

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
library(tibble)
library(xts)

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

1. EDA

```
price = read.csv("./Daily Prices_ICC0.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

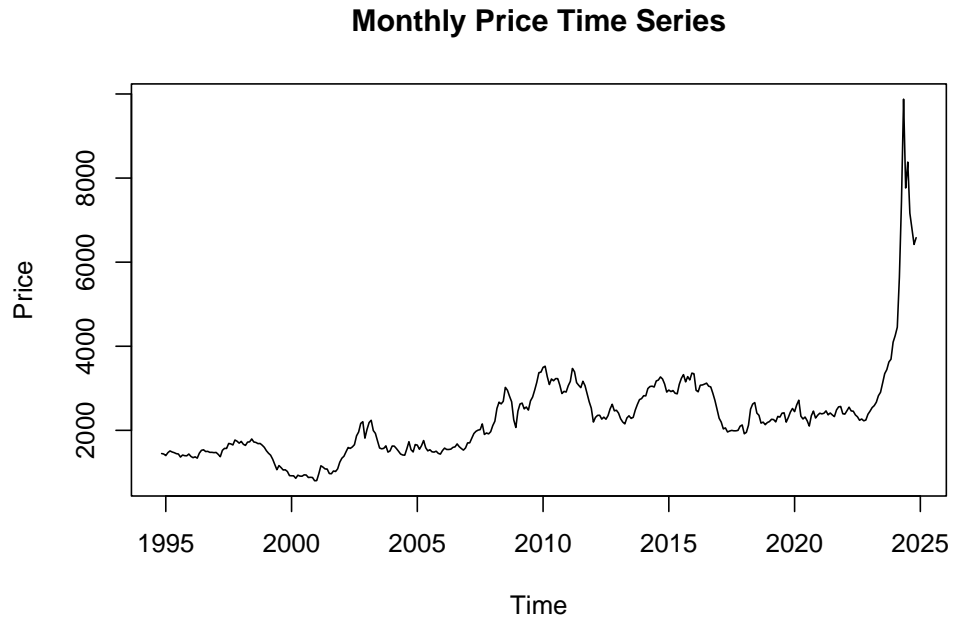
```
price$Date <- as.Date(price$Date, format="%d/%m/%Y")
price$Daily_Price <- as.numeric(gsub(",", "", price$Daily_Price))
price_month <- price |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(month_Price = mean(Daily_Price, na.rm = TRUE)) |> ungroup()
```

```
summary(price)
```

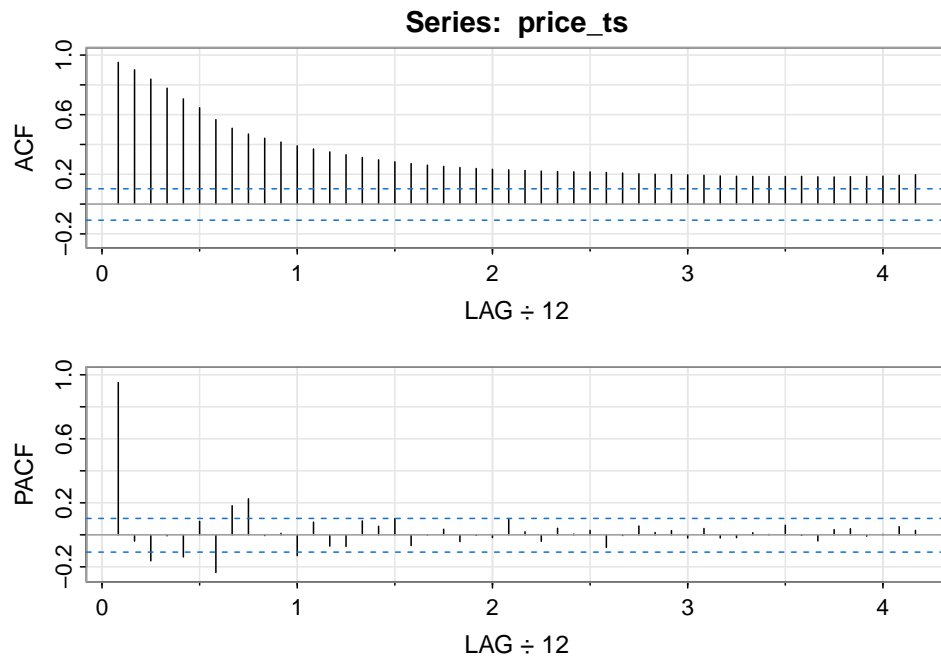
```
##           Date           Daily_Price
##  Min.      :1994-10-03   Min.       : 774.1
##  1st Qu.:2002-05-16   1st Qu.: 1557.8
##  Median :2009-12-17   Median : 2202.0
##  Mean    :2009-12-17   Mean      : 2350.1
##  3rd Qu.:2017-07-24   3rd Qu.: 2738.1
##  Max.    :2025-02-27   Max.       :11984.7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

```
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")
```



```
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
```

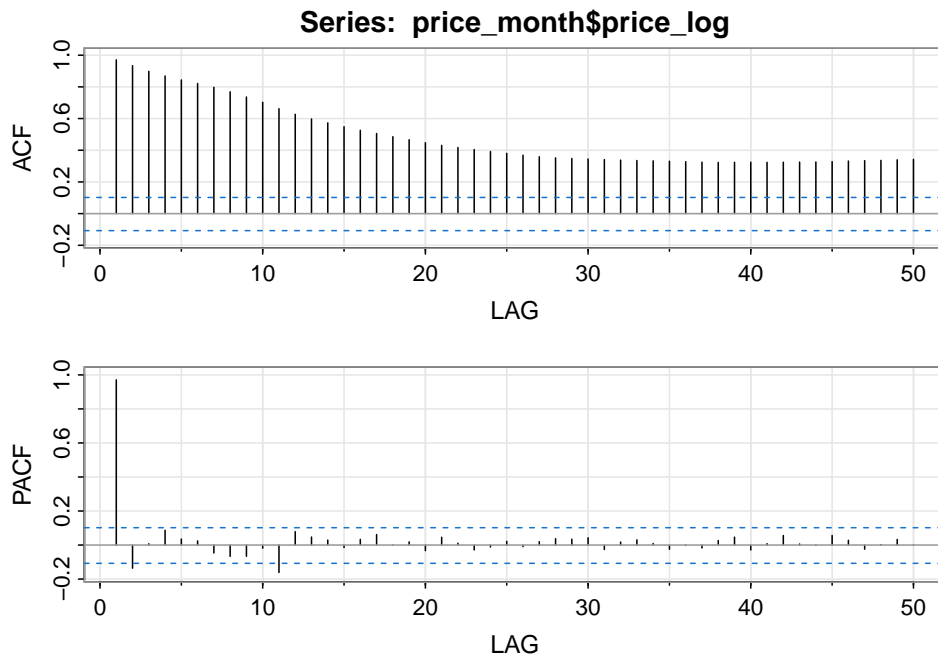
```
ndiffs(price_ts)
```

```
## [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)
```

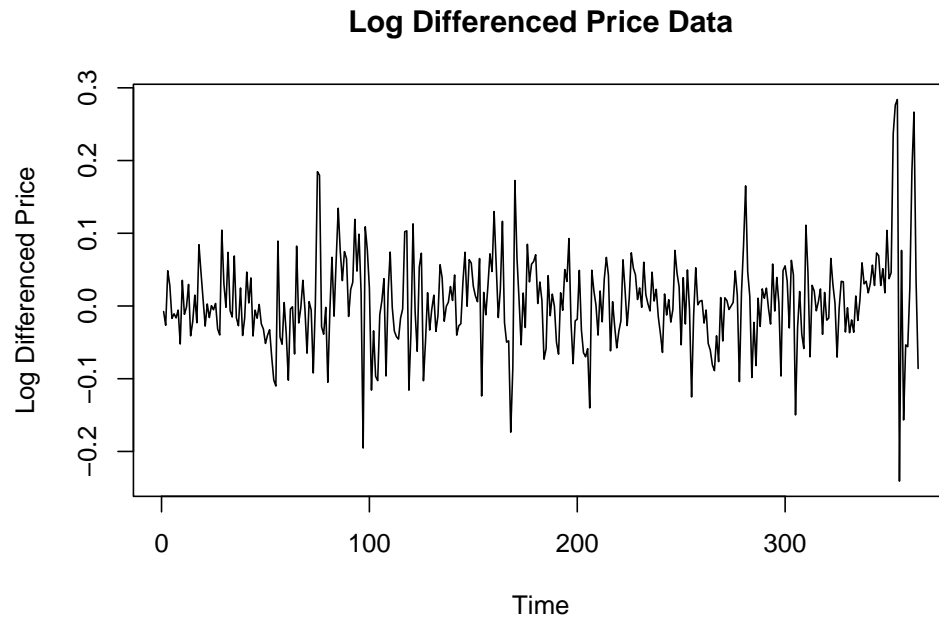


```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.97  0.93  0.90  0.87  0.84  0.82  0.80  0.77  0.74  0.70  0.66  0.63  0.60
## PACF  0.97 -0.14  0.01  0.09  0.04  0.02 -0.05 -0.07 -0.07 -0.02 -0.16  0.08  0.05
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,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")
```

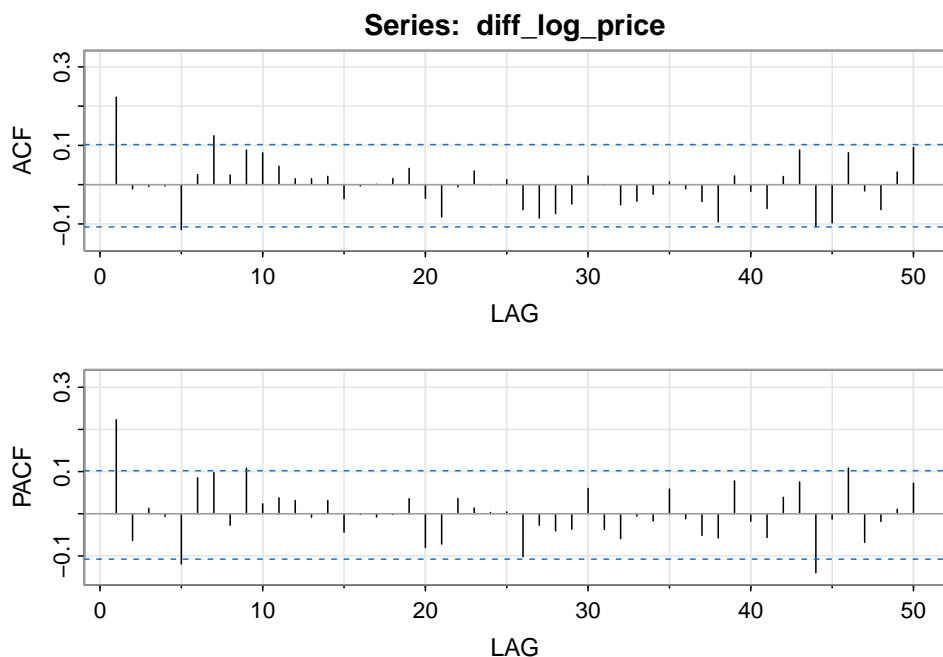


```
adf.test(diff_log_price)
```

```
## 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

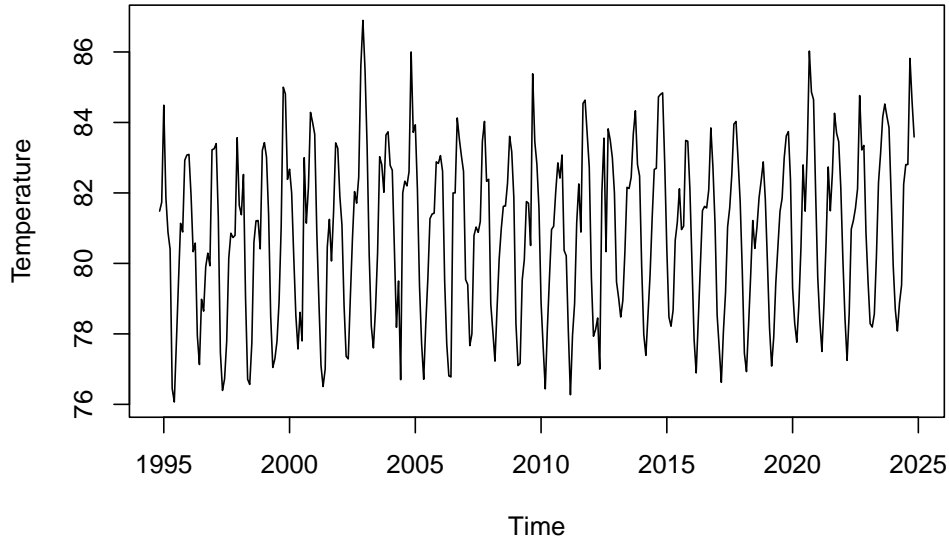
```
weather$Date <- as.Date(weather$Date)
weather$Avg_Temp <- as.numeric(gsub("", "", weather$Avg_Temp))
weather_month <- weather |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(Avg_Temp = mean(Avg_Temp, na.rm = TRUE)) |> ungroup()

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()
```

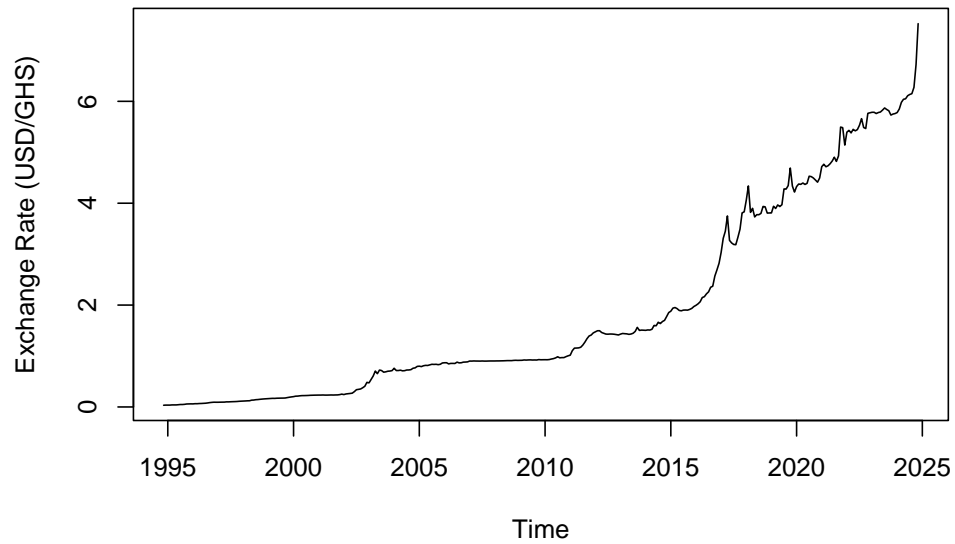
```
summary(exchangerate)
```

```
##      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
```

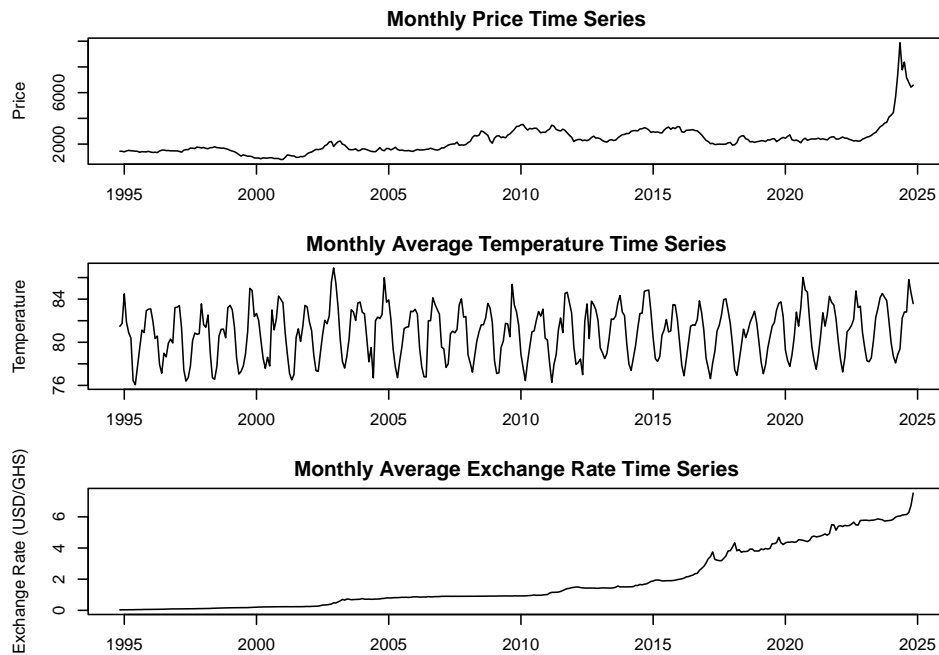
```
rate_ts <- ts(rate_month$exchange_rate, start = c(1994, 11), end = c(2024, 11), frequency = 12)
```

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

Monthly Average Exchange Rate Time Series

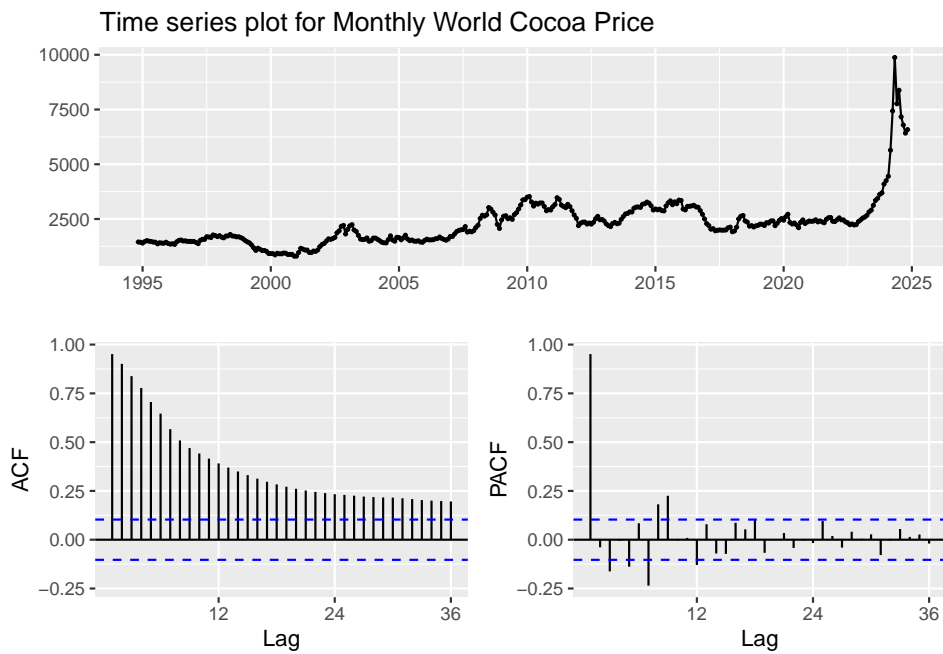


```
par(mfrow=c(3,1), mar = c(3, 4, 2, 2))
# price
plot(price_ts, main="Monthly Price Time Series", ylab="Price", xlab="Time")
# temperature
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")
# exchange rate
ts.plot(rate_ts, main="Monthly Average Exchange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

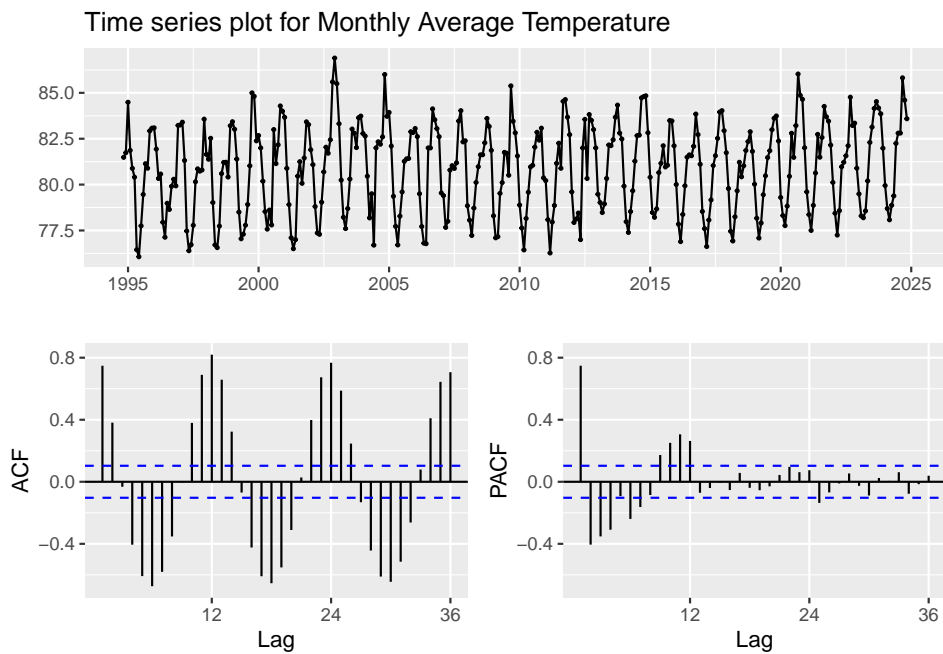


1.4 Time series plots for data


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

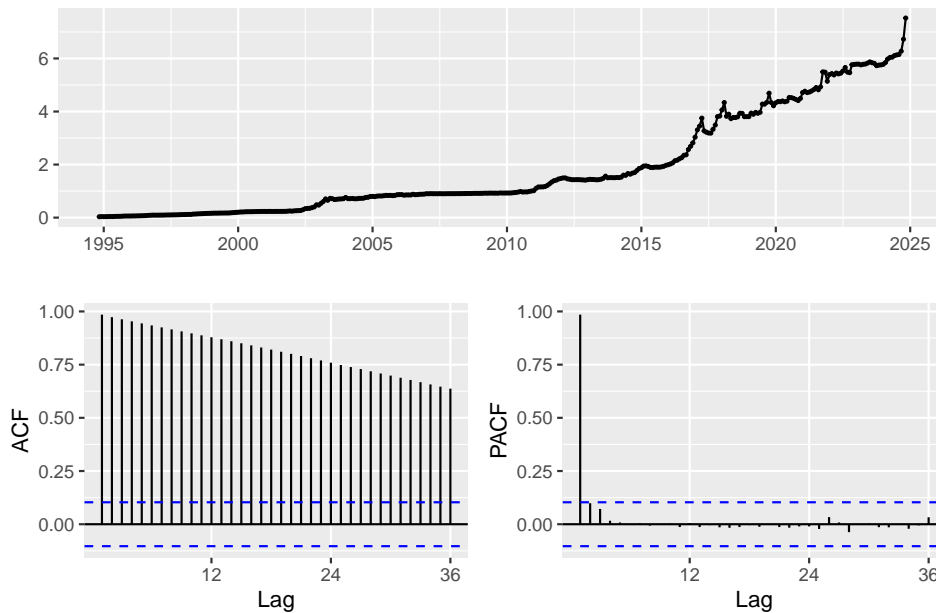


```
ggtsdisplay(weather_ts, main="Time series plot for Monthly Average Temperature")
```

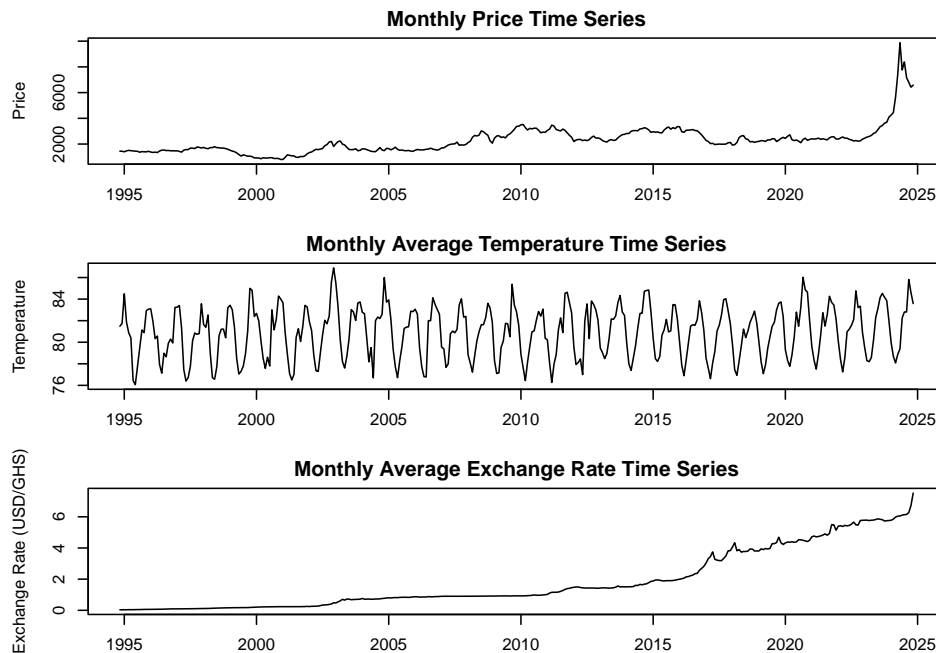


```
ggtsdisplay(rate_ts, main="Time series plot for Monthly Average Exchange Rate(USD/GHS)")
```

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 Datasets

```
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 =
```

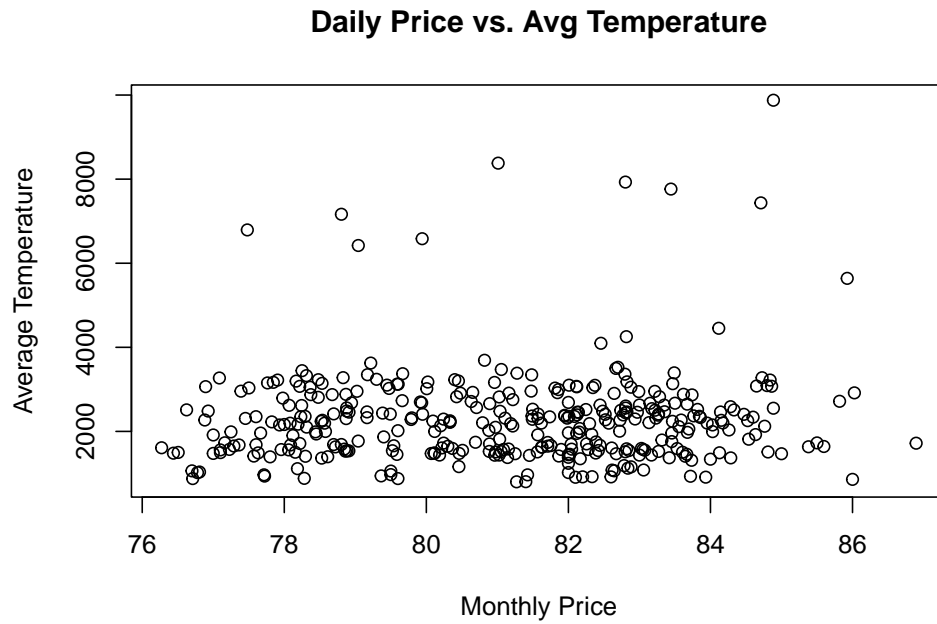
```

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")

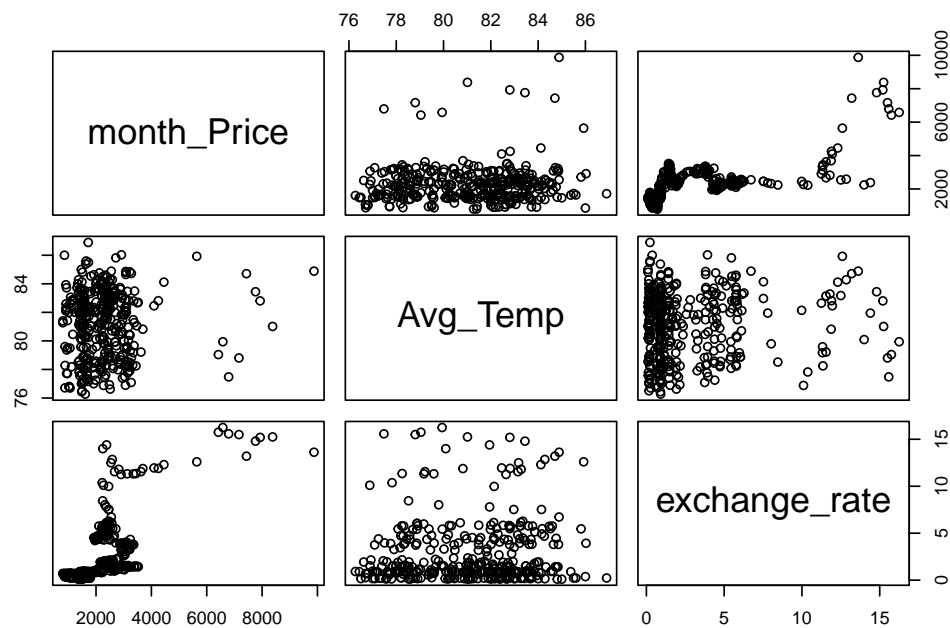
```



```

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

```



1.6 Stationary Check

```

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)

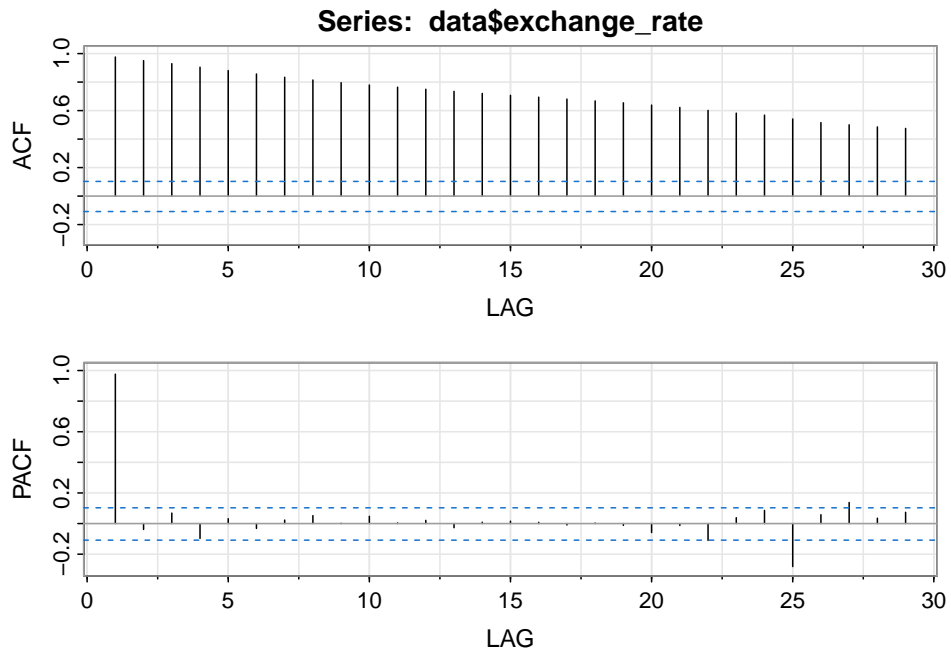
## Warning in adf.test(data$exchange_rate): p-value greater than printed p-value
##
## Augmented Dickey-Fuller Test
##
## data: data$exchange_rate
## Dickey-Fuller = 0.7342, Lag order = 7, p-value = 0.99
## alternative hypothesis: stationary
adf.test(log(data$exchange_rate))

##
## 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
```

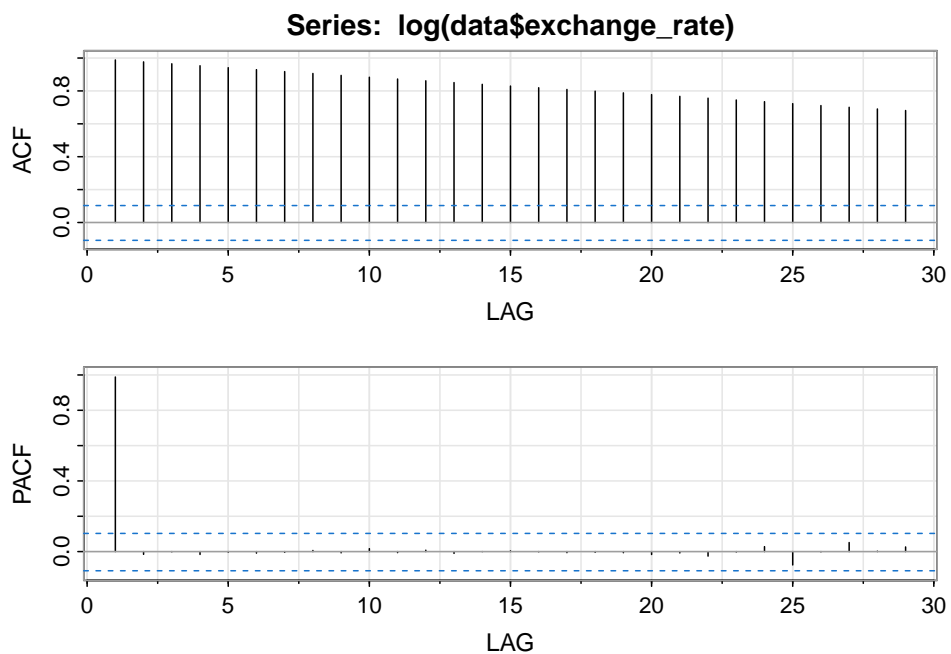
Since monthly average temperature is already stationary, we would do take the differenced and log-transformed exchange rate as our exogenous factors.

```
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
```

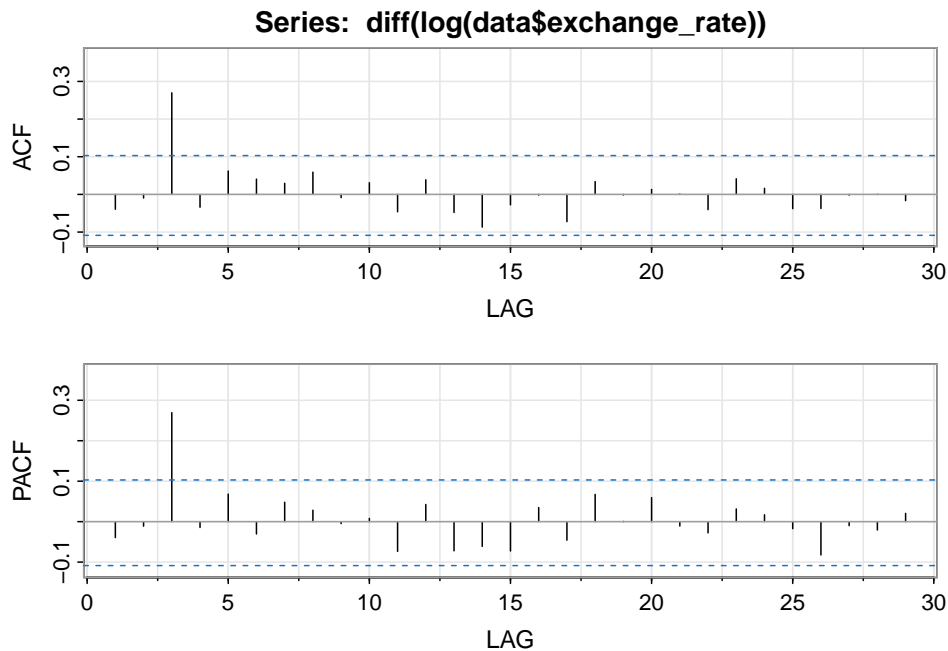
```
acf2(log(data$exchange_rate))
```



```
##      [,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
```

```
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
```

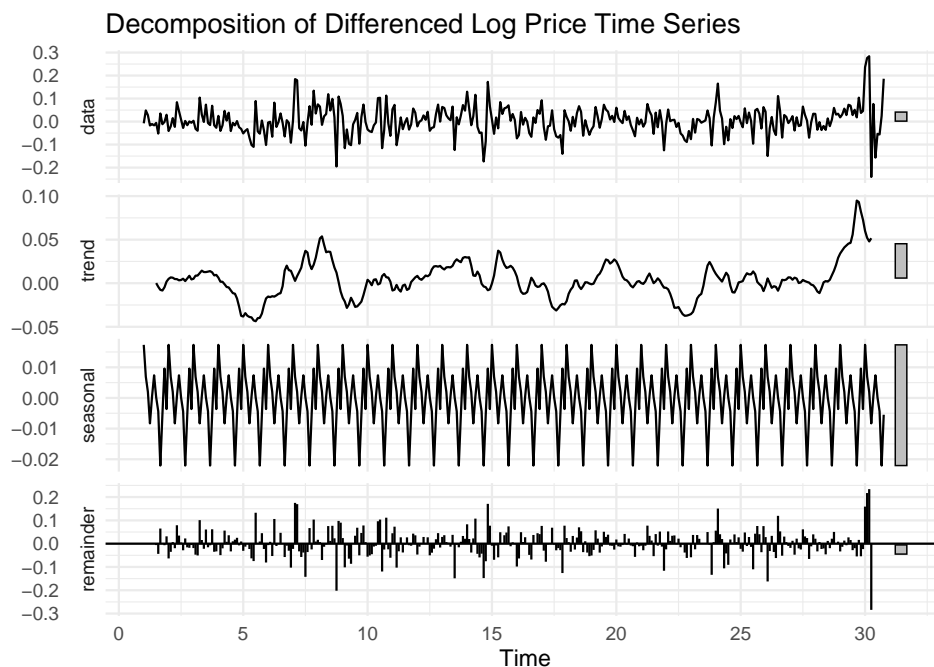
```
adf.test(data$diff_log_price)
```

```
## 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()
```



1.7 Split data

```
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)
```

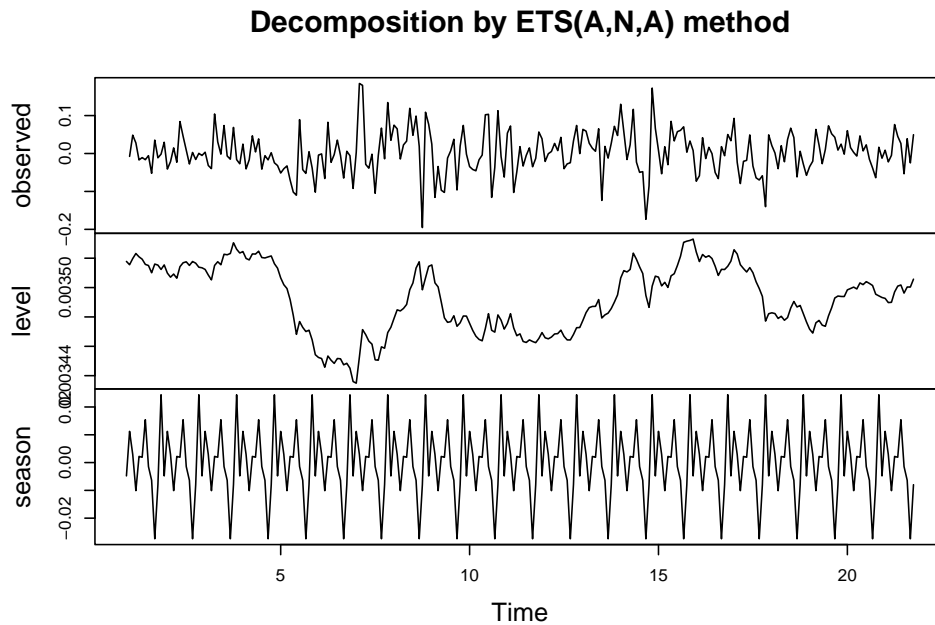
```
## ETS(A,N,A)
##
## Call:
## ets(y = data_train_ts, model = "ANA")
##
## Smoothing parameters:
##   alpha = 1e-04
##   gamma = 1e-04
##
## Initial states:
##   l = 0.0035
##   s = -0.0048 0.0244 -0.008 -0.0274 -0.0064 -0.0014
##         0.0154 0.0019 0.0022 -0.0101 0.0029 0.0112
##
## sigma: 0.057
##
##      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
plot(ets_model)
```

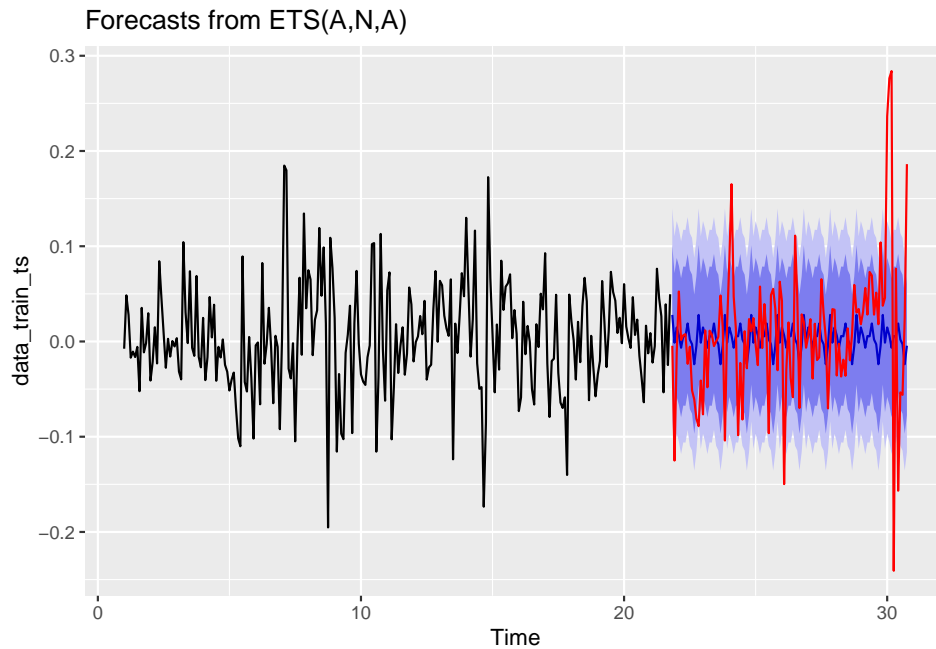


2.1.2 Forecasting and Plotting

```
# 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)

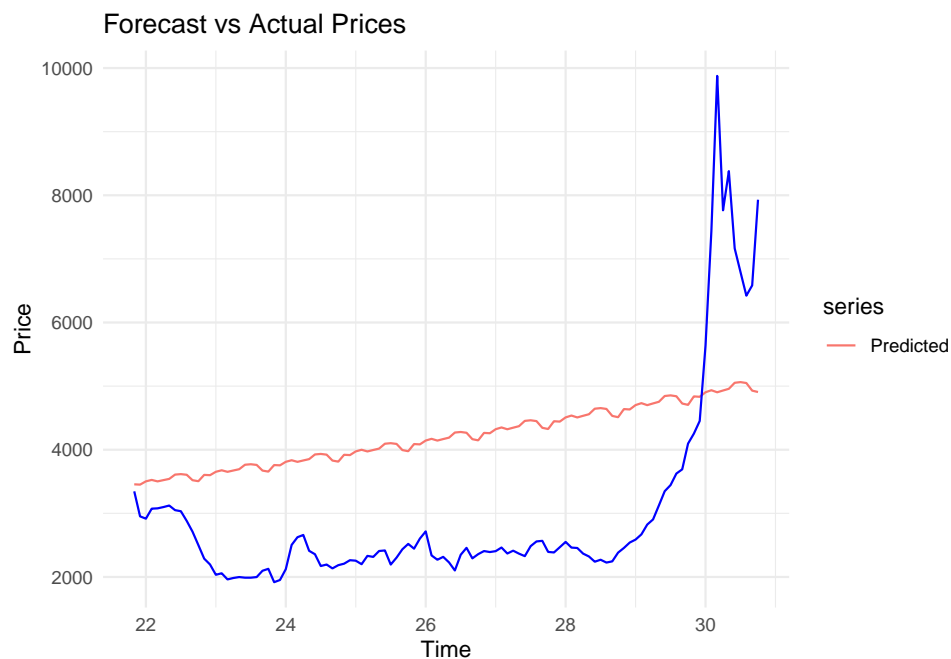
# Convert back to actual price
forecasted_price <- exp(cumsum(forecast_ets$mean) + last_log_price)

actual_price <- exp(testSet$log_price)

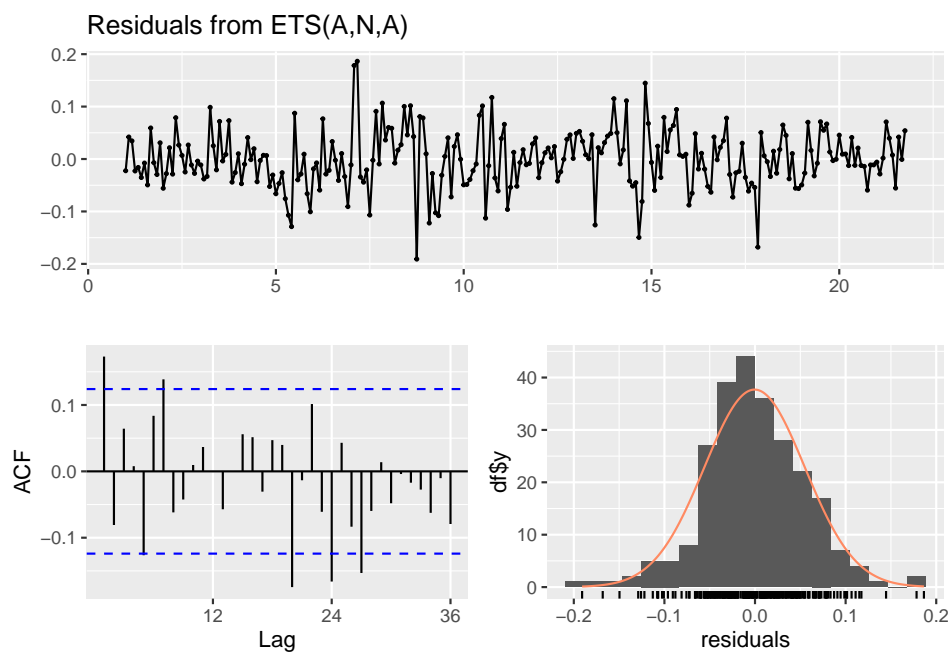
data_test_ts <- ts(testSet$diff_log_price, start = end(data_train_ts) + c(0,1),
                  frequency = 12)

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()
```

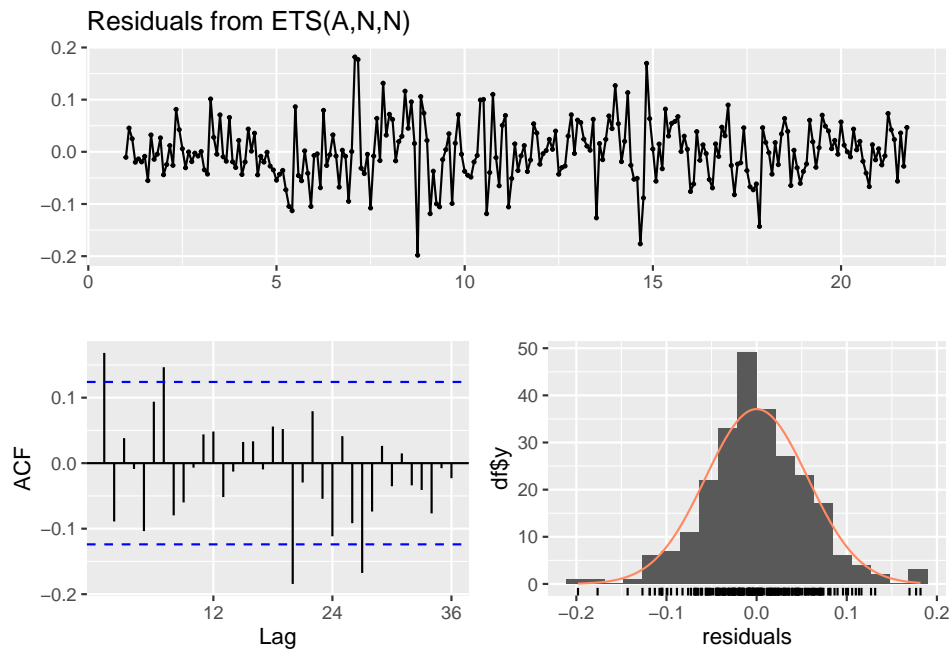


```
checkresiduals(ets_model)
```



```
##
##  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

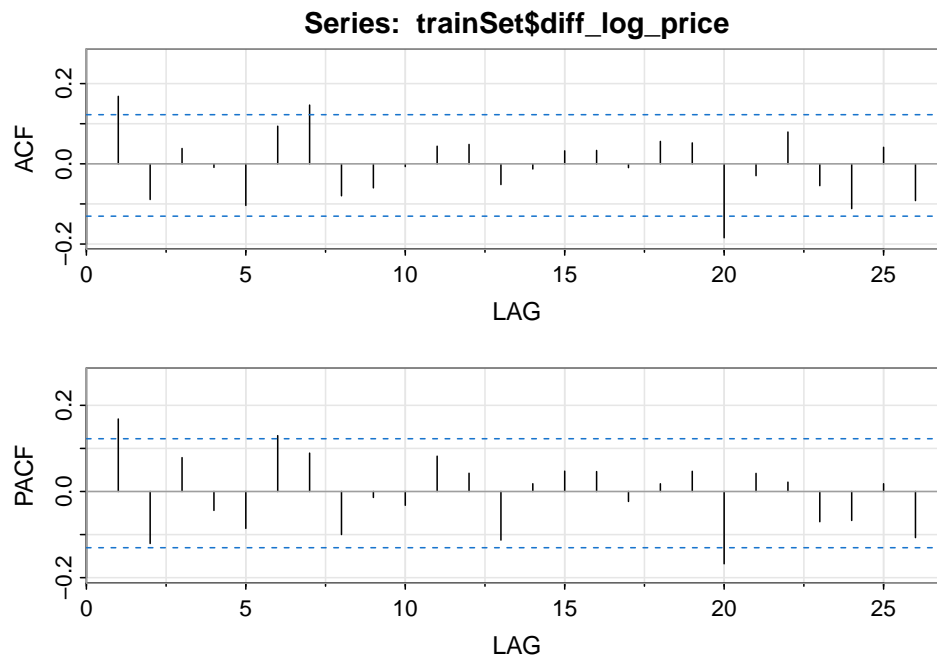
```
adf.test(diff(trainSet$month_Price))
```

```
## 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
```

```
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(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

```
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)

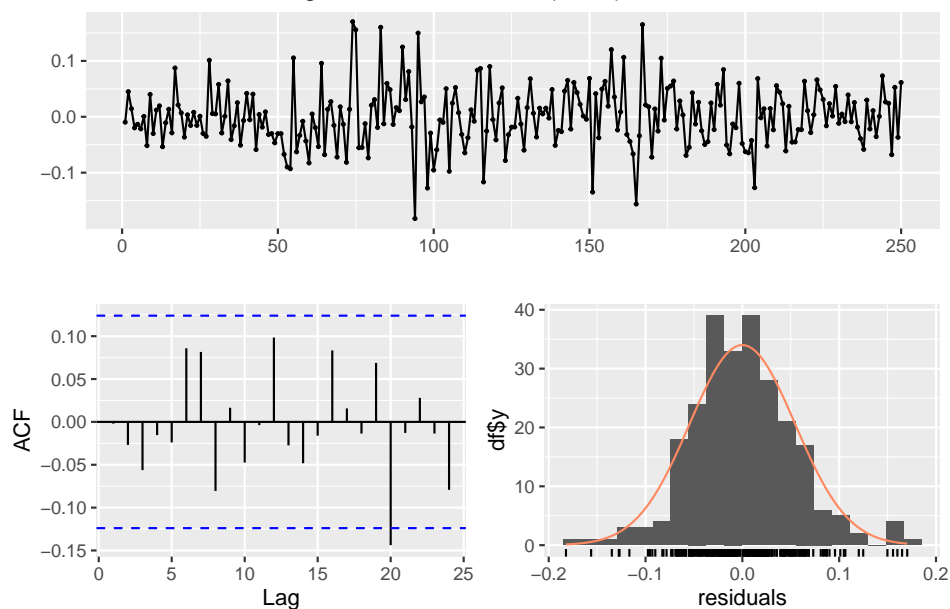
## [1] "Selected order for ARMA: p = 2, q = 3"

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 -1.229987e-06 0.05430941 0.04164443 140.2458 203.7395 0.7523938
##              ACF1
## Training set -0.00203552

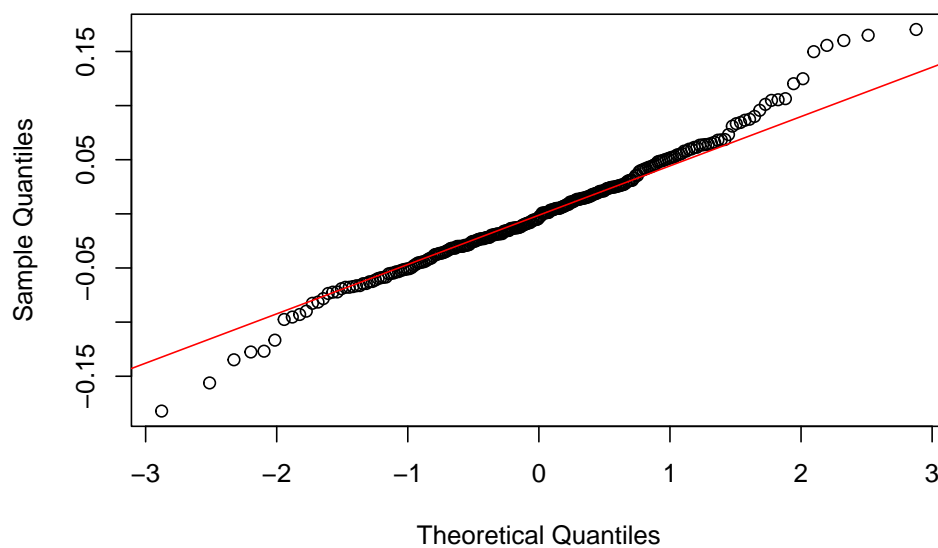
checkresiduals(model.arimax)
```

Residuals from Regression with ARIMA(2,0,3) errors



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

Normal Q-Q Plot



```
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):
 - $Q^* = 7.1814$, $df = 5$, $p\text{-value} = 0.2075$
 - 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.

```
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)

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

## [1] 302.0143
mape(trainSet$month_Price, model.arimax.fitted.converted)

## [1] 0.1877152
rmse(testSet$month_Price, forecast.arimax.final)

## [1] 1701.38
mae(testSet$month_Price, forecast.arimax.final)

## [1] 1559.875
mape(testSet$month_Price, forecast.arimax.final)

## [1] 0.5865196
```

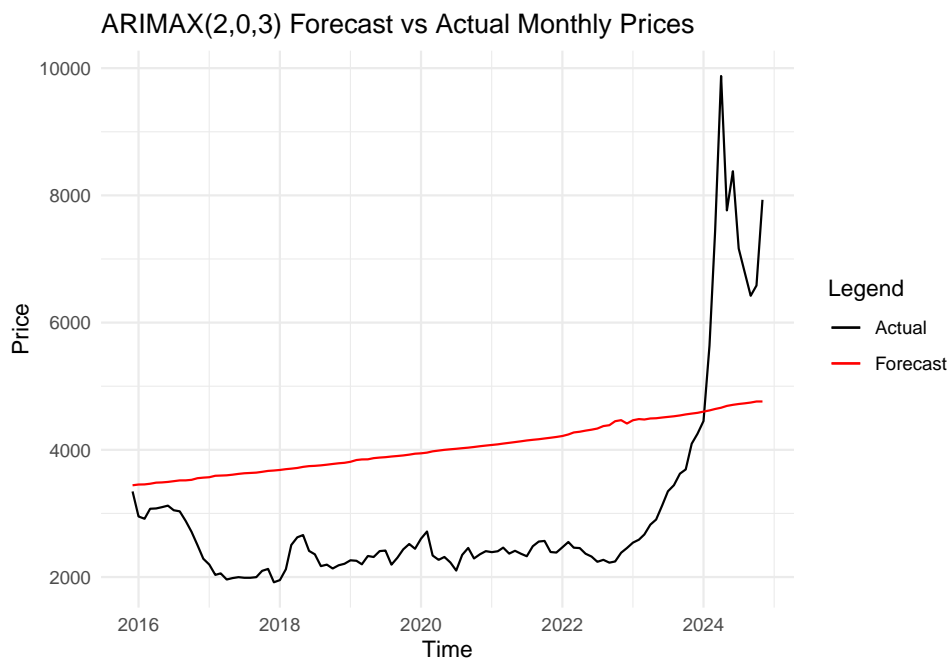


```

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

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 = "Forecast")) +
  labs(
    title = "ARIMAX(2,0,3) Forecast vs Actual Monthly Prices",
    y = "Price",
    x = "Time",
    color = "Legend"
  ) +
  theme_minimal() +
  scale_color_manual(values = c(
    "Actual" = "black",
    "Forecast" = "red"))

```



2.3 GARCH Model

2.3.1 GARCH Parameters (With Xreg)

```

# xreg_matrix is the same as arimax
p = 0:3
q = 0:3
## select ARMA order
aic.arimax.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.armac.garch1[i, j] = AIC(modij)
}
}
aic.armac.min_index <- which(aic.armac.garch1 == min(aic.armac.garch1), arr.ind = TRUE)
aramx.garch.p <- p[aic.armac.min_index[1]]
aramx.garch.q <- q[aic.armac.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

```

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.armac.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.armac.garch2[i, j] = infocriteria(modij)[1]
  }
}

aic.garch.min_index <- which(aic.armac.garch2 == min(aic.armac.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"
```

```

armac.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.armac.garch <- ugarchfit(armac.garch.spec.train,
  data = trainSet$diff_log_price,
  trace = FALSE)
model.armac.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      0.020917   0.131217   0.159405 0.873350
## ar1      0.023795   0.040488   0.587703 0.556732
## ar2     -0.894392   0.031402 -28.482399 0.000000
## ma1      0.180533   0.066946   2.696704 0.007003
## ma2      0.953060   0.013923  68.451358 0.000000
## ma3      0.258470   0.064191   4.026596 0.000057
## mxreg1  -0.000227   0.001627  -0.139236 0.889264
## mxreg2  -0.000305   0.101078  -0.003021 0.997589
## omega    0.000171   0.000126   1.353592 0.175867
## alpha1   0.092076   0.048015   1.917657 0.055155
## beta1    0.858378   0.065243  13.156584 0.000000
## shape    6.610146   3.242426   2.038642 0.041486
##
## Robust Standard Errors:
##      Estimate  Std. Error    t value Pr(>|t|)
## mu      0.020917   0.150659   0.138834 0.889581
## ar1      0.023795   0.049430   0.481394 0.630236
## ar2     -0.894392   0.034242 -26.120005 0.000000
## ma1      0.180533   0.061615   2.930006 0.003390
## ma2      0.953060   0.016170  58.939192 0.000000
## ma3      0.258470   0.063494   4.070783 0.000047
## mxreg1  -0.000227   0.001881  -0.120407 0.904160
## mxreg2  -0.000305   0.129359  -0.002361 0.998116
## omega    0.000171   0.000092   1.858395 0.063113
## alpha1   0.092076   0.040354   2.281704 0.022507
## beta1    0.858378   0.043445  19.757817 0.000000
## shape    6.610146   2.907748   2.273287 0.023009
##
## 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] 0.004822 0.9446
## Lag[2*(p+q)+(p+q)-1][14] 4.789881 1.0000
## Lag[4*(p+q)+(p+q)-1][24] 9.750395 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 0.5772
## 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: 1.955
## Individual Statistics:
## mu 0.21143
## ar1 0.33766
## ar2 0.21801
## ma1 0.09681
## ma2 0.14306
## ma3 0.09043
## mxreg1 0.20585
## mxreg2 0.27455
## omega 0.09041
## alpha1 0.08057
## beta1 0.09023
## shape 0.12346
##
## 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 0.65210 0.5149
## Negative Sign Bias 0.01309 0.9896
## Positive Sign Bias 0.29202 0.7705
## Joint Effect 1.49230 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 : 0.1298318
```

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

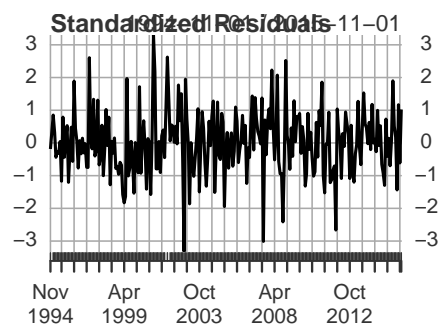
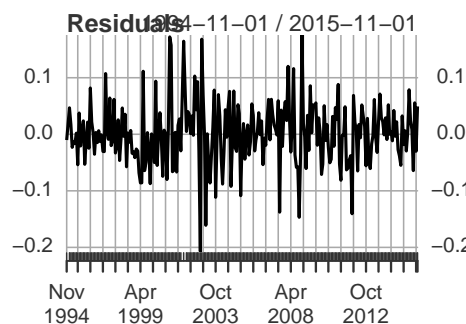
```
##          mu          ar1          ar2          ma1          ma2
## 0.0209166566 0.0237951944 -0.8943915639 0.1805327148 0.9530604439
##          ma3          mxreg1          mxreg2          omega          alpha1
## 0.2584695098 -0.0002265352 -0.0003053705 0.0001710718 0.0920755280
##          beta1          shape
## 0.8583775746 6.6101460622
```

```
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)
```

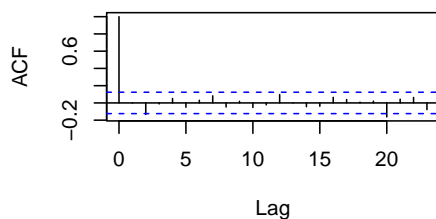
```
# Residual Analysis
par(mfrow = c(2, 2))
```

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

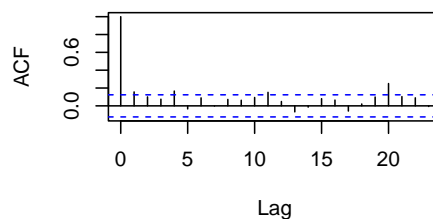
```
# ACF plots
acf(na.omit(as.numeric(residuals(model.armax.garch))), main = "ACF of Residuals")
acf(na.omit(as.numeric(residuals(model.armax.garch)^2)), main = "ACF of Squared Residuals")
```



ACF of Residuals



ACF of Squared Residuals



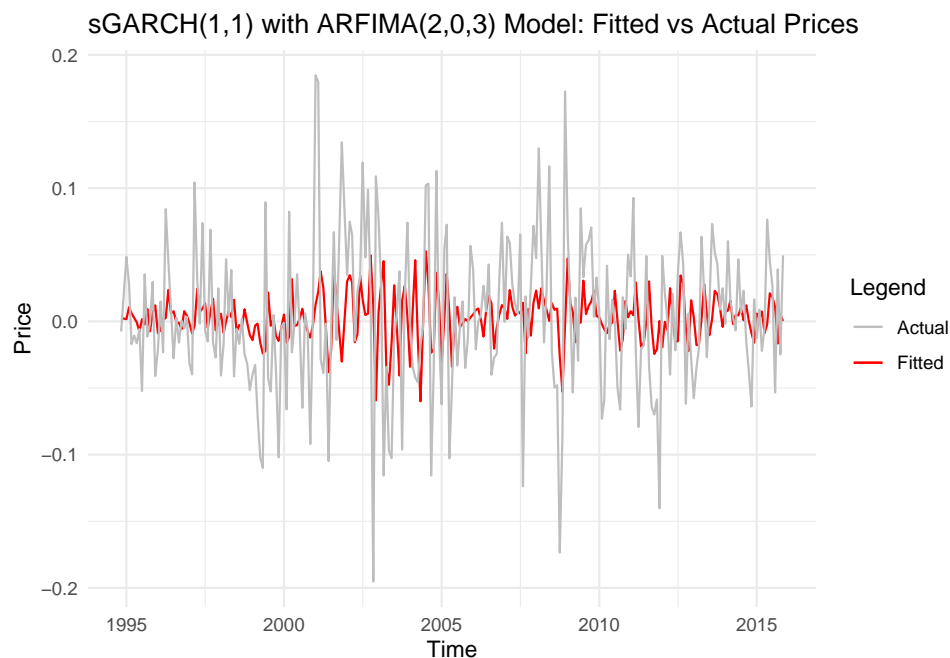
```

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
)

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

```



2.3.2 GARCH Forecast with param selection (With Xreg)

```

# 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)

```

```

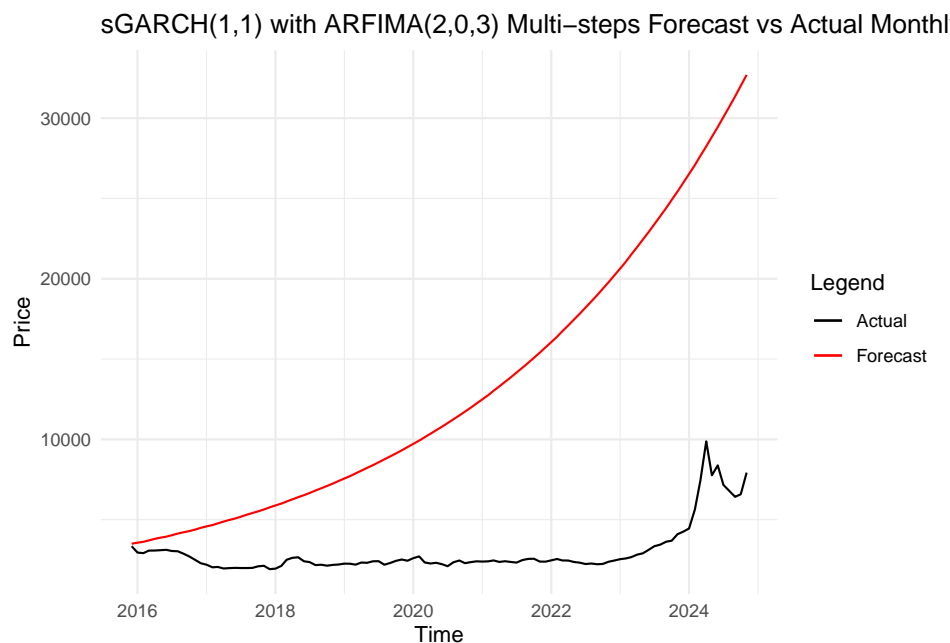
colnames(xreg_test_matrix) <- c("Avg_Temp", "dl_exchange_rate")

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 <- tail(trainSet$month_Price, 1)*exp(cumsum(fore.garch.dl.data))

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
)

library(ggplot2)
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 = "Forecast")) +
  labs(
    title = "sGARCH(1,1) with ARFIMA(2,0,3) Multi-steps Forecast vs Actual Monthly Prices",
    y = "Price",
    x = "Time",
    color = "Legend"
  ) +
  theme_minimal() +
  scale_color_manual(values = c("Actual" = "black", "Forecast" = "red"))

```



RMSE below shows the forecast ability of the final model with one-step forecast

```

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] 383.9596

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

## [1] 299.082

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

## [1] 0.1682952

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

## [1] 12425.02

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

## [1] 10119.37

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

## [1] 3.424806

```

2.5 GAM Model

2.5.1 Fit Model

```

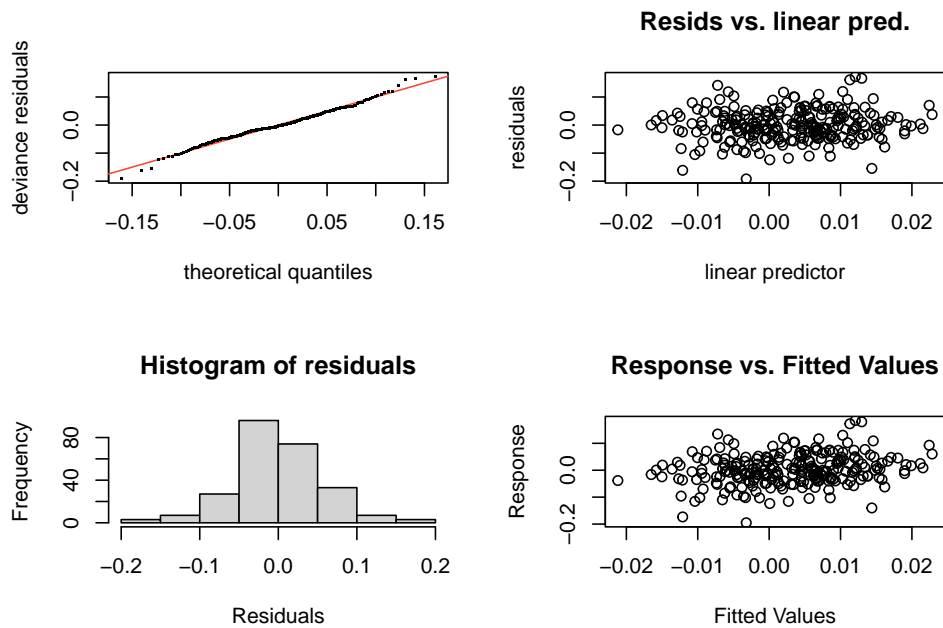
# Uses simple smoothing splines for variables
trainSet$monthFac <- as.factor(format(trainSet$Time, "%m"))
trainSet$month_num <- as.numeric(trainSet$monthFac)

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   0.76
## s(Avg_Temp)      9.00  1.00   1.06   0.84
## s(exchange_rate) 9.00  1.00   0.89   0.03 *
## ---
## 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)
```

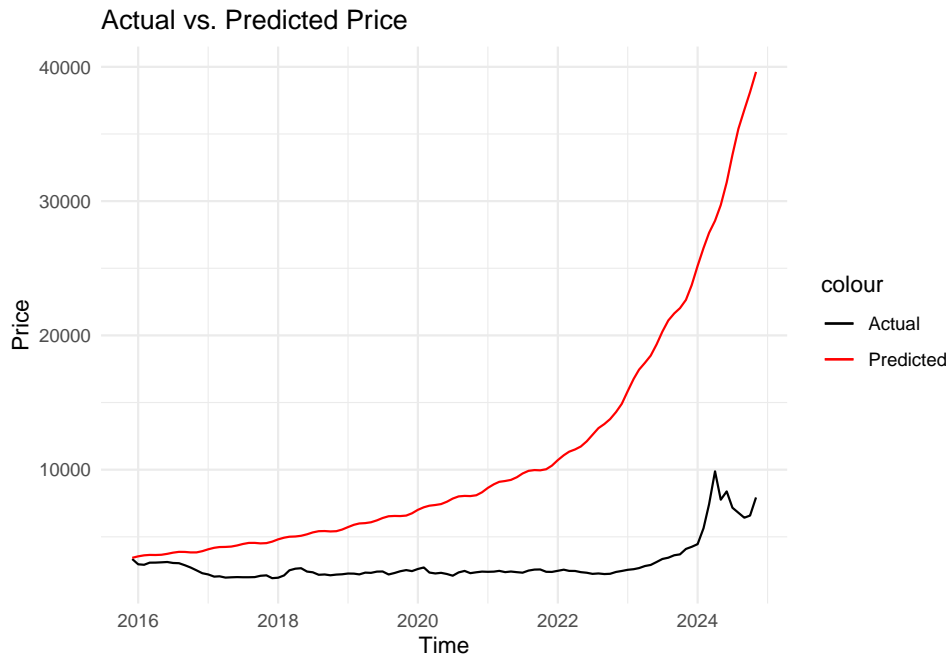
2.5.2 Forecast and Plot

```
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

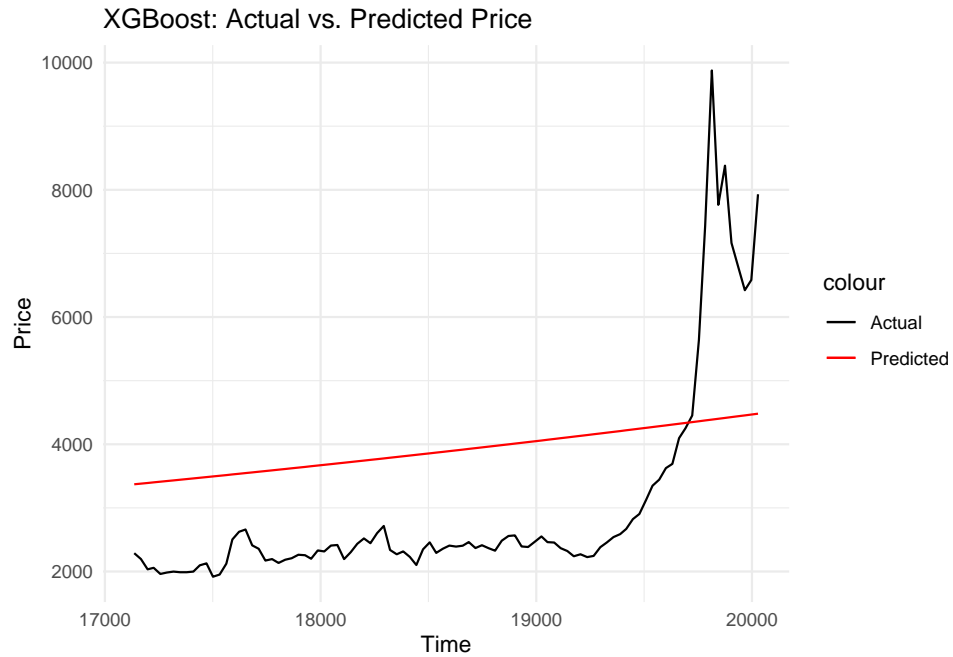
dtest <- xgb.DMatrix(as.matrix(test_data[, predictors]))
test_data$pred_diff_log <- predict(xgb_model, dtest)

# 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)
```