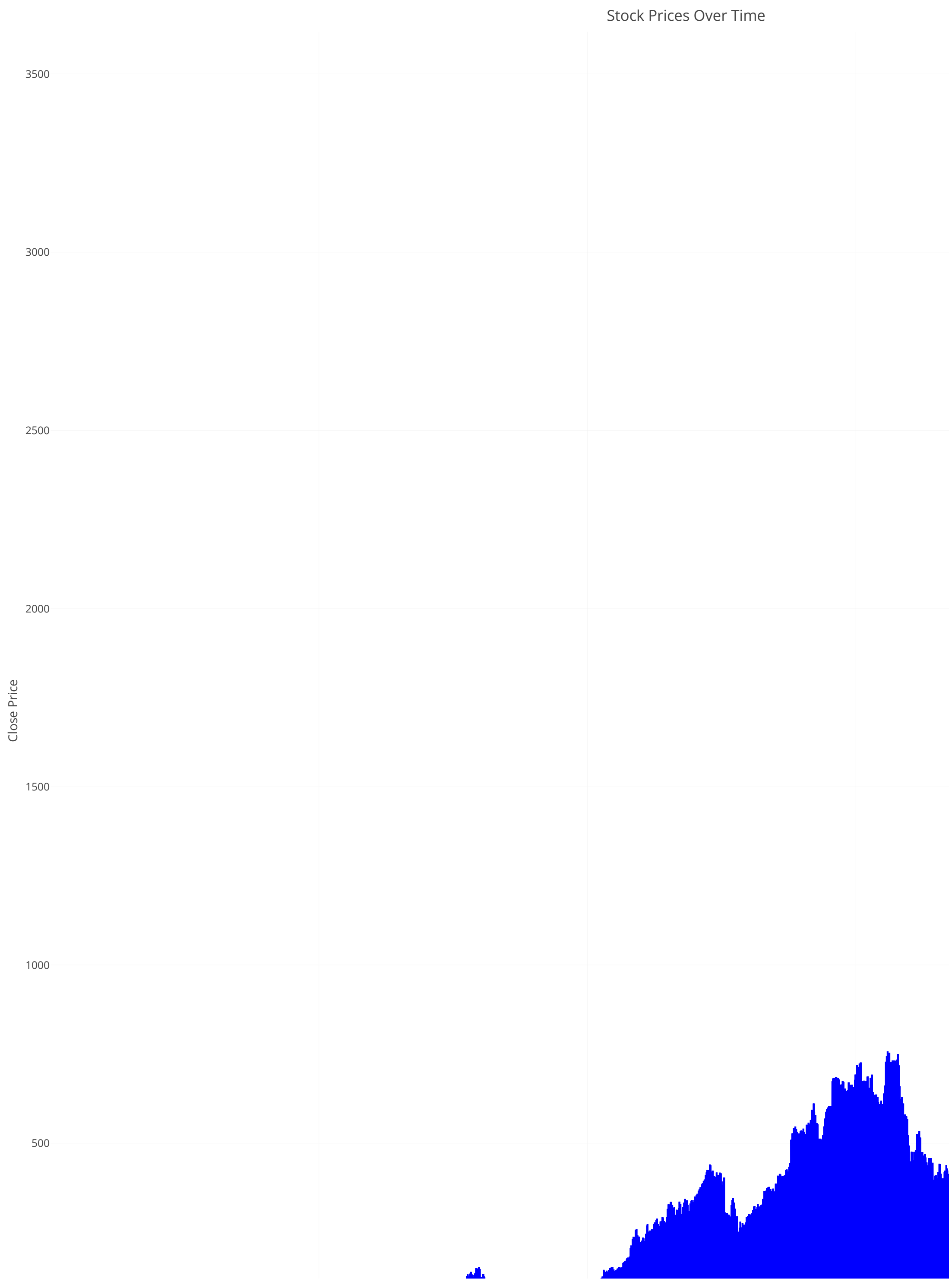
3. Data Visualization

Visualization is a necessary step toward understanding data. We will create various interactive visualizations to understand distribution and trends in stock prices.

3.1 Interactive Line Plot for Stock Prices Over Time

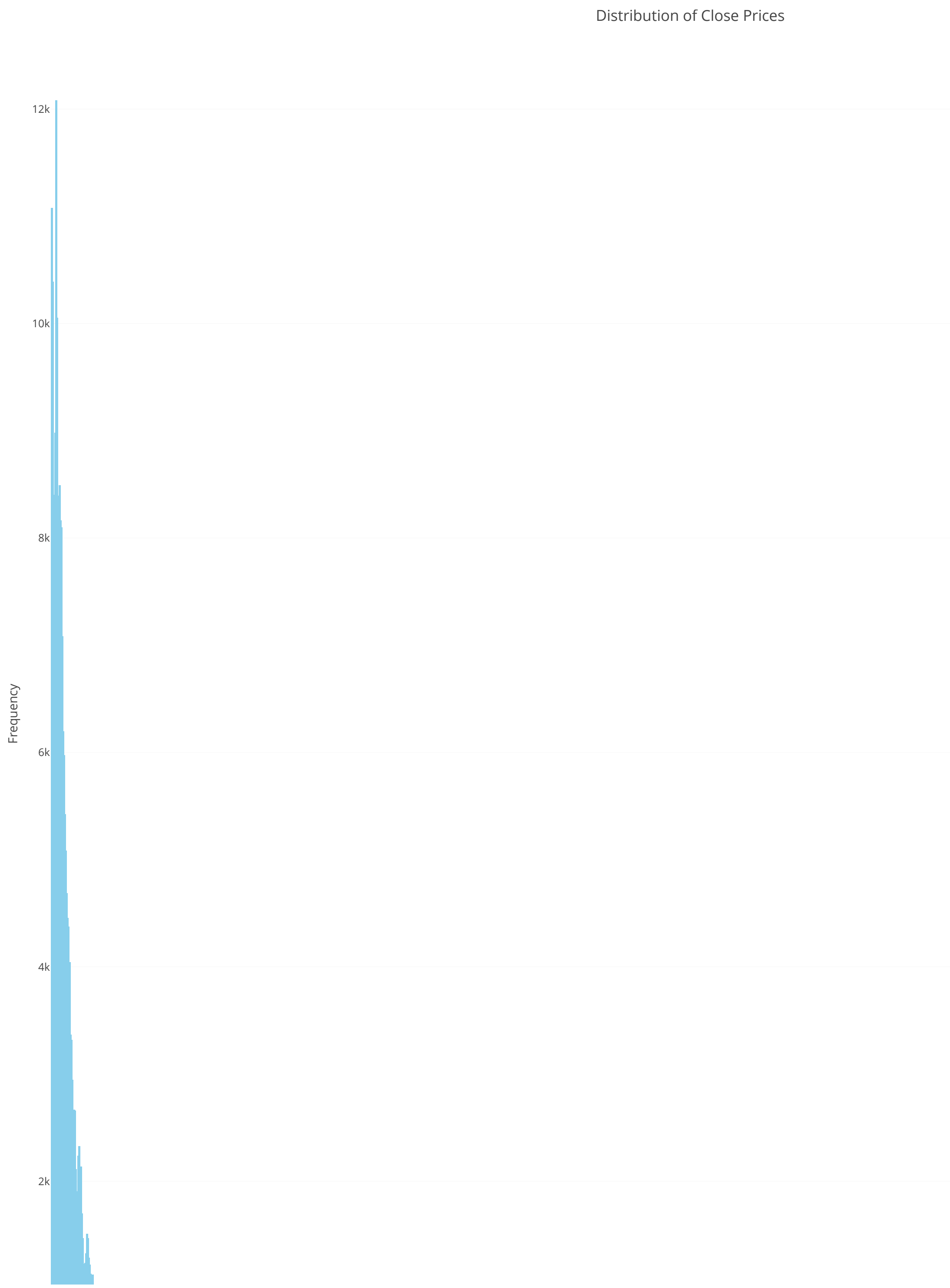
```
plot_ly(data, x = ~Date, y = ~Close, type = 'scatter', mode = 'lines',
        name = 'Close Price', line = list(color = 'blue')) %>%
  layout(title = "Stock Prices Over Time",
         xaxis = list(title = "Date"),
         yaxis = list(title = "Close Price"))
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, please
```


Stock Prices Over Time
Close Price
3500
3000
2500
2000
1500
1000
500

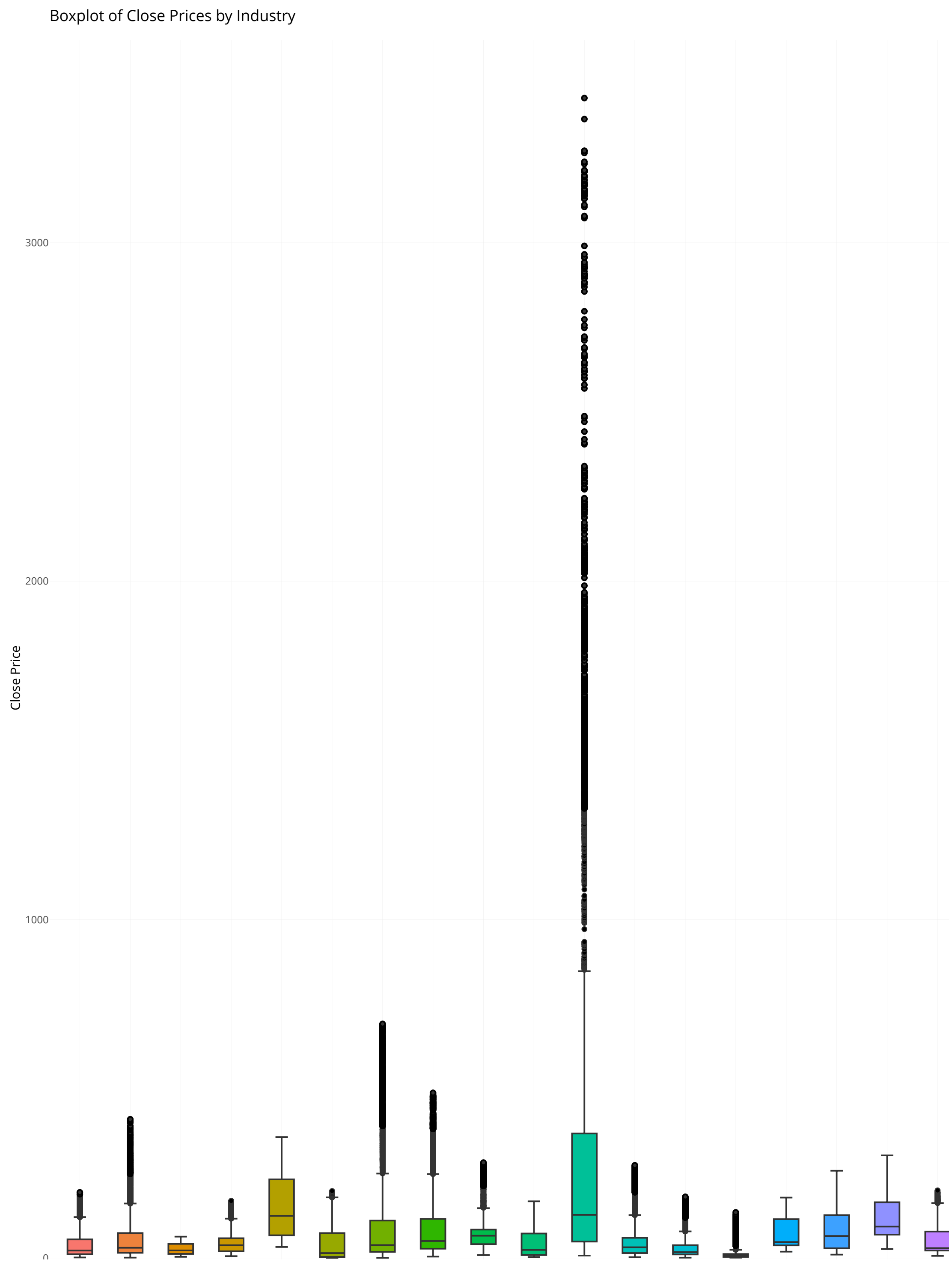
3.2 Interactive Histogram of Close Prices

```
plot_ly(data, x = ~Close, type = 'histogram',
        marker = list(color = 'skyblue')) %>%
  layout(title = "Distribution of Close Prices",
         xaxis = list(title = "Close Price"),
         yaxis = list(title = "Frequency"))
```



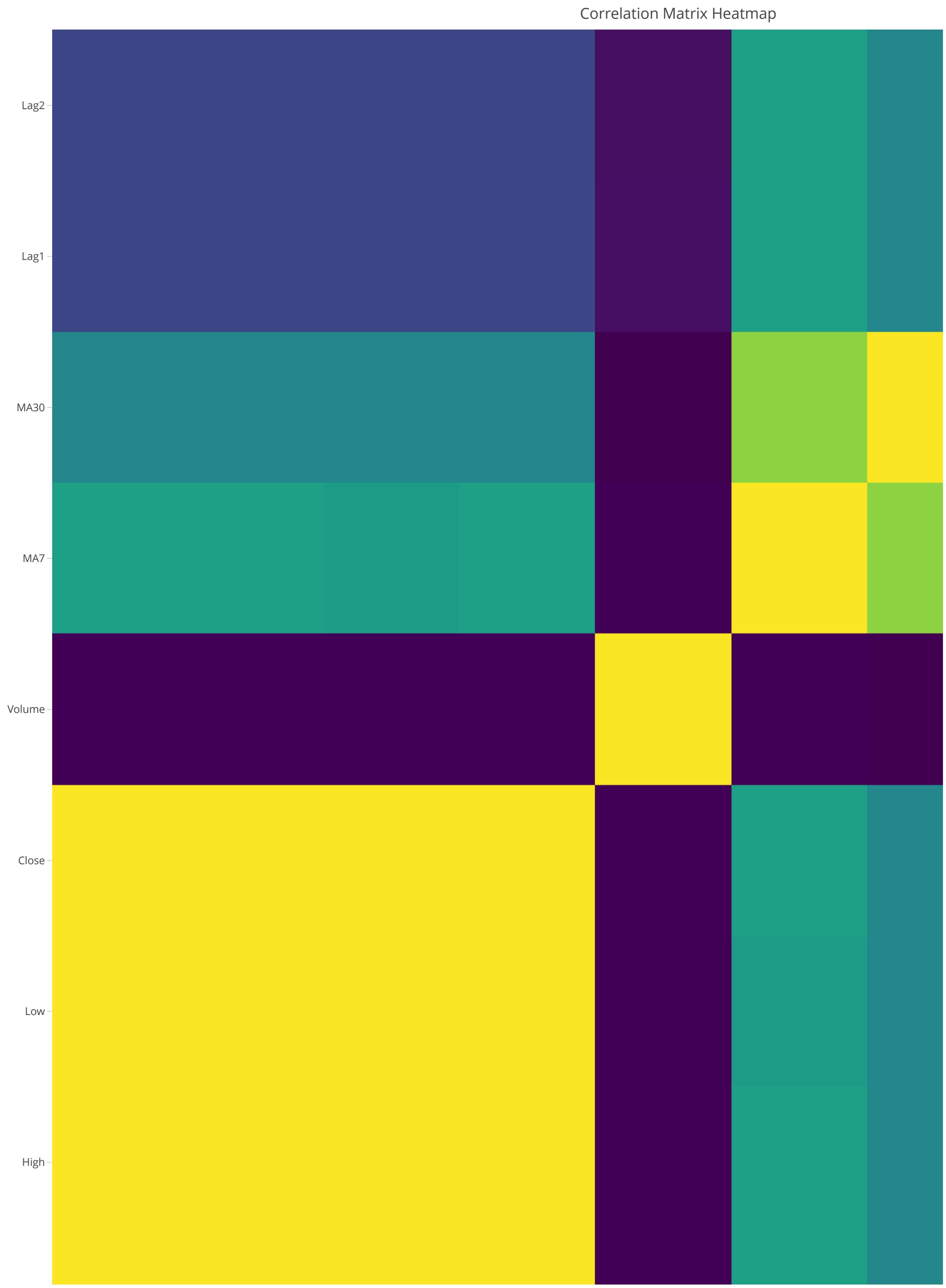
3.3 Interactive Boxplot of Close Prices by Industry

```r
library(ggplot2)
library(plotly)

# Create the ggplot2 boxplot
p <- ggplot(data, aes(x = Industry_Tag, y = Close)) +
  geom_boxplot(aes(fill = Industry_Tag), outlier.colour = "red", outlier.shape = 1) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Boxplot of Close Prices by Industry",
       x = "Industry",
       y = "Close Price")

# Convert the ggplot2 plot to an interactive plotly plot
interactive_plot <- ggplotly(p)

# Display the interactive plot
interactive_plot
```


Boxplot of Close Prices by Industry
Close Price
3000
2000
1000
0

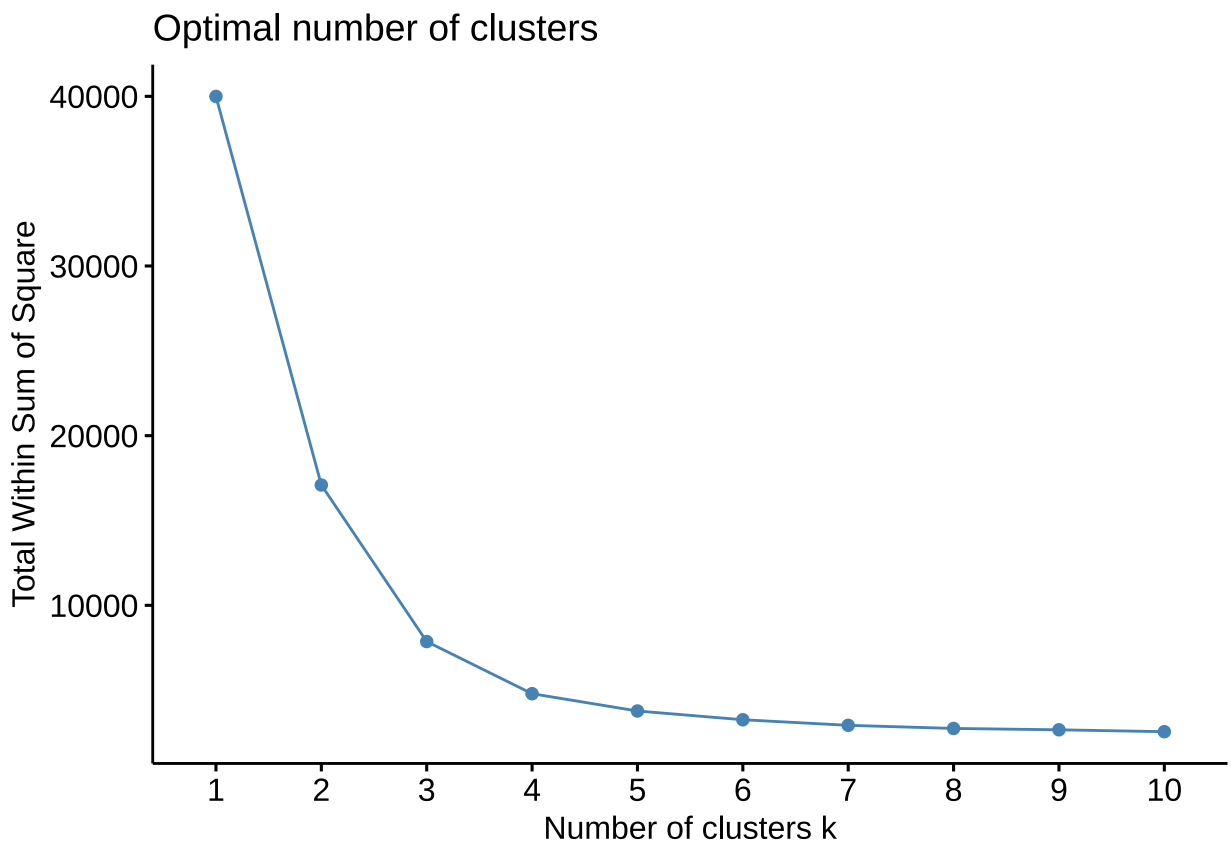
3.4 Interactive Heatmap of Correlation Between Features

```r
# Compute the correlation matrix
corr_matrix <- cor(data %>% select(Open, High, Low, Close, Volume, MA7, MA30, Lag1, Lag2))

# Plot the correlation matrix heatmap
plot_ly(z = corr_matrix, x = colnames(corr_matrix), y = rownames(corr_matrix),
        type = "heatmap", colorscale = "Viridis") %>%
  layout(title = "Correlation Matrix Heatmap",
         xaxis = list(title = ""),
         yaxis = list(title = ""))
```



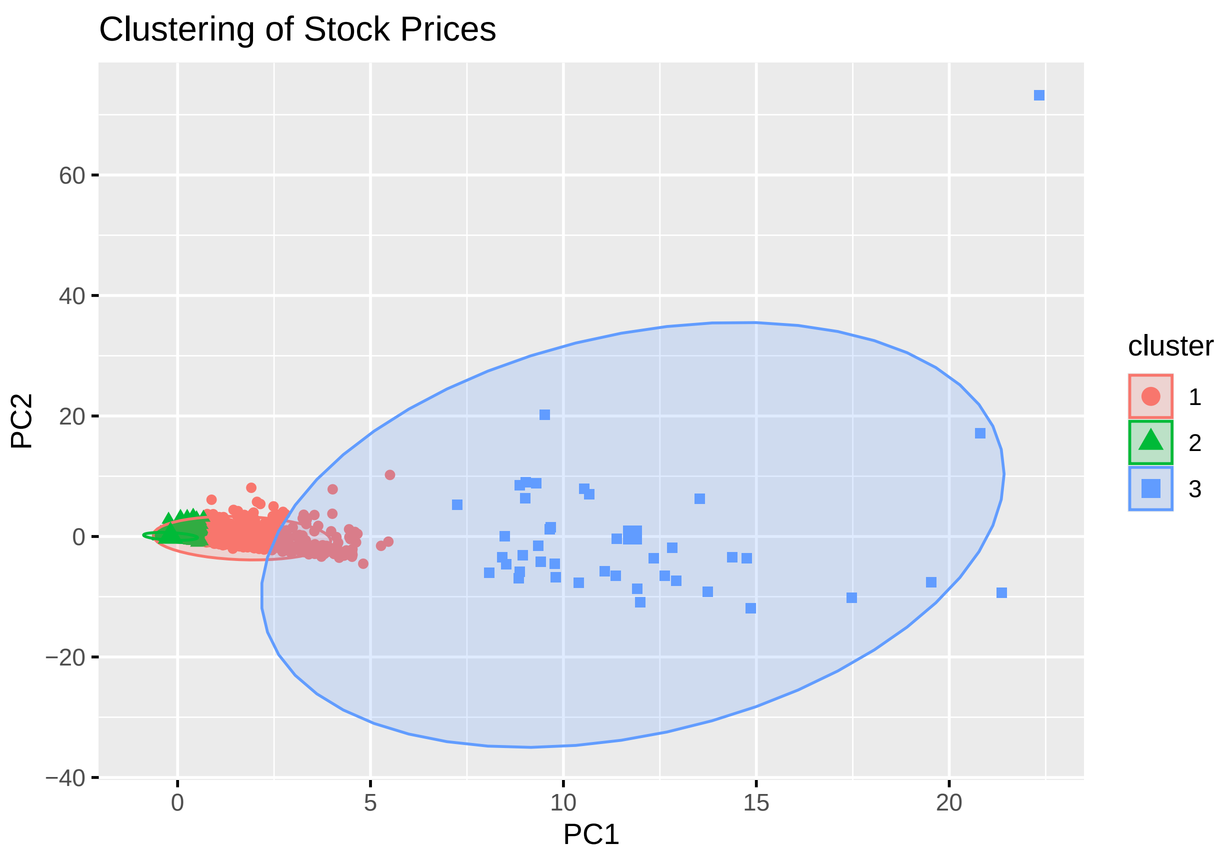
4. Clustering Analysis Clustering will enable us to segment the stock prices according to their characteristics. We can thus reduce the dimensionality by PCA and then use K-means Clustering to group stocks into various volatility clusters.

```r
# Set seed for reproducibility
set.seed(123)

# Sample data for clustering (assuming a large dataset)
sampled_data <- data[sample(1:nrow(data), size = 10000, replace = FALSE),]

# Normalize the selected features
data_normalized <- scale(sampled_data[, c("Open", "High", "Low", "Close")])

# Perform PCA to reduce dimensionality
pca_result <- prcomp(data_normalized, scale. = TRUE)
pca_data <- pca_result$x[, 1:2]  # Use the first two principal components

# Determine optimal number of clusters using the "elbow" method
fviz_nbclust(pca_data, kmeans, method = "wss")
```




```r
# Apply K-means clustering with the chosen number of clusters
kmeans_result <- kmeans(pca_data, centers = 3, nstart = 25)

# Assign cluster names based on volatility
cluster_names <- c("Low Volatility", "Medium Volatility", "High Volatility")
sampled_data$Cluster <- factor(kmeans_result$cluster, labels = cluster_names)

# Visualize clusters
fviz_cluster(kmeans_result, data = pca_data,
```

```
             geom = "point",
             ellipse.type = "norm",
             main = "Clustering of Stock Prices")
```




```
# Calculate and display clustering percentages
cluster_percentages <- round(prop.table(table(sampled_data$Cluster)) * 100, 2)
cat("Clustering Percentages:\n")
```

```
## Clustering Percentages:
```

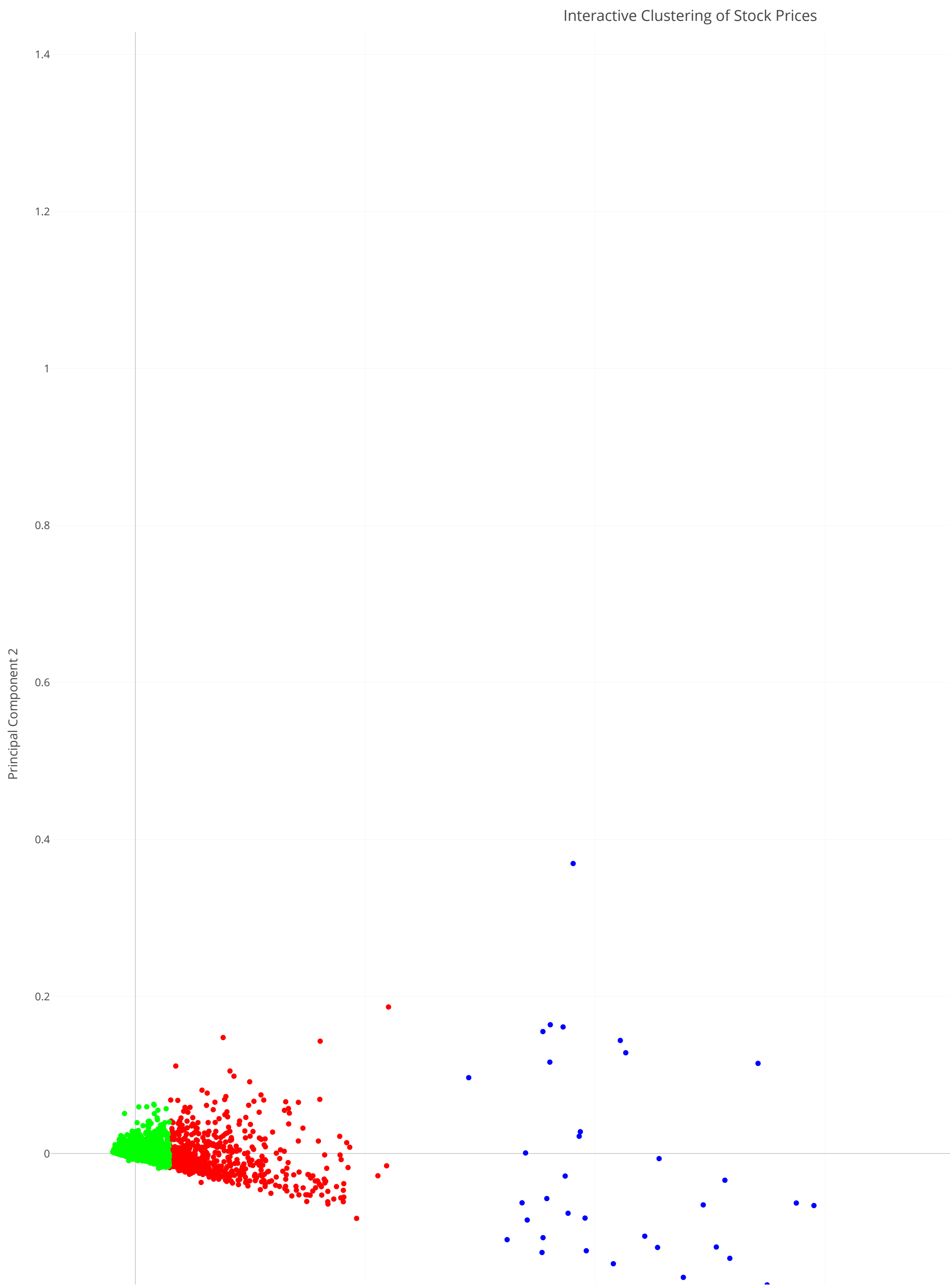
```
print(cluster_percentages)
```

```
##
##    Low Volatility Medium Volatility   High Volatility
##              7.25             92.34              0.41
```

4.1 Interactive Clustering Plot

```
cluster_plot <- plot_ly(sampled_data, x = ~pca_data[, 1], y = ~pca_data[, 2],
                        color = ~Cluster, colors = c('red', 'green', 'blue'),
                        type = 'scatter', mode = 'markers') %>%
  layout(title = "Interactive Clustering of Stock Prices",
```

```
         xaxis = list(title = "Principal Component 1"),
         yaxis = list(title = "Principal Component 2"),
         legend = list(title = list(text = 'Cluster Type')))

# Display the interactive clustering plot
cluster_plot
```


Interactive Clustering of Stock Prices
Principal Component 2
1.4
1.2
1
0.8
0.6
0.4
0.2
0

5. Time Series Forecasting

One of the critical objectives of the said analysis is to forecast the future stock price. We will apply both models, ARIMA and Prophet, to predict future prices and then show their performance comparisons.

5.1 ARIMA Model for Time Series Forecasting

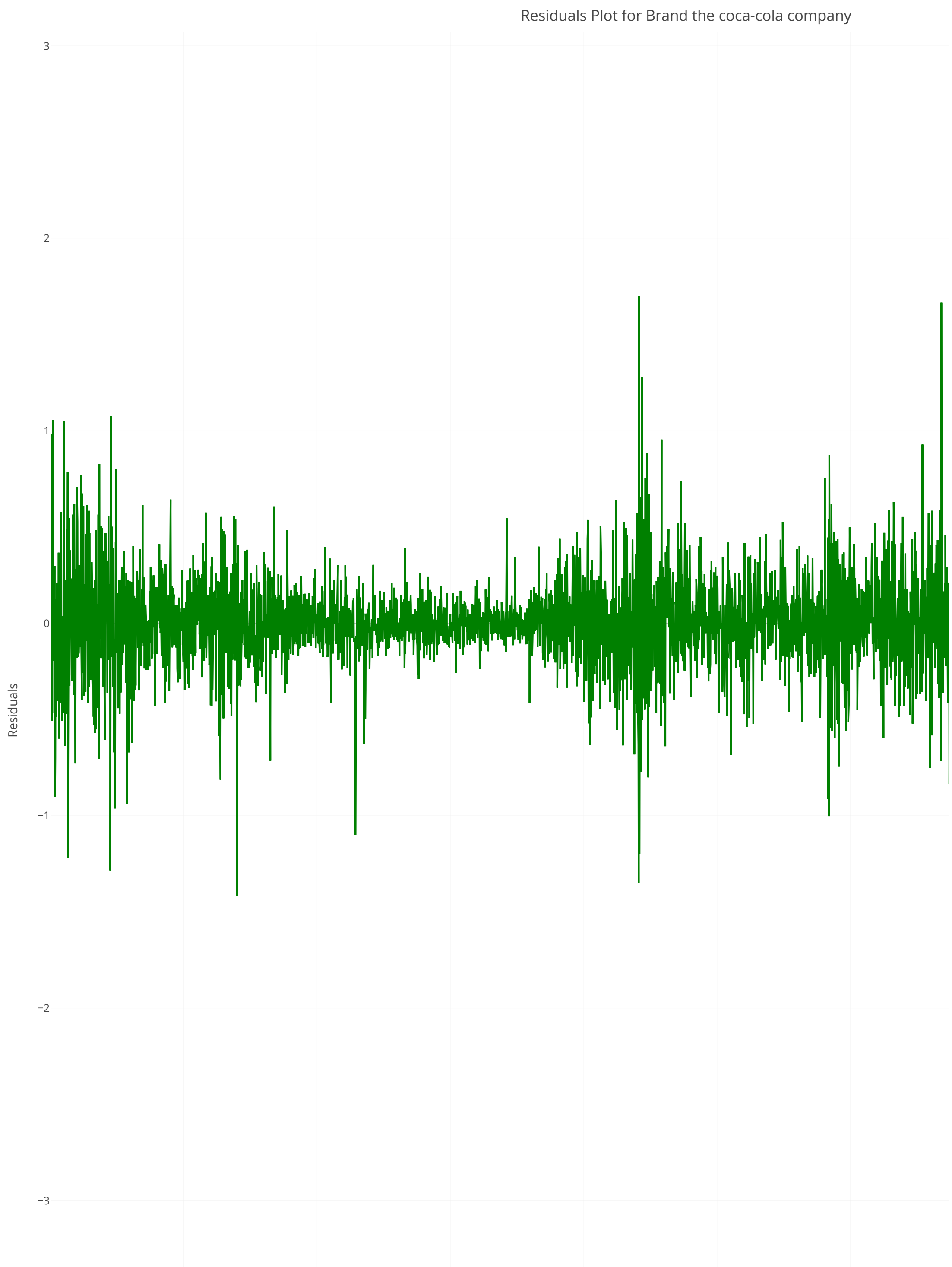
ARIMA is one of the powerful statistical models for time series forecasting. It considers autocorrelation in data and predicts values relative to the previously recorded values.

```r
# Select a brand and industry from the dataset
selected_brand <- unique(data$Brand_Name)[2]
selected_industry <- unique(data$Industry_Tag)[2]

# Filter data for the selected brand and industry
filtered_data <- subset(data, Brand_Name == selected_brand & Industry_Tag == selected_industry)
filtered_data <- filtered_data[order(filtered_data$Date),]

if (nrow(filtered_data) > 0) {

  # Create a time series object for the Close price
  ts_data <- ts(filtered_data$Close, frequency = 252)

  # Train-test split: 80% training, 20% testing
  train_size <- floor(0.8 * length(ts_data))
  train_ts <- ts_data[1:train_size]
  test_ts <- ts_data[(train_size + 1):length(ts_data)]

  # Fit ARIMA model to the training data
  fit_arima <- auto.arima(train_ts)
  summary(fit_arima)

  # Forecast using the ARIMA model for the length of the test data
  forecast_arima <- forecast(fit_arima, h = length(test_ts))

  # Calculate accuracy metrics
  mae <- mean(abs(test_ts - forecast_arima$mean))
  mse <- mean((test_ts - forecast_arima$mean)^2)
  rmse <- sqrt(mse)
  mape <- mean(abs((test_ts - forecast_arima$mean) / test_ts)) * 100
  accuracy_percentage <- 100 - mape

  # Print accuracy metrics
  cat("ARIMA Model Accuracy for Brand", selected_brand, ":\n")
  cat("Mean Absolute Error (MAE):", mae, "\n")
  cat("Mean Squared Error (MSE):", mse, "\n")
  cat("Root Mean Squared Error (RMSE):", rmse, "\n")
  cat("Mean Absolute Percentage Error (MAPE):", mape, "%\n")
  cat("Model Accuracy Percentage:", accuracy_percentage, "%\n")

  ### 1. Line Plot with Confidence Intervals
  p1 <- plot_ly() %>%
    add_lines(x = 1:length(test_ts), y = as.numeric(test_ts), name = "Actual", line = list(color = 'blu
```

```r
    add_lines(x = 1:length(forecast_arima$mean), y = as.numeric(forecast_arima$mean), name = "Forecast"
    add_ribbons(x = 1:length(forecast_arima$mean),
                ymin = as.numeric(forecast_arima$lower[,2]),
                ymax = as.numeric(forecast_arima$upper[,2]),
                name = "95% Confidence Interval",
                line = list(color = 'rgba(255, 0, 0, 0.2)'),
                fillcolor = 'rgba(255, 0, 0, 0.2)') %>%
    layout(title = paste("ARIMA Forecast vs Actual for Brand", selected_brand),
           xaxis = list(title = "Time"),
           yaxis = list(title = "Close Price"),
           legend = list(orientation = 'h', x = 0.5, y = -0.2))

  ### 2. Scatter Plot with Regression Line
  p2 <- plot_ly(data = data.frame(Actual = as.numeric(test_ts),
                                  Forecast = as.numeric(forecast_arima$mean)),
                x = ~Actual, y = ~Forecast,
                type = 'scatter', mode = 'markers', name = "Actual vs Forecast",
                marker = list(color = 'blue')) %>%
    add_trace(x = ~Actual,
              y = ~fitted(lm(Forecast ~ Actual)),
              mode = 'lines', line = list(color = 'red'), name = "Regression Line") %>%
    layout(title = paste("Scatter Plot with Regression Line for Brand", selected_brand),
           xaxis = list(title = "Actual Values"),
           yaxis = list(title = "Forecasted Values"),
           legend = list(orientation = 'h', x = 0.5, y = -0.2))

  ### 3. Difference Plot
  difference <- as.numeric(test_ts) - as.numeric(forecast_arima$mean)
  p3 <- plot_ly(x = 1:length(difference), y = difference, type = 'scatter', mode = 'lines+markers',
                line = list(color = 'purple'), name = "Difference (Actual - Forecast)") %>%
    layout(title = paste("Difference Plot: Actual - Forecast for Brand", selected_brand),
           xaxis = list(title = "Time"),
           yaxis = list(title = "Difference"),
           legend = list(orientation = 'h', x = 0.5, y = -0.2))

  ### 4. Residuals Plot
  residuals <- residuals(fit_arima)
  p4 <- plot_ly(x = 1:length(residuals), y = residuals, type = 'scatter', mode = 'lines',
                line = list(color = 'green'), name = "Residuals") %>%
    layout(title = paste("Residuals Plot for Brand", selected_brand),
           xaxis = list(title = "Time"),
           yaxis = list(title = "Residuals"),
           legend = list(orientation = 'h', x = 0.5, y = -0.2))

  # Display the interactive plots
    p1
    p2
    p3
    p4


} else {
  cat("No data available for the selected Brand and Industry.\n")
```

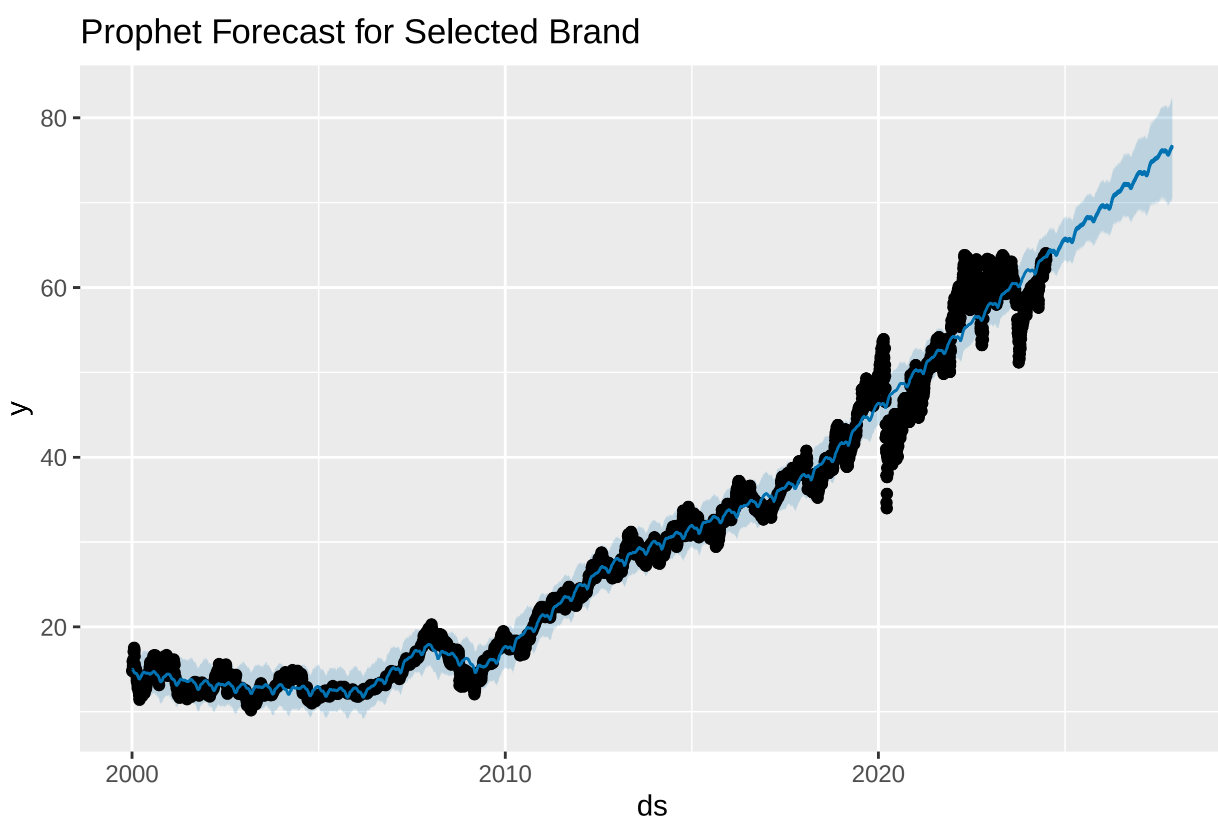
```
}
```

```
## ARIMA Model Accuracy for Brand the coca-cola company :
## Mean Absolute Error (MAE): 4.945435
## Mean Squared Error (MSE): 34.67935
## Root Mean Squared Error (RMSE): 5.888917
## Mean Absolute Percentage Error (MAPE): 8.92855 %
## Model Accuracy Percentage: 91.07145 %
```



5.2. Prophet Model for Time Series Forecasting Prophet is a model developed by Facebook that is particularly good at handling time series data with strong seasonal effects and missing data. We will apply this model to the same data and compare its predictions with the ARIMA model.

```r
# Prepare data for Prophet
prophet_data <- data.frame(ds = filtered_data$Date, y = filtered_data$Close)

# Fit the Prophet model
prophet_model <- prophet(prophet_data)
future <- make_future_dataframe(prophet_model, periods = length(test_ts))
forecast_prophet <- predict(prophet_model, future)

# Plot Prophet forecast
plot(prophet_model, forecast_prophet) + ggtitle("Prophet Forecast for Selected Brand")
```




```r
# Extract the forecasted values for the test period
prophet_forecast_values <- forecast_prophet$yhat[(length(train_ts) + 1):length(ts_data)]
actual_values <- test_ts  # Actual test values
# Calculate accuracy metrics
mae_prophet <- mean(abs(actual_values - prophet_forecast_values))
mse_prophet <- mean((actual_values - prophet_forecast_values)^2)
rmse_prophet <- sqrt(mse_prophet)
mape_prophet <- mean(abs((actual_values - prophet_forecast_values) / actual_values)) * 100
accuracy_percentage_prophet <- 100 - mape_prophet

# Print accuracy metrics
cat("Prophet Model Accuracy for Selected Brand:\n")
```

```
## Prophet Model Accuracy for Selected Brand:
```

```r
cat("Mean Absolute Error (MAE):", mae_prophet, "\n")
```

```
## Mean Absolute Error (MAE): 3.10071
```

```r
cat("Mean Squared Error (MSE):", mse_prophet, "\n")
```

```
## Mean Squared Error (MSE): 14.16086
```

```r
cat("Root Mean Squared Error (RMSE):", rmse_prophet, "\n")
```

```
## Root Mean Squared Error (RMSE): 3.763092
```

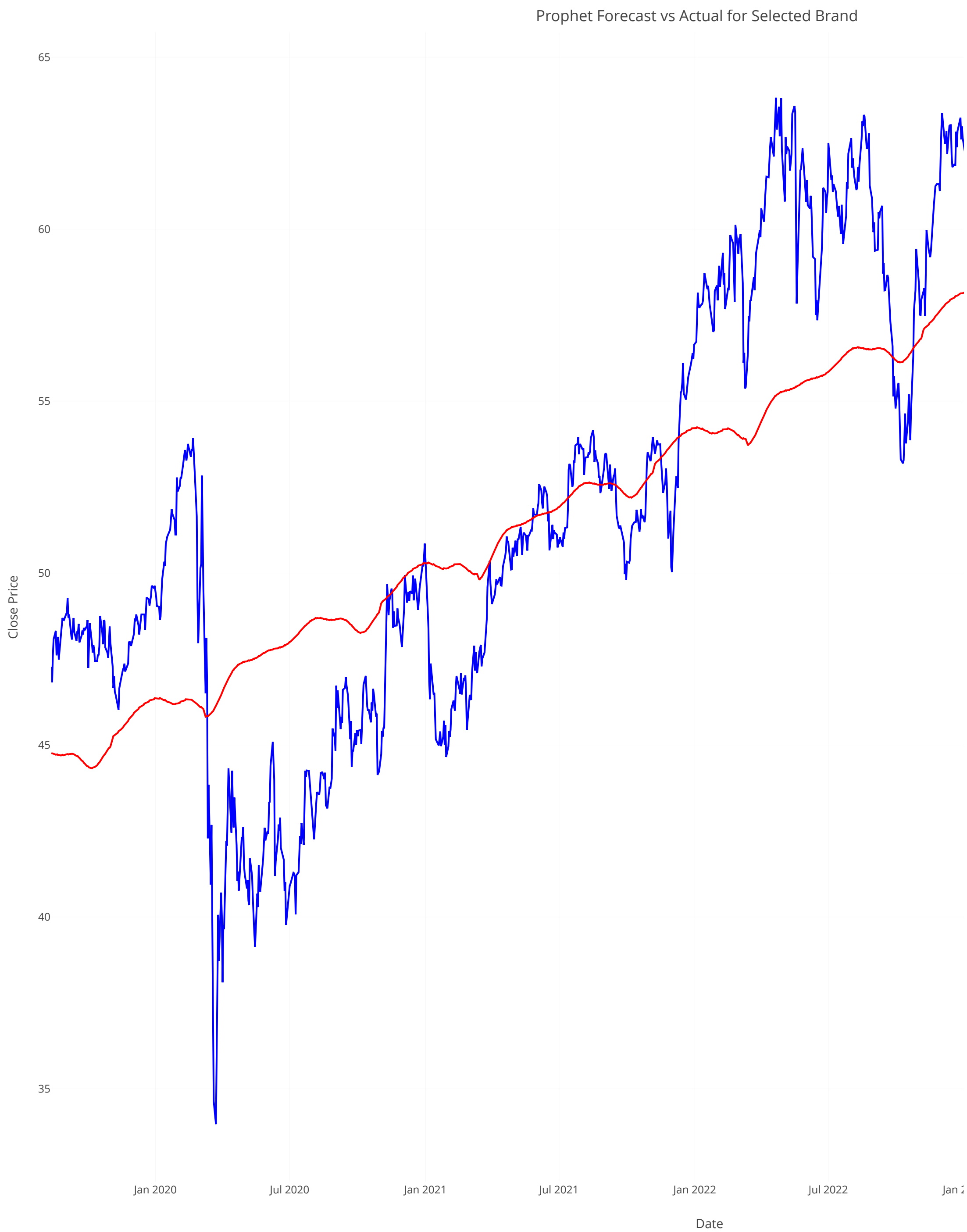
```r
cat("Mean Absolute Percentage Error (MAPE):", mape_prophet, "%\n")
```

```
## Mean Absolute Percentage Error (MAPE): 5.950081 %
```

```r
cat("Model Accuracy Percentage:", accuracy_percentage_prophet, "%\n")
```

```
## Model Accuracy Percentage: 94.04992 %
```

```r
# Visualize the actual vs. forecasted values
plot_ly() %>%
  add_lines(x = as.Date(filtered_data$Date[(length(train_ts) + 1):length(ts_data)]), y = actual_values,
            name = "Actual", line = list(color = 'blue')) %>%
  add_lines(x = as.Date(filtered_data$Date[(length(train_ts) + 1):length(ts_data)]), y = prophet_foreca
            name = "Prophet Forecast", line = list(color = 'red')) %>%
  layout(title = "Prophet Forecast vs Actual for Selected Brand",
         xaxis = list(title = "Date"),
         yaxis = list(title = "Close Price"),
         legend = list(orientation = 'h', x = 0.5, y = -0.2))
```

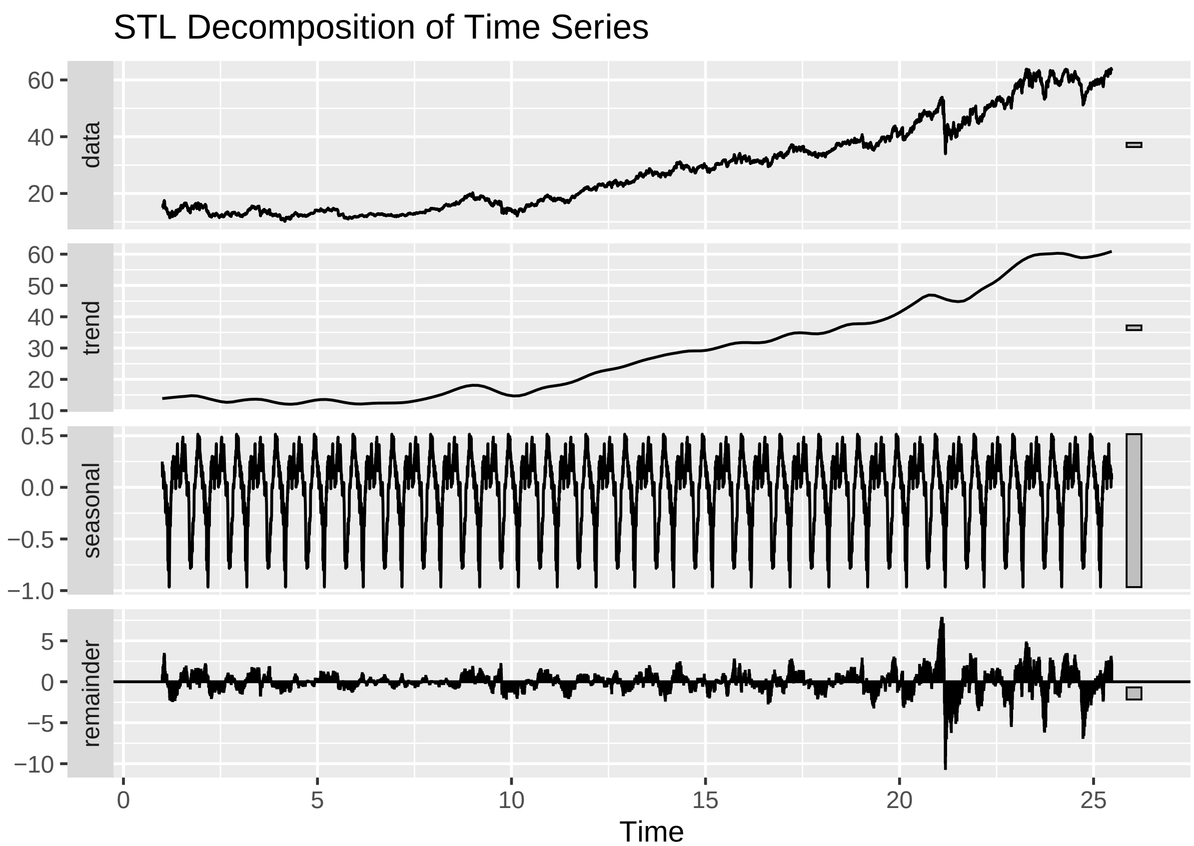


6. Advanced Time Series Analysis Analysis of time series data involves getting a feel for the patterns such as trends and seasonality varying with time. In this section we will apply both ARIMA and Prophet models for forecasting and STL decomposition for the analysis of seasonality.

6.1 STL (Seasonal and Trend Decomposition using Loess) Analysis

This is done by decomposing the time series into STL: trend, seasonal, and remainder components. It will also isolate the seasonal pattern not captured by the simple models

```
# STL Decomposition
stl_decomp <- stl(ts_data, s.window = "periodic")
autoplot(stl_decomp) + ggtitle("STL Decomposition of Time Series")
```



7 Model Comparison

Comparison with Prophet Using Facebook's Prophet for time series forecasting and comparing it with ARIMA

```
# ARIMA Model Forecasting (assuming ARIMA model has already been fitted as fit_arima)
forecast_arima <- forecast(fit_arima, h = length(test_ts))
arima_forecast_values <- as.numeric(forecast_arima$mean)
actual_values <- as.numeric(test_ts)

# Prophet Model Forecasting
prophet_forecast_values <- forecast_prophet$yhat[(length(train_ts) + 1):length(ts_data)]

# Calculate accuracy metrics for ARIMA
mae_arima <- mean(abs(actual_values - arima_forecast_values))
```

```r
mse_arima <- mean((actual_values - arima_forecast_values)^2)
rmse_arima <- sqrt(mse_arima)
mape_arima <- mean(abs((actual_values - arima_forecast_values) / actual_values)) * 100
accuracy_percentage_arima <- 100 - mape_arima

# Calculate accuracy metrics for Prophet
mae_prophet <- mean(abs(actual_values - prophet_forecast_values))
mse_prophet <- mean((actual_values - prophet_forecast_values)^2)
rmse_prophet <- sqrt(mse_prophet)
mape_prophet <- mean(abs((actual_values - prophet_forecast_values) / actual_values)) * 100
accuracy_percentage_prophet <- 100 - mape_prophet

# Print accuracy metrics
cat("ARIMA Model Accuracy:\n")
```

```
## ARIMA Model Accuracy:
```

```r
cat("MAE:", mae_arima, "\n")
```

```
## MAE: 4.945435
```

```r
cat("MSE:", mse_arima, "\n")
```

```
## MSE: 34.67935
```

```r
cat("RMSE:", rmse_arima, "\n")
```

```
## RMSE: 5.888917
```

```r
cat("MAPE:", mape_arima, "%\n")
```

```
## MAPE: 8.92855 %
```

```r
cat("Accuracy Percentage:", accuracy_percentage_arima, "%\n\n")
```

```
## Accuracy Percentage: 91.07145 %
```

```r
cat("Prophet Model Accuracy:\n")
```

```
## Prophet Model Accuracy:
```

```r
cat("MAE:", mae_prophet, "\n")
```

```
## MAE: 3.10071
```

```r
cat("MSE:", mse_prophet, "\n")
```

```
## MSE: 14.16086
```

```r
cat("RMSE:", rmse_prophet, "\n")
```

```
## RMSE: 3.763092
```

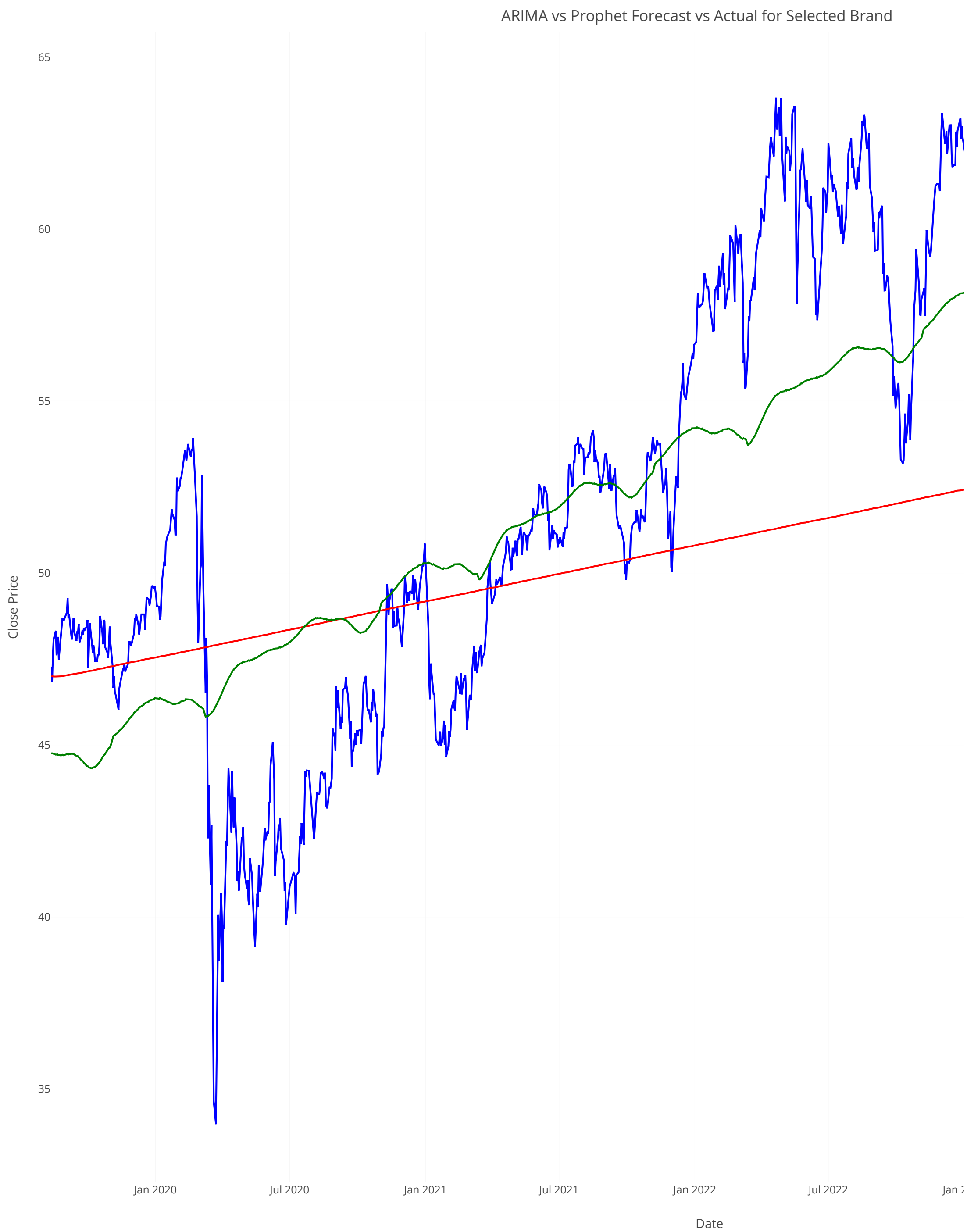
```r
cat("MAPE:", mape_prophet, "%\n")
```

```
## MAPE: 5.950081 %
```

```r
cat("Accuracy Percentage:", accuracy_percentage_prophet, "%\n\n")
```

```
## Accuracy Percentage: 94.04992 %
```

```r
# Visualization: Actual vs Forecast for both models
plot_ly() %>%
  add_lines(x = as.Date(filtered_data$Date[(length(train_ts) + 1):length(ts_data)]), y = actual_values,
            name = "Actual", line = list(color = 'blue')) %>%
  add_lines(x = as.Date(filtered_data$Date[(length(train_ts) + 1):length(ts_data)]), y = arima_forecast_
            name = "ARIMA Forecast", line = list(color = 'red')) %>%
  add_lines(x = as.Date(filtered_data$Date[(length(train_ts) + 1):length(ts_data)]), y = prophet_foreca
            name = "Prophet Forecast", line = list(color = 'green')) %>%
  layout(title = "ARIMA vs Prophet Forecast vs Actual for Selected Brand",
         xaxis = list(title = "Date"),
         yaxis = list(title = "Close Price"),
         legend = list(orientation = 'h', x = 0.5, y = -0.2))
```



ARIMA vs Prophet Forecast vs Actual for Selected Brand
Close Price
65
60
55
50
45
40
35
Jan 2020
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Date