STA457 Project

Xing Yu Wang, Carina Wang, Xinyue Tao, Qinyu Qu

2025-03-29

```
library(dplyr)
library(tidyverse)
library(readr)
library(lubridate)
library(forecast)
library(astsa)
library(tseries)
library(mgcv)
library(Metrics)
library(ggplot2)
library(xgboost)
library(Matrix)
library(caret)
library(rugarch)
library(tibble)
library(xts)
library(gridExtra)
```

1. EDA

```
price = read.csv("./Daily Prices_ICCO.csv")
weather = read.csv("./Ghana_data.csv")
# USD_GHS_Historical_Data <- read_csv("~/sta457/STA457_Project/Project/USD_GHS Historical 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
```

```r
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

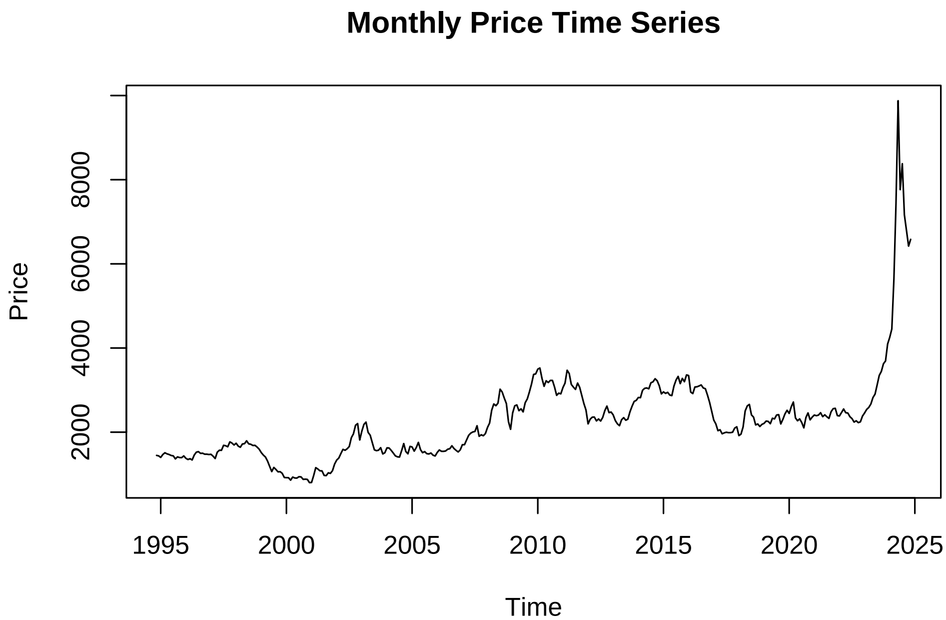
```r
price$Date <- as.Date(price$Date, format="%d/%m/%Y")
price$Daily_Price <- as.numeric(gsub(",", "", price$Daily_Price))
price_month <- price |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(month_Price = mean(Daily_Price, na.rm = TRUE)) |> ungroup()
```

```r
summary(price)
```

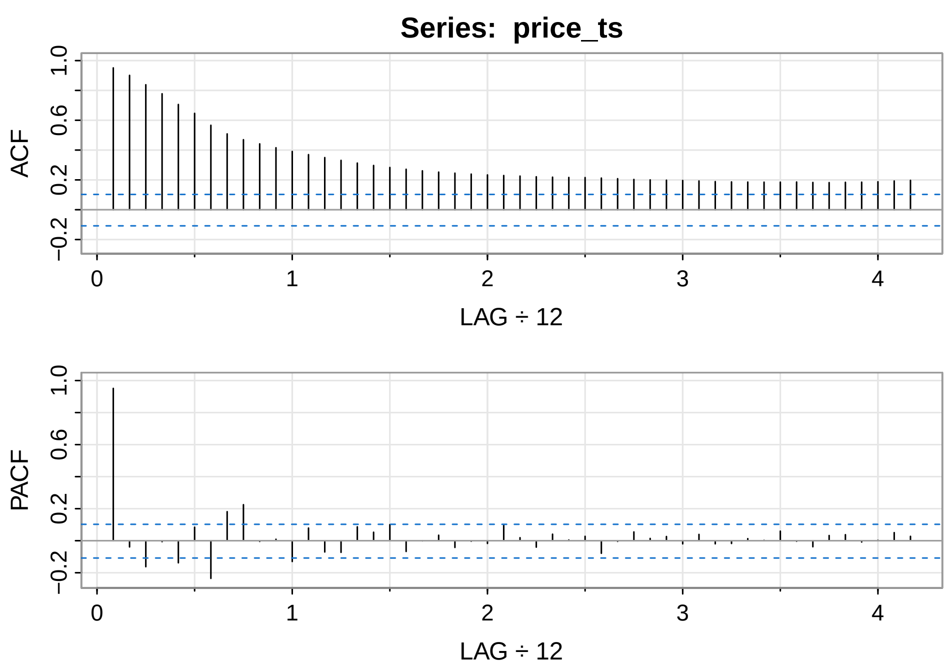
```
##       Date             Daily_Price
##  Min.   :1994-10-03   Min.   :  774.1
##  1st Qu.:2002-05-16   1st Qu.: 1557.8
##  Median :2009-12-17   Median : 2202.0
##  Mean   :2009-12-17   Mean   : 2350.1
##  3rd Qu.:2017-07-24   3rd Qu.: 2738.1
##  Max.   :2025-02-27   Max.   :11984.7
```

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

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




```r
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
```

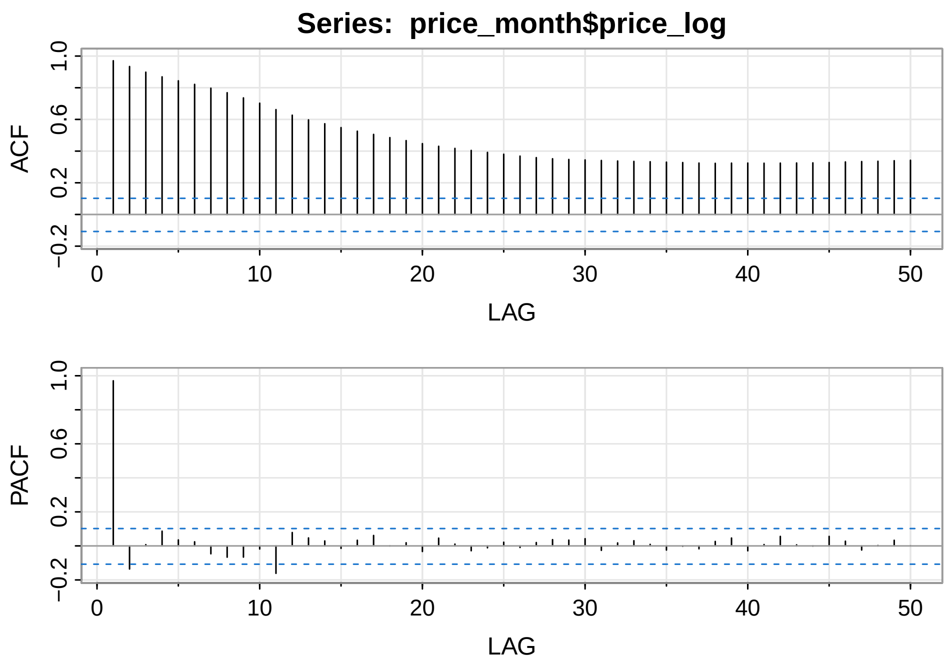
```r
ndiffs(price_ts)
```

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

```r
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
```

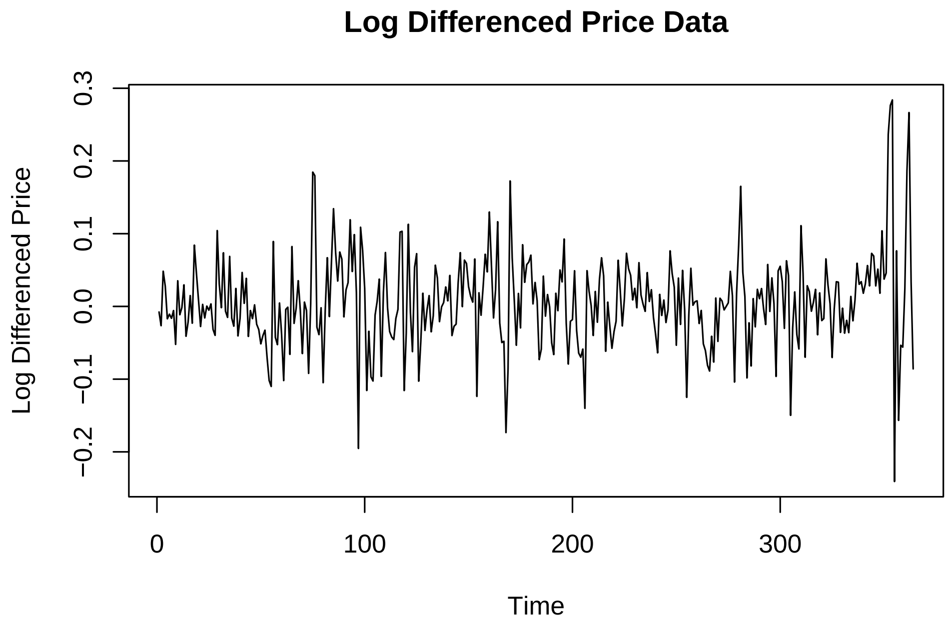
```r
acf2(price_month$price_log, 50)
```




```
##      [,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.

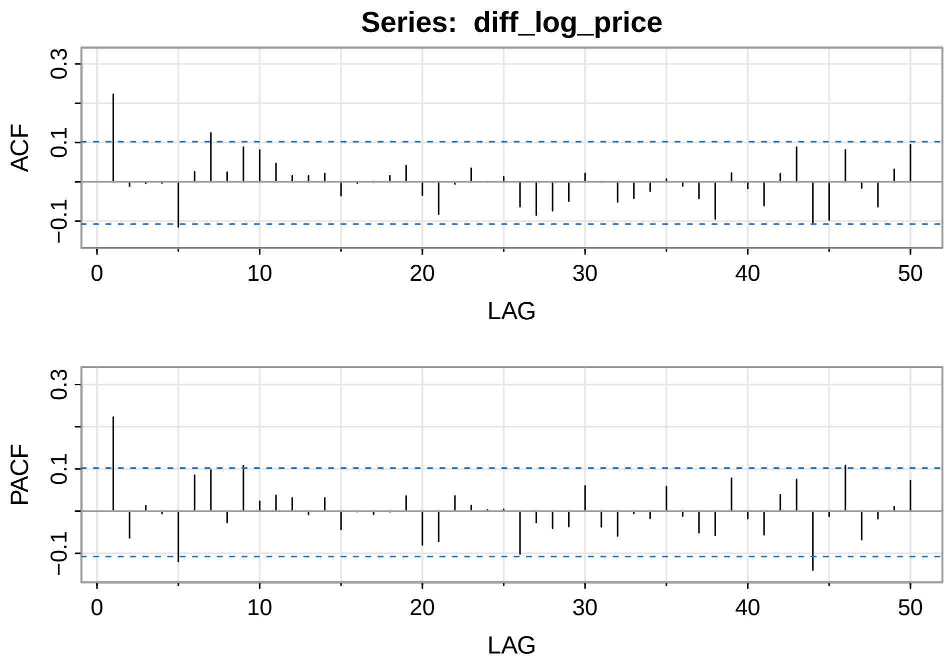
```
diff_log_price = diff(price_month$price_log)
ts.plot(diff_log_price, main = "Log Differenced Price Data", ylab = "Log Differenced Price")
```




```
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

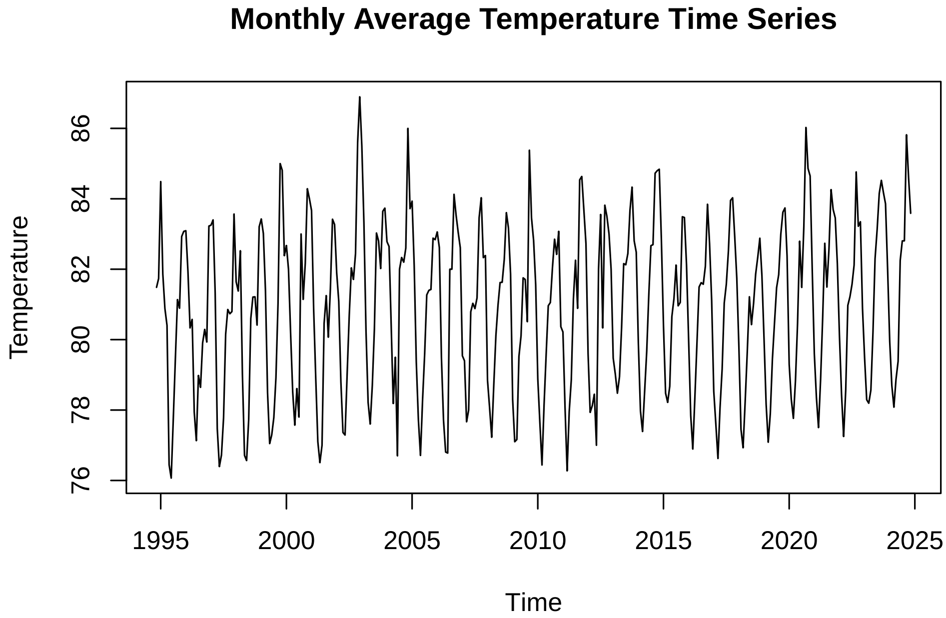
```
weather$Date <- as.Date(weather$Date)
weather$Avg_Temp <- as.numeric(gsub("", "", weather$Avg_Temp))
weather_month <- weather |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(Avg_Temp = mean(Avg_Temp, na.rm = TRUE)) |> ungroup()
```

```
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  
```

```r
weather_ts <- ts(weather_month$Avg_Temp, start = c(1994, 11), end = c(2024, 11), frequency = 12)
```

```r
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")
```



1.3.3 exchange Data

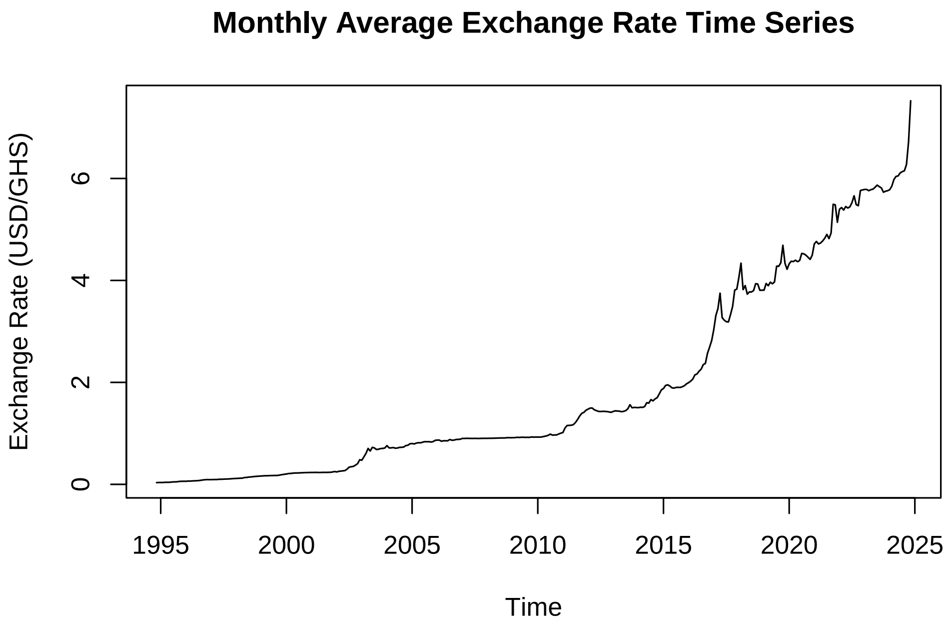
```r
exchangerate$Date <- as.Date(exchangerate$Date)
exchangerate$exchange_rate <- as.numeric(gsub("", "", exchangerate$exchange_rate))
rate_month <- exchangerate |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(exchange_rate = mean(exchange_rate, na.rm = TRUE)) |> ungroup()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

```r
summary(exchangerate)
```

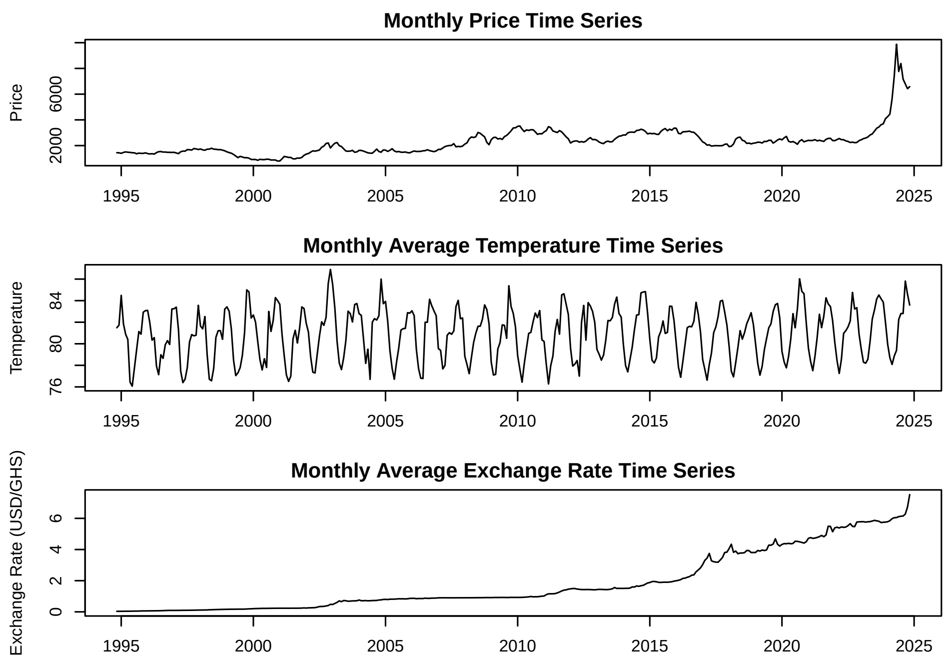
```
##       Date            exchange_rate
##  Min.   :1992-03-01   Min.   : 0.0338
##  1st Qu.:2000-06-01   1st Qu.: 0.5400
##  Median :2008-09-01   Median : 1.1595
##  Mean   :2008-08-31   Mean   : 2.8314
##  3rd Qu.:2016-12-01   3rd Qu.: 4.2805
##  Max.   :2025-03-01   Max.   :16.2500
```

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

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

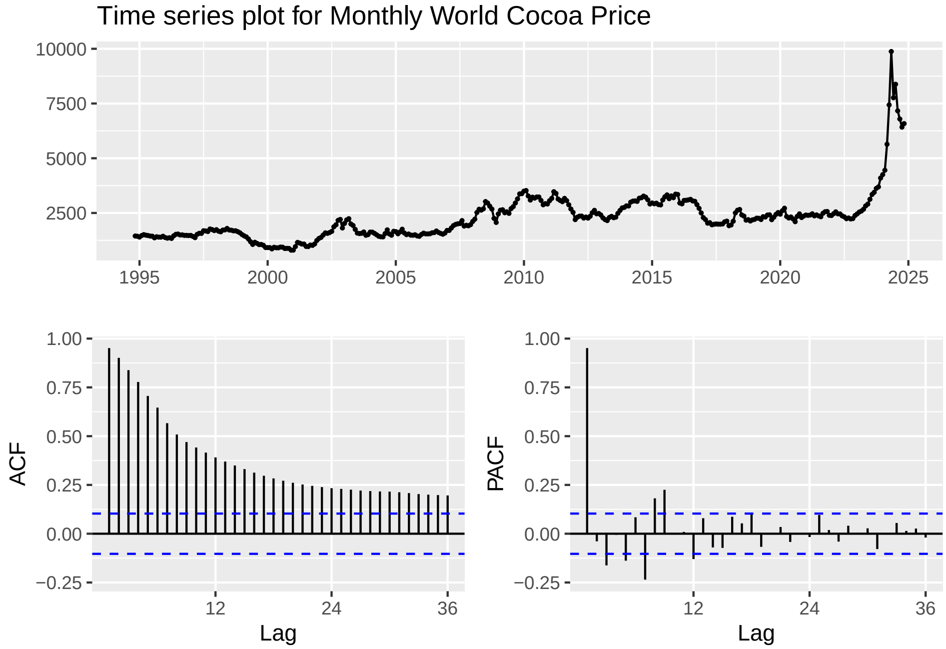



```r
par(mfrow=c(3,1), mar = c(3, 4, 2, 2))
# price
plot(price_ts, main="Monthly Price Time Series", ylab="Price", xlab="Time")
#temperature
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")
# exchange rate
ts.plot(rate_ts, main="Monthly Average Exchange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

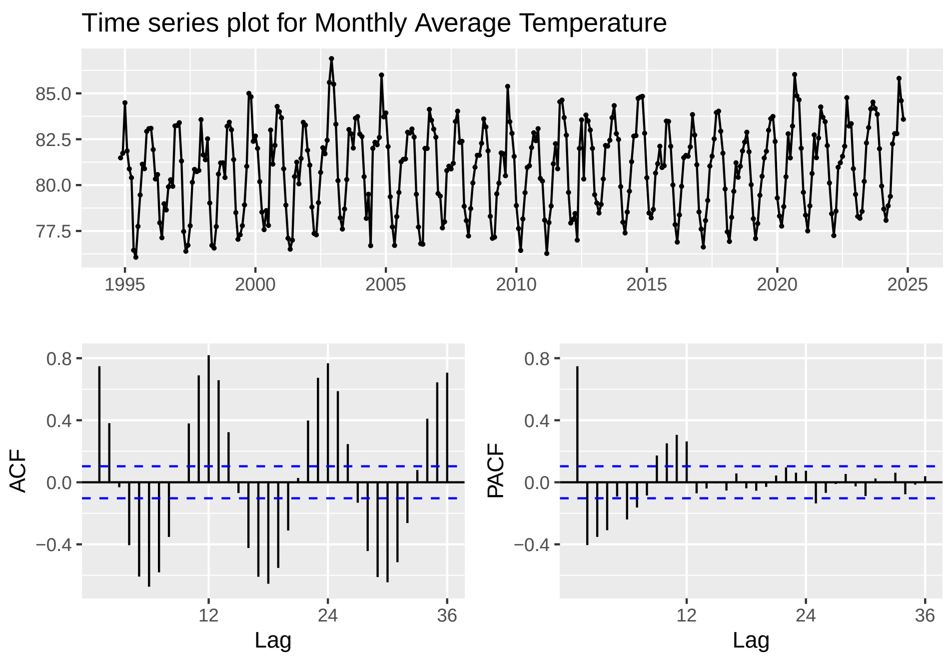


1.4 Time series plots for data

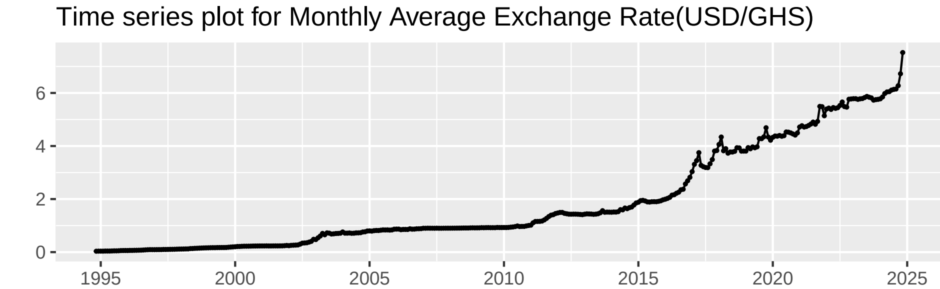
```r
ggtsdisplay(price_ts, main="Time series plot for Monthly World Cocoa Price")
```

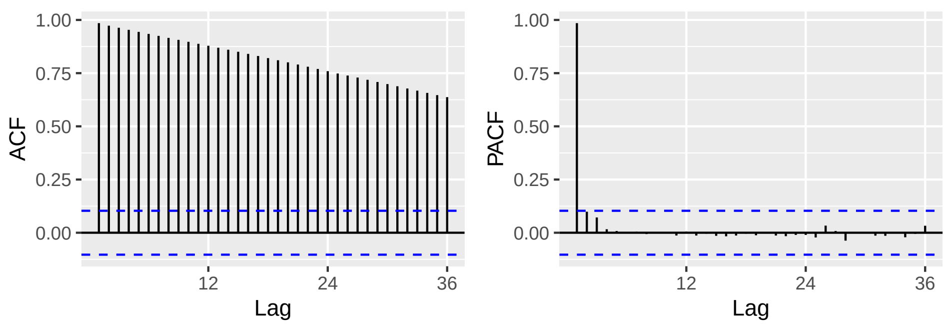



```r
ggtsdisplay(weather_ts, main="Time series plot for Monthly Average Temperature")
```

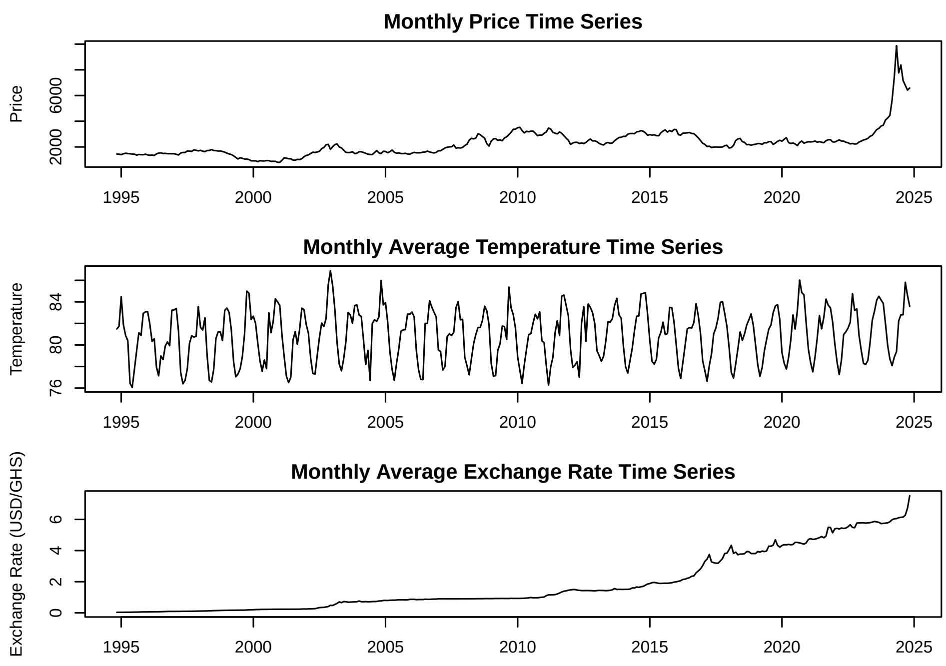



```r
ggtsdisplay(rate_ts, main="Time series plot for Monthly Average Exchange Rate(USD/GHS)")
```






```r
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
```

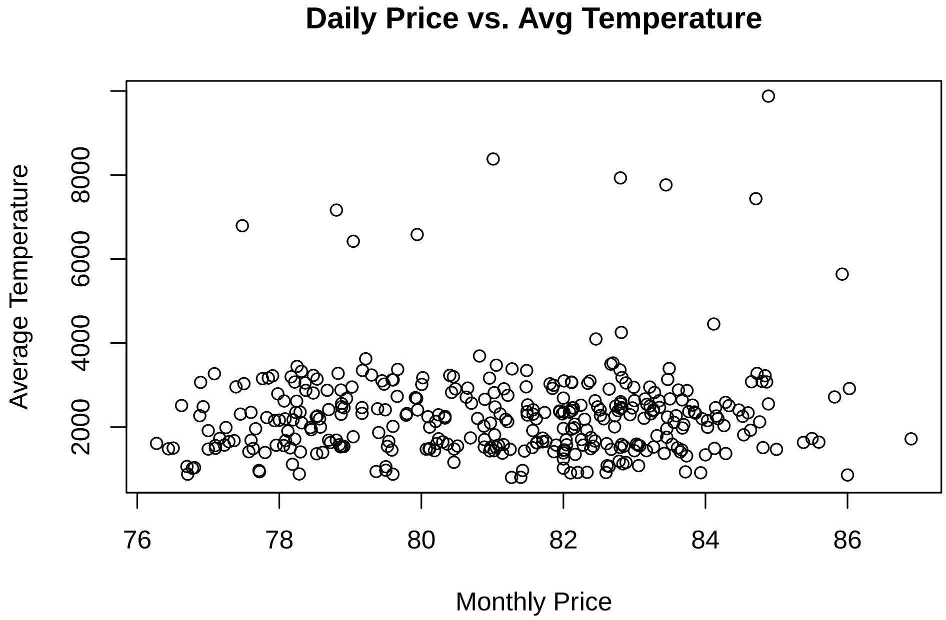


1.5 Combine Datasets

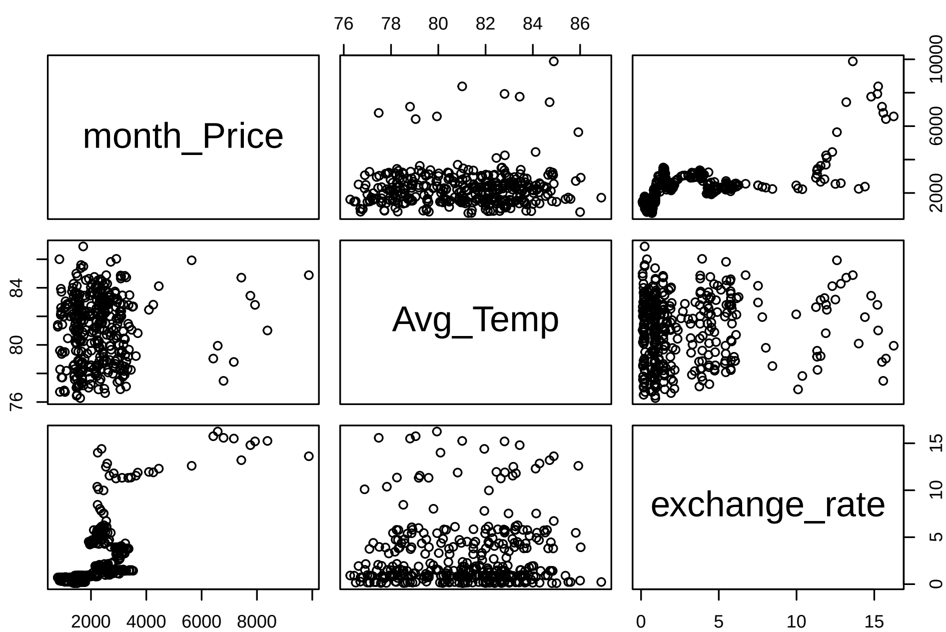
```r
data <- price_month |> left_join(weather_month, by = "Time") |> left_join(rate_month, by = "Time")
data <- data |> mutate(log_price = log(month_Price), diff_log_price =
```

```r
                                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)
```

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




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



1.6 Stationary Check

```r
adf.test(data$Avg_Temp)
```

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

##
##  Augmented Dickey-Fuller Test
##
## data:  data$Avg_Temp
## Dickey-Fuller = -12.411, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

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

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

##
##  Augmented Dickey-Fuller Test
##
## data:  data$exchange_rate
## Dickey-Fuller = 0.7342, Lag order = 7, p-value = 0.99
## alternative hypothesis: stationary
```

```
adf.test(log(data$exchange_rate))
```

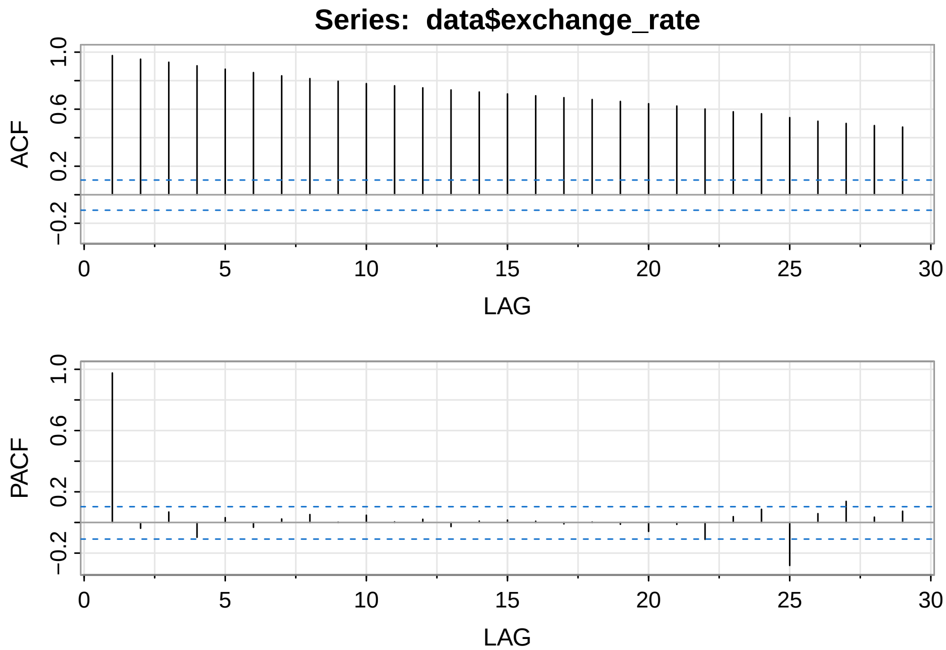
```
##
##  Augmented Dickey-Fuller Test
##
## data:  log(data$exchange_rate)
## Dickey-Fuller = -2.8782, Lag order = 7, p-value = 0.2063
## alternative hypothesis: stationary
```

```
adf.test(diff(log(data$exchange_rate)))
```

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

##
##  Augmented Dickey-Fuller Test
##
## data:  diff(log(data$exchange_rate))
## Dickey-Fuller = -5.3335, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

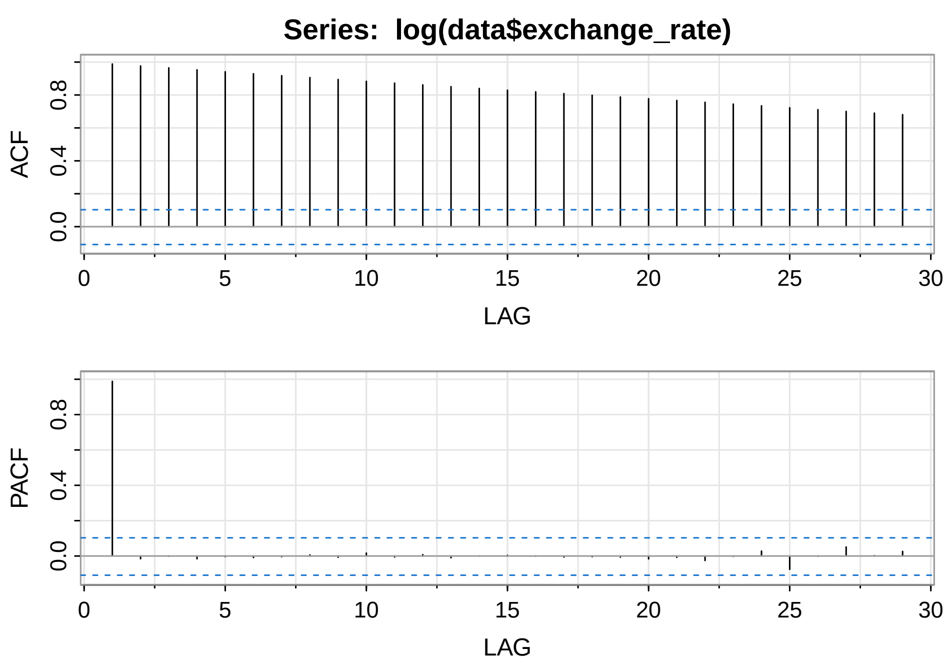
Since monthly average temperature is already stationary, we would do take the differenced and log-transformed exchange rate as our exogenous factors.

```
acf2(data$exchange_rate)
```




```
##      [,1]  [,2] [,3] [,4] [,5]  [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.98  0.95 0.93  0.9 0.88  0.86 0.83 0.81  0.8  0.78  0.76  0.75  0.73
## PACF 0.98 -0.04 0.07 -0.1 0.03 -0.03 0.02 0.05  0.0  0.05  0.00  0.02 -0.03
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.72  0.71  0.69  0.68  0.67  0.65  0.64  0.62  0.60  0.58  0.57  0.54
## PACF  0.01  0.02  0.01 -0.01  0.00 -0.01 -0.06 -0.01 -0.11  0.04  0.09 -0.28
##      [,26] [,27] [,28] [,29]
## ACF   0.52  0.50  0.49  0.47
## PACF  0.06  0.14  0.03  0.07
```

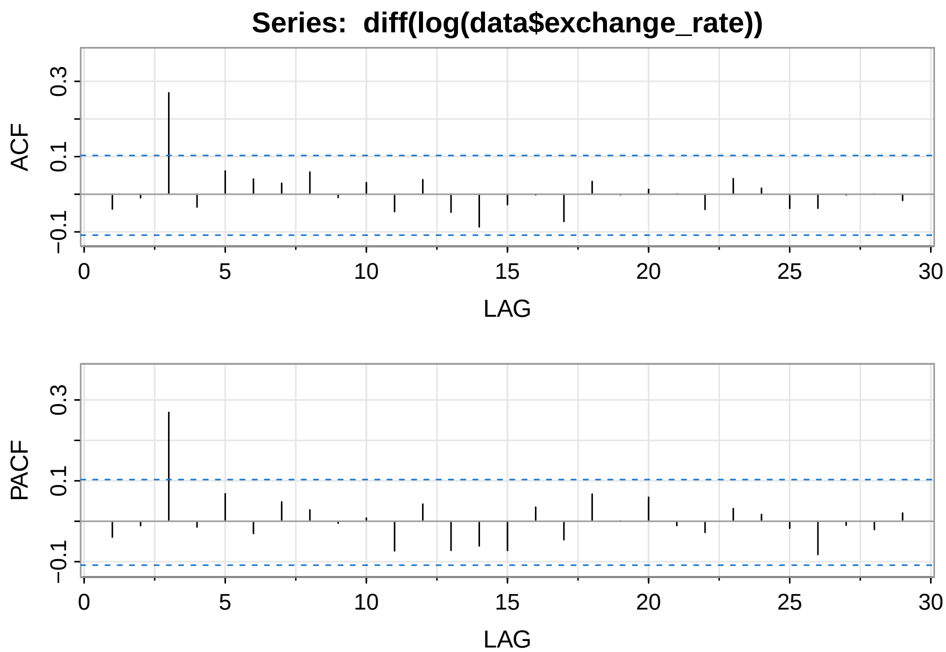
```
acf2(log(data$exchange_rate))
```




```
##      [,1]  [,2] [,3]  [,4] [,5]  [,6] [,7] [,8]  [,9] [,10] [,11] [,12] [,13]
```

```
## ACF  0.99  0.98 0.97  0.95 0.94  0.93 0.92 0.91  0.89  0.88  0.87  0.86  0.85
## PACF 0.99 -0.02 0.00 -0.02 0.00 -0.01 0.00 0.01 -0.01  0.02 -0.01  0.01 -0.01
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.84  0.83  0.82  0.81   0.8  0.79  0.78  0.77  0.76  0.74  0.73  0.72
## PACF  0.00  0.00  0.00 -0.01   0.0 -0.01 -0.02 -0.01 -0.03  0.00  0.03 -0.08
##      [,26] [,27] [,28] [,29]
## ACF   0.71  0.70  0.69  0.68
## PACF  0.00  0.05  0.00  0.03
```

```
acf2(diff(log(data$exchange_rate)))
```




```
##       [,1]  [,2] [,3]  [,4] [,5]  [,6] [,7] [,8]  [,9] [,10] [,11] [,12] [,13]
## ACF  -0.04 -0.01 0.27 -0.03 0.06  0.04 0.03 0.06 -0.01  0.03 -0.05  0.04 -0.05
## PACF -0.04 -0.01 0.27 -0.01 0.07 -0.03 0.05 0.03  0.00  0.01 -0.07  0.04 -0.07
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF  -0.09 -0.03  0.00 -0.07  0.03     0  0.01  0.00 -0.04  0.04  0.02 -0.04
## PACF -0.06 -0.07  0.03 -0.05  0.07     0  0.06 -0.01 -0.03  0.03  0.02 -0.02
##      [,26] [,27] [,28] [,29]
## ACF  -0.04  0.00  0.00 -0.02
## PACF -0.08 -0.01 -0.02  0.02
```

ACF shows similar trend, where only differenced log-transformed exchange rate is stationary. Hence, this differenced and log-transformed exchange rate will be used as one of the external(exogenous) regressors in ARIMAX and GARCHX.

```
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
```

```r
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
```

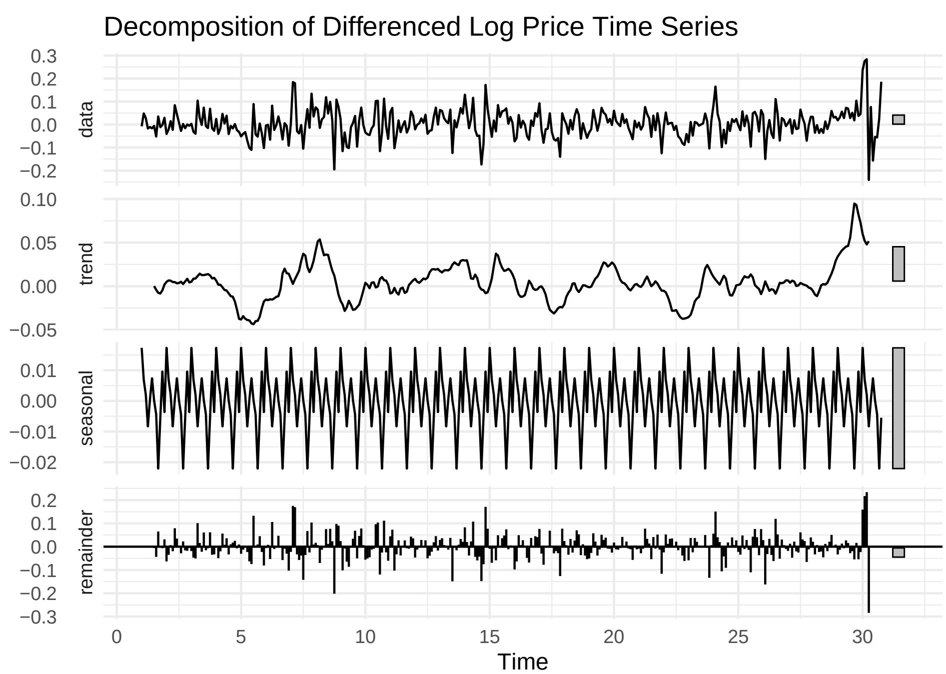
```r
adf.test(data$diff_log_price)
```

```
## 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.

```r
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()
```



1.7 Split data

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

2. Method

2.1 ETS Model

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

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

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)
```

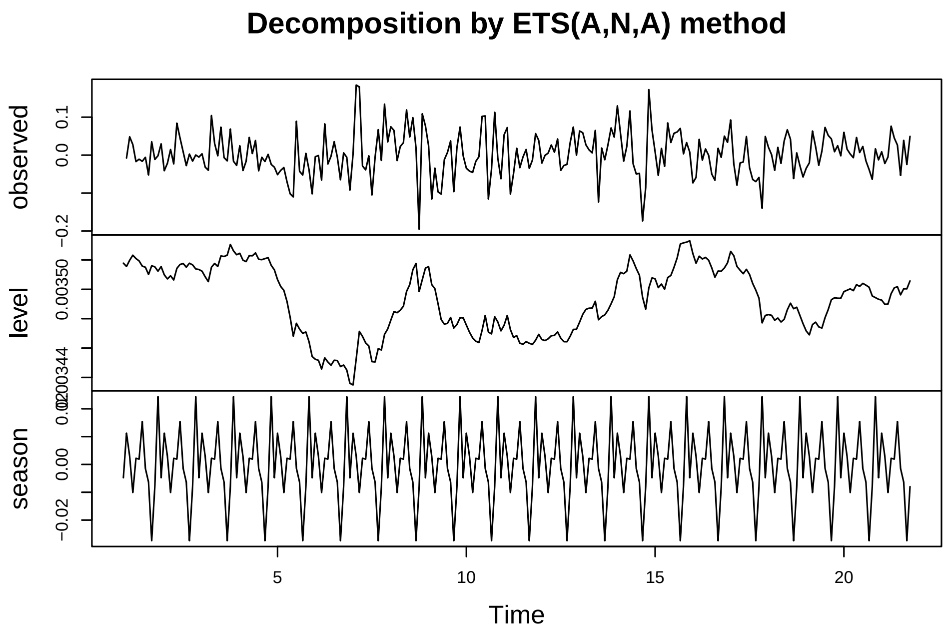
```
## ETS(A,N,A)
##
## Call:
##  ets(y = data_train_ts, model = "ANA")
##
##   Smoothing parameters:
##     alpha = 1e-04
##     gamma = 1e-04
##
##   Initial states:
##     l = 0.0035
##     s = -0.0048 0.0244 -0.008 -0.0274 -0.0064 -0.0014
##            0.0154 0.0019 0.0022 -0.0101 0.0029 0.0112
##
##   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)
```

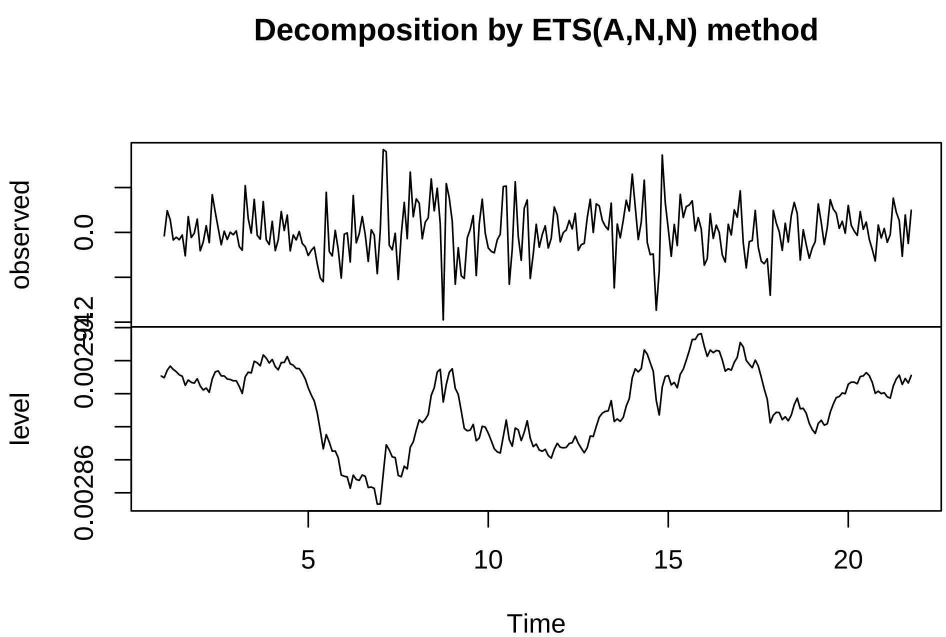
```
## ETS(A,N,N)
##
## Call:
##  ets(y = data_train_ts, model = "ZZZ")
##
##   Smoothing parameters:
##     alpha = 1e-04
##
##   Initial states:
##     l = 0.0029
##
##   sigma:  0.0569
##
##       AIC      AICc       BIC
```

```
## -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
```

```r
plot(ets_model)
```

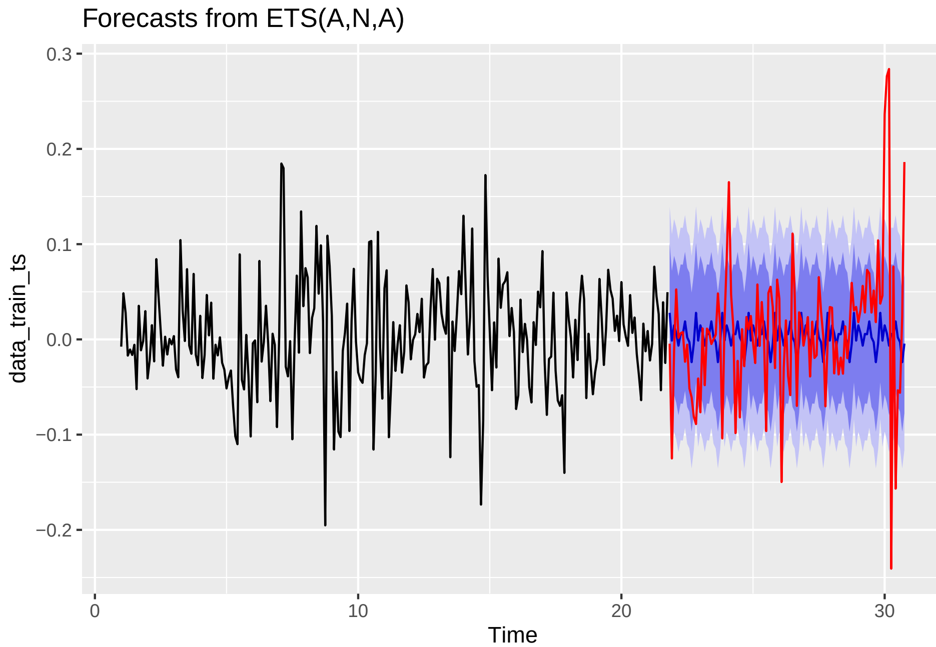



```r
plot(ets_zmodel)
```

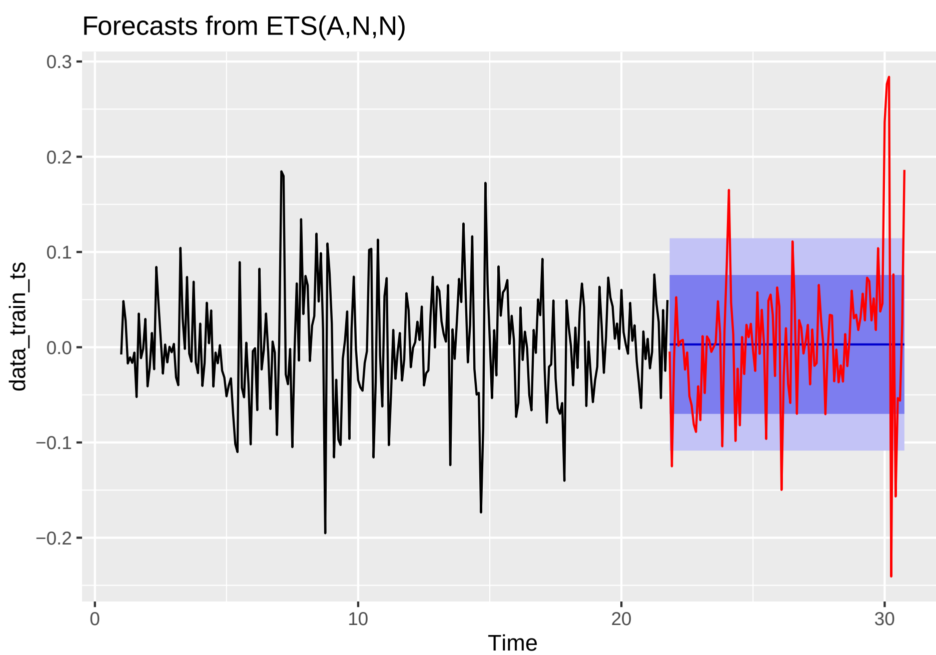


2.1.2 Forecasting and Plotting

```
# 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)
forecast_zets <- forecast(ets_zmodel, h = h)

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




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

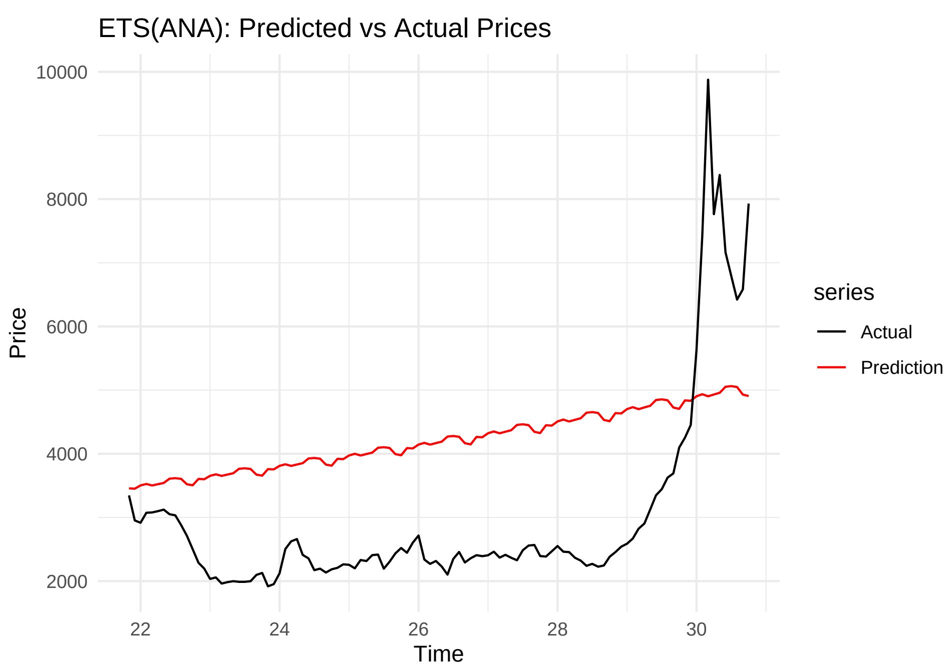


The red line is the observed actual values. The forecasted values are the central blue line within the blue shaded prediction intervals.

```
last_log_price <- tail(trainSet$log_price, 1)

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

actual_price <- exp(testSet$log_price)
```

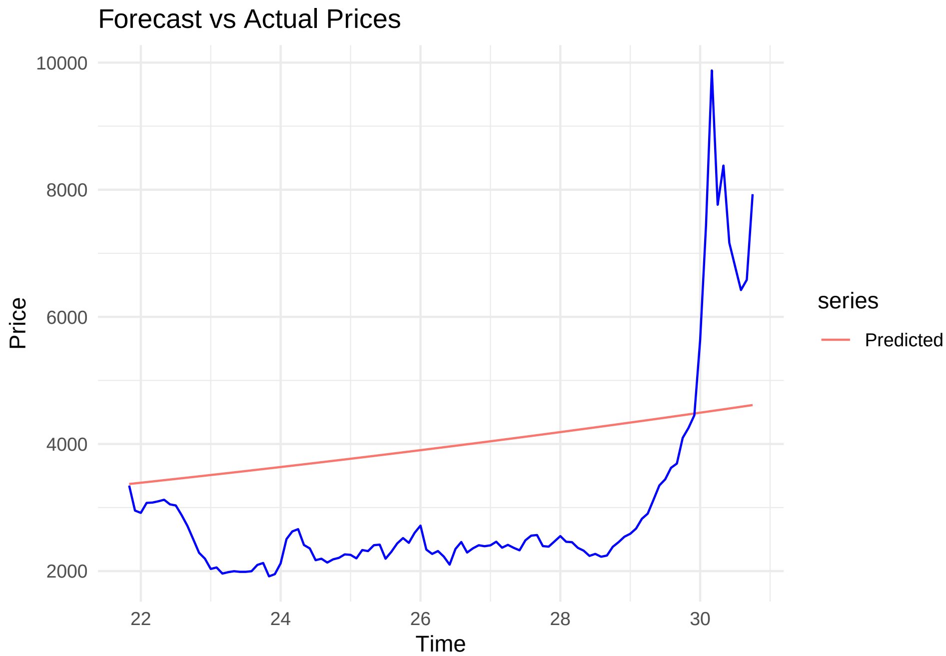
```
data_test_ts <- ts(testSet$diff_log_price, start = end(data_train_ts) + c(0,1),
                   frequency = 12)

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

```
# ANA
p1 <- autoplot(forecast_ets_ts, series = "Prediction") +
  autolayer(actual_ets_ts, series = "Actual") +
  ggtitle("ETS(ANA): Predicted vs Actual Prices") +
  ylab("Price") +
  xlab("Time") +
  theme_minimal()+
  scale_color_manual(values = c(
    "Actual" = "black",
    "Prediction" = "red"))
p1
```

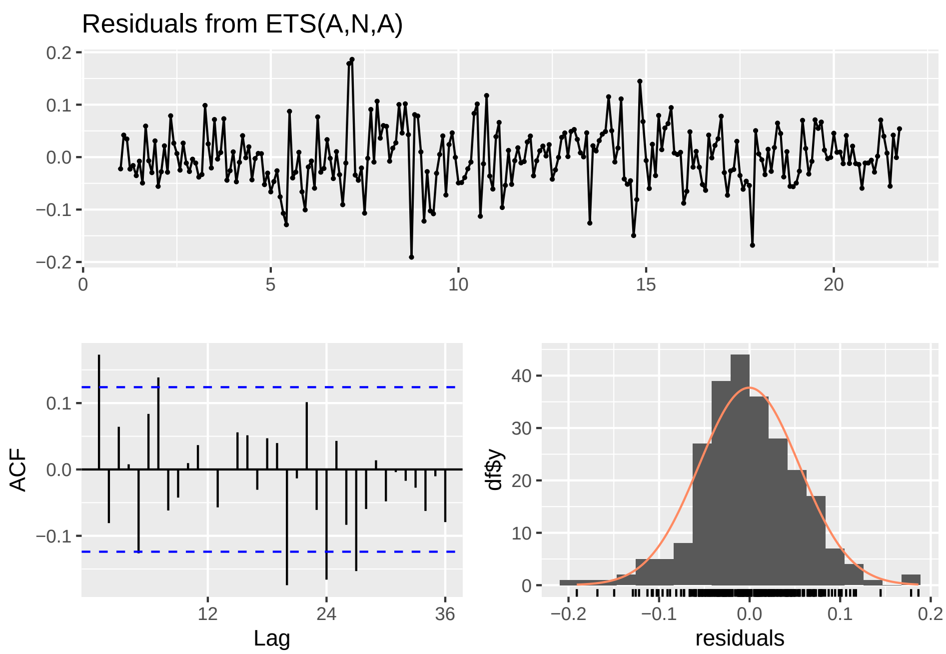



```
# ANN
autoplot(forecast_zets_ts, series = "Predicted") +
  autolayer(actual_ets_ts, series = "Actual", color = "blue") +
  ggtitle("Forecast vs Actual Prices") +
```

```r
  ylab("Price") +
  xlab("Time") +
  theme_minimal()
```



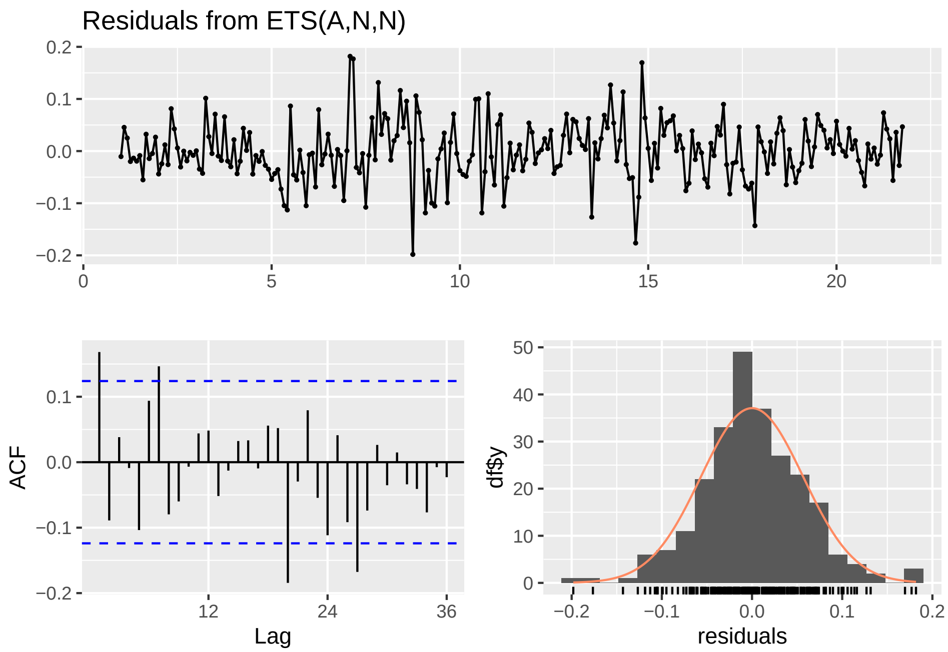

```r
checkresiduals(ets_model)
```




```
##
##  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
```

```r
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
```

```r
# RMSE
sqrt(mean((actual_price - forecasted_price)^2))
```

```
## [1] 1798.181
```

```r
sqrt(mean((actual_price - forecasted_zprice)^2))
```

```
## [1] 1670.782
```

```r
#MAE
mean(abs(actual_ets_ts - forecast_ets_ts))
```

```
## [1] 1674.821
```

```r
mean(abs(actual_ets_ts - forecast_zets_ts))
```

```
## [1] 1509.149
```

```r
# MAPE
mean(abs((actual_ets_ts - forecast_ets_ts) / actual_ets_ts)) * 100
```

```
## [1] 63.88981
```

```r
mean(abs((actual_ets_ts - forecast_zets_ts) / actual_ets_ts)) * 100
```

```
## [1] 56.269
```

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 before we fit the model.

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

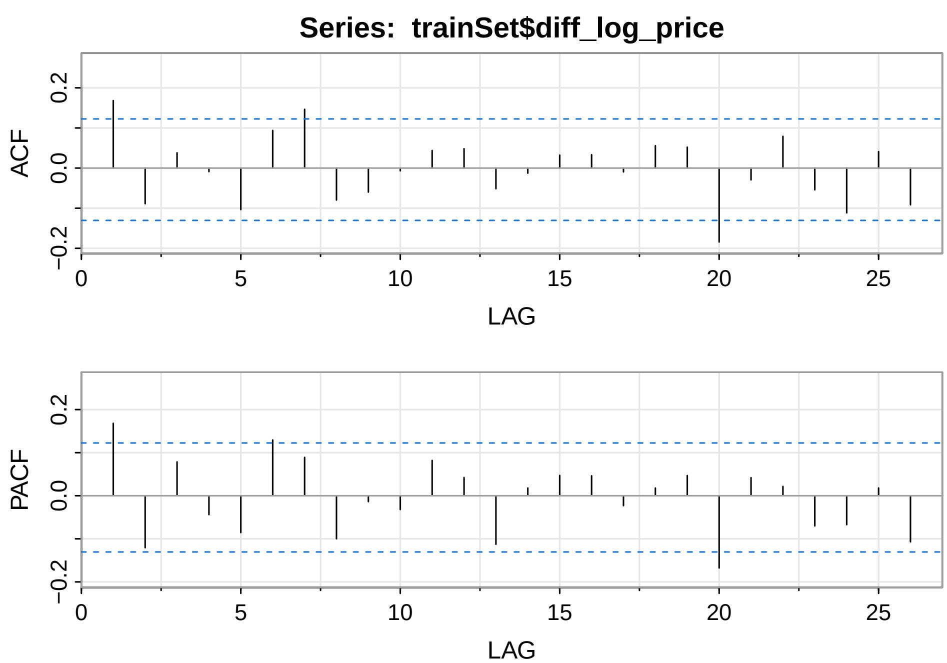
```
## Warning in adf.test(diff(trainSet$month_Price)): p-value smaller than printed
## p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(trainSet$month_Price)
## Dickey-Fuller = -5.2038, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

```
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
```

```
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
```

```r
adf.test(data$Avg_Temp)
```

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

##
##  Augmented Dickey-Fuller Test
##
## data:  data$Avg_Temp
## Dickey-Fuller = -12.411, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

```r
adf.test(diff(log(data$exchange_rate)))
```

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

##
##  Augmented Dickey-Fuller Test
##
## data:  diff(log(data$exchange_rate))
## Dickey-Fuller = -5.3335, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

2.2.1 Fit ARIMAX Model

```r
dl.rate.train <- c(0, diff(log(trainSet$exchange_rate)))
xreg_matrix <- cbind(trainSet$Avg_Temp, dl.rate.train)
colnames(xreg_matrix) <- c("Avg_Temp", "dl_exchange_rate")

p <- 0:10
q <- 0:10
aic.arimax <- matrix(0, length(p), length(q))
for (i in 1:length(p)) {
  for (j in 1:length(q)) {
      modij = Arima(trainSet$diff_log_price, order = c(p[i], 0, q[j]),
                    method = "ML", xreg=xreg_matrix)
      aic.arimax[i, j] = AIC(modij)
  }
}

aic.arimax.min_index <- which(aic.arimax == min(aic.arimax), arr.ind = TRUE)
ariamx.p <- p[aic.arimax.min_index[1]]
ariamx.q <- q[aic.arimax.min_index[2]]
sprintf("Selected order for ARMA: p = %d, q = %d", ariamx.p, ariamx.q)
```

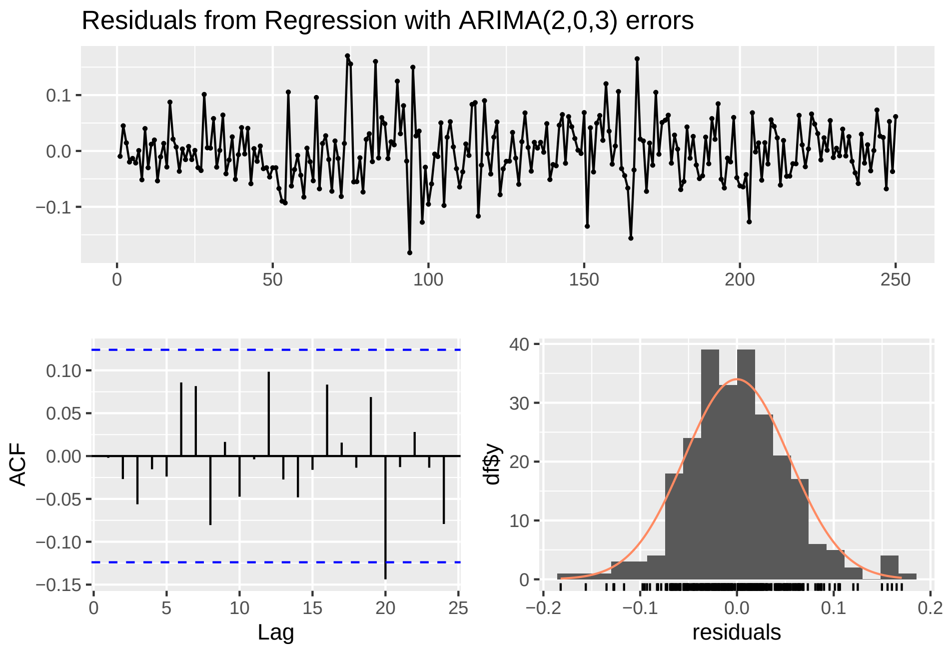
```
## [1] "Selected order for ARMA: p = 2, q = 3"
```

```r
model.arimax <- Arima(trainSet$diff_log_price, order=c(ariamx.p,0,ariamx.q), xreg = xreg_matrix)
summary(model.arimax)
```

```
## Series: trainSet$diff_log_price
## Regression with ARIMA(2,0,3) errors
##
## Coefficients:
##           ar1      ar2     ma1     ma2     ma3  intercept  Avg_Temp
```

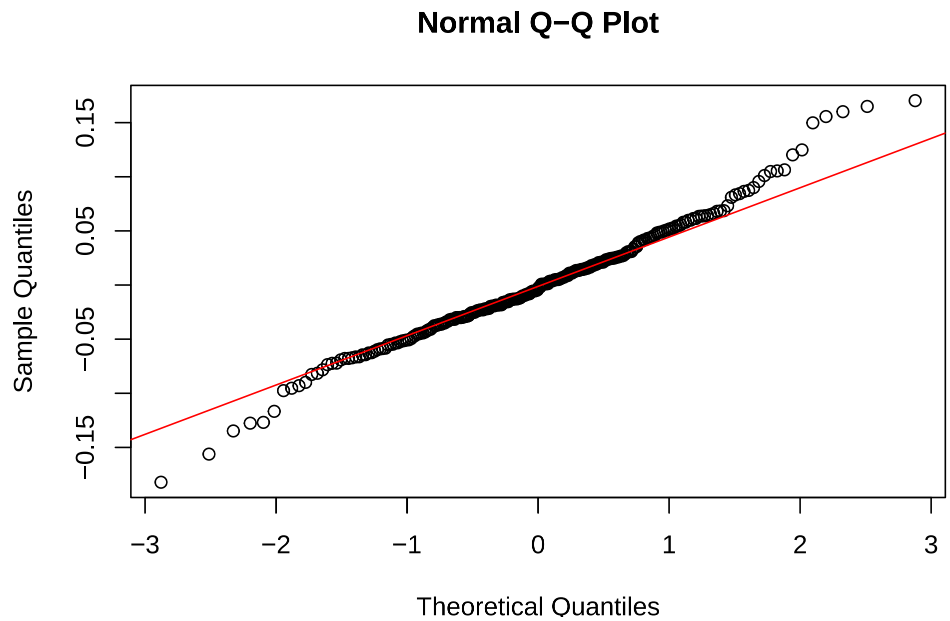
```
##        -0.2295  -0.8933  0.4381  0.9099  0.2695    -0.0090    0.0001
## s.e.    0.0716   0.0884  0.0907  0.0974  0.0622     0.1392    0.0017
##        dl_exchange_rate
##                  0.0398
## s.e.             0.1033
##
## sigma^2 = 0.003047:  log likelihood = 373.33
## AIC=-728.67   AICc=-727.92   BIC=-696.98
##
## Training set error measures:
##                         ME       RMSE        MAE      MPE     MAPE      MASE
## Training set -1.229988e-06 0.05430941 0.04164443 140.2458 203.7395 0.7523938
##                      ACF1
## Training set -0.002035519
```

```r
checkresiduals(model.arimax)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(2,0,3) errors
## Q* = 7.1814, df = 5, p-value = 0.2075
##
## Model df: 5.   Total lags used: 10
```

```r
qqnorm(model.arimax$residuals)
qqline(model.arimax$residuals, col="red")
```




```r
adf.test(model.arimax$residuals)
```

```
## Warning in adf.test(model.arimax$residuals): p-value smaller than printed
## p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  model.arimax$residuals
## Dickey-Fuller = -5.2873, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

- ADF Test on ARIMAX Model Residuals: Failed to reject H_0, indicating that the residuals do not exhibit significant autocorrelation.
- Histogram and QQ-Plot of Residuals: Residuals align well with the 45-degree line, suggesting normality.
- ACF of Residuals: Appears random, with all lags within the range of -0.15 to 0.1, indicating no strong autocorrelations.
- Standardized Residuals Plot: No discernible trend observed, further supporting the model's adequacy.
- Ljung-Box Test (Residuals from ARIMA(2,0,3) model):
  - Q^* = 7.1814, df = 5, p-value = 0.2075
  - Model degrees of freedom: 5, Total lags used: 10 Conclusion: The ARIMAX model effectively captures the trend of the training dataset.

2.2.2 Forecasting With ARIMAX Model

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

```r
dl.rate.test <- c(0, diff(log(testSet$exchange_rate)))
forecast.arimax.xreg <- cbind(testSet$Avg_Temp, dl.rate.test)
colnames(forecast.arimax.xreg) <- c("Avg_Temp", "dl_exchange_rate")
forecast.arimax <- forecast(model.arimax, xreg=forecast.arimax.xreg,
                            h=nrow(testSet))

last_log_price <- tail(trainSet$log_price, 1)
# Convert back to actual price
forecast.arimax.final <- exp(cumsum(forecast.arimax$mean) + last_log_price)
```

```r
model.arimax.fitted <- as.numeric(model.arimax$fitted)
model.arimax.fitted.converted <- exp(log(trainSet$month_Price[1])
                                     + cumsum(model.arimax.fitted))
rmse(trainSet$month_Price, model.arimax.fitted.converted)
```

```
## [1] 376.4071
```

```r
mae(trainSet$month_Price, model.arimax.fitted.converted)
```

```
## [1] 302.0143
```

```r
mape(trainSet$month_Price, model.arimax.fitted.converted)
```

```
## [1] 0.1877152
```

```r
rmse(testSet$month_Price, forecast.arimax.final)
```

```
## [1] 1701.38
```

```r
mae(testSet$month_Price, forecast.arimax.final)
```

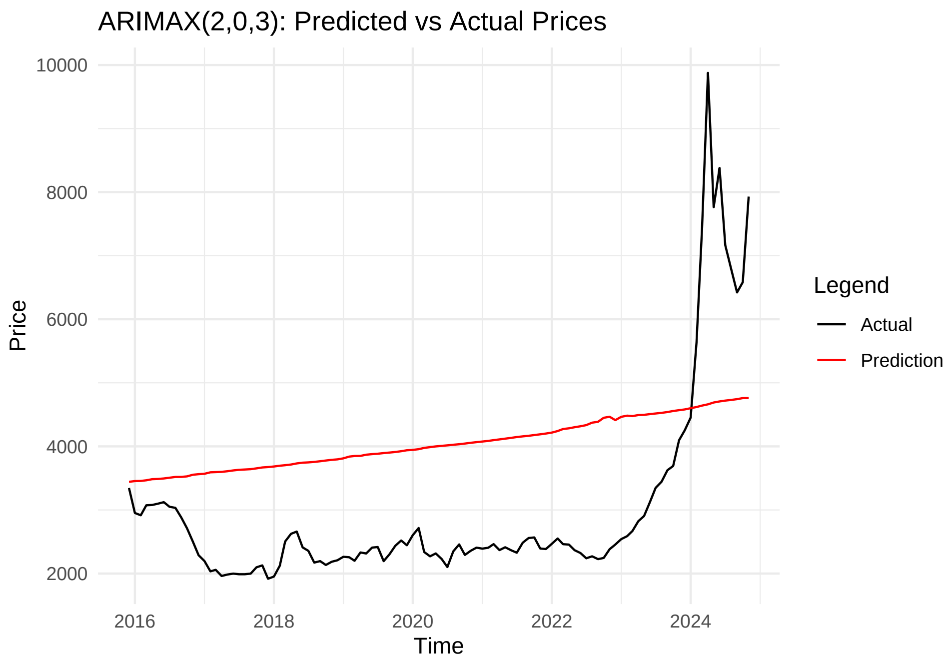
```
## [1] 1559.875
```

```r
mape(testSet$month_Price, forecast.arimax.final)
```

```
## [1] 0.5865196
```

```r
forecast.arimax.df <- tibble(
  Time = testSet$Time,
  Price = forecast.arimax.final
)
test.arimax.df <- tibble(
  Time = testSet$Time,
  Price = testSet$month_Price
)
```

```r
p2 <- ggplot() +
  geom_line(data = test.arimax.df, aes(x = Time, y = Price, color = "Actual")) +
  geom_line(data = forecast.arimax.df, aes(x = Time, y = Price, color = "Prediction")) +
  labs(
    title = "ARIMAX(2,0,3): Predicted vs Actual Prices",
    y = "Price",
    x = "Time",
    color = "Legend"
  ) +
  theme_minimal() +
  scale_color_manual(values = c(
    "Actual" = "black",
    "Prediction" = "red"))
p2
```



2.3 ARMAX-GARCH Model

2.3.1 ARMAX-GARCH Parameters

```
# xreg_matrix is the same as arimax
p = 0:3
q = 0:3
## select ARMA order
aic.armax.garch1 <- matrix(0, length(p), length(q))
for (i in 1:length(p)) {
  for (j in 1:length(q)) {
      modij = Arima(trainSet$diff_log_price, order = c(p[i], 0, q[j]),
                    method = "ML", xreg=xreg_matrix)
      aic.armax.garch1[i, j] = AIC(modij)
  }
}
aic.armax.min_index <- which(aic.armax.garch1 == min(aic.armax.garch1), arr.ind = TRUE)
aramx.garch.p <- p[aic.armax.min_index[1]]
aramx.garch.q <- q[aic.armax.min_index[2]]
sprintf("Selected order for ARMA: p = %d, q = %d", aramx.garch.p, aramx.garch.q)
```

```
## [1] "Selected order for ARMA: p = 2, q = 3"
```

This is the same as what we have for ARIMAX. Then we use the similar method, where we fix the armax order, and systematically search for the combination of orders for garch configuration with smallest AIC.

```
m = 1:3
n = 1:3
# dl.rate.train <- c(0, diff(log(trainSet$exchange_rate)))
# xreg_matrix <- cbind(trainSet$Avg_Temp, dl.rate.train)
# colnames(xreg_matrix) <- c("Avg_Temp", "dl_exchange_rate")
## select GARCH order
aic.armax.garch2 <- matrix(0, length(m), length(n))
for (i in 1:length(m)) {
```

```
  for (j in 1:length(n)) {
      spec = ugarchspec(variance.model=list(model="sGARCH",
                                            garchOrder=c(m[i],n[j])),
                mean.model=list(armaOrder=c(aramx.garch.p, aramx.garch.q),
                                include.mean=T,
                                external.regressors = xreg_matrix),
                distribution.model="std")
      modij = ugarchfit(spec=spec, data = trainSet$diff_log_price,
                        trace = FALSE)
      aic.armax.garch2[i, j] = infocriteria(modij)[1]
  }
}


aic.garch.min_index <- which(aic.armax.garch2 == min(aic.armax.garch2), arr.ind = TRUE)
aramx.garch.m <- m[aic.garch.min_index[1]]
aramx.garch.n <- n[aic.garch.min_index[2]]
sprintf("Selected order for GARCH: m = %d, n = %d", aramx.garch.m, aramx.garch.n)
```

```
## [1] "Selected order for GARCH: m = 1, n = 1"
```

```
armax.garch.spec.train <- ugarchspec(variance.model=list(model="sGARCH",
                                            garchOrder=c(aramx.garch.m,aramx.garch.n)),
                mean.model=list(armaOrder=c(aramx.garch.p, aramx.garch.q),
                                include.mean=T,
                                external.regressors = xreg_matrix),
                distribution.model="std")
model.armax.garch <- ugarchfit(armax.garch.spec.train,
                               data = trainSet$diff_log_price,
                               trace = FALSE)
model.armax.garch
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(2,0,3)
## Distribution : std
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error    t value Pr(>|t|)
## mu      0.020916    0.131218   0.159399 0.873355
## ar1     0.023794    0.040489   0.587662 0.556759
## ar2    -0.894393    0.031402 -28.482252 0.000000
## ma1     0.180535    0.066946   2.696709 0.007003
## ma2     0.953060    0.013924  68.449605 0.000000
## ma3     0.258471    0.064191   4.026575 0.000057
## mxreg1 -0.000227    0.001627  -0.139230 0.889268
## mxreg2 -0.000302    0.101092  -0.002988 0.997616
```

```
## omega   0.000171    0.000126   1.353567 0.175874
## alpha1  0.092076    0.048016   1.917618 0.055160
## beta1   0.858375    0.065245  13.156115 0.000000
## shape   6.610258    3.242791   2.038447 0.041505
##
## Robust Standard Errors:
##         Estimate  Std. Error    t value Pr(>|t|)
## mu      0.020916    0.150658   0.138831 0.889584
## ar1     0.023794    0.049430   0.481361 0.630260
## ar2    -0.894393    0.034242 -26.120012 0.000000
## ma1     0.180535    0.061616   2.929988 0.003390
## ma2     0.953060    0.016171  58.937321 0.000000
## ma3     0.258471    0.063495   4.070706 0.000047
## mxreg1 -0.000227    0.001881  -0.120404 0.904163
## mxreg2 -0.000302    0.129392  -0.002335 0.998137
## omega   0.000171    0.000092   1.858301 0.063126
## alpha1  0.092076    0.040356   2.281594 0.022513
## beta1   0.858375    0.043446  19.757096 0.000000
## shape   6.610258    2.908315   2.272883 0.023033
##
## LogLikelihood : 383.864
##
## Information Criteria
## ------------------------------------
##
## Akaike       -2.9749
## Bayes        -2.8059
## Shibata      -2.9792
## Hannan-Quinn -2.9069
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                          statistic p-value
## Lag[1]                    0.004823  0.9446
## Lag[2*(p+q)+(p+q)-1][14]  4.789866  1.0000
## Lag[4*(p+q)+(p+q)-1][24]  9.750399  0.8540
## d.o.f=5
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.3107  0.5773
## Lag[2*(p+q)+(p+q)-1][5]    1.5176  0.7356
## Lag[4*(p+q)+(p+q)-1][9]    3.1776  0.7293
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.6921 0.500 2.000  0.4054
## ARCH Lag[5]    2.3180 1.440 1.667  0.4051
## ARCH Lag[7]    3.3988 2.315 1.543  0.4416
##
```

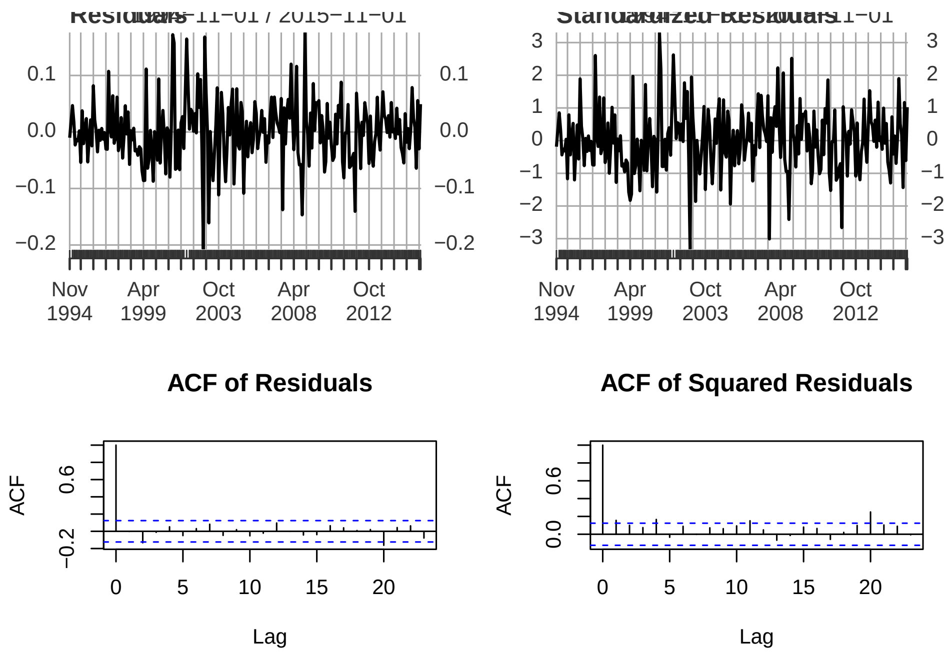
```
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1.9549
## Individual Statistics:
## mu     0.21144
## ar1    0.33765
## ar2    0.21800
## ma1    0.09682
## ma2    0.14306
## ma3    0.09044
## mxreg1 0.20586
## mxreg2 0.27457
## omega  0.09039
## alpha1 0.08056
## beta1  0.09022
## shape  0.12348
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          2.69 2.96 3.51
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias          0.65211 0.5149
## Negative Sign Bias 0.01309 0.9896
## Positive Sign Bias 0.29201 0.7705
## Joint Effect       1.49229 0.6840
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     12.56       0.8603
## 2    30     25.28       0.6636
## 3    40     35.12       0.6475
## 4    50     44.40       0.6599
##
##
## Elapsed time : 0.7453628
```

```r
model.armax.garch@fit$coef
```

```
##            mu           ar1           ar2           ma1           ma2
##  0.0209159682  0.0237936527 -0.8943927883  0.1805351003  0.9530600211
##           ma3        mxreg1        mxreg2         omega        alpha1
##  0.2584712375 -0.0002265262 -0.0003021024  0.0001710821  0.0920758276
##         beta1         shape
##  0.8583750421  6.6102581111
```

```r
garch_time_index <- as.POSIXct(trainSet$Time)
residuals_armax_garch_xts <- xts(residuals(model.armax.garch),
                                 order.by = garch_time_index)
std_resid_armax_garch_xts <- xts(model.armax.garch@fit$z,
                                 order.by = garch_time_index)
```

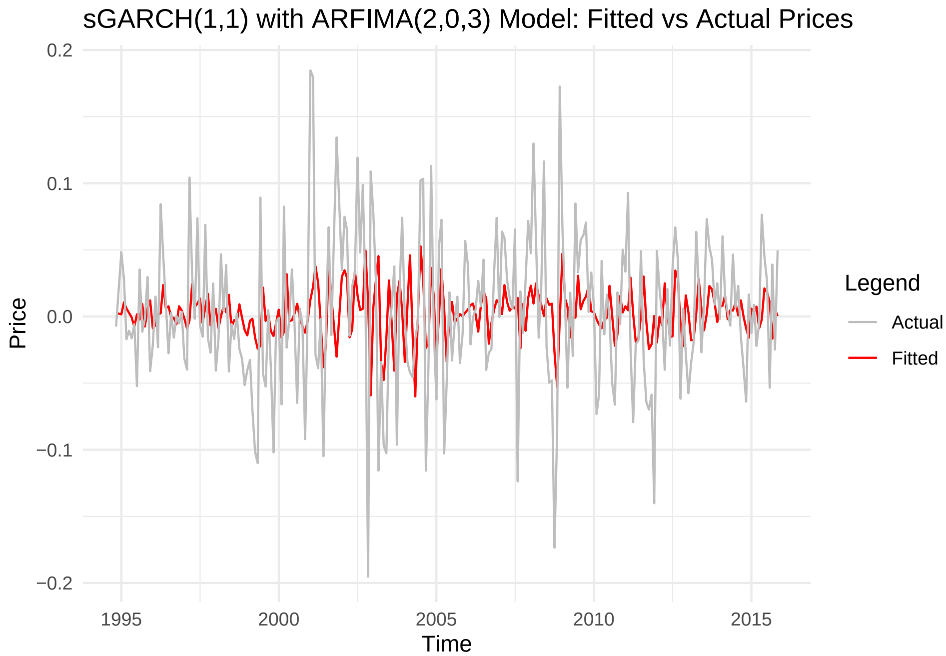
```r
# Residual Analysis
par(mfrow = c(2, 2))

# Residual plots
plot(residuals_armax_garch_xts, main = "Residuals")
plot(std_resid_armax_garch_xts, main = "Standardized Residuals")

# ACF plots
acf(na.omit(as.numeric(residuals(model.armax.garch))), main = "ACF of Residuals")
acf(na.omit(as.numeric(residuals(model.armax.garch)^2)), main = "ACF of Squared Residuals")
```




```r
armax.garch.actual.values <- trainSet$diff_log_price
armax.garch.fitted.values <- as.numeric(fitted(model.armax.garch))

armax.garch.fit.df <- tibble(
  Time = trainSet$Time,
  Price = armax.garch.fitted.values
)
armax.garch.train.df <- tibble(
  Time = trainSet$Time,
  Price = trainSet$diff_log_price
)
```

```r
ggplot() +
  geom_line(data = armax.garch.fit.df, aes(x = Time, y = Price, color = "Fitted")) +
  geom_line(data = armax.garch.train.df, aes(x = Time, y = Price, color = "Actual")) +
  labs(
    title = "sGARCH(1,1) with ARFIMA(2,0,3) Model: Fitted vs Actual Prices",
    y = "Price",
    x = "Time",
    color = "Legend"
  ) +
  theme_minimal() +
```

```r
  scale_color_manual(values = c("Actual" = "grey", "Fitted" = "red"))
```



2.3.2 ARMAX-GARCH Forecast

```r
# multi-step forecast
ngarchfore = length(testSet$diff_log_price)
xreg_test_matrix <- cbind(testSet$Avg_Temp, diff(log(testSet$exchange_rate)))
```

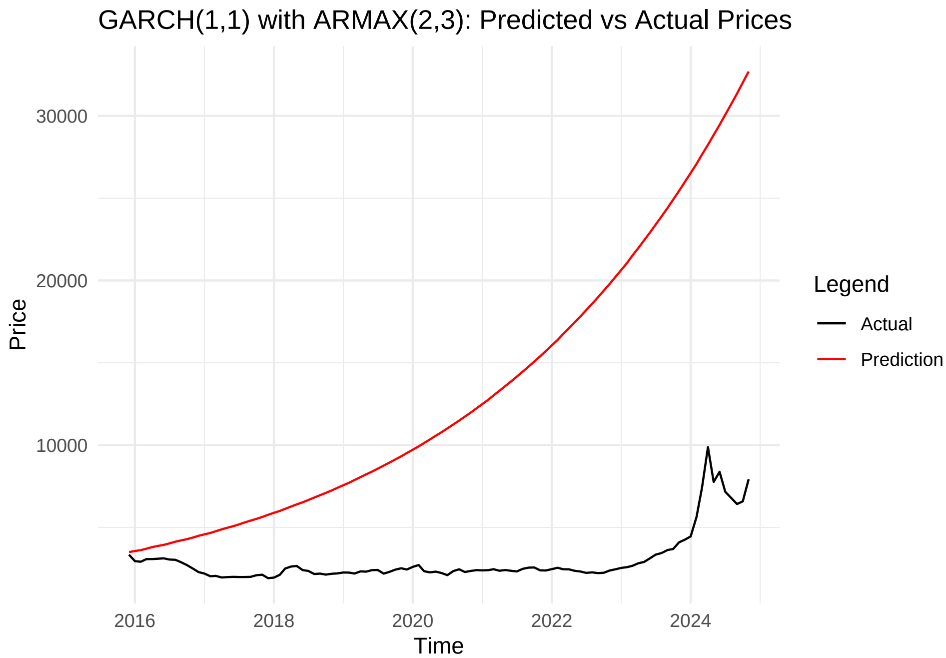
```
## Warning in cbind(testSet$Avg_Temp, diff(log(testSet$exchange_rate))): number of
## rows of result is not a multiple of vector length (arg 2)
```

```r
colnames(xreg_test_matrix) <- c("Avg_Temp", "dl_exchange_rate")

fore.garch.dl = ugarchforecast(model.armax.garch,
                               n.ahead = ngarchfore,
                              external.forecasts = list(mreg=xreg_test_matrix))
fore.garch.dl.data <- as.numeric(fore.garch.dl@forecast$seriesFor)
last_log_price <- tail(trainSet$log_price, 1)
forecast.armax.garch.multi <- exp(cumsum(fore.garch.dl.data) + last_log_price)
```

```r
library(tibble)
forecast.garch.multi.df <- tibble(
  Time = testSet$Time,
  Price = forecast.armax.garch.multi
)
test.garch.df <- tibble(
  Time = testSet$Time,
  Price = testSet$month_Price
)
```

```r
p3 <- ggplot() +
  geom_line(data = test.garch.df, aes(x = Time, y = Price, color = "Actual")) +
  geom_line(data = forecast.garch.multi.df, aes(x = Time, y = Price, color = "Prediction")) +
```

```r
  labs(
    title = "GARCH(1,1) with ARMAX(2,3): Predicted vs Actual Prices",
    y = "Price",
    x = "Time",
    color = "Legend"
  ) +
  theme_minimal() +
  scale_color_manual(values = c("Actual" = "black", "Prediction" = "red"))
p3
```




```r
model.armax.garch.fitted <- as.numeric(fitted(model.armax.garch))
model.armax.garch.fitted.converted <- exp(log(trainSet$month_Price[1])
                                          + cumsum(model.armax.garch.fitted))
rmse(trainSet$month_Price, model.armax.garch.fitted.converted)
```

```
## [1] 383.951
```

```r
mae(trainSet$month_Price, model.armax.garch.fitted.converted)
```

```
## [1] 299.0776
```

```r
mape(trainSet$month_Price, model.armax.garch.fitted.converted)
```

```
## [1] 0.1682957
```

```r
rmse(testSet$month_Price, forecast.armax.garch.multi[1:length(testSet$Time)])
```

```
## [1] 12424.11
```

```r
mae(testSet$month_Price, forecast.armax.garch.multi[1:length(testSet$Time)])
```

```
## [1] 10118.71
```

```r
mape(testSet$month_Price, forecast.armax.garch.multi[1:length(testSet$Time)])
```

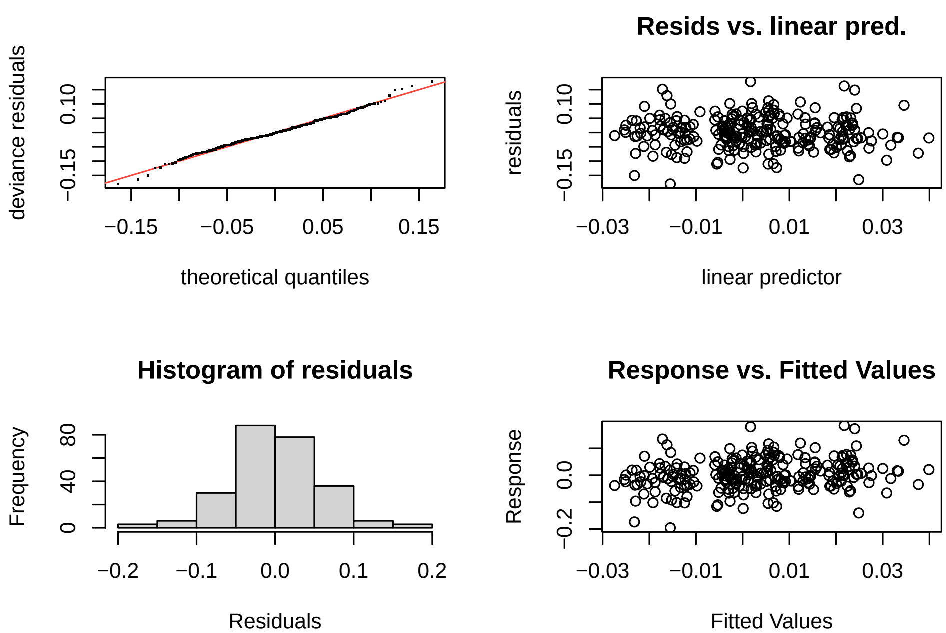
```
## [1] 3.424606
```

2.4 GAM Model

2.4.1 Fit Model

```
trainSet$Time = as.Date(trainSet$Time)
trainSet$monthFac = as.factor(format(trainSet$Time, "%m"))
trainSet$Ndays = days_in_month(trainSet$Time)
trainSet$logdays = log(trainSet$Ndays)

gam_model <- gam(diff_log_price ~ s(as.numeric(Time), k=12) + s(Avg_Temp) + s(exchange_rate) +
                   s(monthFac, bs = "re") + sinpi(yday(Time) / 182.625) +
                   cospi(yday(Time) / 182.625) + sinpi(yday(Time) / 91.3125) +
                   cospi(yday(Time) / 91.3125) + offset(logdays),
                   data = trainSet, method = "ML", family = gaussian())

gam_model2 <- gam(diff_log_price ~ s(as.numeric(Time), k=100) + s(Avg_Temp) +
                   s(log(exchange_rate)) + s(monthFac, bs = "re") + sinpi(yday(Time) / 182.625) +
                   cospi(yday(Time) / 182.625) + sinpi(yday(Time) / 91.3125) +
                   cospi(yday(Time) / 91.3125) + offset(logdays),
                   data = trainSet, method = "REML")

gam_model3 <- gam(diff_log_price ~ s(as.numeric(Time), k=100) + s(Avg_Temp) +
                   s(log(exchange_rate)) + s(monthFac, bs = "re") +
                    s(yday(Time), bs = "cc", k = 10) + offset(logdays),
                   data = trainSet, method = "REML")
```

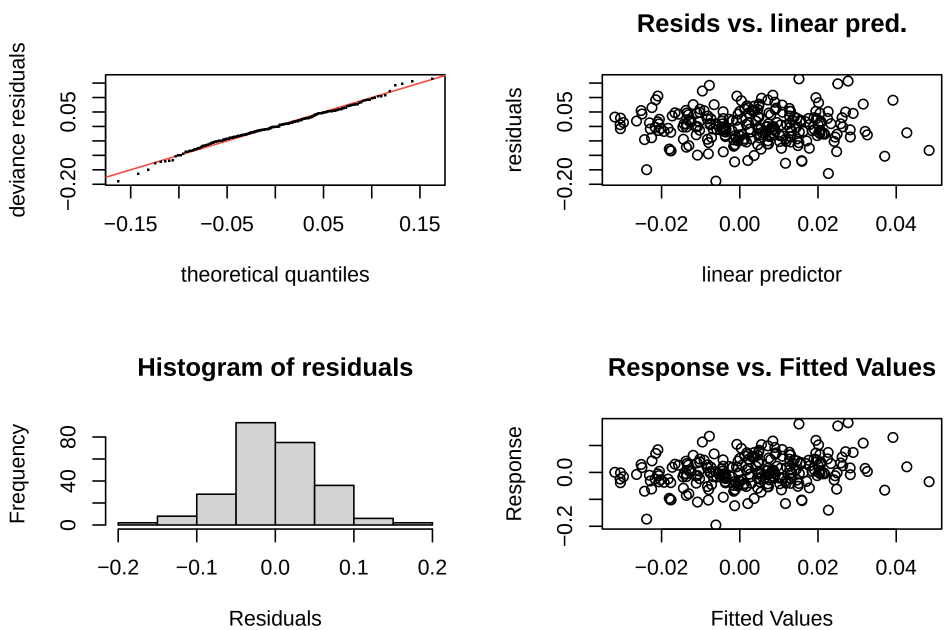
```
gam.check(gam_model)
```




```
##
## Method: ML   Optimizer: outer newton
## full convergence after 11 iterations.
## Gradient range [-0.0001044506,4.709133e-05]
## (score -354.0039 & scale 0.003234713).
## Hessian positive definite, eigenvalue range [1.846249e-05,125.1959].
## Model rank =  46 / 46
```

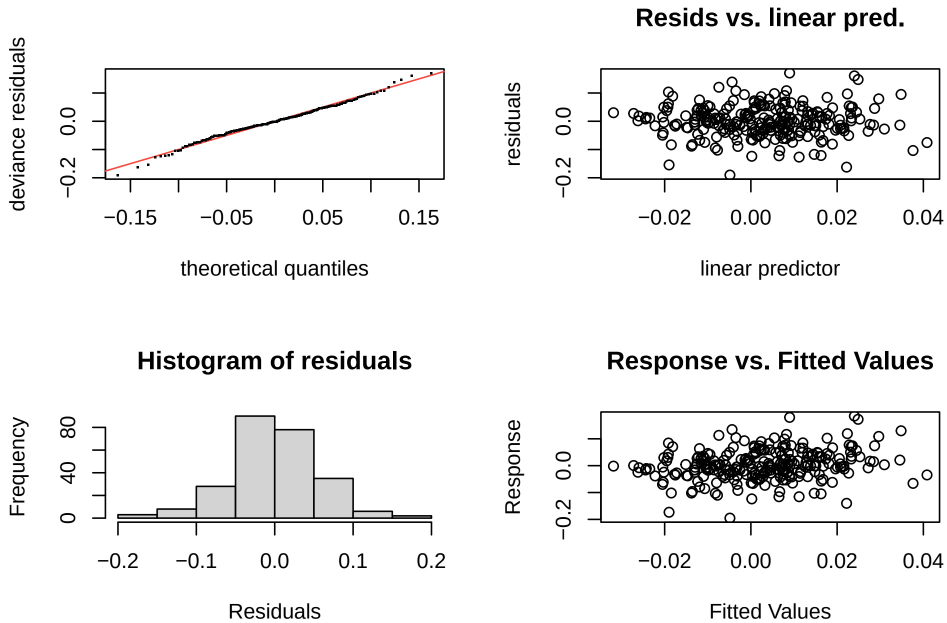
```
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                          k'   edf k-index p-value
## s(as.numeric(Time)) 11.00  1.00    0.86   0.005 **
## s(Avg_Temp)          9.00  1.00    1.06   0.800
## s(exchange_rate)     9.00  1.00    0.87   0.030 *
## s(monthFac)         12.00  5.67      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
gam.check(gam_model2)
```




```
##
## Method: REML   Optimizer: outer newton
## full convergence after 12 iterations.
## Gradient range [-0.0001138432,0.0001596367]
## (score -325.3884 & scale 0.003205777).
## Hessian positive definite, eigenvalue range [1.228084e-05,121.0816].
## Model rank =  134 / 134
##
## 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(as.numeric(Time))   99.00  1.00    0.86   0.010 **
## s(Avg_Temp)            9.00  1.00    1.05   0.735
## s(log(exchange_rate))  9.00  1.00    0.88   0.025 *
## s(monthFac)           12.00  6.22      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
gam.check(gam_model3)
```




```
##
## Method: REML   Optimizer: outer newton
## full convergence after 13 iterations.
## Gradient range [-3.368377e-05,8.44348e-05]
## (score -336.8505 & scale 0.003206499).
## Hessian positive definite, eigenvalue range [1.109847e-05,123.1779].
## Model rank =  138 / 138
##
## 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(as.numeric(Time))   9.90e+01 1.00e+00    0.85  <2e-16 ***
## s(Avg_Temp)           9.00e+00 1.00e+00    1.05   0.710
## s(log(exchange_rate)) 9.00e+00 1.00e+00    0.88   0.025 *
## s(monthFac)           1.20e+01 9.21e+00      NA      NA
## s(yday(Time))         8.00e+00 7.79e-05    1.20   0.995
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(gam_model)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(as.numeric(Time), k = 12) + s(Avg_Temp) +
##     s(exchange_rate) + s(monthFac, bs = "re") + sinpi(yday(Time)/182.625) +
##     cospi(yday(Time)/182.625) + sinpi(yday(Time)/91.3125) + cospi(yday(Time)/91.3125) +
##     offset(logdays)
##
## Parametric coefficients:
```

```
##                            Estimate Std. Error  t value Pr(>|t|)
## (Intercept)               -3.412302   0.008311 -410.557   <2e-16 ***
## sinpi(yday(Time)/182.625)  0.013344   0.013951    0.957    0.340
## cospi(yday(Time)/182.625)  0.015835   0.015280    1.036    0.301
## sinpi(yday(Time)/91.3125)  0.009482   0.011766    0.806    0.421
## cospi(yday(Time)/91.3125)  0.014453   0.012311    1.174    0.242
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                       edf Ref.df     F p-value
## s(as.numeric(Time)) 1.000      1 0.072   0.788
## s(Avg_Temp)         1.000      1 0.108   0.742
## s(exchange_rate)    1.000      1 0.460   0.498
## s(monthFac)         5.672     11 4.562  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  -0.0036   Deviance explained = 26.3%
## -ML =   -354  Scale est. = 0.0032347  n = 250
```

```r
summary(gam_model2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(as.numeric(Time), k = 100) + s(Avg_Temp) +
##     s(log(exchange_rate)) + s(monthFac, bs = "re") + sinpi(yday(Time)/182.625) +
##     cospi(yday(Time)/182.625) + sinpi(yday(Time)/91.3125) + cospi(yday(Time)/91.3125) +
##     offset(logdays)
##
## Parametric coefficients:
##                            Estimate Std. Error  t value Pr(>|t|)
## (Intercept)               -3.412316   0.010789 -316.273   <2e-16 ***
## sinpi(yday(Time)/182.625)  0.013120   0.017024    0.771    0.442
## cospi(yday(Time)/182.625)  0.015518   0.018129    0.856    0.393
## sinpi(yday(Time)/91.3125)  0.009725   0.015240    0.638    0.524
## cospi(yday(Time)/91.3125)  0.014619   0.015713    0.930    0.353
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                         edf Ref.df     F p-value
## s(as.numeric(Time))   1.000  1.001 1.089   0.298
## s(Avg_Temp)           1.000  1.000 0.095   0.758
## s(log(exchange_rate)) 1.000  1.000 1.683   0.196
## s(monthFac)           6.221 11.000 5.019  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.00538   Deviance explained = 27.2%
## -REML = -325.39  Scale est. = 0.0032058  n = 250
```

```r
summary(gam_model3)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(as.numeric(Time), k = 100) + s(Avg_Temp) +
##     s(log(exchange_rate)) + s(monthFac, bs = "re") + s(yday(Time),
##     bs = "cc", k = 10) + offset(logdays)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.41221    0.01009  -338.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                             edf Ref.df     F p-value
## s(as.numeric(Time))   1.000e+00      1 1.164   0.282
## s(Avg_Temp)           1.000e+00      1 0.086   0.770
## s(log(exchange_rate)) 1.000e+00      1 1.812   0.180
## s(monthFac)           9.215e+00     11 6.357  <2e-16 ***
## s(yday(Time))         7.789e-05      8 0.000   0.628
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.00516   Deviance explained = 26.8%
## -REML = -336.85  Scale est. = 0.0032065  n = 250
```

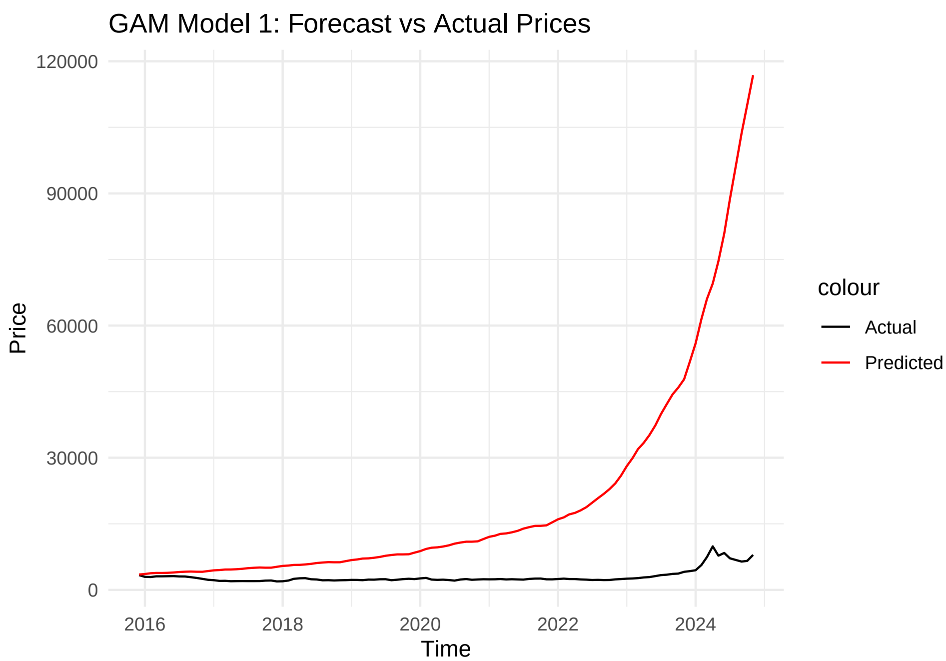
2.4.2 Forecast and Plot

```r
testSet$Time = as.Date(testSet$Time)
testSet$monthFac = as.factor(format(testSet$Time, "%m"))
# trainSet$month_num = as.numeric(trainSet$monthFac)
# trainSet$timeNumeric = as.numeric(trainSet$date)
testSet$Ndays = days_in_month(testSet$Time)
testSet$logdays = log(testSet$Ndays)

testSet2 <- testSet
testSet2$log_exchange_rate <- log(testSet2$exchange_rate)
```

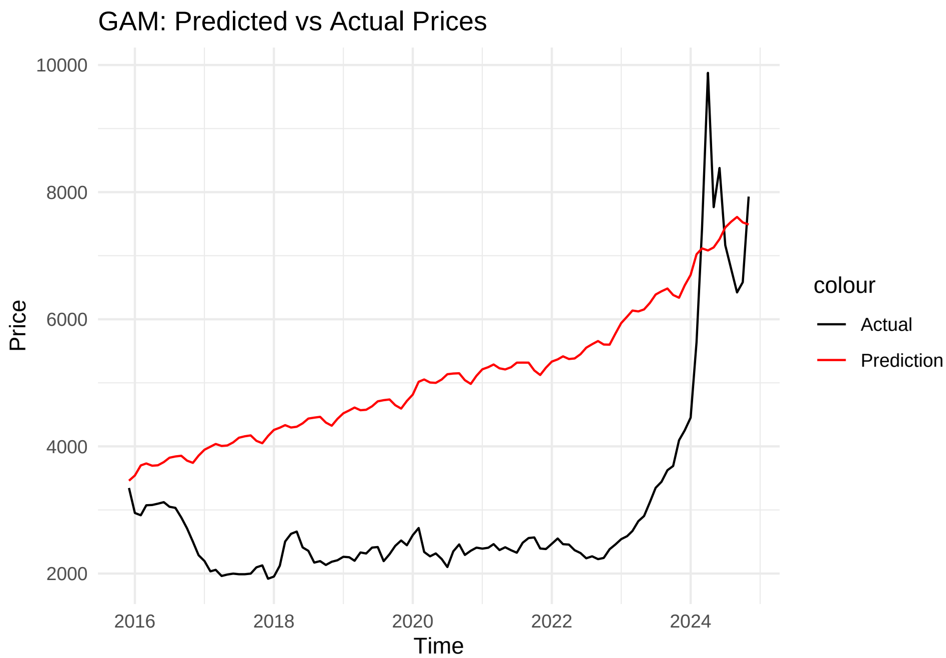
```r
# gam1
testSet$pred_log <- predict(gam_model, newdata = testSet)
testSet$pred_log_price <- last_log_price + cumsum(testSet$pred_log)
testSet$pred_price <- exp(testSet$pred_log_price)

# gam2
testSet2$pred_log2 <- predict(gam_model2, newdata = testSet2)
testSet2$pred_log_price2 <- last_log_price + cumsum(testSet2$pred_log2)
testSet2$pred_price2 <- exp(testSet2$pred_log_price2)

# gam3
testSet2$pred_log3 <- predict(gam_model3, newdata = testSet2)
```

```r
testSet2$pred_log_price3 <- last_log_price + cumsum(testSet2$pred_log3)
testSet2$pred_price3 <- exp(testSet2$pred_log_price3)
```

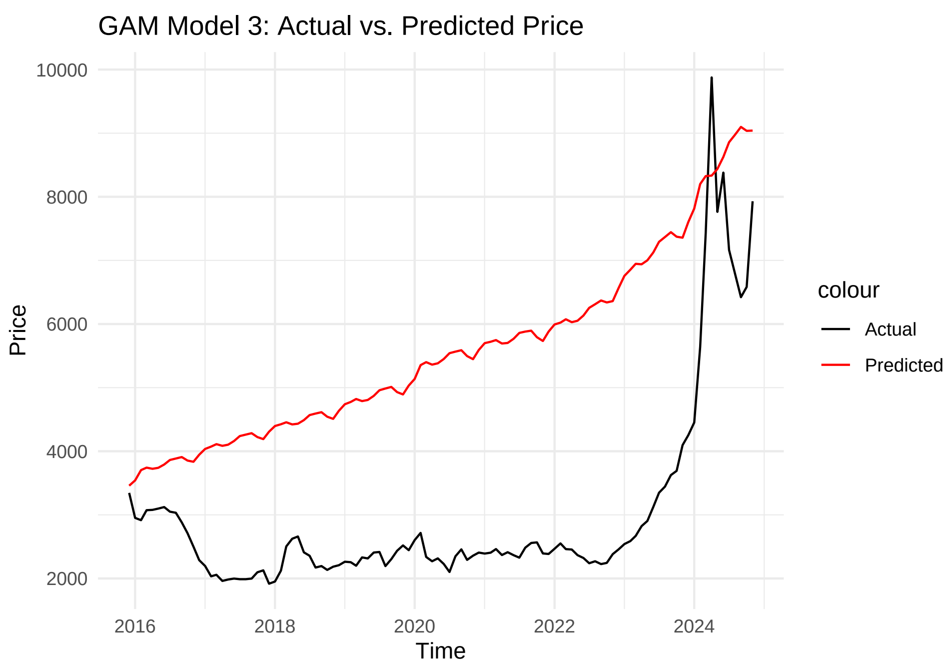
```r
ggplot(testSet, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price, color = "Predicted")) +
  labs(title = "GAM Model 1: Forecast vs Actual Prices",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red")) +
  theme_minimal()
```




```r
p4 <- ggplot(testSet2, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price2, color = "Prediction")) +
  labs(title = "GAM: Predicted vs Actual Prices",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Prediction" = "red")) +
  theme_minimal()
p4
```




```r
ggplot(testSet2, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price3, color = "Predicted")) +
  labs(title = "GAM Model 3: Actual vs. Predicted Price",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red")) +
  theme_minimal()
```




```r
anova(gam_model, gam_model2, gam_model3)
```

```
## Analysis of Deviance Table
##
## Model 1: diff_log_price ~ s(as.numeric(Time), k = 12) + s(Avg_Temp) +
```

```
##     s(exchange_rate) + s(monthFac, bs = "re") + sinpi(yday(Time)/182.625) +
##     cospi(yday(Time)/182.625) + sinpi(yday(Time)/91.3125) + cospi(yday(Time)/91.3125) +
##     offset(logdays)
## Model 2: diff_log_price ~ s(as.numeric(Time), k = 100) + s(Avg_Temp) +
##     s(log(exchange_rate)) + s(monthFac, bs = "re") + sinpi(yday(Time)/182.625) +
##     cospi(yday(Time)/182.625) + sinpi(yday(Time)/91.3125) + cospi(yday(Time)/91.3125) +
##     offset(logdays)
## Model 3: diff_log_price ~ s(as.numeric(Time), k = 100) + s(Avg_Temp) +
##     s(log(exchange_rate)) + s(monthFac, bs = "re") + s(yday(Time),
##     bs = "cc", k = 10) + offset(logdays)
##   Resid. Df Resid. Dev        Df   Deviance
## 1    234.79    0.76445
## 2    234.75    0.75585  0.041669  0.0085981
## 3    235.10    0.75925 -0.347735 -0.0033964
```

```r
# RMSE
sqrt(mean((testSet$month_Price - testSet$pred_price)^2))
```

```
## [1] 29140.14
```

```r
sqrt(mean((testSet2$month_Price - testSet2$pred_price2)^2))
```

```
## [1] 2368.678
```

```r
sqrt(mean((testSet2$month_Price - testSet2$pred_price3)^2))
```

```
## [1] 2819.93
```

```r
# MAE
mean(abs(testSet$month_Price - testSet$pred_price))
```

```
## [1] 17292.64
```

```r
mean(abs(testSet2$month_Price - testSet2$pred_price2))
```

```
## [1] 2213.002
```

```r
mean(abs(testSet2$month_Price - testSet2$pred_price3))
```

```
## [1] 2623.059
```

```r
# MAPE
mean(abs((testSet$month_Price - testSet$pred_price) / testSet$month_Price)) * 100
```

```
## [1] 468.1819
```

```r
mean(abs((testSet2$month_Price - testSet2$pred_price2) / testSet2$month_Price)) * 100
```

```
## [1] 88.60513
```

```r
mean(abs((testSet2$month_Price - testSet2$pred_price3) / testSet2$month_Price)) * 100
```

```
## [1] 103.173
```

2.5 Walk-Forward Validation with XGBoost Model

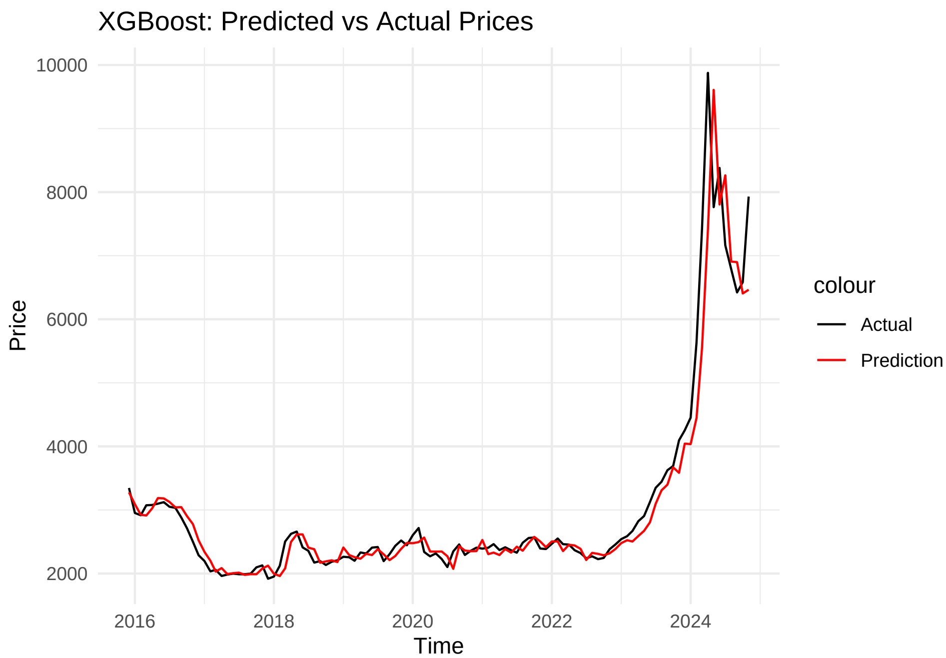
2.5.1 Fit and Forecast

```r
ntest <- nrow(data) - cutoff
predictions <- c()
```

```r
actuals <- c()
dates <- c()
```

```r
data$monthFac <- as.factor(format(data$Time, "%m"))
data$Time <- as.numeric(as.Date(data$Time))
data$monthFac <- as.numeric(data$monthFac)
data$log_exchange_rate <- log(data$exchange_rate)
features <- c("monthFac", "Time", "Avg_Temp", "log_exchange_rate")
for (i in 1:ntest) {
  train_data <- data[1:(cutoff + i - 1), ]
  test_data <- data[(cutoff + i), ]
  x_train <- train_data %>% select(all_of(features))
  y_train <- train_data$log_price
  x_test <- test_data %>% select(all_of(features))
  dtrain <- xgb.DMatrix(data = as.matrix(x_train), label = y_train)
  dtest <- xgb.DMatrix(data = as.matrix(x_test))
  xgb_model <- xgboost(data = dtrain, nrounds = 100, objective = "reg:squarederror", verbose = 0)
  pred_log <- predict(xgb_model, dtest)
  pred_price <- exp(pred_log)
  predictions <- c(predictions, pred_price)
  actuals <- c(actuals, exp(test_data$log_price))
  dates <- c(dates, test_data$Time)
}

xgb_walk_df <- tibble(Time = as.Date(dates),
                      Actual = actuals,
                      Predicted = predictions)
```

```r
p5 <- ggplot(xgb_walk_df, aes(x = Time)) + geom_line(aes(y = Actual, color = "Actual")) +
  geom_line(aes(y = Predicted, color = "Prediction")) +
  labs(title = "XGBoost: Predicted vs Actual Prices", x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Prediction" = "red")) +
  theme_minimal()
p5
```




```r
# RMSE
sqrt(mean((xgb_walk_df$Actual - xgb_walk_df$Predicted)^2))
```

```
## [1] 435.0732
```

```r
# MAE
mean(abs(xgb_walk_df$Actual - xgb_walk_df$Predicted))
```

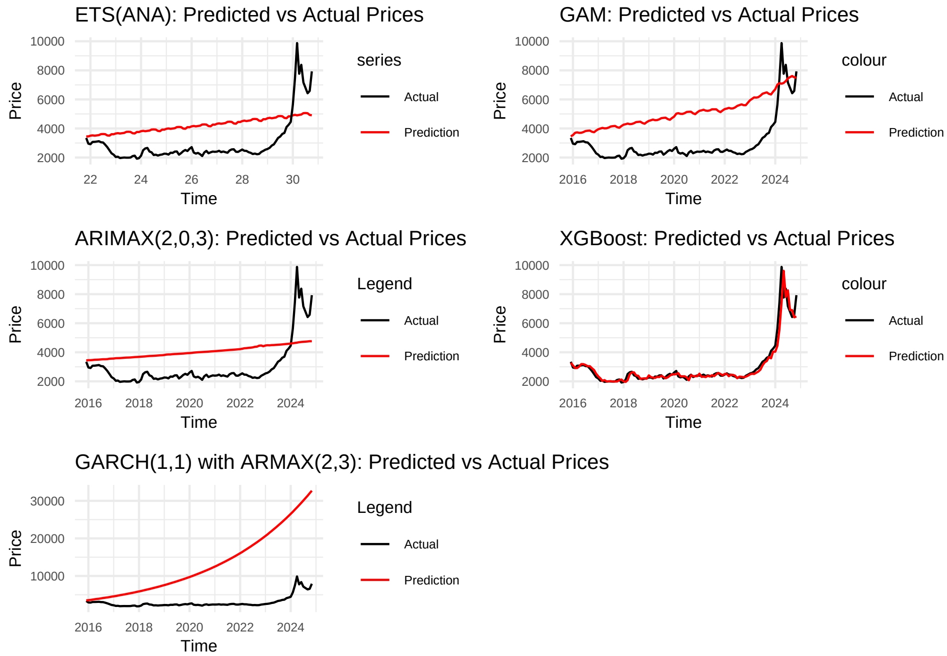
```
## [1] 203.8842
```

```r
# MAPE
mean(abs((xgb_walk_df$Actual - xgb_walk_df$Predicted) / xgb_walk_df$Actual)) * 100
```

```
## [1] 5.191014
```

2.6 Model Selection

```r
p1 <- p1 + theme(text = element_text(size = 8))
p2 <- p2 + theme(text = element_text(size = 8))
p3 <- p3 + theme(text = element_text(size = 8))
p4 <- p4 + theme(text = element_text(size = 8))
p5 <- p5 + theme(text = element_text(size = 8))

p.combined <- grid.arrange(p1, p4, p2, p5, p3,
             ncol = 2,
             widths = c(2, 2),
             heights = c(1.2, 1.2, 1.2))
```




```r
ggsave("plot_all_models.png", p.combined, dpi = 300)
```

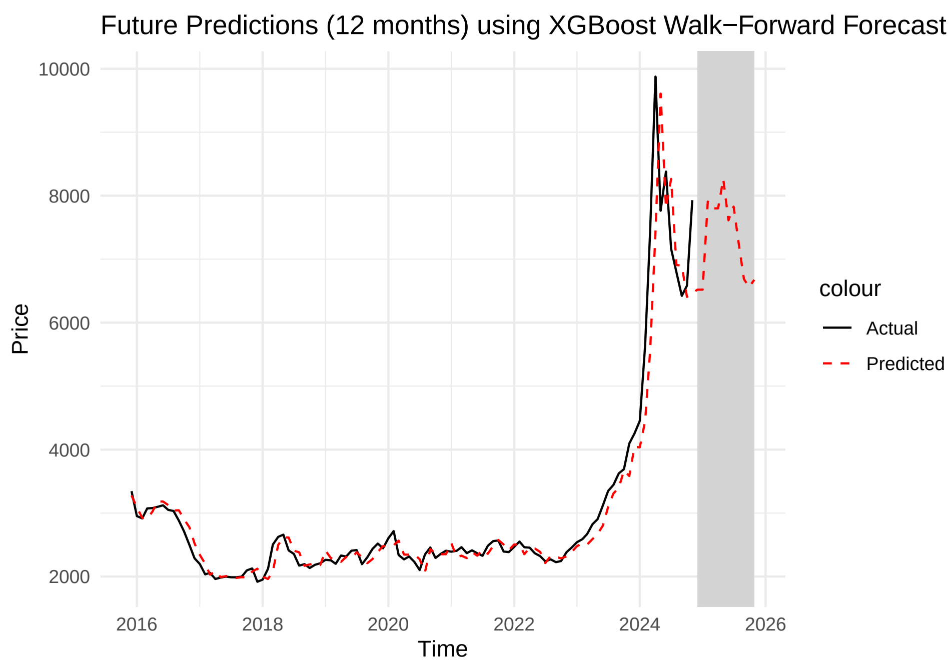
```
## Saving 6.5 x 4.5 in image
```

3. Prediction

```r
# Predict 12 months
future_months <- 12

last_row <- data[nrow(data), ]
future_predictions <- c()
future_dates <- c()

for (i in 1:future_months) {
  future_time <- last_row$Time + (i * 30)
  future_monthFac <- as.numeric(format(as.Date(future_time, origin = "1970-01-01"), "%m"))

  future_data <- last_row
  future_data$Time <- future_time
  future_data$monthFac <- future_monthFac

  x_future <- future_data %>% select(all_of(features))
  dfuture <- xgb.DMatrix(data = as.matrix(x_future))

  pred_log_future <- predict(xgb_model, dfuture)
  pred_price_future <- exp(pred_log_future)

  future_predictions <- c(future_predictions, pred_price_future)
  future_dates <- c(future_dates, as.Date(future_time, origin = "1970-01-01"))

  last_row$log_price <- pred_log_future
}
```

```r
future_xgb_df <- tibble(Time = as.Date(future_dates),
                        Predicted = future_predictions)

combined_df <- bind_rows(xgb_walk_df, future_xgb_df)
```

```r
forecast_start <- min(future_xgb_df$Time)
forecast_end <- max(future_xgb_df$Time)

ggplot(combined_df, aes(x = Time)) +
  geom_rect(aes(xmin = forecast_start, xmax = forecast_end, ymin = -Inf, ymax = Inf),
            fill = "lightgray", alpha = 0.3) +
  geom_line(aes(y = Actual, color = "Actual"), na.rm = TRUE) +
  geom_line(aes(y = Predicted, color = "Predicted"), linetype = "dashed") +
  labs(title = "Future Predictions (12 months) using XGBoost Walk-Forward Forecast",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red")) +
  theme_minimal()
```




```r
summary(combined_df)
```

```
##       Time                Actual        Predicted
##  Min.   :2015-12-01   Min.   :1918   Min.   :1960
##  1st Qu.:2018-05-24   1st Qu.:2261   1st Qu.:2304
##  Median :2020-11-16   Median :2415   Median :2478
##  Mean   :2020-11-15   Mean   :2977   Mean   :3356
##  3rd Qu.:2023-05-08   3rd Qu.:2907   3rd Qu.:3183
##  Max.   :2025-10-27   Max.   :9877   Max.   :9609
##                       NA's   :12
```