

# STA457 Project

Xing Yu Wang

2025-03-29

```
library(dplyr)
library(tidyverse)
library(readr)
library(lubridate)
library(forecast)

## Warning: package 'forecast' was built under R version 4.3.3
library(astsa)

## Warning: package 'astsa' was built under R version 4.3.3
library(tseries)

## Warning: package 'tseries' was built under R version 4.3.3
library(mgcv)
library(Metrics)

## Warning: package 'Metrics' was built under R version 4.3.3
library(ggplot2)
library(xgboost)

## Warning: package 'xgboost' was built under R version 4.3.3
# library(XGBClassifier)
library(rugarch)

## Warning: package 'rugarch' was built under R version 4.3.3
```

## 1. EDA

```
price = read.csv("./Daily Prices_ICCO.csv")
weather = read.csv("./Ghana_data.csv")
USD_GHS_Historical_Data = read.csv("./USD_GHS Historical Data.csv")
```

### 1.1 Clean Data

```
weather <- weather |> dplyr::select(
  DATE, TAVG)
exchangerate <- USD_GHS_Historical_Data |> dplyr::select(
  Date, Price)

colnames(price)[colnames(price) == 'ICCO.daily.price..US..tonne.'] <- 'Daily_Price'
colnames(weather)[colnames(weather) == 'DATE'] <- 'Date'
```

```
colnames(weather)[colnames(weather) == 'TAVG'] <- 'Avg_Temp'
colnames(exchangerate)[colnames(exchangerate) == 'Price'] <- 'exchange_rate'
```

## 1.2 Check duplicated values

```
price |> group_by(Date) |> filter(n() > 1) |> ungroup()
```

```
## # A tibble: 8 x 2
##   Date       Daily_Price
##   <chr>      <chr>
## 1 31/01/2024 4,798.20
## 2 31/01/2024 10,888.05
## 3 30/01/2024 4,775.17
## 4 30/01/2024 10,676.42
## 5 09/01/2024 4,171.24
## 6 09/01/2024 4,171.24
## 7 15/12/2023 4,272.15
## 8 15/12/2023 4,272.15
```

```
price <- price |> filter(!(Date == "31/01/2024" & Daily_Price == "10,888.05"))
price <- price |> filter(!(Date == "30/01/2024" & Daily_Price == "10,676.42"))
price <- distinct(price)
```

## 1.3 Convert to Time Series Data

### 1.3.1 price Dataset

```
price$Date <- as.Date(price$Date, format="%d/%m/%Y")
price$Daily_Price <- as.numeric(gsub(",", "", price$Daily_Price))
price_month <- price |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(month_Price = mean(Daily_Price, na.rm = TRUE)) |> ungroup()
```

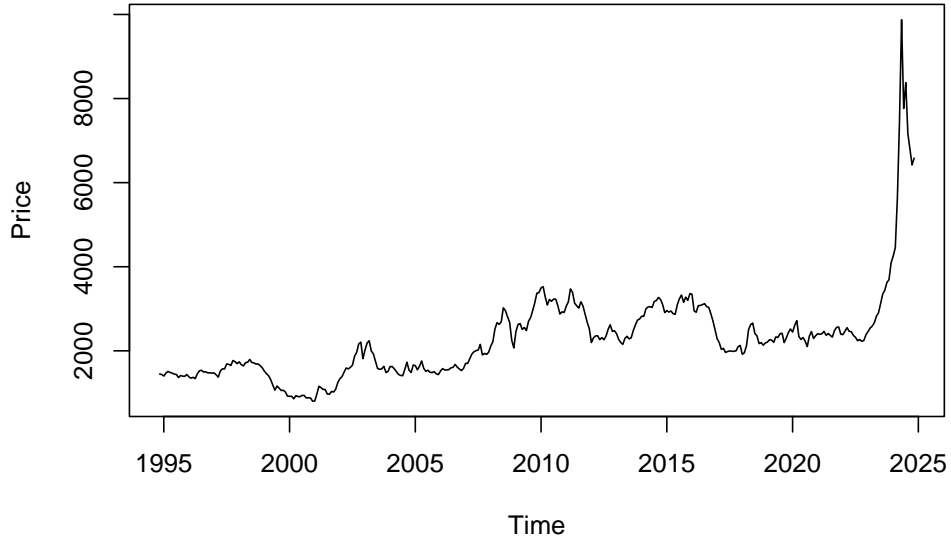
```
summary(price)
```

```
##           Date           Daily_Price
##  Min.   :1994-10-03   Min.    : 774.1
##  1st Qu.:2002-05-16   1st Qu.: 1557.8
##  Median :2009-12-17   Median : 2202.0
##  Mean   :2009-12-17   Mean    : 2350.1
##  3rd Qu.:2017-07-24   3rd Qu.: 2738.1
##  Max.   :2025-02-27   Max.    :11984.7
```

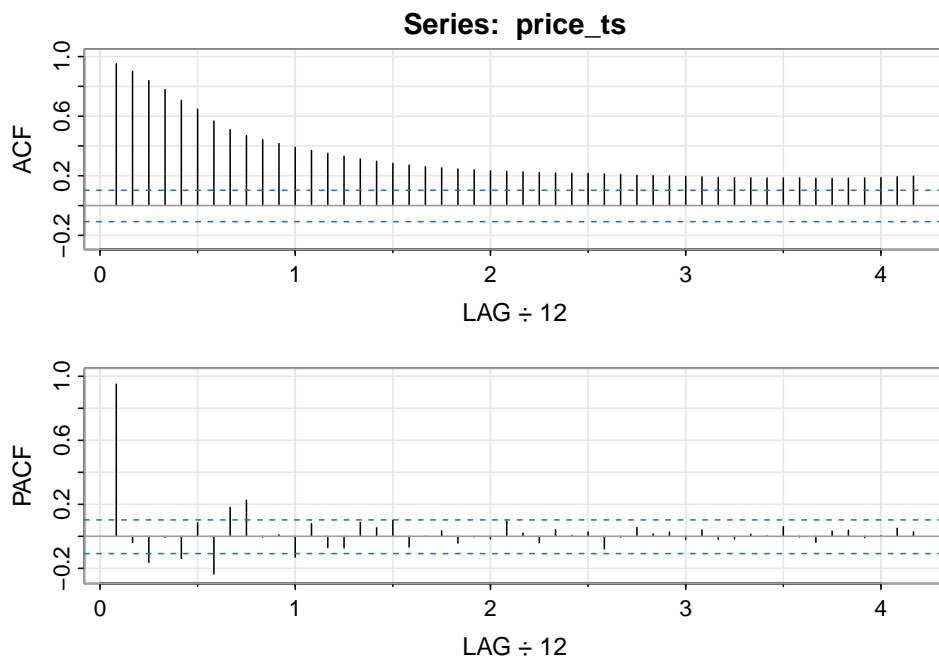
```
price_ts <- ts(price_month$month_Price, start = c(1994, 11), end = c(2024, 11), frequency = 12)
```

```
plot(price_ts, main="Monthly Price Time Series", ylab="Price", xlab="Time")
```

## Monthly Price Time Series



```
acf2(price_ts, 50)
```

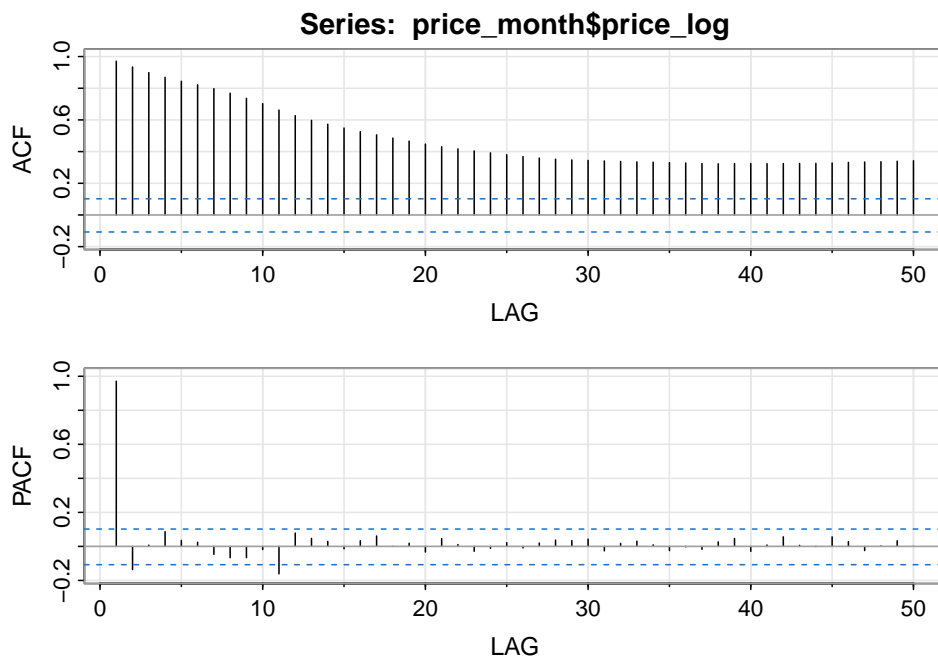


```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.95  0.90  0.84  0.78  0.71  0.65  0.57  0.51  0.47  0.44  0.42  0.39  0.37
## PACF  0.95 -0.04 -0.16 -0.01 -0.14  0.08 -0.24  0.18  0.23  0.00  0.01 -0.13  0.08
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF  0.35  0.33  0.31  0.30  0.28  0.27  0.26  0.25  0.25  0.24  0.23  0.23
## PACF -0.07 -0.07  0.09  0.05  0.10 -0.07  0.00  0.03 -0.04  0.00 -0.02  0.10
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF  0.23  0.22  0.22  0.22  0.22  0.21  0.21  0.20  0.20  0.20  0.20  0.19
## PACF  0.02 -0.04  0.04  0.00  0.03 -0.08  0.00  0.05  0.01  0.03 -0.02  0.04
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
## ACF  0.19  0.19  0.19  0.18  0.18  0.19  0.18  0.18  0.18  0.18  0.19  0.19
## PACF -0.02 -0.02  0.01  0.00  0.06  0.00 -0.04  0.03  0.04 -0.01  0.00  0.05
```

```
##      [,50]
## ACF    0.20
## PACF   0.03
ndiffs(price_ts)
```

```
## [1] 1
price_month$price_log <- log(price_month$month_Price)
adf.test(price_month$price_log)
```

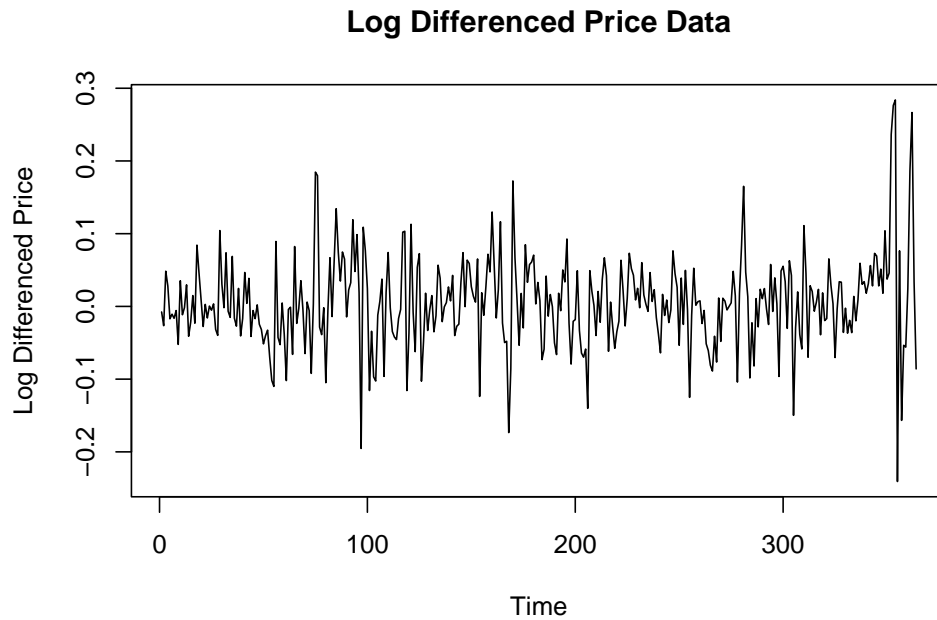
```
##
## Augmented Dickey-Fuller Test
##
## data: price_month$price_log
## Dickey-Fuller = -1.736, Lag order = 7, p-value = 0.6883
## alternative hypothesis: stationary
acf2(price_month$price_log, 50)
```



```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.97  0.93  0.90  0.87  0.84  0.82  0.80  0.77  0.74  0.70  0.66  0.63  0.60
## PACF  0.97 -0.14  0.01  0.09  0.04  0.02 -0.05 -0.07 -0.07 -0.02 -0.16  0.08  0.05
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF  0.57  0.55  0.53  0.51  0.49  0.47  0.45  0.43  0.42  0.40  0.39  0.38
## PACF  0.03 -0.01  0.03  0.06  0.00  0.02 -0.03  0.05  0.01 -0.03 -0.01  0.02
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF  0.37  0.36  0.35  0.35  0.34  0.34  0.34  0.34  0.33  0.33  0.33  0.32
## PACF -0.01  0.02  0.04  0.03  0.04 -0.03  0.02  0.03  0.01 -0.02  0.00 -0.02
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
## ACF  0.32  0.32  0.32  0.32  0.32  0.33  0.33  0.33  0.33  0.33  0.34  0.34
## PACF  0.03  0.05 -0.03  0.01  0.06  0.01  0.00  0.06  0.03 -0.02  0.00  0.03
##      [,50]
## ACF    0.34
## PACF   0.00
```

Hence, we want to difference the price data.

```
diff_log_price = diff(price_month$price_log)
ts.plot(diff_log_price, main = "Log Differenced Price Data", ylab = "Log Differenced Price")
```



```
adf.test(diff_log_price)
```

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

```
##
```

```
## Augmented Dickey-Fuller Test
```

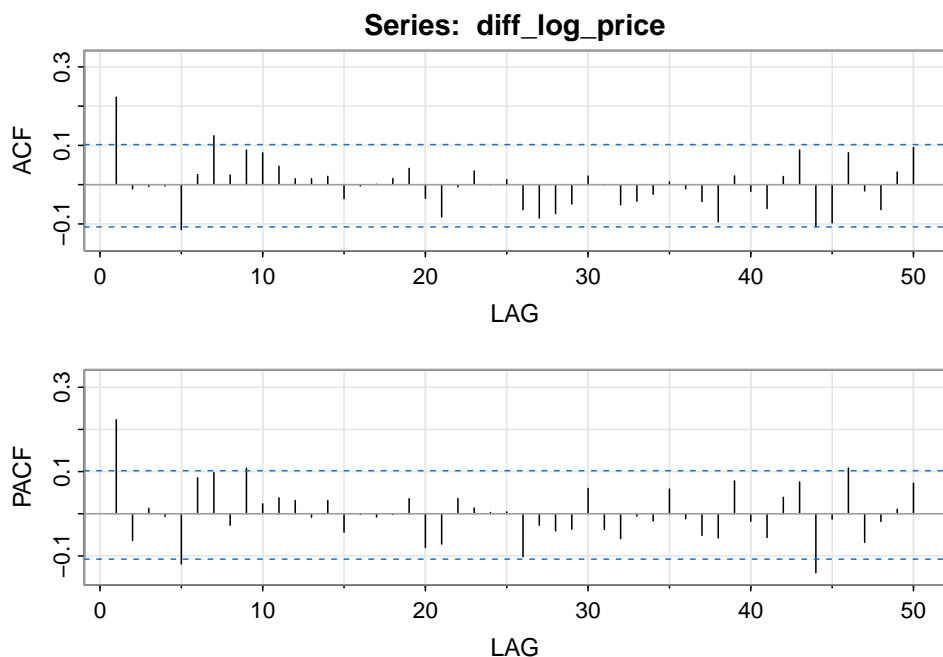
```
##
```

```
## data: diff_log_price
```

```
## Dickey-Fuller = -6.1385, Lag order = 7, p-value = 0.01
```

```
## alternative hypothesis: stationary
```

```
acf2(diff_log_price, 50)
```



```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.22 -0.01 0.00  0.00 -0.11 0.03 0.12  0.02 0.09  0.08  0.05  0.02  0.02
## PACF 0.22 -0.06 0.01 -0.01 -0.12 0.09 0.10 -0.03 0.11  0.02  0.04  0.03 -0.01
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.02 -0.04  0  0.00  0.02  0.04 -0.03 -0.08 -0.01  0.03  0  0.01
## PACF  0.03 -0.04  0 -0.01  0.00  0.04 -0.08 -0.07  0.04  0.01  0  0.00
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF  -0.06 -0.09 -0.07 -0.05  0.02  0.00 -0.05 -0.04 -0.02  0.01 -0.01 -0.04
## PACF -0.10 -0.03 -0.04 -0.04  0.06 -0.04 -0.06 -0.01 -0.02  0.06 -0.01 -0.05
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
## ACF  -0.09  0.02 -0.02 -0.06  0.02  0.09 -0.11 -0.10  0.08 -0.02 -0.06  0.03
## PACF -0.06  0.08 -0.02 -0.06  0.04  0.08 -0.14 -0.01  0.11 -0.07 -0.02  0.01
##      [,50]
## ACF   0.09
## PACF   0.07
```

### 1.3.2 ghana Dataset

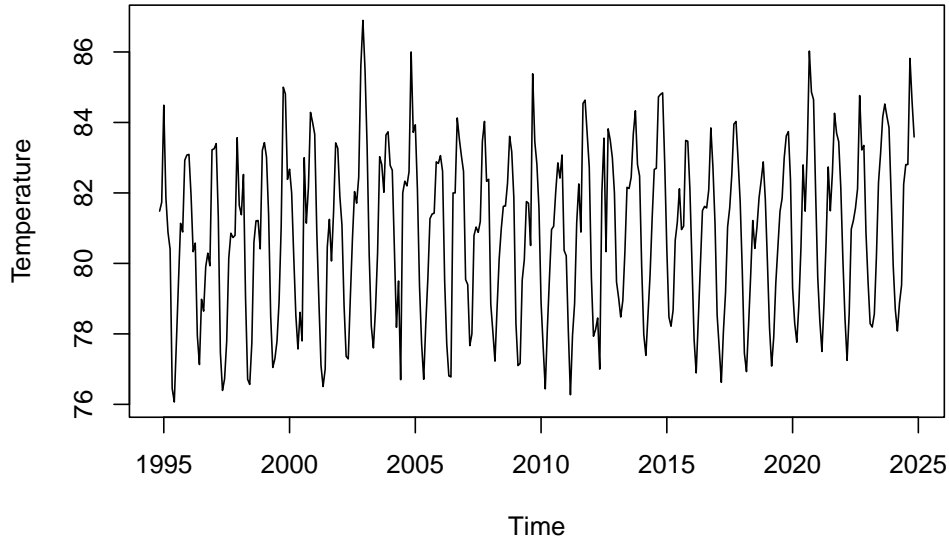
```
weather$Date <- as.Date(weather$Date)
weather$Avg_Temp <- as.numeric(gsub("", "", weather$Avg_Temp))
weather_month <- weather |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(Avg_Temp = mean(Avg_Temp, na.rm = TRUE)) |> ungroup()

summary(weather_month)
```

```
##      Time      Avg_Temp
## Min.   :1990-01-01  Min.   :76.07
## 1st Qu.:1998-09-23  1st Qu.:78.90
## Median :2007-07-16  Median :81.20
## Mean   :2007-06-22  Mean   :80.97
## 3rd Qu.:2016-03-08  3rd Qu.:82.82
## Max.   :2024-11-01  Max.   :86.90
```

```
weather_ts <- ts(weather_month$Avg_Temp, start = c(1994, 11), end = c(2024, 11), frequency = 12)
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")
```

### Monthly Average Temperature Time Series



### 1.3.3 exchange Data

```
exchangerate$Date <- as.Date(exchangerate$Date)
exchangerate$exchange_rate <- as.numeric(gsub(" ", "", exchangerate$exchange_rate))
rate_month <- exchangerate |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(exchange_rate = mean(exchange_rate, na.rm = TRUE)) |> ungroup()
```

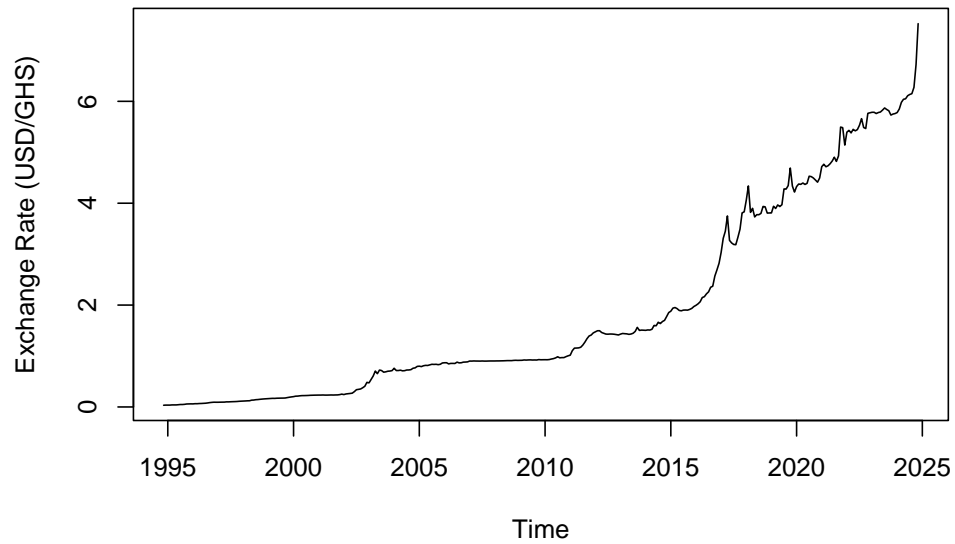
```
summary(exchangerate)
```

```
##      Date      exchange_rate
## Min.   :1992-03-01   Min.    : 0.0338
## 1st Qu.:2000-06-01   1st Qu.: 0.5400
## Median :2008-09-01   Median : 1.1595
## Mean   :2008-08-31   Mean    : 2.8314
## 3rd Qu.:2016-12-01   3rd Qu.: 4.2805
## Max.   :2025-03-01   Max.    :16.2500
```

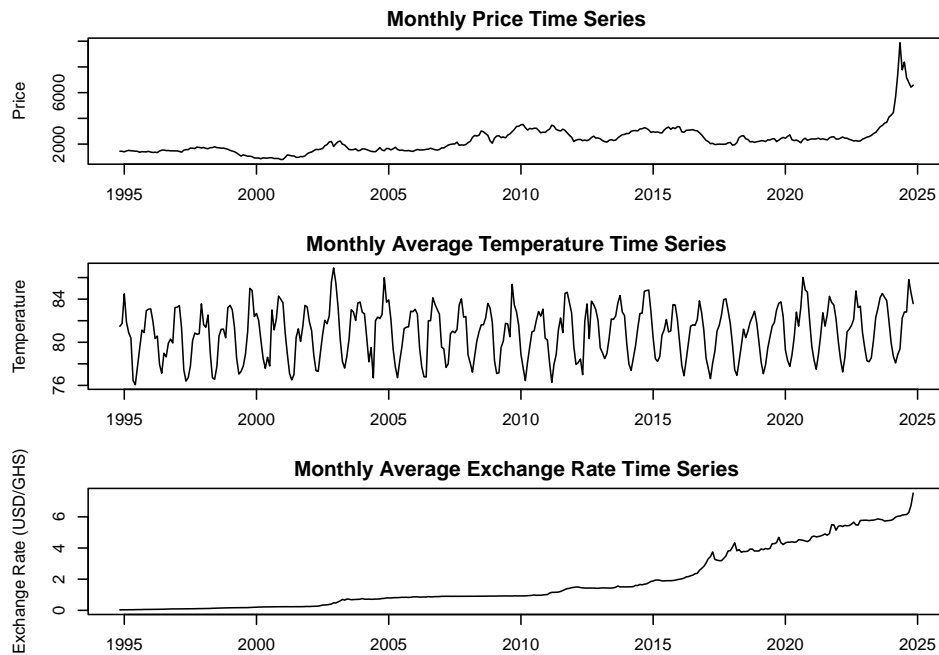
```
rate_ts <- ts(rate_month$exchange_rate, start = c(1994, 11), end = c(2024, 11), frequency = 12)
```

```
ts.plot(rate_ts, main="Monthly Average Exchange Rate Time Series", ylab="Exchange Rate (USD/GHS)", xlab="Time")
```

## Monthly Average Exchange Rate Time Series



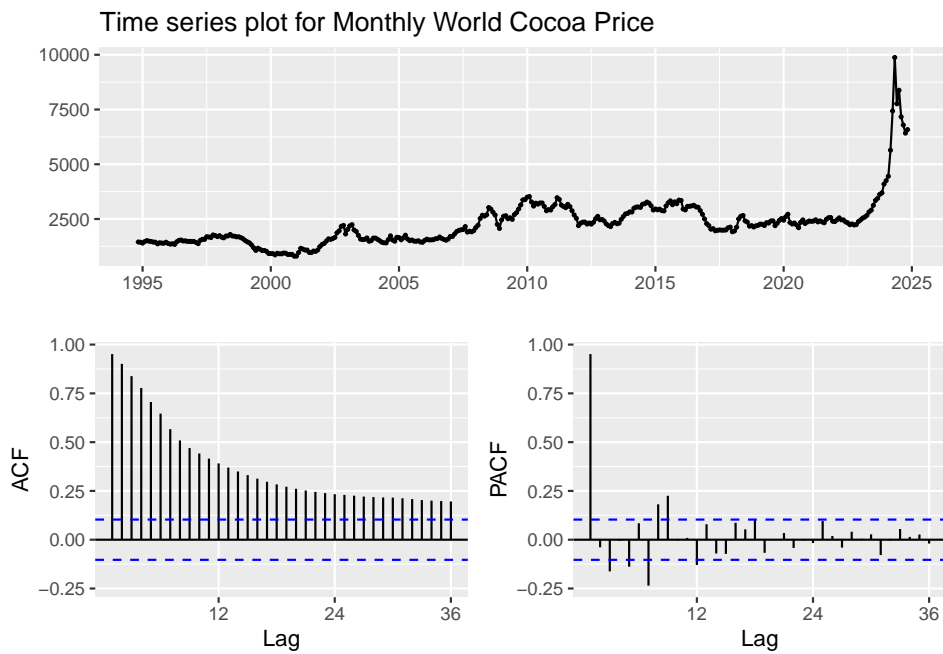
```
par(mfrow=c(3,1), mar = c(3, 4, 2, 2))  
# price  
plot(price_ts, main="Monthly Price Time Series", ylab="Price", xlab="Time")  
# temperature  
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")  
# exchange rate  
ts.plot(rate_ts, main="Monthly Average Exchange Rate Time Series", ylab="Exchange Rate (USD/GHS)", xlab="Time")
```



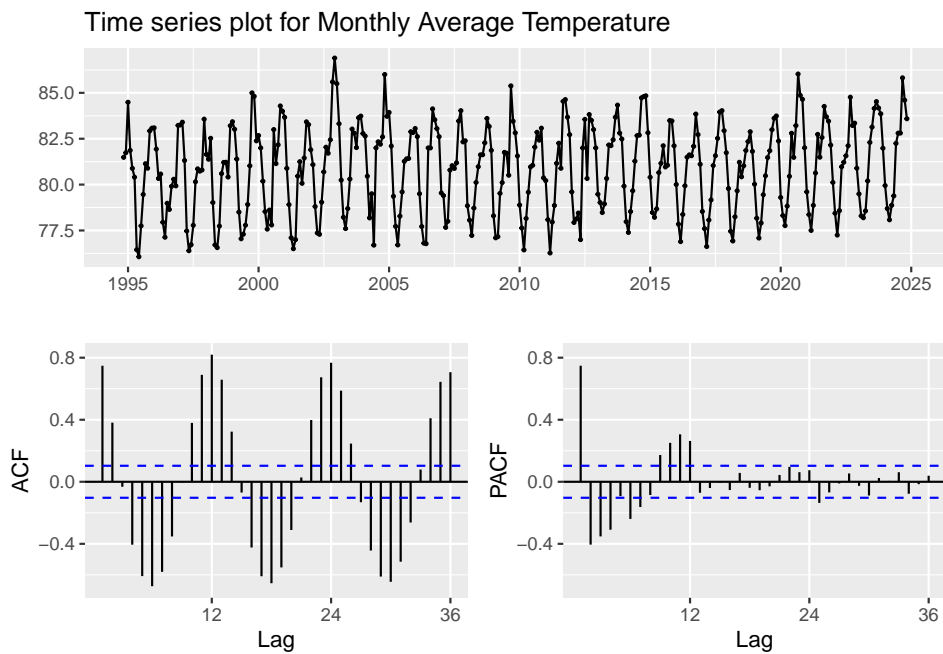
### 1.4 Time series plots for data



```
ggtsdisplay(price_ts, main="Time series plot for Monthly World Cocoa Price")
```

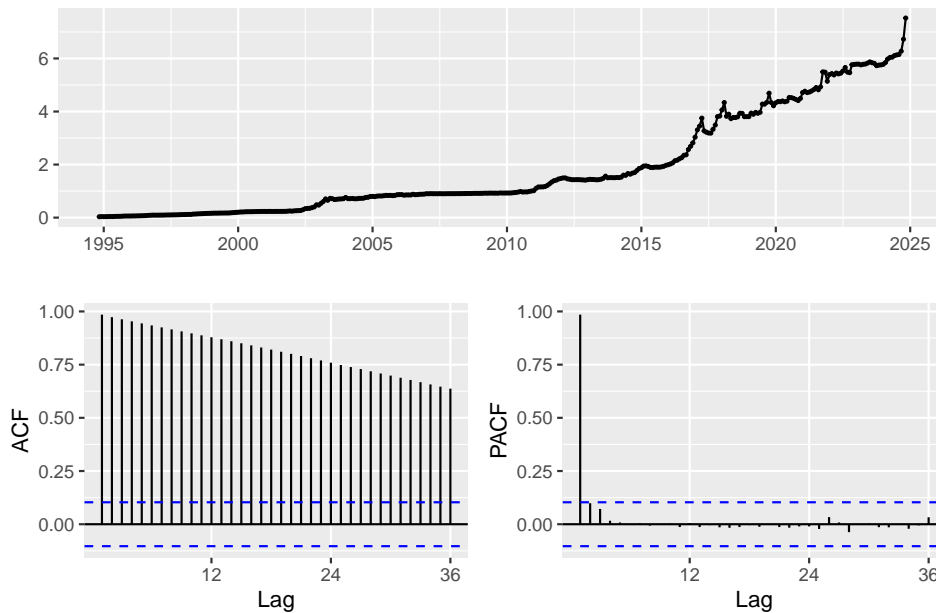


```
ggtsdisplay(weather_ts, main="Time series plot for Monthly Average Temperature")
```

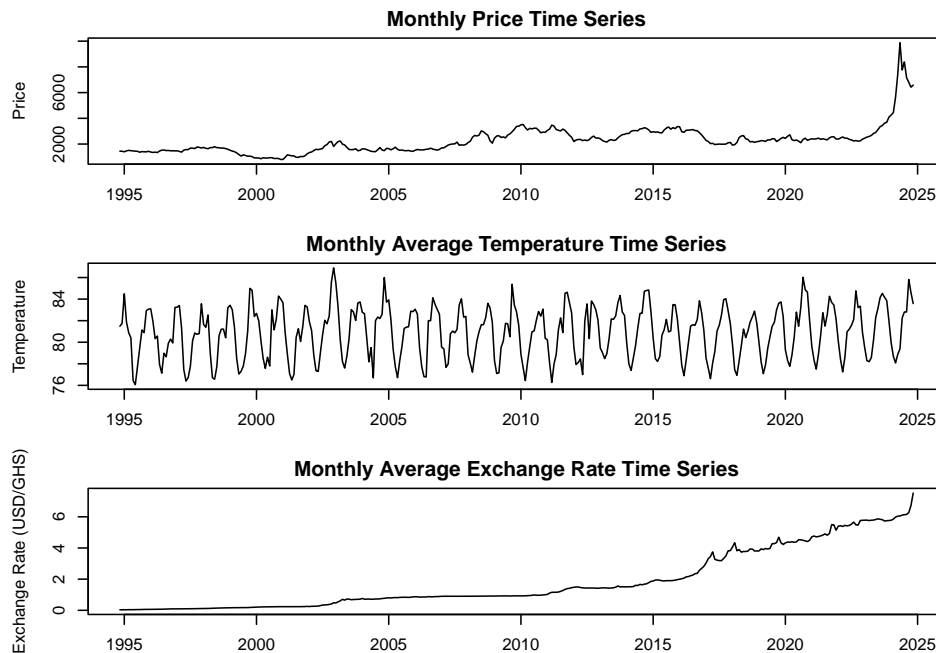


```
ggtsdisplay(rate_ts, main="Time series plot for Monthly Average Exchange Rate(USD/GHS)")
```

Time series plot for Monthly Average Exchange Rate(USD/GHS)



```
par(mfrow=c(3,1), mar = c(3, 4, 2, 2))
# price
plot(price_ts, main="Monthly Price Time Series", ylab="Price", xlab="Time")
# temperature
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")
# exchange rate
ts.plot(rate_ts, main="Monthly Average Exchange Rate Time Series", ylab="Exchange Rate (USD/GHS)", xlab="Time")
```



## 1.5 Combine and Split data

```
data <- price_month |> left_join(weather_month, by = "Time") |> left_join(rate_month, by = "Time")
data <- data |> mutate(log_price = log(month_Price), diff_log_price =
```

```

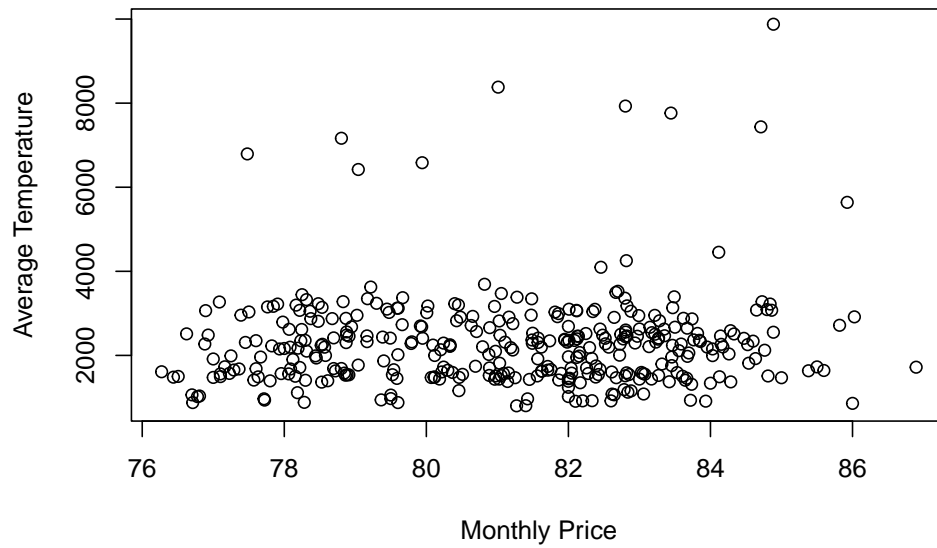
      c(NA, diff(price_month$price_log))) |> drop_na()
data <- data |> dplyr::select(Time, Avg_Temp, exchange_rate, diff_log_price, log_price, month_Price)

data$Time <- as.Date(data$Time)

plot(data$Avg_Temp, data$month_Price, xlab = "Monthly Price", ylab = "Average Temperature",
      main = "Daily Price vs. Avg Temperature")

```

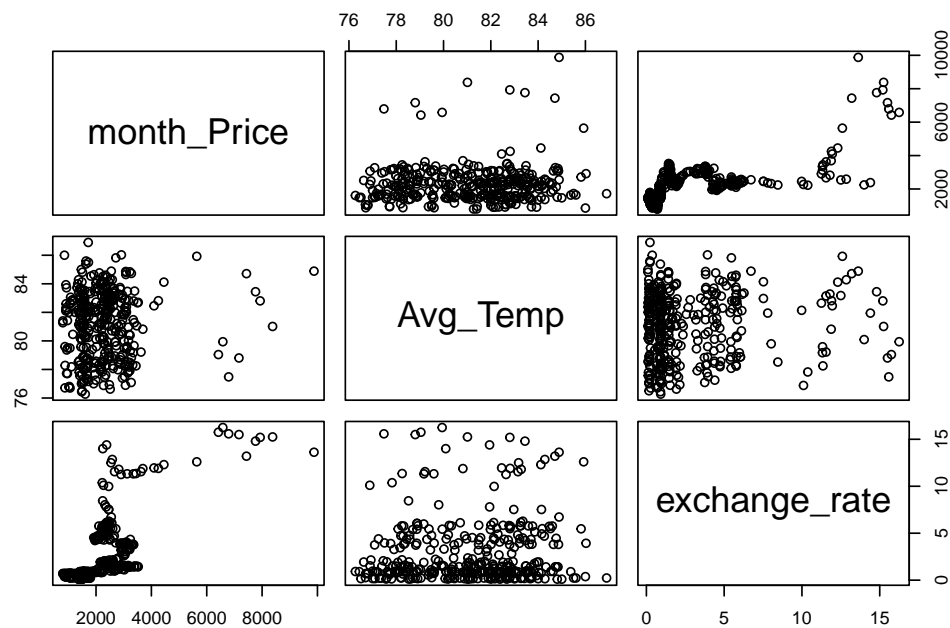
Daily Price vs. Avg Temperature



```

pairs(data[, c("month_Price", "Avg_Temp", "exchange_rate")])

```



```

data <- data[order(data$Time), ]
cutoff <- floor(0.7 * nrow(data))
trainSet <- data[1:cutoff, ]
testSet <- data[(cutoff+1):nrow(data), ]

```

```
data_train_ts <- ts(trainSet$diff_log_price, frequency = 12)
```

## 1.6 Stationarity check and Decomposition

```
adf.test(data$month_Price)
```

```
##
## Augmented Dickey-Fuller Test
##
## data: data$month_Price
## Dickey-Fuller = -1.7041, Lag order = 7, p-value = 0.7017
## alternative hypothesis: stationary
```

```
adf.test(data$log_price)
```

```
##
## Augmented Dickey-Fuller Test
##
## data: data$log_price
## Dickey-Fuller = -2.3875, Lag order = 7, p-value = 0.4133
## alternative hypothesis: stationary
```

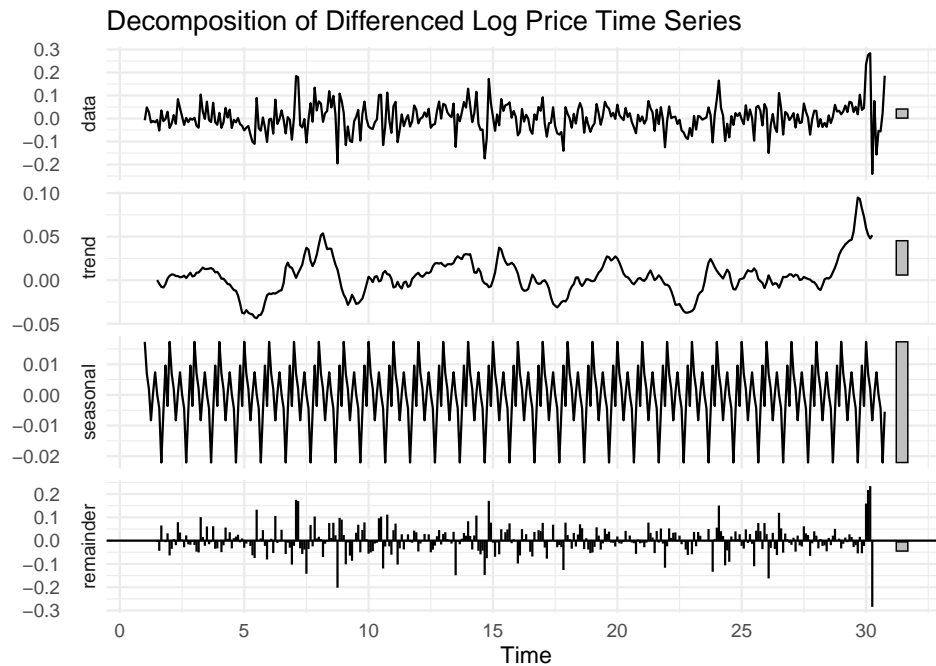
```
adf.test(data$diff_log_price)
```

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

```
##
## Augmented Dickey-Fuller Test
##
## data: data$diff_log_price
## Dickey-Fuller = -6.2103, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

Since only the diff\_log\_price is stationary, we choose differenced monthly log price when fitting the model.

```
diff_price_ts <- ts(data$diff_log_price, frequency = 12)
autoplot(decompose(diff_price_ts, type="additive")) +
  ggtitle("Decomposition of Differenced Log Price Time Series") +
  theme_minimal()
```



## 2. Method

### 2.1 ETS Model

ETS is a purely univariate model and cannot directly handle external regressors.

#### 2.1.1 Fit Model

```
ets_model <- ets(data_train_ts, model = "ANA")
ets_zmodel <- ets(data_train_ts, model = "ZZZ") # Automatically selects best model
summary(ets_model)
```

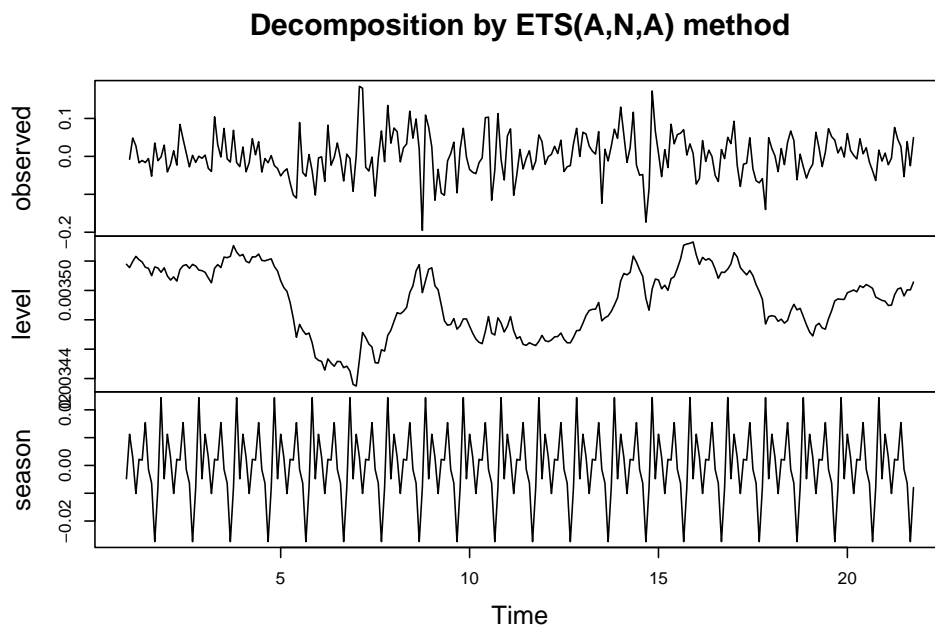
```
## ETS(A,N,A)
##
## Call:
## ets(y = data_train_ts, model = "ANA")
##
## Smoothing parameters:
##   alpha = 1e-04
##   gamma = 1e-04
##
## Initial states:
##   l = 0.0035
##   s = -0.0048 0.0244 -0.0008 -0.0274 -0.0064 -0.0014
##         0.0154 0.0019 0.0022 -0.0101 0.0029 0.0112
##
## sigma: 0.057
##
##      AIC      AICc      BIC
## -36.76439 -34.71311  16.05752
##
## Training set error measures:
```

```
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -0.000482889 0.05534 0.04218605 116.6964 180.0324 0.6834589
##               ACF1
## Training set 0.1729102

summary(ets_zmodel)

## ETS(A,N,N)
##
## Call:
## ets(y = data_train_ts, model = "ZZZ")
##
## Smoothing parameters:
##   alpha = 1e-04
##
## Initial states:
##   l = 0.0029
##
## sigma: 0.0569
##
##      AIC      AICc      BIC
## -48.96308 -48.86552 -38.39869
##
## Training set error measures:
##               ME      RMSE      MAE      MPE      MAPE      MASE
## Training set 1.567182e-05 0.05666171 0.04285329 109.1957 114.6766 0.694269
##               ACF1
## Training set 0.1682833

plot(ets_model)
```



### 2.1.2 Forecasting and Plotting

```
# Plot using log differenced price
data_test_ts <- ts(testSet$diff_log_price, start = end(data_train_ts) + c(0,1),
```

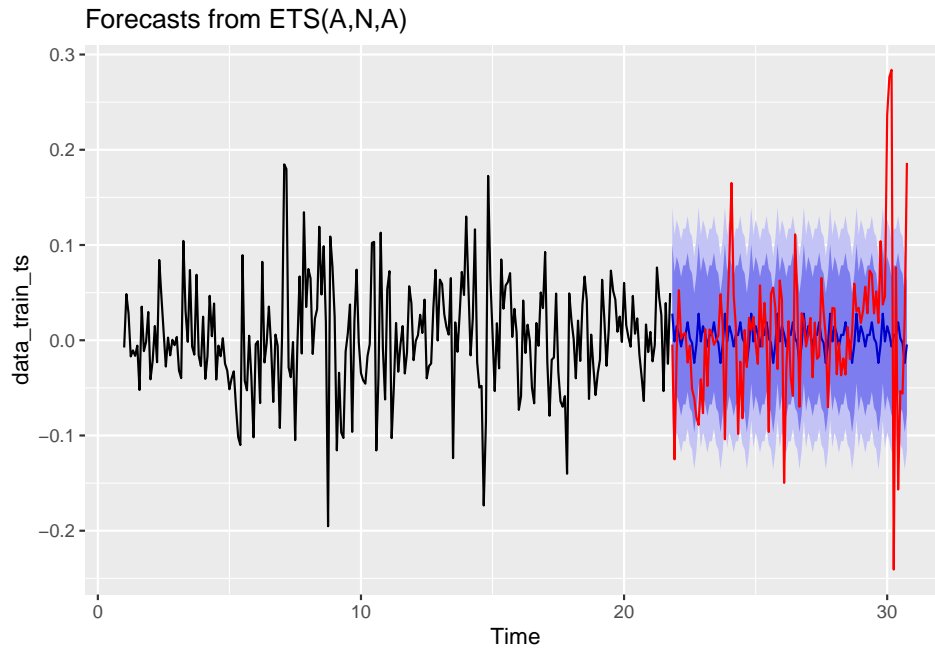
```

frequency = 12)

h <- nrow(testSet)
forecast_ets <- forecast(ets_model, h = h)

autoplot(forecast_ets) + autolayer(data_test_ts, series = "Actual", color = "red")

```



The red line is the observed actual values. The forecasted values are the central blue line within the blue shaded prediction intervals.

```

last_log_price <- tail(trainSet$log_price, 1)

# Convert back to actual price
forecasted_price <- exp(cumsum(forecast_ets$mean) + last_log_price)

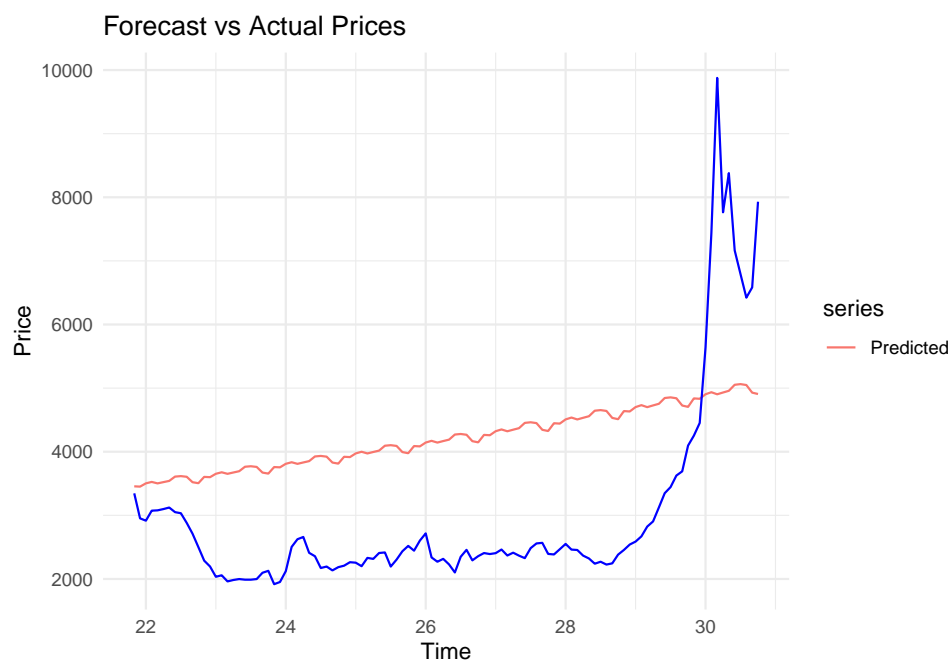
actual_price <- exp(testSet$log_price)

data_test_ts <- ts(testSet$diff_log_price, start = end(data_train_ts) + c(0,1),
frequency = 12)

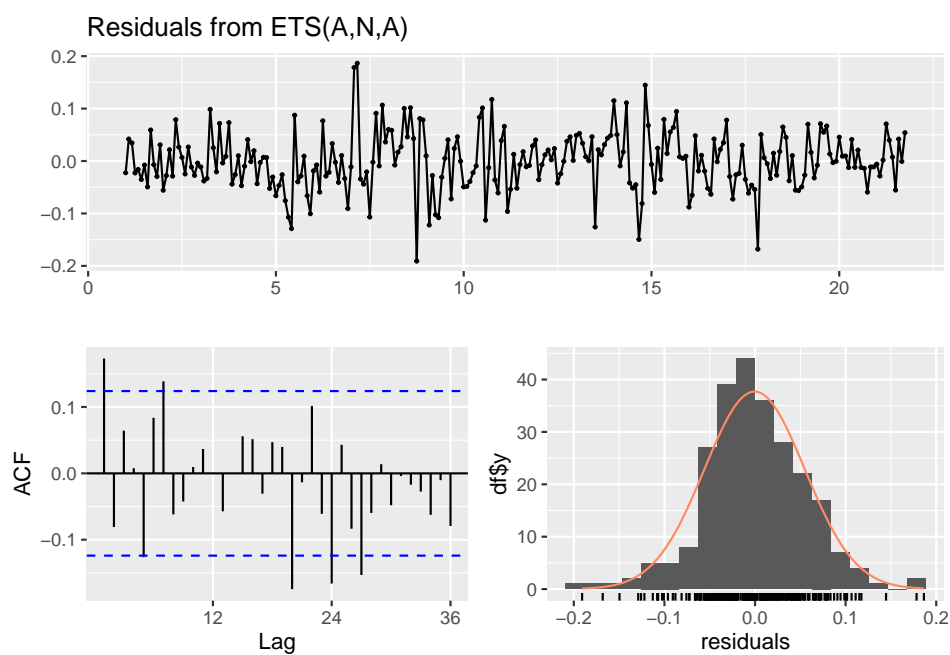
forecast_ets_ts <- ts(forecasted_price, start = start(data_test_ts), frequency = 12)
actual_ets_ts <- ts(actual_price, start = start(data_test_ts), frequency = 12)

# Plot using actual price
autoplot(forecast_ets_ts, series = "Predicted") +
  autolayer(actual_ets_ts, series = "Actual", color = "blue") +
  ggtitle("Forecast vs Actual Prices") +
  ylab("Price") +
  xlab("Time") +
  theme_minimal()

```



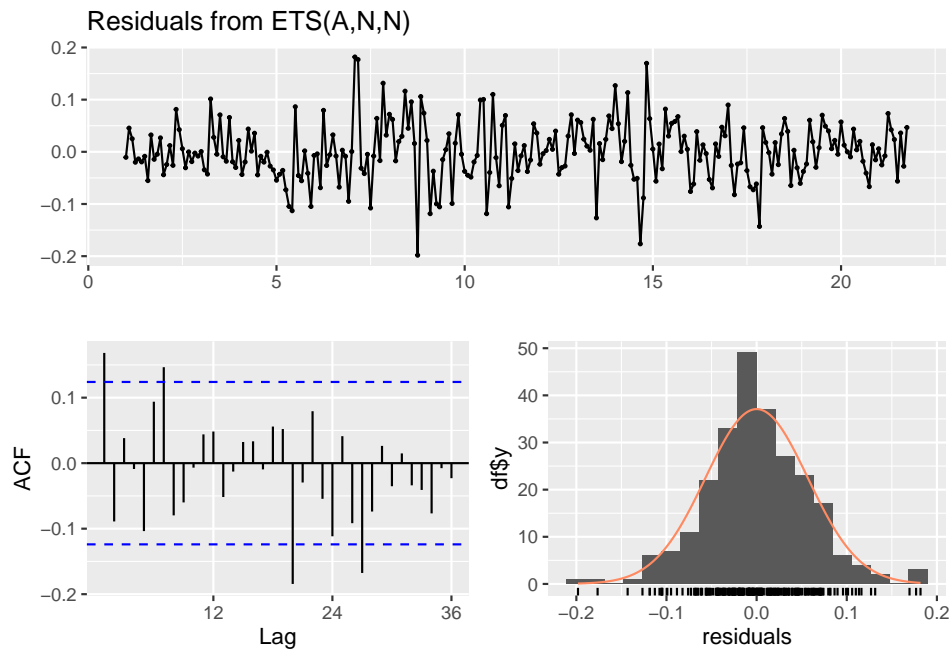
```
checkresiduals(ets_model)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,N,A)
## Q* = 46.672, df = 24, p-value = 0.003672
##
## Model df: 0.   Total lags used: 24
```



```
checkresiduals(ets_zmodel)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from ETS(A,N,N)
## Q* = 42.424, df = 24, p-value = 0.01156
##
## Model df: 0.   Total lags used: 24
```

## 2.2 ARIMAX Model

Recall that in Section 1.3.1, we have tested the `acf` and `adf.test`, and determined that we would be using the differenced price data. To fit the trainset, we evaluate `p` and `q` for ARIMA model.

```
adf.test(trainSet$log_price)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  trainSet$log_price
## Dickey-Fuller = -2.5744, Lag order = 6, p-value = 0.334
## alternative hypothesis: stationary
```

Next, we check if applying 1st differencing is good enough

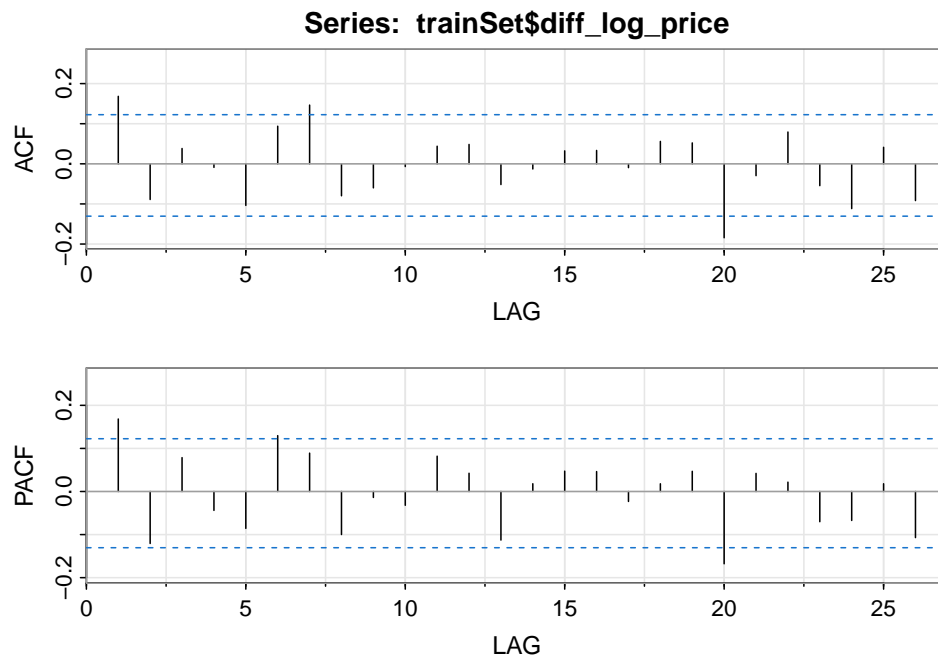
```
adf.test(diff(trainSet$month_Price))
```

```
## Warning in adf.test(diff(trainSet$month_Price)): p-value smaller than printed
## p-value
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(trainSet$month_Price)
## Dickey-Fuller = -5.2038, Lag order = 6, p-value = 0.01
```

```
## alternative hypothesis: stationary
```

P-value is smaller than 0.01 for differenced log price, and we are

```
acf2(trainSet$diff_log_price)
```



```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.17 -0.09 0.04 -0.01 -0.10 0.09 0.15 -0.08 -0.06 -0.01 0.04 0.05 -0.05
## PACF 0.17 -0.12 0.08 -0.04 -0.09 0.13 0.09 -0.10 -0.01 -0.03 0.08 0.04 -0.11
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF  -0.01 0.03 0.03 -0.01 0.06 0.05 -0.18 -0.03 0.08 -0.05 -0.11 0.04
## PACF 0.02 0.05 0.05 -0.02 0.02 0.05 -0.17 0.04 0.02 -0.07 -0.07 0.02
##      [,26]
## ACF  -0.09
## PACF -0.11
```

### 2.2.1 Fit ARIMAX Model

```
xreg_matrix <- cbind(trainSet$Avg_Temp, trainSet$exchange_rate)
colnames(xreg_matrix) <- c("Avg_Temp", "exchange_rate")
arimax_model <- Arima(trainSet$log_price, order=c(1,1,1), xreg = xreg_matrix)
summary(arimax_model)
```

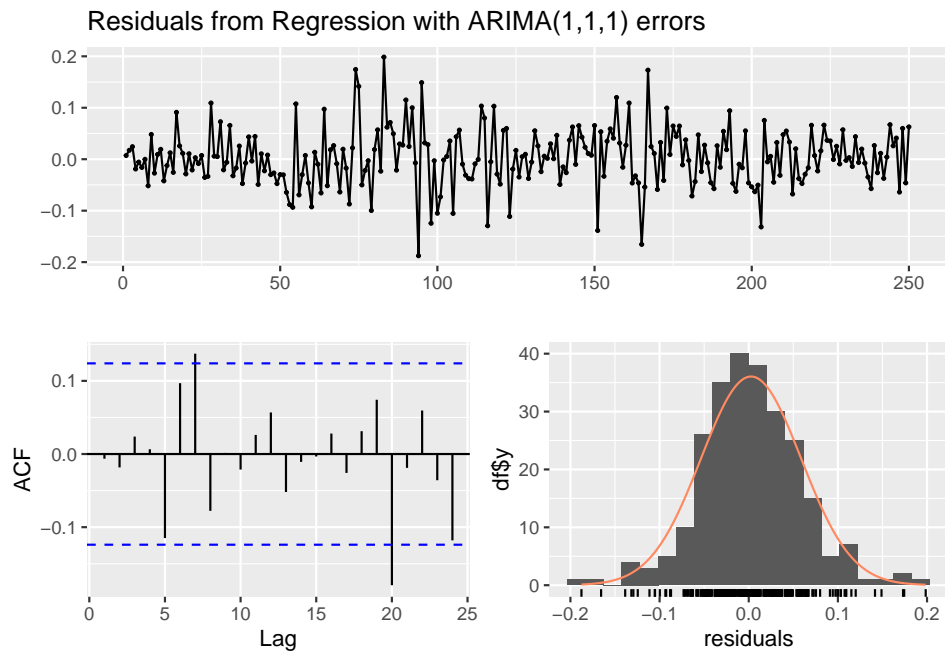
```
## Series: trainSet$log_price
## Regression with ARIMA(1,1,1) errors
##
## Coefficients:
##          ar1      ma1  Avg_Temp  exchange_rate
##        -0.2875  0.5124   0.0010    0.0296
## s.e.      0.1971  0.1743   0.0022    0.0485
##
## sigma^2 = 0.003219: log likelihood = 363.12
## AIC=-716.24  AICc=-715.99  BIC=-698.65
##
```

```
## Training set error measures:
##           ME           RMSE           MAE           MPE           MAPE           MASE
## Training set 0.002593798 0.05617033 0.04245345 0.03138562 0.5658759 0.9793631
##           ACF1
## Training set -0.006404069
```

```
AIC(arimax_model)
```

```
## [1] -716.2394
```

```
checkresiduals(arimax_model)
```

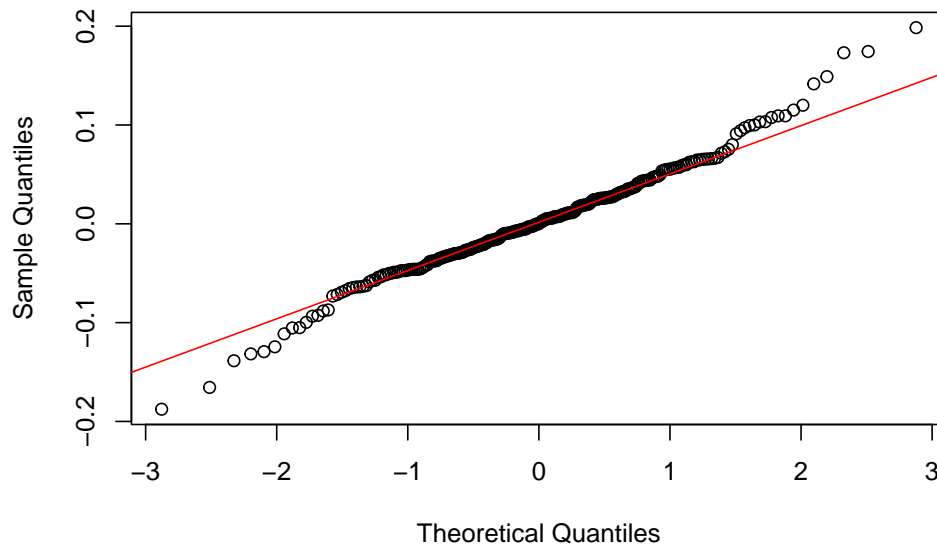


```
##
## Ljung-Box test
##
## data: Residuals from Regression with ARIMA(1,1,1) errors
## Q* = 12.635, df = 8, p-value = 0.125
##
## Model df: 2. Total lags used: 10
```

```
qqnorm(arimax_model$residuals)
```

```
qqline(arimax_model$residuals, col="red")
```

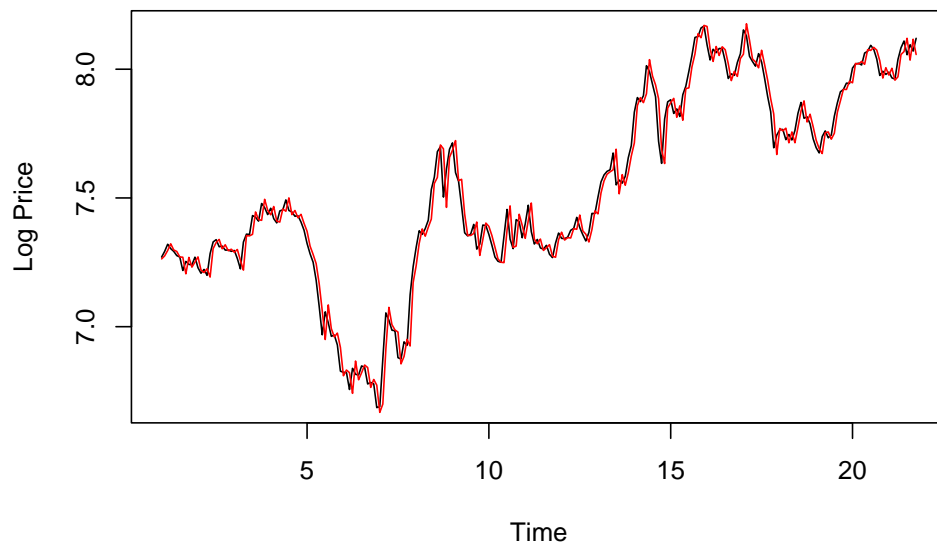
Normal Q-Q Plot



Fail to reject  $H_0$ , hence residuals of this plot do not show significant autocorrelation. - QQ-plot shows: .... - ACF shows: ... - Residuals shows: ...

```
arimax_train_ts <- ts(trainSet$log_price, start = start(min(trainSet$Time)), frequency = 12)
arimax_fitted_ts <- ts(fitted(arimax_model), start = start(min(trainSet$Time)), frequency = 12)
plot(arimax_train_ts, type='l', col='black', main="ARIMAX: Train Set Log Prices vs Fitted",
     ylab="Log Price", xlab="Time")
lines(arimax_fitted_ts, col='red')
```

ARIMAX: Train Set Log Prices vs Fitted



The ARIMAX model fit the trainSet very accurately.

### 2.2.2 Forecasting With ARIMAX Model

Next we try to fit this ARIMAX model to testing set.

```

forecast_arimax_xreg <- cbind(testSet$Avg_Temp, testSet$exchange_rate)
colnames(forecast_arimax_xreg) <- c("Avg_Temp", "exchange_rate")
forecast_arimax <- forecast(arimax_model, xreg=forecast_arimax_xreg, h=nrow(testSet))

```

Then we convert the log prediction back to original price.

```

start_year <- format(min(testSet$Time), "%Y")
start_month <- format(min(testSet$Time), "%m")
start_arimax_test = c(as.numeric(start_year), as.numeric(start_month))

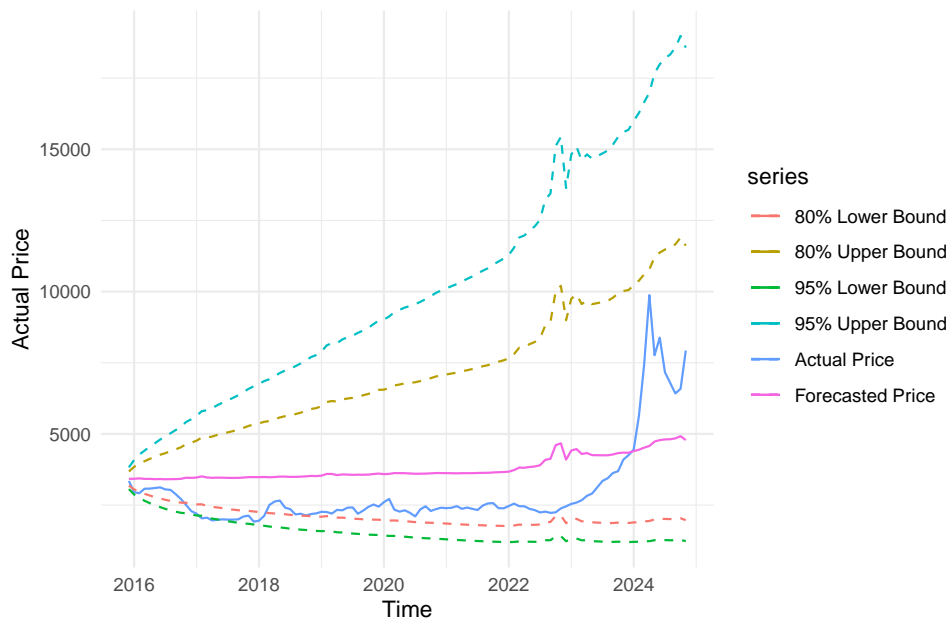
actual_price_arimax <- testSet$month_Price
forecasted_price_arimax <- exp(forecast_arimax$mean)
forecasted_arimax_lower95 <- exp(forecast_arimax$lower[,2])
forecasted_arimax_lower80 <- exp(forecast_arimax$lower[,1])
forecasted_arimax_upper95 <- exp(forecast_arimax$upper[,2])
forecasted_arimax_upper80 <- exp(forecast_arimax$upper[,1])

actual_arimax_ts <- ts(actual_price_arimax, start = start_arimax_test, frequency = 12)
forecast_arimax_ts <- ts(forecasted_price_arimax, start = start_arimax_test, frequency = 12)
forecasted_arimax_lower80_ts <- ts(forecasted_arimax_lower80, start = start_arimax_test, frequency = 12)
forecasted_arimax_lower95_ts <- ts(forecasted_arimax_lower95, start = start_arimax_test, frequency = 12)
forecasted_arimax_upper80_ts <- ts(forecasted_arimax_upper80, start = start_arimax_test, frequency = 12)
forecasted_arimax_upper95_ts <- ts(forecasted_arimax_upper95, start = start_arimax_test, frequency = 12)

# Plot with proper transformation
autoplot(actual_arimax_ts, series="Actual Price") +
  autolayer(forecast_arimax_ts, series="Forecasted Price") +
  autolayer(forecasted_arimax_lower95_ts, series="95% Lower Bound", linetype="dashed") +
  autolayer(forecasted_arimax_lower80_ts, series="80% Lower Bound", linetype="dashed") +
  autolayer(forecasted_arimax_upper80_ts, series="80% Upper Bound", linetype="dashed") +
  autolayer(forecasted_arimax_upper95_ts, series="95% Upper Bound", linetype="dashed") +
  ggtitle("ARIMAX(1,1,1) Forecast (Transformed to Original Scale)") +
  ylab("Actual Price") +
  xlab("Time") +
  theme_minimal()

```

ARIMAX(1,1,1) Forecast (Transformed to Original Scale)



```
accuracy(forecast_arimax$mean, testSet$log_price)
```

```
## [1] 0
```

```
accuracy(forecast_arimax_ts, actual_arimax_ts)
```

```
## [1] 0
```

80% lower and upper bound from forecasts create a tighter bounds for the forecasting the actual price as shown in graph. However, since our price data is non-stationary, and there is a sudden increase towards the end, the ARIMAX model, which relies on historical patterns, struggles to capture this trend, leading to poorer performance on the test set.

## 2.3 GARCH Model

### ARCH/GARCH Model (Monthly)

```
garch_spec <- ugarchspec(
  variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
  mean.model = list(armaOrder = c(1, 1), include.mean = TRUE),
  distribution.model = "norm"
)
garch_fit <- ugarchfit(spec = garch_spec, data = trainSet$diff_log_price)
garch_fit
```

```
##
## *-----*
## *          GARCH Model Fit          *
## *-----*
##
## Conditional Variance Dynamics
## -----
## GARCH Model   : sGARCH(1,1)
## Mean Model    : ARFIMA(1,0,1)
## Distribution   : norm
```

```

##
## Optimal Parameters
## -----
##      Estimate   Std. Error   t value   Pr(>|t|)
## mu      0.003483    0.003505    0.99364  0.320399
## ar1     -0.361441    0.218279   -1.65586  0.097750
## ma1      0.530997    0.194176    2.73462  0.006245
## omega    0.000203    0.000140    1.44321  0.148961
## alpha1   0.123035    0.056972    2.15958  0.030805
## beta1    0.817532    0.078868   10.36586  0.000000
##
## Robust Standard Errors:
##      Estimate   Std. Error   t value   Pr(>|t|)
## mu      0.003483    0.003503    0.9944  0.320027
## ar1     -0.361441    0.167744   -2.1547  0.031184
## ma1      0.530997    0.145785    3.6423  0.000270
## omega    0.000203    0.000130    1.5581  0.119212
## alpha1   0.123035    0.042833    2.8724  0.004073
## beta1    0.817532    0.068735   11.8939  0.000000
##
## LogLikelihood : 377.0571
##
## Information Criteria
## -----
##
## Akaike      -2.9685
## Bayes       -2.8839
## Shibata     -2.9696
## Hannan-Quinn -2.9344
##
## Weighted Ljung-Box Test on Standardized Residuals
## -----
##                                statistic p-value
## Lag[1]                                0.2817  0.5956
## Lag[2*(p+q)+(p+q)-1] [5]      0.8050  1.0000
## Lag[4*(p+q)+(p+q)-1] [9]      4.1960  0.6445
## d.o.f=2
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## -----
##                                statistic p-value
## Lag[1]                                0.0002722  0.9868
## Lag[2*(p+q)+(p+q)-1] [5]  3.0820246  0.3923
## Lag[4*(p+q)+(p+q)-1] [9]  4.5840493  0.4929
## d.o.f=2
##
## Weighted ARCH LM Tests
## -----
##      Statistic Shape Scale P-Value
## ARCH Lag[3]      2.868 0.500 2.000 0.09035
## ARCH Lag[5]      3.749 1.440 1.667 0.19802
## ARCH Lag[7]      4.452 2.315 1.543 0.28605
##

```

```
## Nyblom stability test
## -----
## Joint Statistic: 1.2432
## Individual Statistics:
## mu      0.06455
## ar1     0.05972
## ma1     0.14971
## omega   0.13292
## alpha1  0.10723
## beta1   0.13036
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:      1.49 1.68 2.12
## Individual Statistic: 0.35 0.47 0.75
##
## Sign Bias Test
## -----
##              t-value  prob sig
## Sign Bias      1.0710 0.2852
## Negative Sign Bias 0.7394 0.4604
## Positive Sign Bias 0.2215 0.8249
## Joint Effect    2.0718 0.5576
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## -----
##   group statistic p-value(g-1)
## 1    20      26.96      0.1056
## 2    30      26.24      0.6126
## 3    40      40.24      0.4151
## 4    50      54.00      0.2892
##
##
## Elapsed time : 0.04367089
```

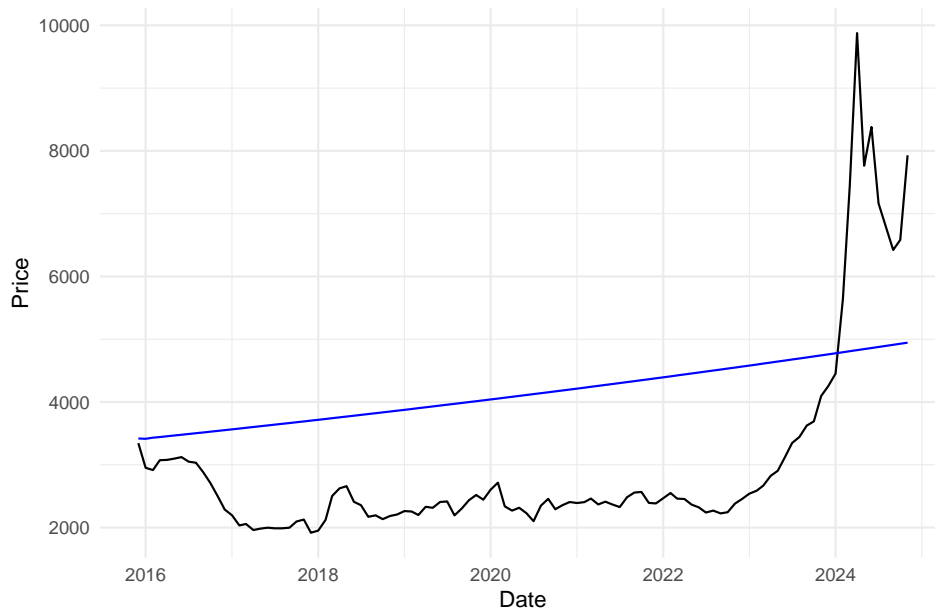
### 2.3.2 GARCH Forecast

```
garch_forecast <- ugarchforecast(garch_fit, n.ahead = length(testSet$diff_log_price))
last_train_price <- tail(trainSet$month_Price, 1)
garch_forecast_prices <- last_train_price * exp(cumsum(as.numeric(fitted(garch_forecast))))
garch_df <- tibble(
  Date = testSet$Time,
  Price = garch_forecast_prices
)
test_df <- tibble(
  Date = testSet$Time,
  Price = testSet$month_Price
)

ggplot() +
  geom_line(data = test_df, aes(x = Date, y = Price), color = "black") +
  geom_line(data = garch_df, aes(x = Date, y = Price), color = "blue") +
  labs(title = "GARCH Forecast vs Actual Prices (Monthly)", y = "Price", x = "Date") +
  theme_minimal()
```



GARCH Forecast vs Actual Prices (Monthly)



```
garch_predicted_vol <- sigma(garch_forecast)
actual_vol <- abs(testSet$log_price) # Assuming you have log-returns
rmse_garch_test <- sqrt(mean((garch_predicted_vol - actual_vol)^2, na.rm = TRUE))
print(rmse_garch_test)
```

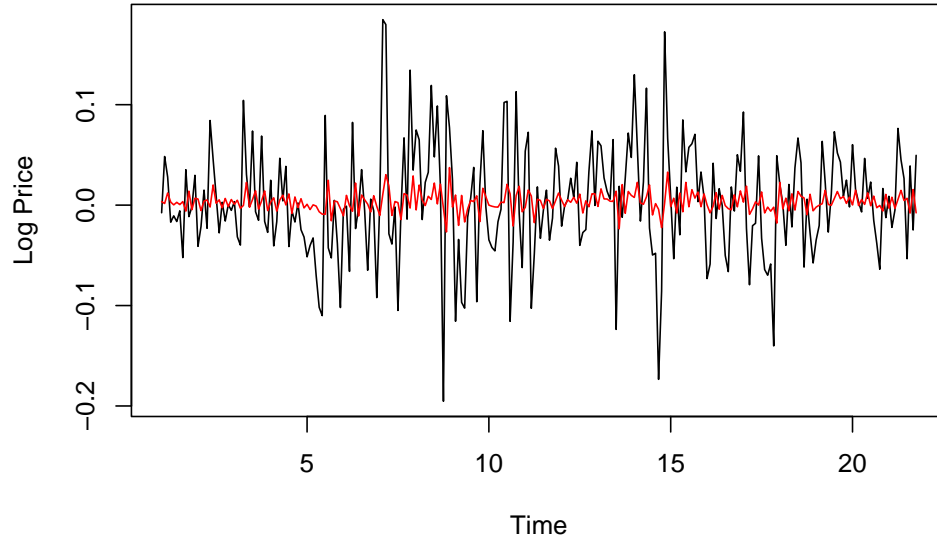
```
## [1] 7.869751
```

```
garch_train_vol <- sigma(garch_fit)
actual_train_vol <- abs(trainSet$log_price) # Assuming you have log-returns
rmse_garch <- sqrt(mean((garch_train_vol - actual_train_vol)^2, na.rm = TRUE))
print(rmse_garch)
```

```
## [1] 7.488708
```

```
train_ts <- ts(trainSet$diff_log_price, start = start(min(trainSet$Time)), frequency = 12)
garch_fitted_ts <- ts(fitted(garch_fit), start = start(min(trainSet$Time)), frequency = 12)
plot(train_ts, type='l', col='black', main="Garch: Train Set Log Prices vs Fitted",
     ylab="Log Price", xlab="Time")
lines(garch_fitted_ts, col='red')
```

### Garch: Train Set Log Prices vs Fitted



## 2.5 GAM Model

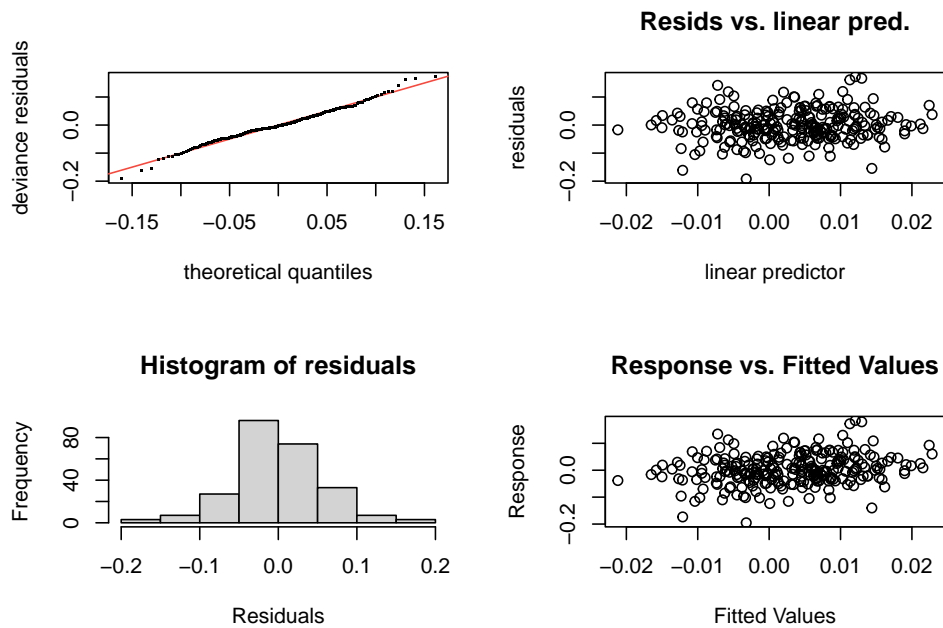
### 2.5.1 Fit Model

```
# Uses simple smoothing splines for variables
trainSet$monthFac <- as.factor(format(trainSet$Time, "%m"))
trainSet$month_num <- as.numeric(trainSet$monthFac)

gam_basic <- gam(diff_log_price ~ s(month_num, bs="cc", k=12) + s(Avg_Temp) + s(exchange_rate),
  data = trainSet, method = "REML")
```

```
gam.check(gam_basic)
```

#### 2.5.1.1 Basic Model



```
##
## Method: REML   Optimizer: outer newton
## full convergence after 10 iterations.
## Gradient range [-0.0001167052,0.0001905195]
## (score -350.1765 & scale 0.003138989).
## Hessian positive definite, eigenvalue range [3.540616e-06,123.5188].
## Model rank = 29 / 29
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##      k'    edf k-index p-value
## s(month_num)    10.00  3.05   1.05   0.77
## s(Avg_Temp)      9.00  1.00   1.06   0.84
## s(exchange_rate) 9.00  1.00   0.89   0.05 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(gam_basic)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(month_num, bs = "cc", k = 12) + s(Avg_Temp) +
##      s(exchange_rate)
##
## Parametric coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.002929   0.003543   0.827   0.409
##
## Approximate significance of smooth terms:
##      edf Ref.df    F p-value
## s(month_num)    3.054 10.000 0.795 0.0278 *
```

```
## s(Avg_Temp)      1.001  1.001 1.289  0.2571
## s(exchange_rate) 1.000  1.000 0.694  0.4055
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0261   Deviance explained = 4.59%
## -REML = -350.18   Scale est. = 0.003139   n = 250
```

```
trainSet$dateInt = as.integer(trainSet$Time)
trainSet$Time_num <- as.numeric(trainSet$Time) / 365 # Convert to years
gam_year <- gam(diff_log_price ~ s(month_num, bs="cc", k=12) +
               sinpi(dateInt / 182.625) + cospi(dateInt / 182.625) +
               sinpi(dateInt / 91.3125) + cospi(dateInt / 91.3125) +
               s(Avg_Temp) + s(exchange_rate), data = trainSet, method = "REML")
```

```
summary(gam_basic)
```

### 2.5.1.2 Complex Model

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(month_num, bs = "cc", k = 12) + s(Avg_Temp) +
##               s(exchange_rate)
##
## Parametric coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.002929   0.003543   0.827   0.409
##
## Approximate significance of smooth terms:
##               edf Ref.df    F p-value
## s(month_num)    3.054 10.000 0.795  0.0278 *
## s(Avg_Temp)      1.001  1.001 1.289  0.2571
## s(exchange_rate) 1.000  1.000 0.694  0.4055
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0261   Deviance explained = 4.59%
## -REML = -350.18   Scale est. = 0.003139   n = 250
```

```
summary(gam_year)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(month_num, bs = "cc", k = 12) + sinpi(dateInt/182.625) +
##               cospi(dateInt/182.625) + sinpi(dateInt/91.3125) + cospi(dateInt/91.3125) +
##               s(Avg_Temp) + s(exchange_rate)
##
## Parametric coefficients:
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.936e-03  3.542e-03   0.829   0.4081
## sinpi(dateInt/182.625) 8.070e-03  8.529e-03   0.946   0.3450
## cospi(dateInt/182.625) 9.264e-03  1.058e-02   0.876   0.3820
## sinpi(dateInt/91.3125) -7.343e-05  5.084e-03  -0.014   0.9885
## cospi(dateInt/91.3125) 1.449e-02  6.045e-03   2.397   0.0173 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df      F p-value
## s(month_num)    0.0003453 10.000 0.000   0.707
## s(Avg_Temp)     1.0003821   1.001 0.289   0.592
## s(exchange_rate) 1.0000069   1.000 0.820   0.366
##
## R-sq.(adj) =  0.0271   Deviance explained = 5.06%
## -REML = -338.66   Scale est. = 0.0031356   n = 250
# plot(gam_complex, pages = 1, shade = TRUE)
# gam.check(gam_complex)
```

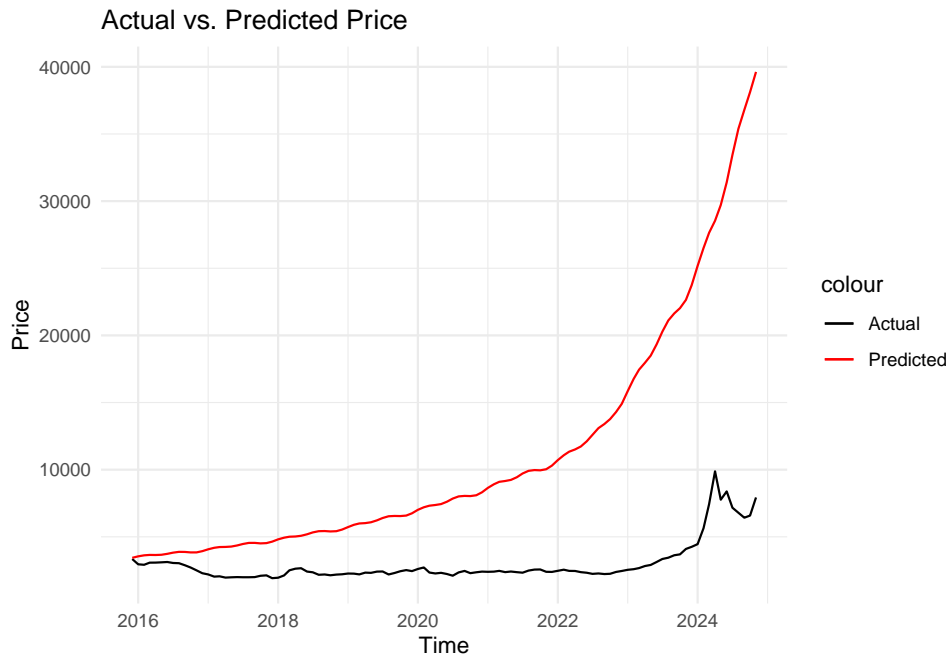
## 2.5.2 Forecast and Plot

```
testSet$month_num <- as.numeric(format(testSet$Time, "%m"))
testSet$Time_num <- as.numeric(difftime(testSet$Time, min(trainSet$Time), units="days")) / 365
testSet$dateInt = as.integer(testSet$Time)

testSet$pred_log <- predict(gam_year, newdata = testSet)
testSet$pred_log_price <- last_log_price + cumsum(testSet$pred_log)
testSet$pred_price <- exp(testSet$pred_log_price)

# testSet$pred_price <- exp(log(last_price) + cumsum(testSet$pred_log))
# testSet$pred_price <- exp(cumsum(testSet$pred_log) + last_log_price)

ggplot(testSet, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price, color = "Predicted")) +
  labs(title = "Actual vs. Predicted Price",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red")) +
  theme_minimal()
```



```
rmse(testSet$month_Price, testSet$pred_price)
```

```
## [1] 10954.54
```

## 2.6 XGBoost Model

```
trainSet$Lag1 <- lag(trainSet$diff_log_price, 1)
trainSet$Lag12 <- lag(trainSet$diff_log_price, 12)
testSet$Lag1 <- lag(testSet$diff_log_price, 1)
testSet$Lag12 <- lag(testSet$diff_log_price, 12)

predictors <- c("Time", "Avg_Temp", "exchange_rate", "Lag1", "Lag12")
target <- "diff_log_price"

train_data <- trainSet[complete.cases(trainSet[, c(predictors, target)]), ]
test_data <- testSet[complete.cases(testSet[, predictors]), ]

train_data <- train_data |>
  mutate(across(all_of(predictors), as.numeric))
test_data <- test_data |>
  mutate(across(all_of(predictors), as.numeric))

# Convert to DMatrix (XGBoost's optimized format)
dtrain <- xgb.DMatrix(
  data = as.matrix(train_data[, predictors]),
  label = train_data[[target]]
)

params <- list(
  objective = "reg:squarederror", # For regression
  eta = 0.05, # Learning rate (lower for time series)
  max_depth = 6, # Tree depth (avoid overfitting)
  subsample = 0.8, # Random subset of data per tree
```

```

    colsample_bytree = 0.8,          # Random subset of features per tree
    gamma = 1,                      # Minimum loss reduction for splits
    min_child_weight = 5            # Prevent overfitting to small groups
)

set.seed(123)
xgb_model <- xgb.train(
  params,
  data = dtrain,
  nrounds = 1000,                  # Large number (early stopping will handle)
  watchlist = list(train = dtrain),
  early_stopping_rounds = 50,      # Stop if no improvement for 50 rounds
  print_every_n = 10
)

## [1] train-rmse:0.475762
## Will train until train_rmse hasn't improved in 50 rounds.
##
## [11] train-rmse:0.289145
## [21] train-rmse:0.179561
## [31] train-rmse:0.117399
## [41] train-rmse:0.084142
## [51] train-rmse:0.068493
## [61] train-rmse:0.061738
## [71] train-rmse:0.059145
## [81] train-rmse:0.058259
## [91] train-rmse:0.057915
## [101] train-rmse:0.057826
## [111] train-rmse:0.057774
## [121] train-rmse:0.057760
## [131] train-rmse:0.057755
## [141] train-rmse:0.057752
## [151] train-rmse:0.057749
## [161] train-rmse:0.057748
## [171] train-rmse:0.057748
## [181] train-rmse:0.057749
## [191] train-rmse:0.057748
## [201] train-rmse:0.057748
## [211] train-rmse:0.057748
## [221] train-rmse:0.057750
## [231] train-rmse:0.057749
## [241] train-rmse:0.057748
## [251] train-rmse:0.057749
## [261] train-rmse:0.057748
## Stopping. Best iteration:
## [214] train-rmse:0.057748

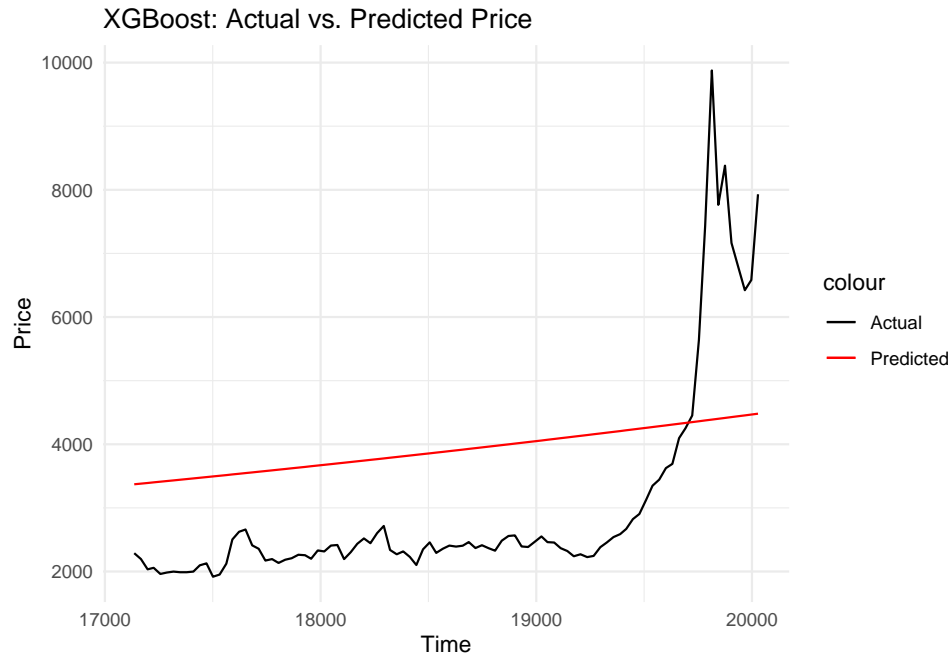
dtest <- xgb.DMatrix(as.matrix(test_data[, predictors]))
test_data$pred_diff_log <- predict(xgb_model, dtest)

# Convert to actual price predictions
test_data <- test_data %>%
  mutate(
    pred_log_price = last_log_price + cumsum(pred_diff_log), # Only if modeling differences

```

```
pred_price = exp(pred_log_price)
)
```

```
ggplot(test_data, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price, color = "Predicted")) +
  labs(title = "XGBoost: Actual vs. Predicted Price",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red")) +
  theme_minimal()
```



```
# xgb = XGBClassifier(random_state=42)
# xgb.fit(predictors, target)
```