# Appendix

Gursimar Singh

2025-03-30

```
library(tidyverse)
library(imputeTS)
library(forecast)
library(tseries)
```

```
#### Loading and Preprocessing all the Data ####

# Process the Price data
price_data = read.csv("Daily Prices_ICCO.csv")

# new dataframe containing date and price
price_data_modified <- data.frame(
  date = as.Date(price_data[[1]], format = "%d/%m/%Y"),
  price = as.numeric(gsub(",", "", price_data[[2]]))
)

# Remove rows that have duplicate values for 'date'
price_data_modified <- price_data_modified %>%
  distinct(date, .keep_all = TRUE)

price_data_modified <- price_data_modified[nrow(price_data_modified):1, ]
row.names(price_data_modified) <- NULL

# Process the weather data
ghana_data <- read_csv("Ghana_data.csv") %>%
  distinct() %>%
  mutate(date = as.Date(DATE, format = "%Y-%m-%d")) %>%
  filter(!(is.na(PRCP) & is.na(TAVG) & is.na(TMAX) & is.na(TMIN)))
```

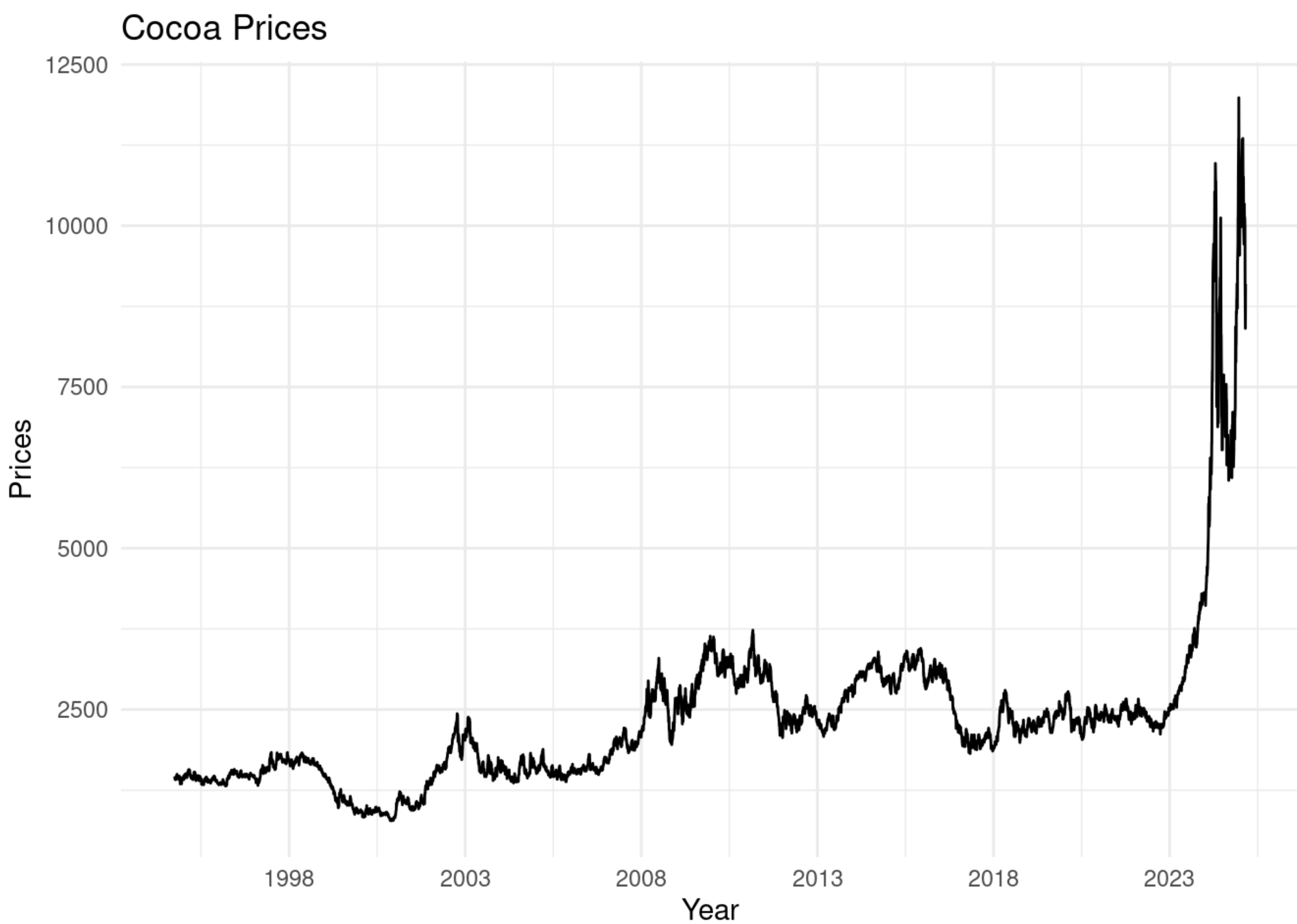
```
## Rows: 53231 Columns: 7
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (2): STATION, NAME
## dbl  (4): PRCP, TAVG, TMAX, TMIN
## date (1): DATE
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Impute missing values by using interpolation for TAVG and fill by 0 for PRCP.
ghana_imputed <- ghana_data %>%
  arrange(date) %>%
  mutate(
    TAVG = na_interpolation(TAVG),
    PRCP = ifelse(is.na(PRCP), 0, PRCP)
  )

# avg over all the different stations
weather_avg <- ghana_imputed %>%
  group_by(date) %>%
  summarise(PRCP = mean(PRCP, na.rm = TRUE),
            TAVG = mean(TAVG, na.rm = TRUE))

# Process the sentiment data
sentiment_data = read.csv('sentiment_data.csv')
sentiment_data <- sentiment_data %>% mutate(date = as.Date(date, format = "%Y-%m-%
d"))

# Ensure sentiment_data has only one row per date
sentiment_data <- sentiment_data %>%
  group_by(date) %>%
  summarise(sentiment_score = mean(sentiment_score, na.rm = TRUE))
```

```
#### Merging all three datasets ####

#First we join price with weather
merged_price_weather <- left_join(price_data_modified, weather_avg, by = "date")

#Impute the non-matched dates
merged_price_weather <- merged_price_weather %>%
  mutate(
    TAVG = na_interpolation(TAVG),
    PRCP = ifelse(is.na(PRCP), 0, PRCP)
  )

full_data <- left_join(merged_price_weather, sentiment_data, by = "date") %>%
  mutate(sentiment_score = ifelse(is.na(sentiment_score), 0, sentiment_score))
```

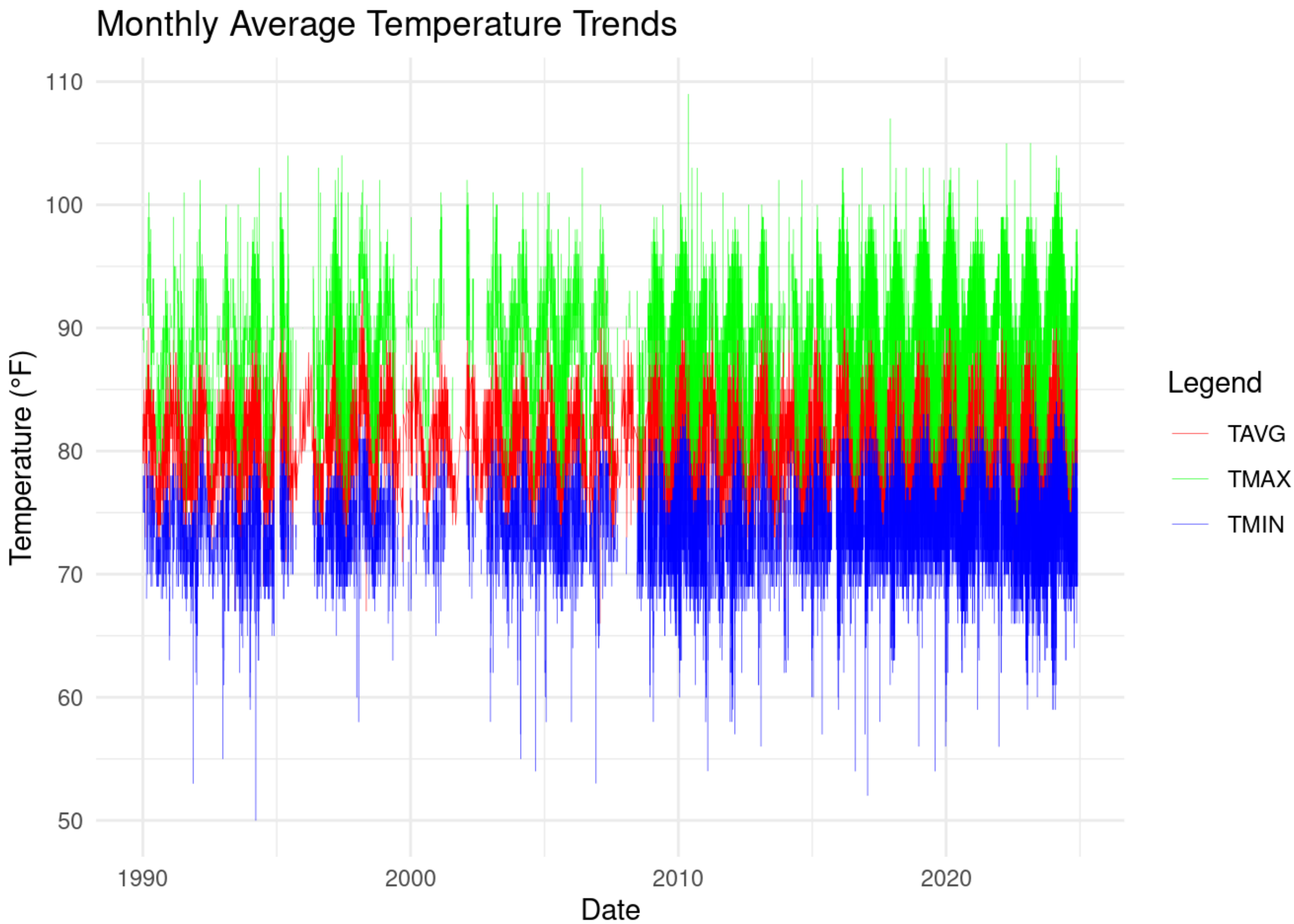
```
#### General Data Plots ####

# Price data
prices <- ts(full_data$price, frequency = 365)

df <- data.frame(ds = as.Date(full_data$date), y = prices)

ggplot(df, aes(x = ds, y = y)) +
  geom_line() +
  scale_x_date(date_breaks = "5 years", date_labels = "%Y") +
  labs(title = "Cocoa Prices", x = "Year", y = "Prices") +
  theme_minimal()
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```

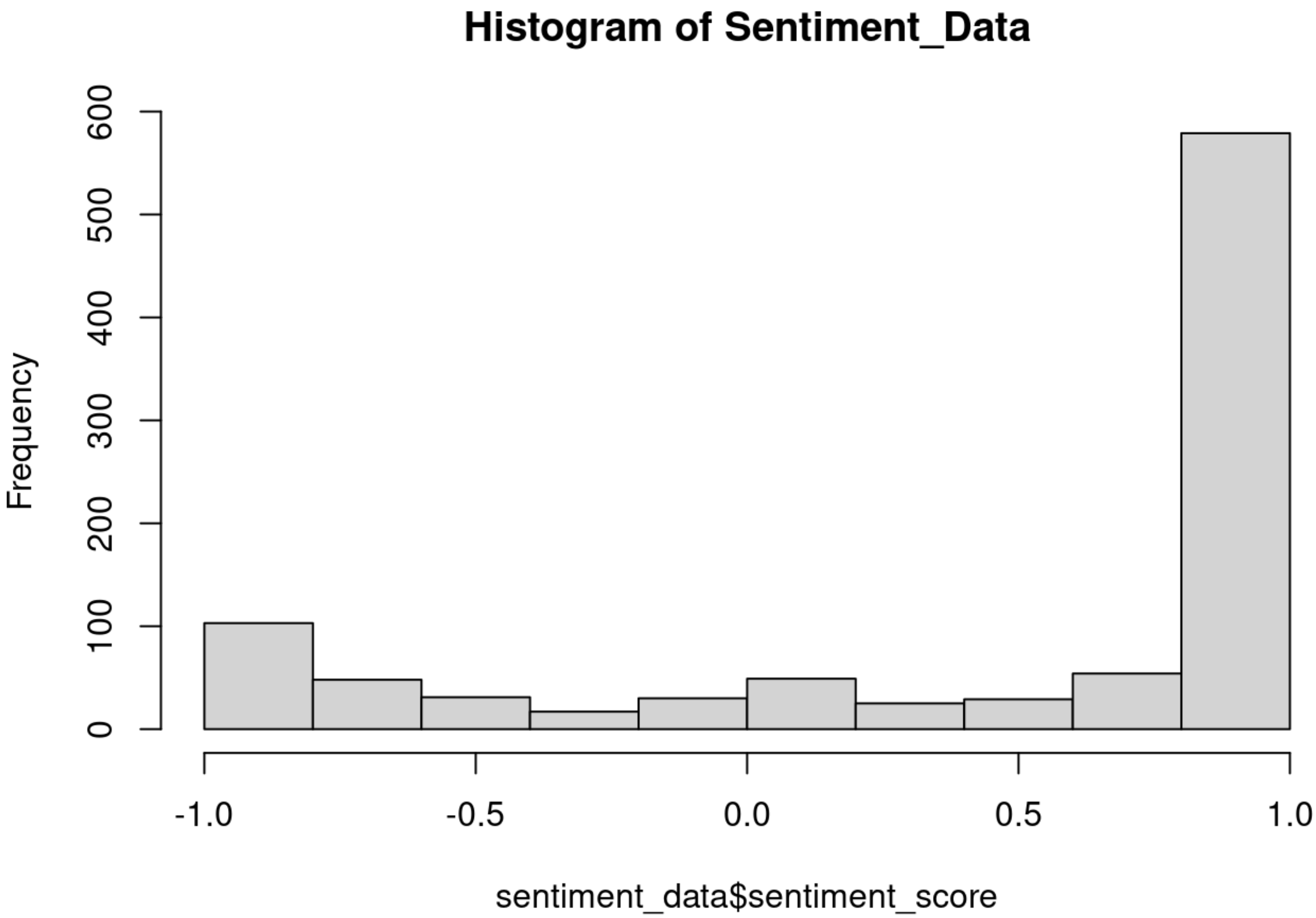



```
# Climate Data

ggplot(ghana_imputed, aes(x = date)) +
  geom_line(aes(y = TAVG, color = "TAVG"), size = 0.1) +
  geom_line(aes(y = TMAX, color = "TMAX"), size = 0.1) +
  geom_line(aes(y = TMIN, color = "TMIN"), size = 0.1) +
  labs(title = "Monthly Average Temperature Trends",
       x = "Date",
       y = "Temperature (°F)",
       color = "Legend") +
  scale_color_manual(values = c("TAVG" = "red", "TMAX" = "green", "TMIN" = "blue")) +
  theme_minimal()
```

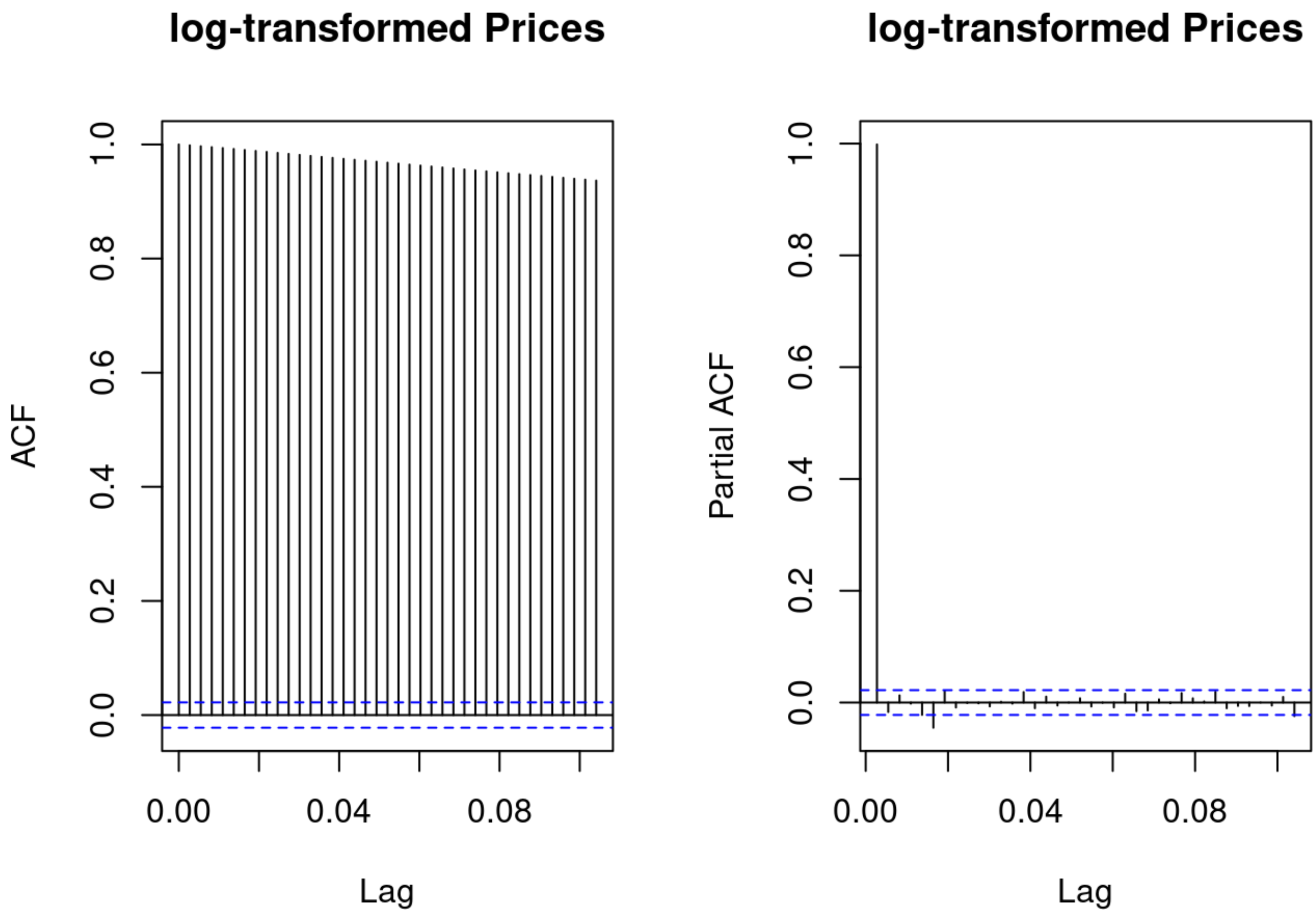
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```
# Sentiment Data
hist(sentiment_data$sentiment_score, main = "Histogram of Sentiment_Data")
```




```r
# Log Transformed Price Data
transformed_prices <- log(prices)
par(mfrow=c(1, 2))
acf(transformed_prices, main = "log-transformed Prices")
pacf(transformed_prices, main = "log-transformed Prices")
```




```
df <- data.frame(ds = as.Date(full_data$date), y = transformed_prices)

ggplot(df, aes(x = ds, y = y)) +
  geom_line() +
  scale_x_date(date_breaks = "5 years", date_labels = "%Y") +
  labs(title = "log-transformed Cocoa Prices", x = "Year", y = "log-transformed Price
s") +
  theme_minimal()
```

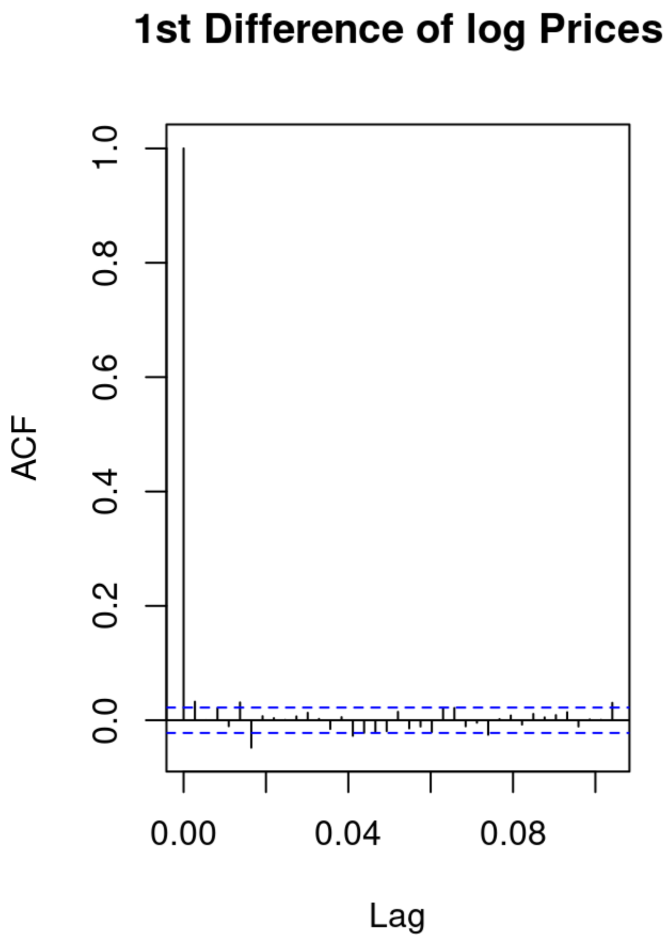
```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```

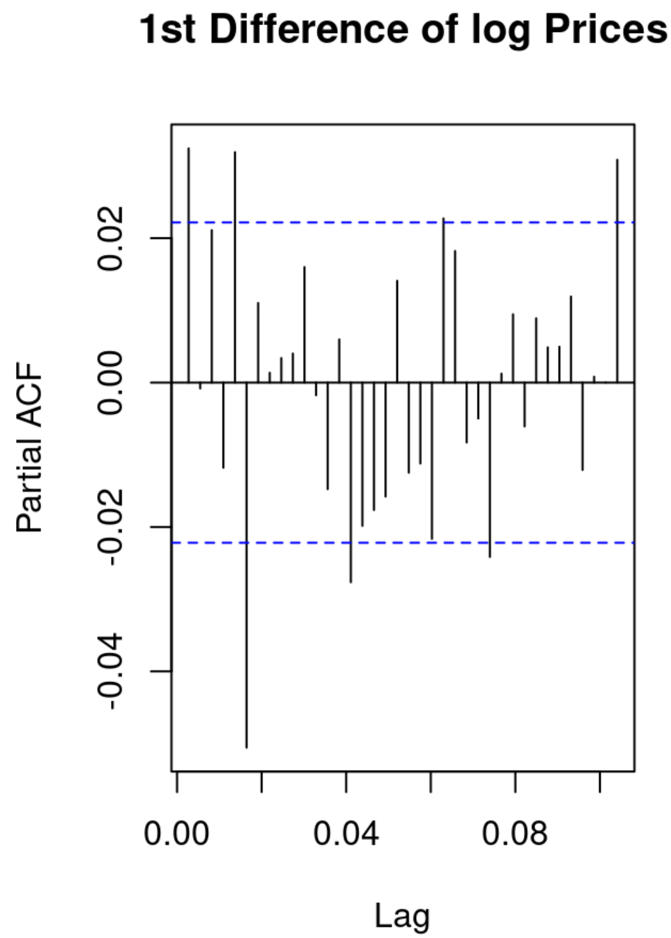

```
adf.test(transformed_prices)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  transformed_prices
## Dickey-Fuller = -1.6096, Lag order = 19, p-value = 0.7437
## alternative hypothesis: stationary
```

p-value > 0.05, not yet stationary

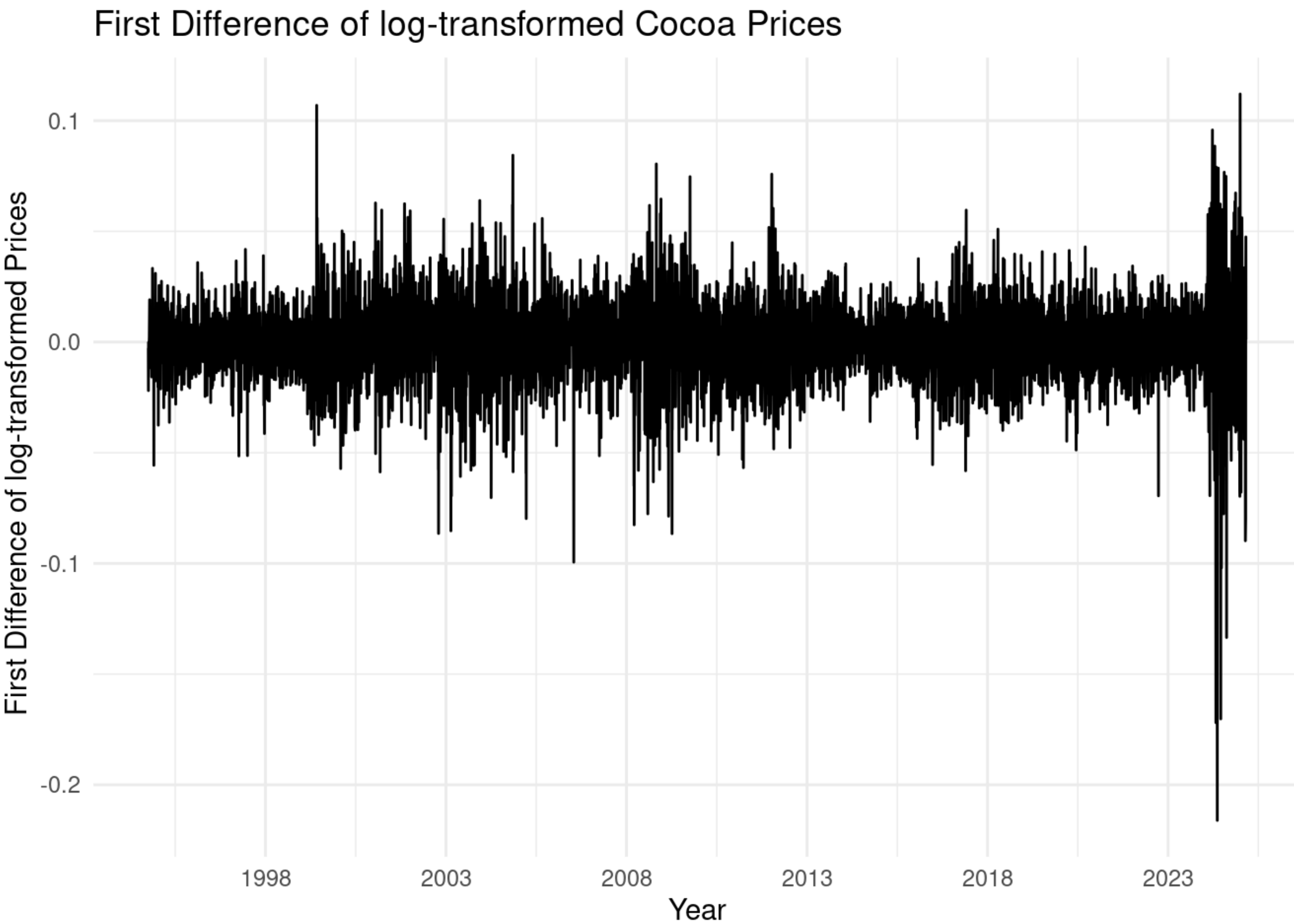
```
# Performing first-order differencing
transformed_prices_1 <- diff(transformed_prices)
par(mfrow=c(1, 2))
acf(transformed_prices_1, main = "1st Difference of log Prices")
pacf(transformed_prices_1,main = "1st Difference of log Prices")
```




```
dates_diff <- as.Date(full_data$date[-1])
df <- data.frame(ds = dates_diff, y = transformed_prices_1)


ggplot(df, aes(x = ds, y = y)) +
  geom_line() +
  scale_x_date(date_breaks = "5 years", date_labels = "%Y") +
  labs(title = "First Difference of log-transformed Cocoa Prices", x = "Year", y = "
First Difference of log-transformed Prices") +
  theme_minimal()
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```




```
adf.test(transformed_prices_1)
```

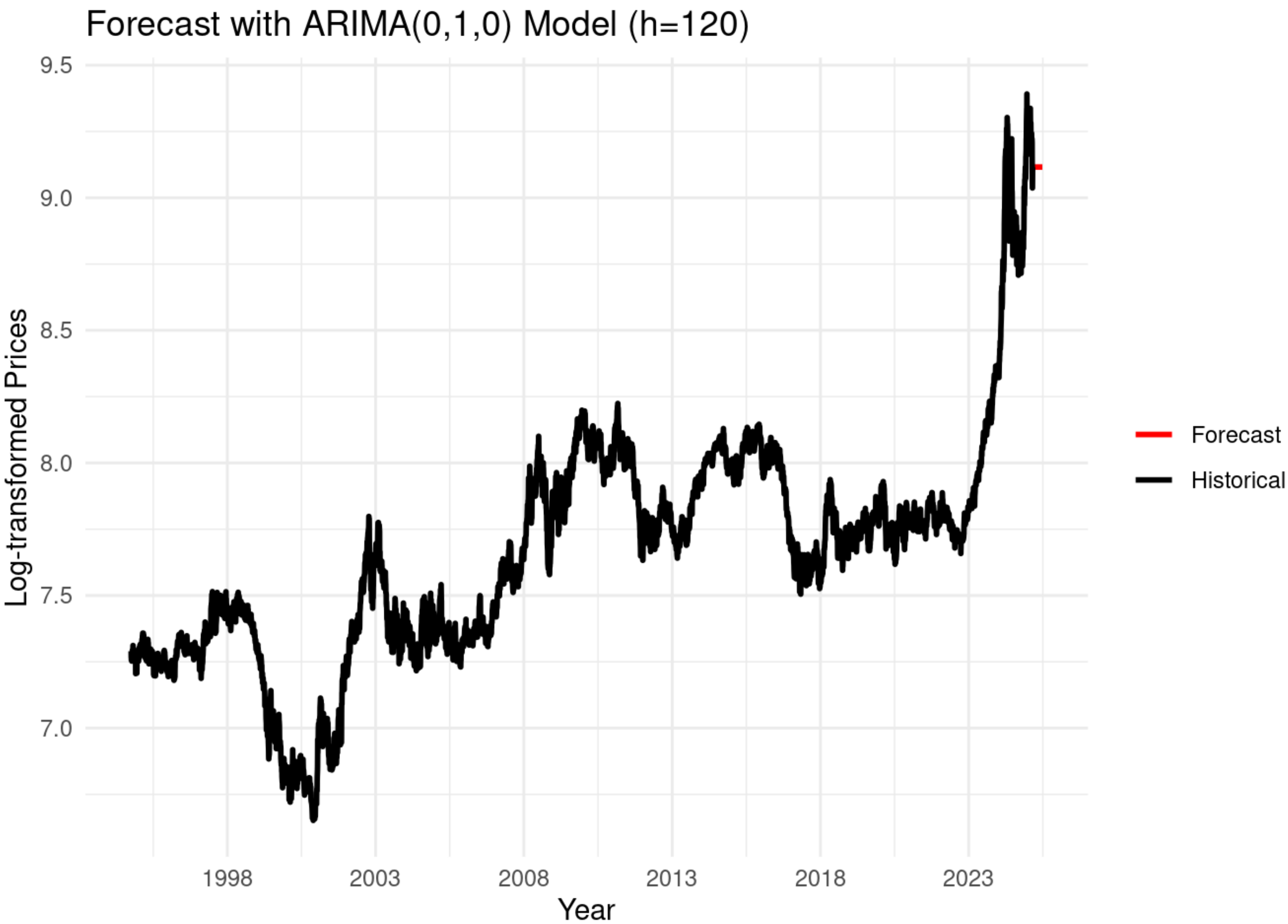
```
## Warning in adf.test(transformed_prices_1): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  transformed_prices_1
## Dickey-Fuller = -20.894, Lag order = 19, p-value = 0.01
## alternative hypothesis: stationary
```

```
# Initial Attempts at Forecasting (ARIMA(0,1,0))
arima_prices_1 <- arima(transformed_prices, order = c(0, 1, 0))
arima_prices_1
```

```
##
## Call:
## arima(x = transformed_prices, order = c(0, 1, 0))
##
##
## sigma^2 estimated as 0.0002877:  log likelihood = 20749.69,  aic = -41497.38
```

```
forecast_values_1 <- forecast(arima_prices_1, h = 120 )

forecast_values_numeric <- forecast_values_1$mean
forecast_dates <- seq(from = as.Date(full_data$date[nrow(full_data)]), by = "day", le
ngth.out = 120)

df <- data.frame(ds = c(as.Date(full_data$date), forecast_dates) , y = c(transformed_
prices, forecast_values_numeric), type = c(rep("Historical", length(transformed_price
s)), rep("Forecast", length(forecast_values_numeric))))

ggplot(df, aes(x = ds, y = y, color = type)) +
  geom_line(size = 1) +  # Historical and forecasted lines
  scale_x_date(date_breaks = "5 years", date_labels = "%Y") +
  labs(title = "Forecast with ARIMA(0,1,0) Model (h=120)", x = "Year", y = "Log-trans
formed Prices") +
  theme_minimal() +
  scale_color_manual(values = c("Historical" = "black", "Forecast" = "red")) +  # Dif
ferent colors for historical and forecast
  theme(legend.title = element_blank())  # Remove legend title
```




```
print(forecast_values_1$mean[1:5]) # we notice all values are the same
```

```
## [1] 9.115992 9.115992 9.115992 9.115992 9.115992
```

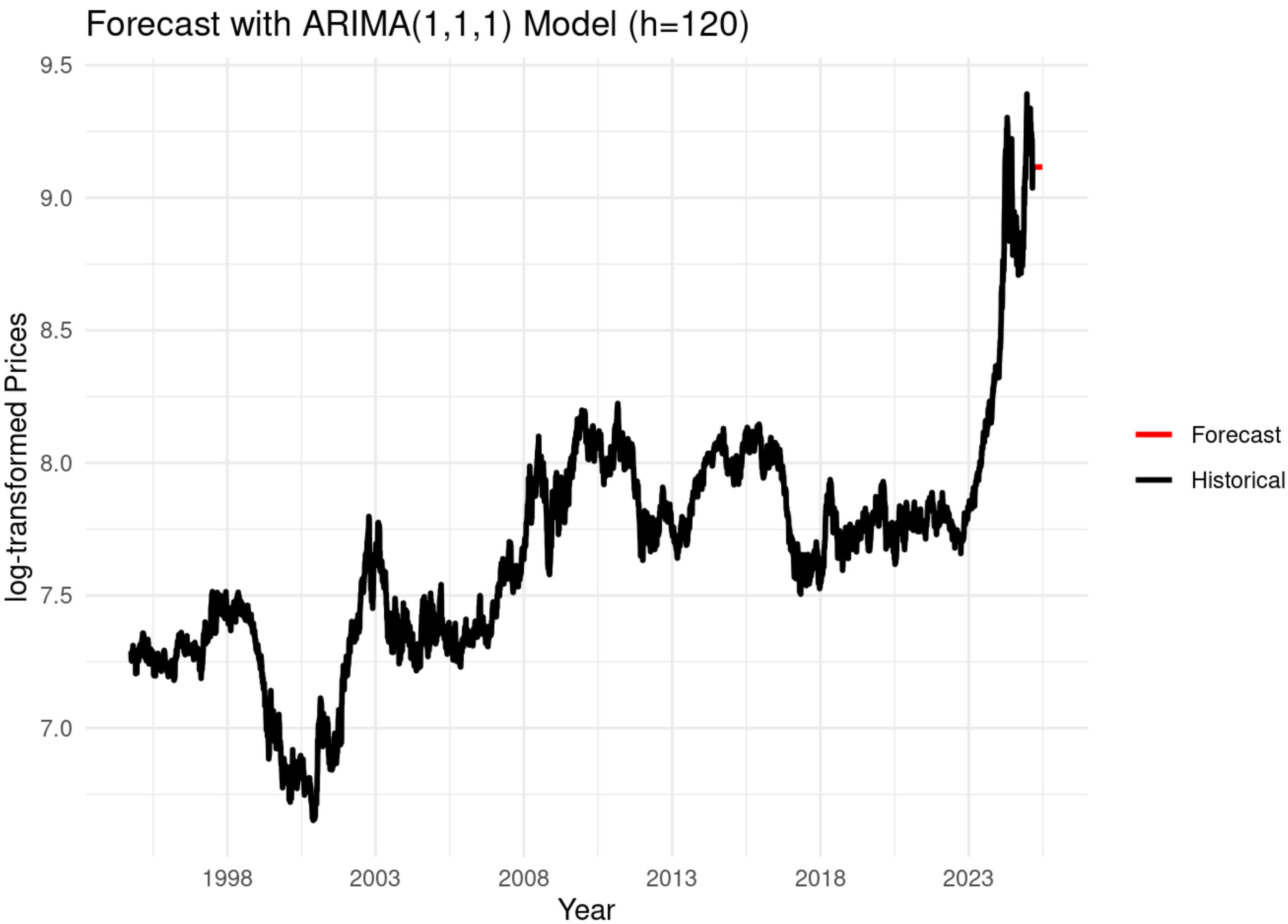
```
# Another Simple Attempt at Forecasting (ARIMA(1,1,1))

arima_prices_2 <- arima(transformed_prices, order = c(1, 1, 1))
arima_prices_2
```

```
##
## Call:
## arima(x = transformed_prices, order = c(1, 1, 1))
##
## Coefficients:
```

```
## Warning in sqrt(diag(x$var.coef)): NaNs produced
```

```
##          ar1     ma1
##       0.0163  0.0163
## s.e.     NaN     NaN
##
## sigma^2 estimated as 0.0002874:  log likelihood = 20753.85,  aic = -41501.7
```

```
forecast_values_2 <- forecast(arima_prices_2, h = 120 )

forecast_values_numeric_2 <- forecast_values_2$mean
forecast_dates_2 <- seq(from = as.Date(full_data$date[nrow(full_data)]), by = "day",
length.out = 120)

df <- data.frame(ds = c(as.Date(full_data$date), forecast_dates_2) , y = c(transforme
d_prices, forecast_values_numeric_2), type = c(rep("Historical", length(transformed_p
rices)), rep("Forecast", length(forecast_values_numeric_2))))

ggplot(df, aes(x = ds, y = y, color = type)) +
  geom_line(size = 1) +  # Historical and forecasted lines
  scale_x_date(date_breaks = "5 years", date_labels = "%Y") +
  labs(title = "Forecast with ARIMA(1,1,1) Model (h=120)", x = "Year", y = "log-trans
formed Prices") +
  theme_minimal() +
  scale_color_manual(values = c("Historical" = "black", "Forecast" = "red")) +  # Dif
ferent colors for historical and forecast
  theme(legend.title = element_blank())  # Remove legend title
```




```
print(forecast_values_2$mean[1:5]) # Again we notice all values are the same
```

```
## [1] 9.116002 9.116002 9.116002 9.116002 9.116002
```

```
# ACF of TAVG
library(TSA)
```

```
## Registered S3 methods overwritten by 'TSA':
##   method       from
##   fitted.Arima forecast
##   plot.Arima   forecast
```

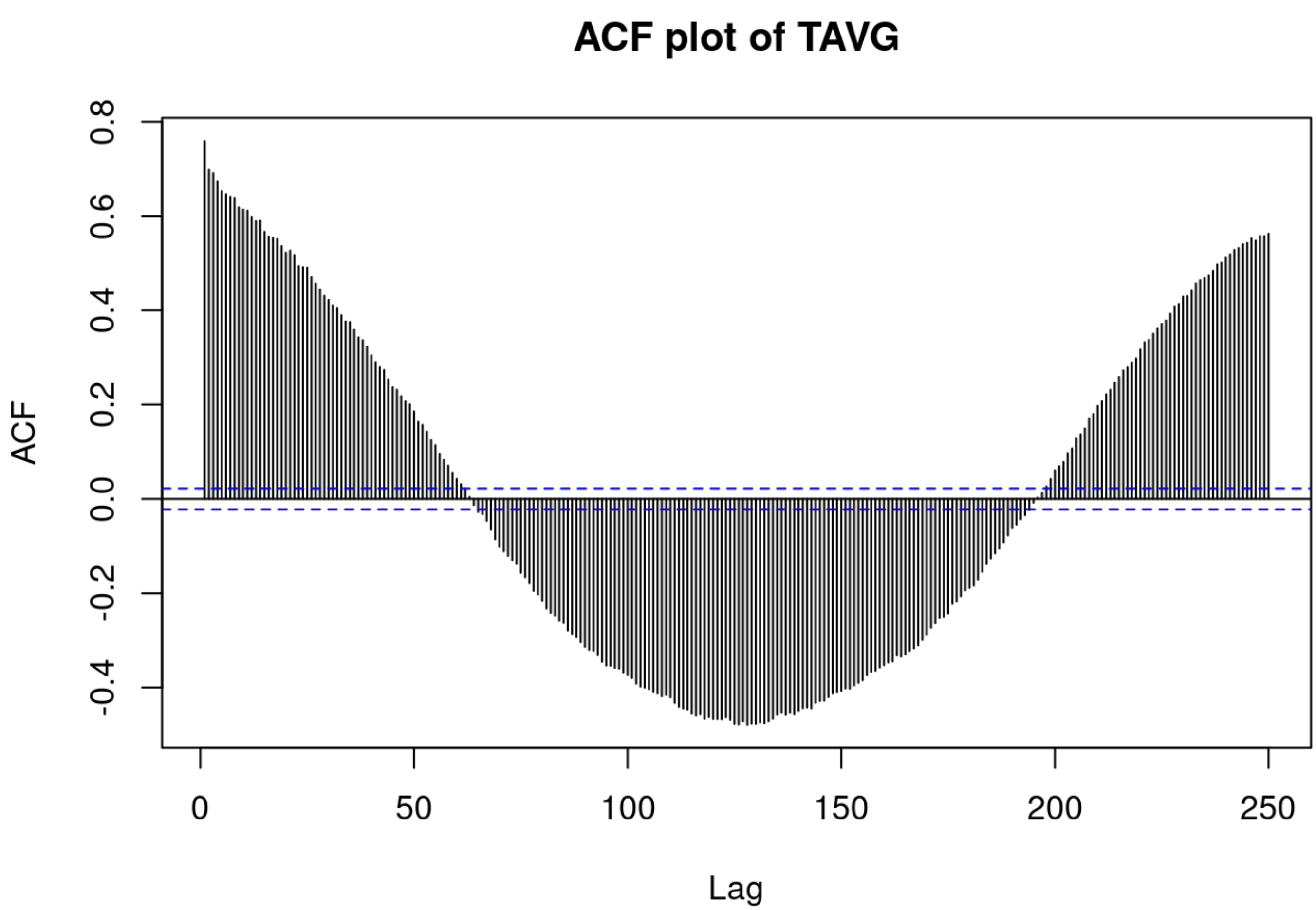
```
##
## Attaching package: 'TSA'
```

```
## The following object is masked from 'package:readr':
##
##     spec
```

```
## The following objects are masked from 'package:stats':
##
##     acf, arima
```

```
## The following object is masked from 'package:utils':
##
##     tar
```

```
acf(ts(full_data[[4]]), lag.max = 250, main = "ACF plot of TAVG")
```

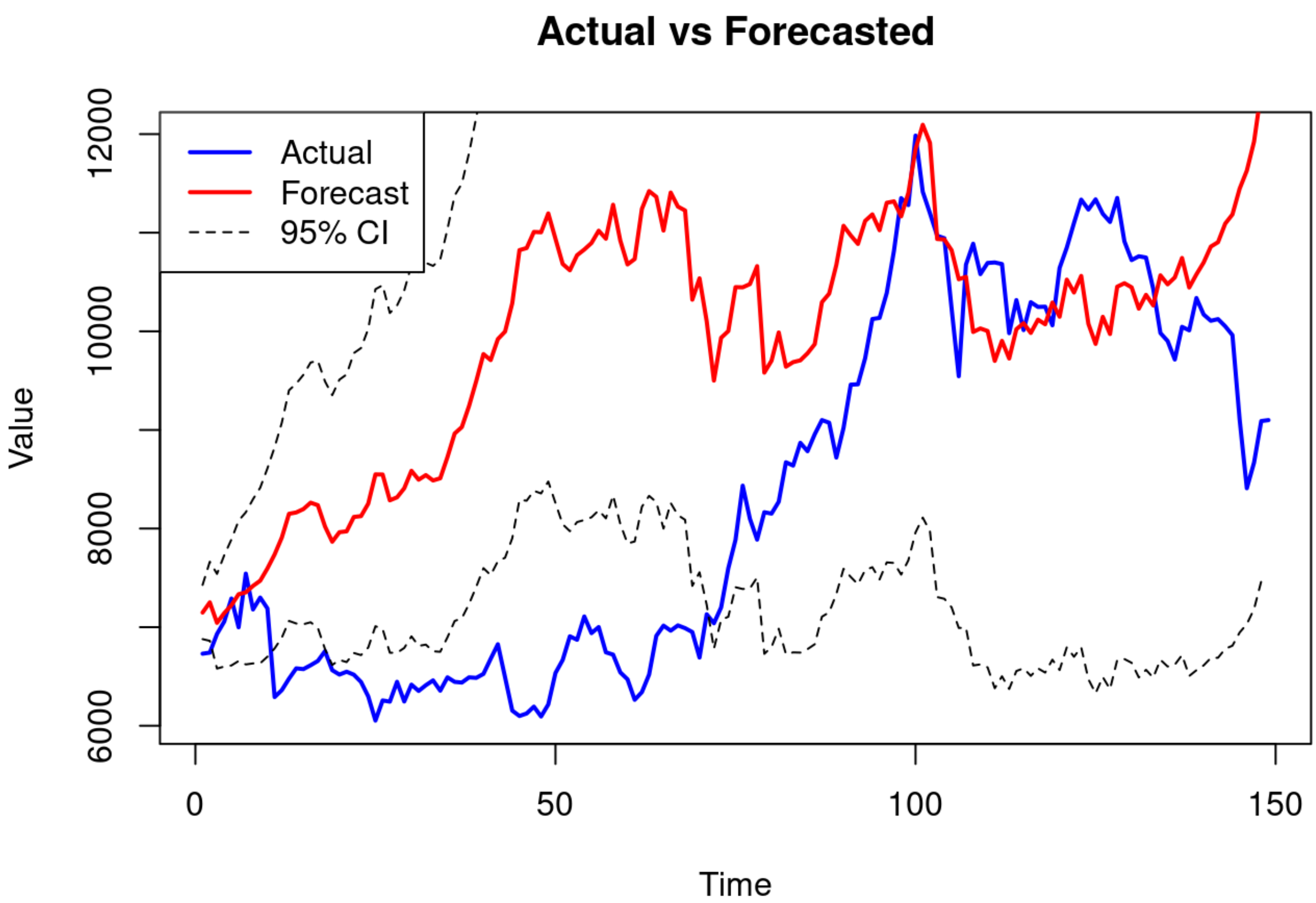


```
#### Splitting Dataset into Train and Test ####

# Define the split date (e.g., 2020-01-01)
split_date <- as.Date("2024-08-01")

# Split the dataset
train_df <- full_data %>% filter(date < split_date)
test_df  <- full_data %>% filter(date >= split_date)

train_price = log(ts(train_df[2]))
test_price = ts(test_df[2])
```

```
#### Predicting External Regressors ####

# 1. PRCP
forecast_PRCP = function(data, forecast_steps) {
  PRCP_ts <- ts(data)

  # Model decided by looking at immediate ACF cut off and slower PACF     decay, the 
series is stationary by adfuller test
  model_PRCP <- Arima(PRCP_ts,
                      order = c(8,0,0),
                      seasonal = list(order = c(1,1,1), period = 135),
                      method = 'CSS')

  # Generate forecasts
  forecasted_values <- forecast(model_PRCP, h = forecast_steps)$mean

  return(forecasted_values)
}

# 2. TAVG

forecast_TAVG <- function(data, forecast_steps, K = 5) {
  K = 5
  TAVG_ts = ts(data)

  # Generate Fourier terms to capture periodicity
  fourier_terms <- fourier(ts(TAVG_ts, frequency = 365), K = K)
  fourier_model <- Arima(TAVG_ts, order = c(1,0,1), xreg = fourier_terms)

  # Create Fourier terms for forecasting
  new_fourier <- fourier(ts(TAVG_ts, frequency = 365), K = K, h = forecast_steps)

  # Generate forecasts
  forecasted_values <- forecast(fourier_model, xreg = new_fourier)$mean
```

```
  return(forecasted_values)
}

# 3. Sentiment_score

# Beta distribution approach for sentiment forecasting
forecast_sentiment <- function(n_forecasts) {
  Sentiment_Data = ts(sentiment_data[[2]])
  mean_value = mean(Sentiment_Data)
  var_value = var(Sentiment_Data)

  mean_scaled <- (mean_value + 1) / 2
  var_scaled <- var_value / 4

  # Calculate parameters for Beta distribution
  shape1 <- mean_scaled * (mean_scaled * (1 - mean_scaled) / var_scaled - 1)
  shape2 <- (1 - mean_scaled) * (mean_scaled * (1 - mean_scaled) / var_scaled - 1)

  # Generate values from Beta distribution
  sentiment_values <- rbeta(n_forecasts, shape1, shape2)

  # Convert back to [-1,1] scale
  sentiment_values <- sentiment_values * 2 - 1

  return(sentiment_values)
}

# Reusable function to combine all three predictions
create_exog_matrix <- function(df, forecast_steps) {
  # Generate forecasts for each variable
  forecasted_PRCP <- forecast_PRCP(df[[3]], forecast_steps)
  forecasted_TAVG <- forecast_TAVG(df[[4]], forecast_steps)
  forecasted_sentiment <- forecast_sentiment(forecast_steps)

  # Combine into exogenous variable matrix
  exog_matrix <- cbind(forecasted_PRCP, forecasted_TAVG, forecasted_sentiment)
  colnames(exog_matrix) <- c("PRCP", "TAVG", "sentiment_score")

  return(exog_matrix)
}
```

```
#### Price Predictions and Plotting ####

exog_train <- as.matrix(train_df[, c(3, 4, 5)])
exog_test <- create_exog_matrix(train_df, length(test_price))

# Fit ARIMA model with three exogenous variables
model_sarima <- Arima(train_price, order = c(1,1,0), seasonal = list(order = c(1,1,
0), period = 135),
               xreg = exog_train, method = 'CSS')

forecasted = forecast(model_sarima, length(test_price), xreg=exog_test)

# Get the indices of the test data
test_index <- 1:length(test_price)

# Get the corresponding indices of the forecast (should match test indices)
forecast_index <- test_index # Since predictions are for test itself

# Determine y-axis range
y_range <- range(test_price, exp(forecasted$mean), exp(forecasted$lower), exp(forecas
ted$upper))


# Plot the test data
plot(test_index, test_price, col = "blue", lwd = 2, type = "l",
     main = "Actual vs Forecasted", xlab = "Time", ylab = "Value")

# Add forecasted values at the same indices as test
lines(forecast_index, exp(forecasted$mean), col = "red", lwd = 2)

# Add confidence intervals (optional)
lines(forecast_index, exp(forecasted$lower[,2]), col = "black", lty = "dashed")
lines(forecast_index, exp(forecasted$upper[,2]), col = "black", lty = "dashed")

# Add a legend
legend("topleft", legend = c("Actual", "Forecast", "95% CI"),
       col = c("blue", "red", "black"), lty = c(1,1,2), lwd = c(2,2,1))
```




```
#### Metrics for SARIMA ####
pred = as.numeric(exp(forecasted$mean))
actual = as.numeric(test_price)

# Calculate RMSE
rmse <- sqrt(mean((actual - pred)^2))
print(paste("RMSE:", rmse))
```

```
## [1] "RMSE: 2341.94167938522"
```

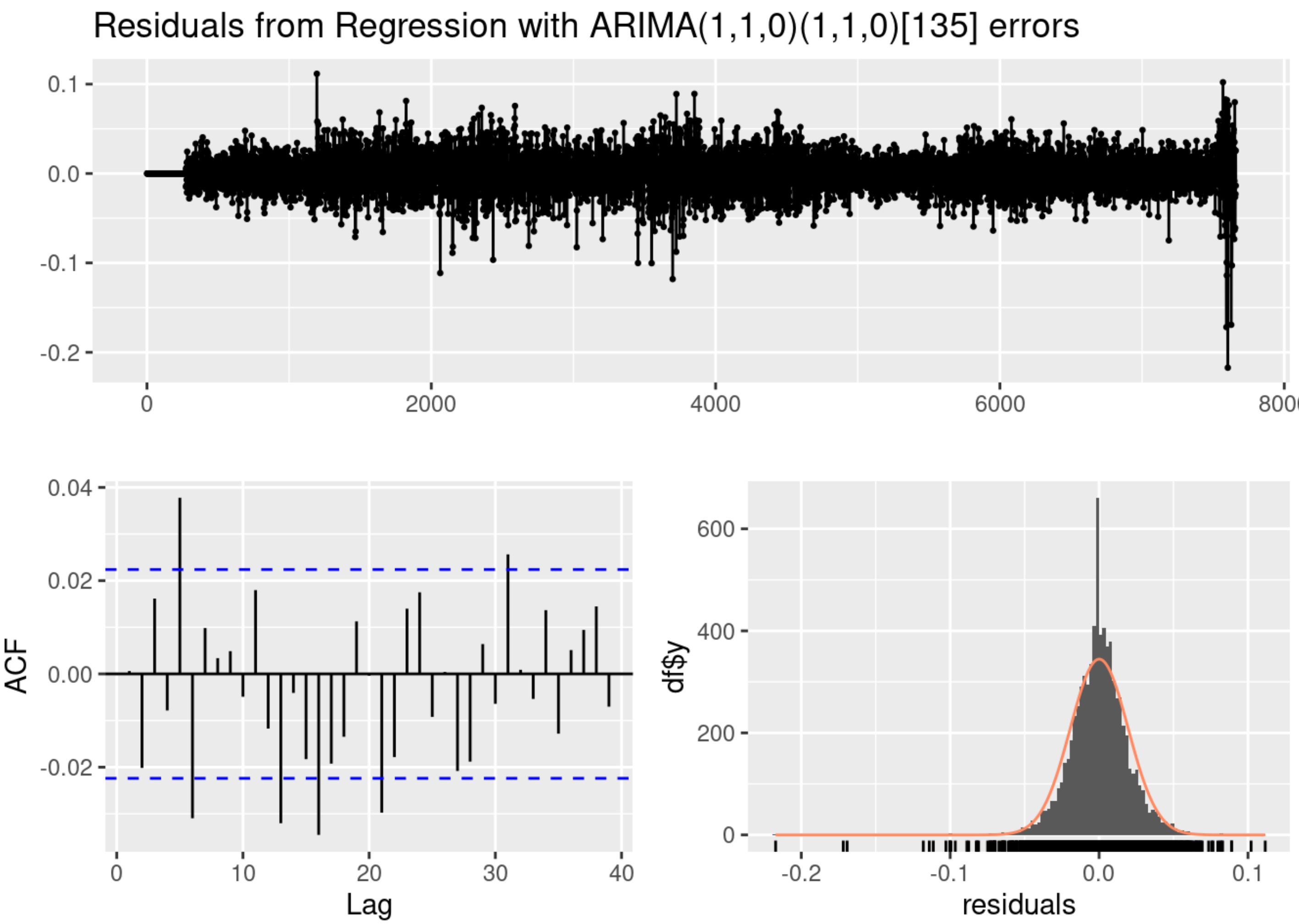
```
# Calculate MAPE
mape <- mean(abs((actual - pred) / actual)) * 100
print(paste("MAPE:", mape))
```

```
## [1] "MAPE: 25.676080506698"
```

```
# Calculate MAE
mae <- mean(abs(actual - pred))
print(paste("MAE:", mae))
```

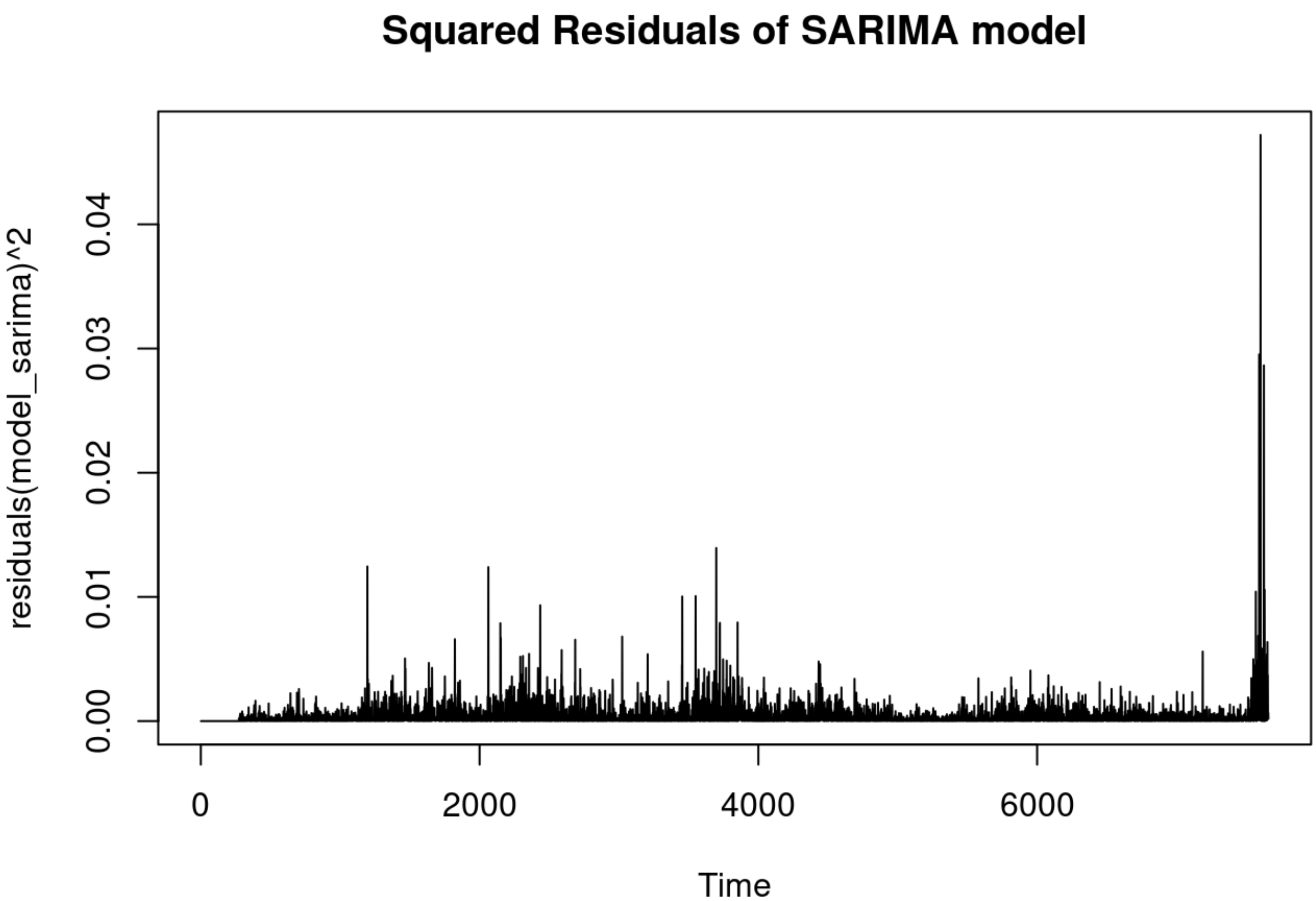
```
## [1] "MAE: 1838.44152916104"
```

```
#### Residuals of SARIMA ####
checkresiduals(model_sarima)
```

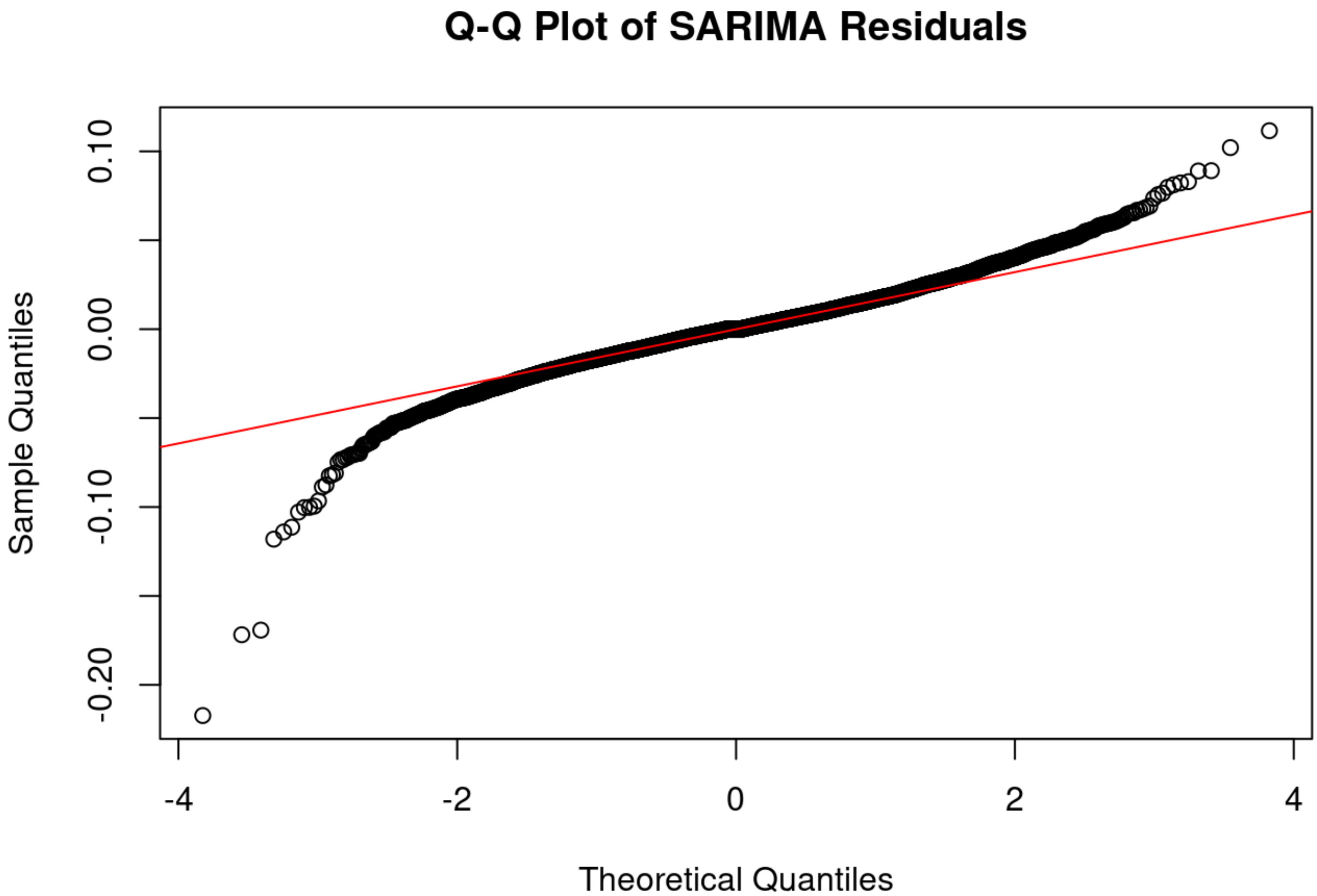



```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,1,0)(1,1,0)[135] errors
## Q* = 25.078, df = 8, p-value = 0.001508
##
## Model df: 2.   Total lags used: 10
```

```
plot(residuals(model_sarima)^2, main = "Squared Residuals of SARIMA model")
```




```
qqnorm(residuals(model_sarima), main = "Q-Q Plot of SARIMA Residuals")
qqline(residuals(model_sarima), col="red")
```




```
#### GARCH model and adjusted prediction ####

# Extract residuals from ARIMA model
library(rugarch)
```

```
## Loading required package: parallel
```

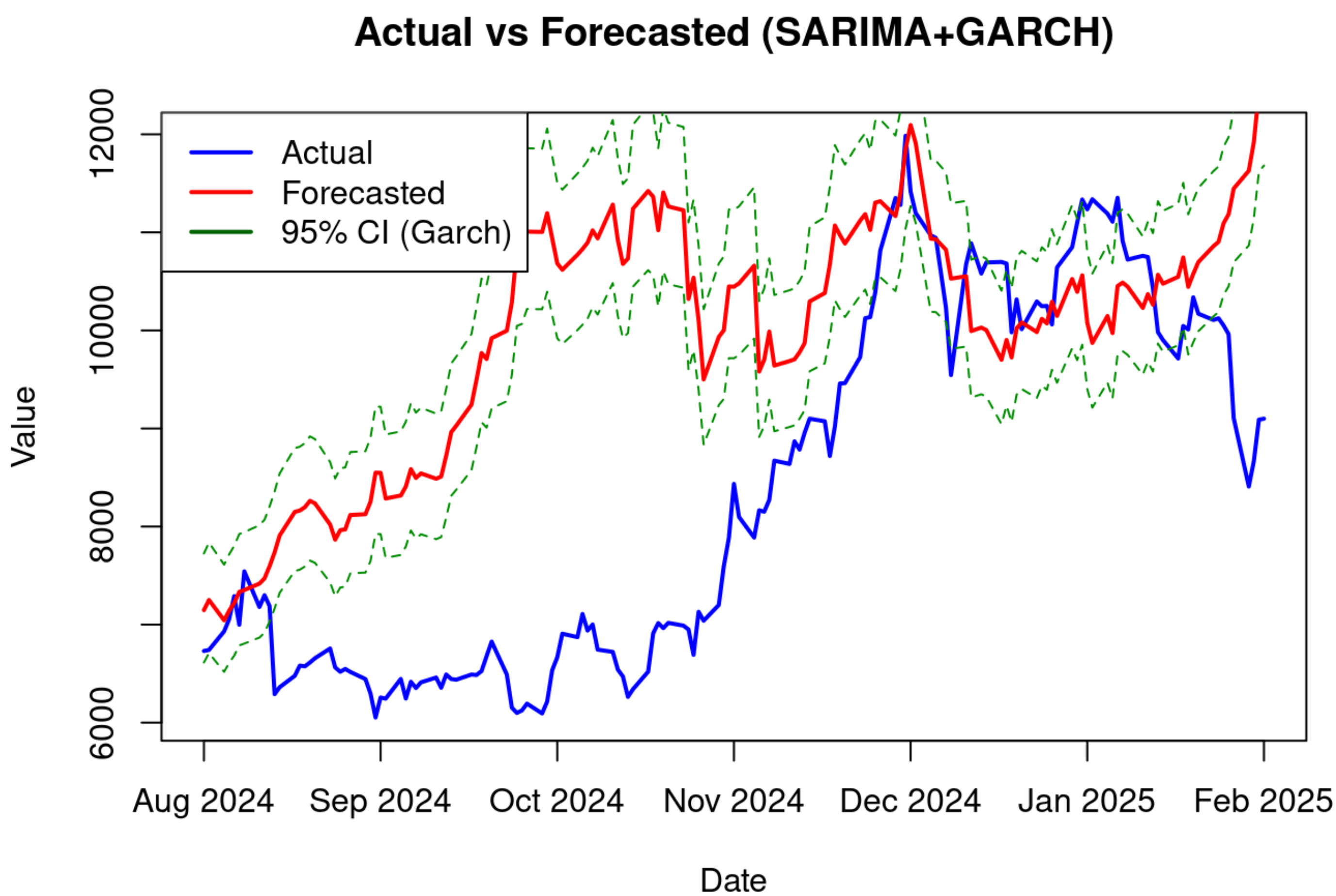
```
##
## Attaching package: 'rugarch'
```

```
## The following object is masked from 'package:purrr':
##
##     reduce
```

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```
residuals_sarima <- residuals(model_sarima)

# Define GARCH(1,1) model specification
garch_spec <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,
1)),
                         mean.model = list(armaOrder = c(0,0), include.mean = FALSE),
# No mean needed
                         distribution.model = "norm")

# Fit GARCH model to ARIMA residuals
garch_fit = ugarchfit(spec = garch_spec, data = residuals_sarima)

sarima_forecast <- forecast(model_sarima, length(test_price), xreg = exog_test)

# Forecast GARCH residual variance
garch_forecast <- ugarchforecast(garch_fit, n.ahead = length(test_price))

# Extract standard deviation predictions (volatility)
garch_adjustment <- sigma(garch_forecast)

dates <- test_df$date

# Create the plot without x-axis labels initially
plot(dates, test_price, col = "blue", lwd = 2, type = "l",
     main = "Actual vs Forecasted (SARIMA+GARCH)",
     xlab = "", ylab = "Value", xaxt = "n") # Suppress default x-axis

# Add forecasted values
# lines(dates, final_forecast, col = "pink", lwd = 2)
lines(dates, exp(sarima_forecast$mean), col = "red", lwd = 2)

# Add confidence intervals
lines(dates, exp(sarima_forecast$mean - 1.96 * garch_adjustment), col = "green4", lty
= "dashed")
lines(dates, exp(sarima_forecast$mean + 1.96 * garch_adjustment), col = "green4", lty
= "dashed")

# Create custom x-axis with month names and years
# Calculate appropriate number of ticks based on date range
date_range <- as.numeric(diff(range(dates)))
num_ticks <- min(7, max(3, floor(date_range/30))) # Adjust based on date range
```

```r
# Create evenly spaced tick positions
date_ticks <- seq(min(dates), max(dates), length.out = num_ticks)

# Add the custom axis with Month Year format
axis.Date(1, at = date_ticks, format = "%b %Y")

# Add proper x-axis label
mtext("Date", side = 1, line = 3)

# Add legend
legend("topleft", legend = c("Actual",  "Forecasted", "95% CI (Garch)"),
       col = c("blue", "red", "darkgreen"), lty = c(1, 1, 1, 2), lwd = c(2, 2, 2, 1))
```




```r
#### Residuals for SARIMA + GARCH ####

library(ggplot2)
library(ggpubr)
```

```
##
## Attaching package: 'ggpubr'
```

```
## The following object is masked from 'package:forecast':
##
##     gghistogram
```

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

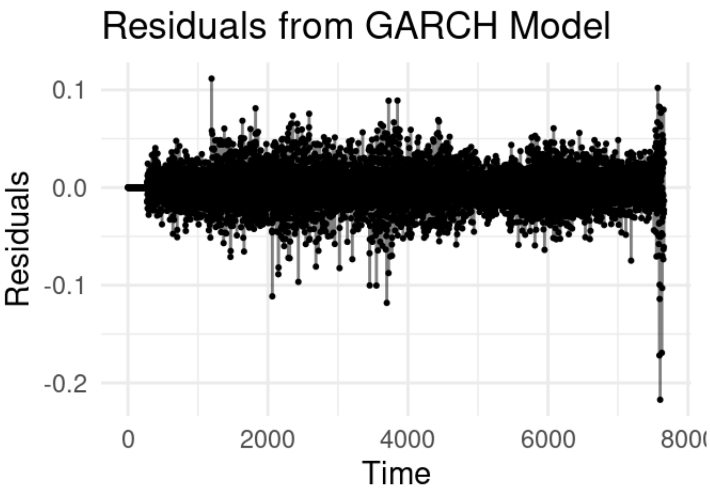
```
## The following object is masked from 'package:dplyr':
##
##     combine
```

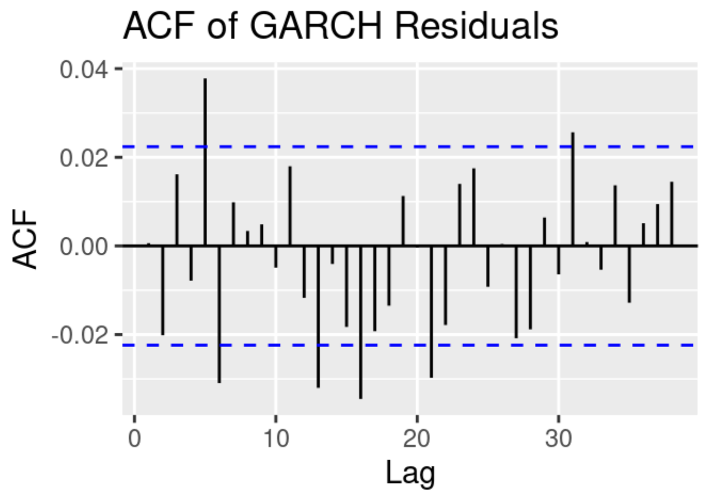
```
library(forecast)

# Assuming garch_residuals contains residuals from your GARCH model
garch_residuals <- residuals(garch_fit)

# Convert to data frame
resid_df <- data.frame(
  Time = 1:length(garch_residuals),
  Residuals = garch_residuals
)

# 1️⃣ **Residuals Time Series Plot**
p1 <- ggplot(resid_df, aes(x = Time, y = Residuals)) +
  geom_point(color = "black", size = 0.4) +
  geom_line(color = "black", alpha = 0.5) +
  theme_minimal() +
  ggtitle("Residuals from GARCH Model")

# 2️⃣ **Autocorrelation Function (ACF) Plot**
p2 <- ggAcf(garch_residuals, main = "ACF of GARCH Residuals")
```

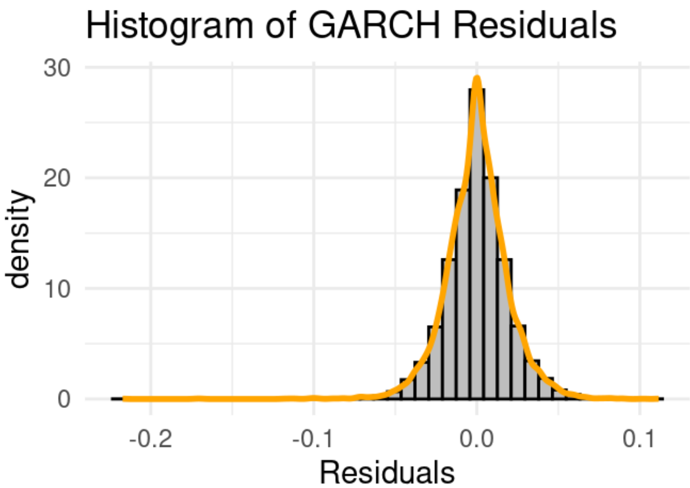
```
## Warning in ggplot2::geom_segment(lineend = "butt", ...): Ignoring unknown
## parameters: `main`
```

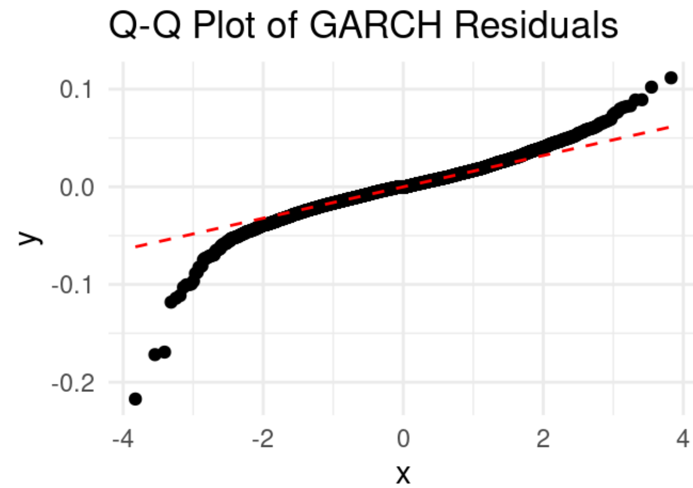
```
# 3️⃣ **Histogram with Density Plot**
p3 <- ggplot(resid_df, aes(x = Residuals)) +
  geom_histogram(aes(y = ..density..), bins = 40, fill = "gray", color = "black") +
  geom_density(color = "orange", size = 1) +
  theme_minimal() +
  ggtitle("Histogram of GARCH Residuals")

# 4️⃣ **Q-Q Plot**
p4 <- ggplot(resid_df, aes(sample = Residuals)) +
  stat_qq() +
  stat_qq_line(color = "red", linetype = "dashed") +
  theme_minimal() +
  ggtitle("Q-Q Plot of GARCH Residuals")

# 🏗️ **Arrange all plots in a single grid**
grid.arrange(p1, p2, p3, p4, ncol = 2)
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## ℹ Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```


Residuals from GARCH Model
Residuals
0.1
0.0
-0.1
-0.2
0
2000
4000
6000
Time


ACF of GARCH Residuals
ACF
0.04
0.02
0.00
-0.02
0
10
20
30
Lag


Histogram of GARCH Residuals
density
30
20
10
0
-0.2
-0.1
0.0
0.1
Residuals


Q-Q Plot of GARCH Residuals
y
0.1
0.0
-0.1
-0.2
-4
-2
0
2
4
x

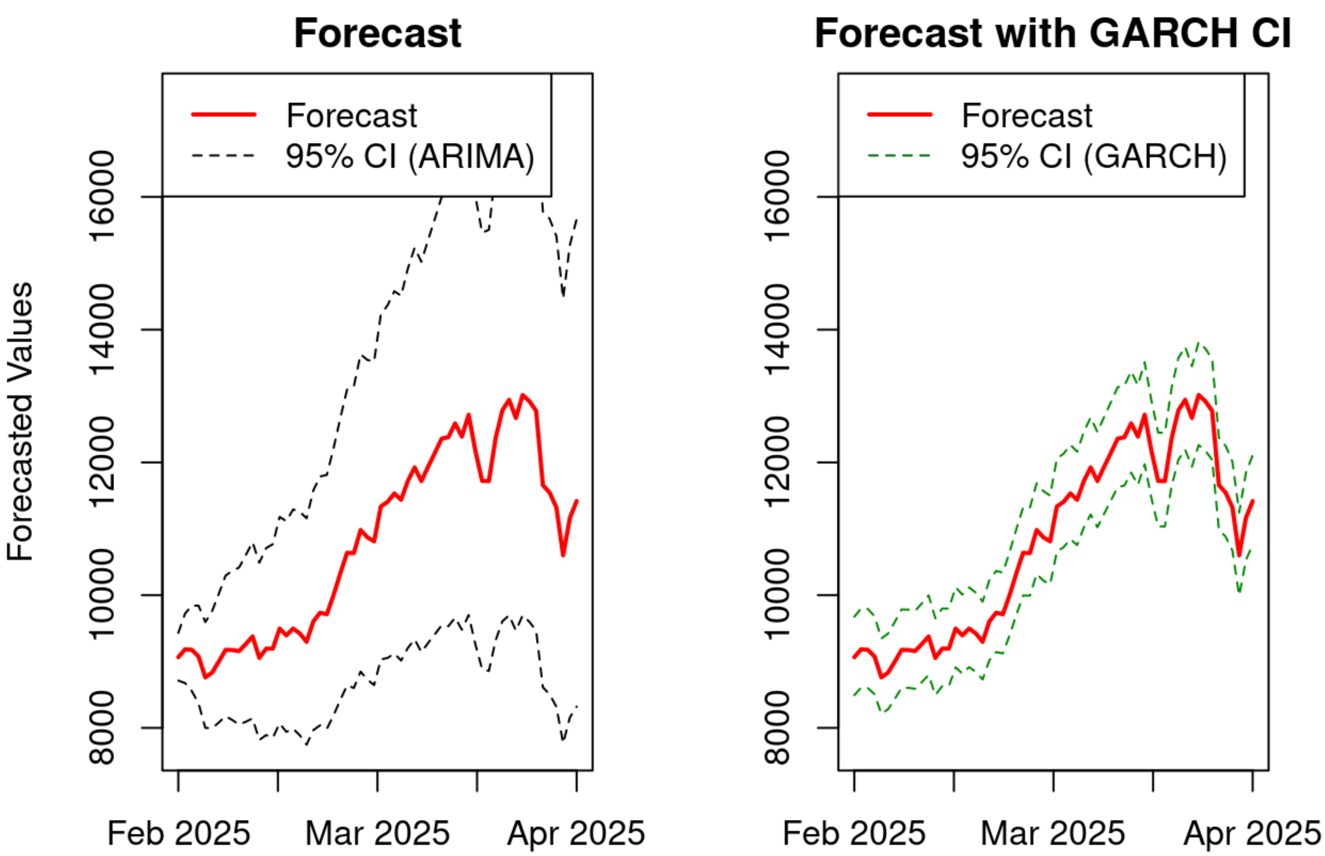
```
#### Final forecasting using full data ####

# Prepare the data
forecast_steps_full = 60
price_full = log(ts(full_data$price))
exog_full = as.matrix(full_data[, 3:5])
exog_forecast = create_exog_matrix(full_data, forecast_steps_full)

# Fit both SARIMAX and GARCH models
sarima_model_full = Arima(price_full, order = c(1,1,0), seasonal = list(order = c(1,
1,0), period = 135),
                          xreg = exog_full, method = 'CSS')

residuals_full = residuals(sarima_model_full)

garch_spec_full <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(
1,1)),
                         mean.model = list(armaOrder = c(0,0), include.mean = FALSE),
# No mean needed
                         distribution.model = "norm")

# Forecast both and combine
garch_fit_full = ugarchfit(spec = garch_spec, data = residuals_full)

sarima_forecast_full <- forecast(sarima_model_full, forecast_steps_full, xreg = exog_
forecast)

garch_forecast_full <- ugarchforecast(garch_fit_full, n.ahead = forecast_steps_full)

garch_adjustment_full <- sigma(garch_forecast_full)

print(exp(sarima_forecast_full$mean[1:5]))
```

```
## [1] 9065.172 9183.954 9176.442 9074.927 8761.109
```

```
#### Plotting the predictions with both SARIMA CI and GARCH CI ####

# Set up side-by-side plotting and adjust margins
par(mfrow = c(1, 2), mar = c(4, 4, 2, 2), oma = c(0, 0, 2, 2))  # More right margin

# Extract forecasted dates
last_date <- max(full_data$date)
dates <- seq(from = last_date, by = "day", length.out = length(sarima_forecast_full$m
ean))
```

```
# Determine common y-axis range across both plots
y_min <- min(exp(sarima_forecast_full$mean - 1.96 * garch_adjustment_full),
             exp(sarima_forecast_full$lower[,2]))

y_max <- max(exp(sarima_forecast_full$mean + 1.96 * garch_adjustment_full),
             exp(sarima_forecast_full$upper[,2]))

# Ensure the same x-axis range
x_min <- min(dates)
x_max <- max(dates)

#### Plot 1: Forecast with Standard ARIMA 95% CI ####
plot(dates, exp(sarima_forecast_full$mean), type = "l", col = "red", lwd = 2,
     xaxt = "n", xlab = "", ylab = "Forecasted Values", main = "Forecast",
     ylim = c(y_min, y_max), xlim = c(x_min, x_max))  # Force same x-scale

# Add standard ARIMA confidence intervals
lines(dates, exp(sarima_forecast_full$lower[,2]), col = "black", lty = "dashed")
lines(dates, exp(sarima_forecast_full$upper[,2]), col = "black", lty = "dashed")

# Create shared custom x-axis
date_ticks <- seq(x_min, x_max, length.out = 5)
axis.Date(1, at = date_ticks, format = "%b %Y")

# Add legend
legend("topleft", legend = c("Forecast", "95% CI (ARIMA)"),
       col = c("red", "black"), lty = c(1, 2), lwd = c(2, 1))


#### Plot 2: Forecast with GARCH-Adjusted Confidence Intervals ####
plot(dates, exp(sarima_forecast_full$mean), type = "l", col = "red", lwd = 2,
     xaxt = "n", xlab = "", ylab = "", main = "Forecast with GARCH CI",
     ylim = c(y_min, y_max), xlim = c(x_min, x_max))  # Ensure same x-scale

# Add GARCH confidence intervals
lines(dates, exp(sarima_forecast_full$mean - 1.96 * garch_adjustment_full), col = "gr
een4", lty = "dashed")
lines(dates, exp(sarima_forecast_full$mean + 1.96 * garch_adjustment_full), col = "gr
een4", lty = "dashed")

# Use shared x-axis
axis.Date(1, at = date_ticks, format = "%b %Y")

# Add legend
legend("topleft", legend = c("Forecast", "95% CI (GARCH)"),
       col = c("red", "darkgreen"), lty = c(1, 2), lwd = c(2, 1))
```

```
# Add a single x-axis label below both plots
mtext("Date", side = 1, line = 2.5, outer = TRUE)
```




```
# Reset plot layout
par(mfrow = c(1, 1))
```