STA457 Project

Xing Yu Wang

2025-03-29

```r
# install.packages("forecast")
# install.packages("astsa")
# install.packages("Metrics")
# install.packages("xgboost")

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)
```

1. EDA

```r
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")
```

```
## Rows: 397 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): Change %
## dbl  (4): Price, Open, High, Low
## lgl  (1): Vol.
## date (1): Date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

1.1 Clean Data

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

```r
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

```r
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

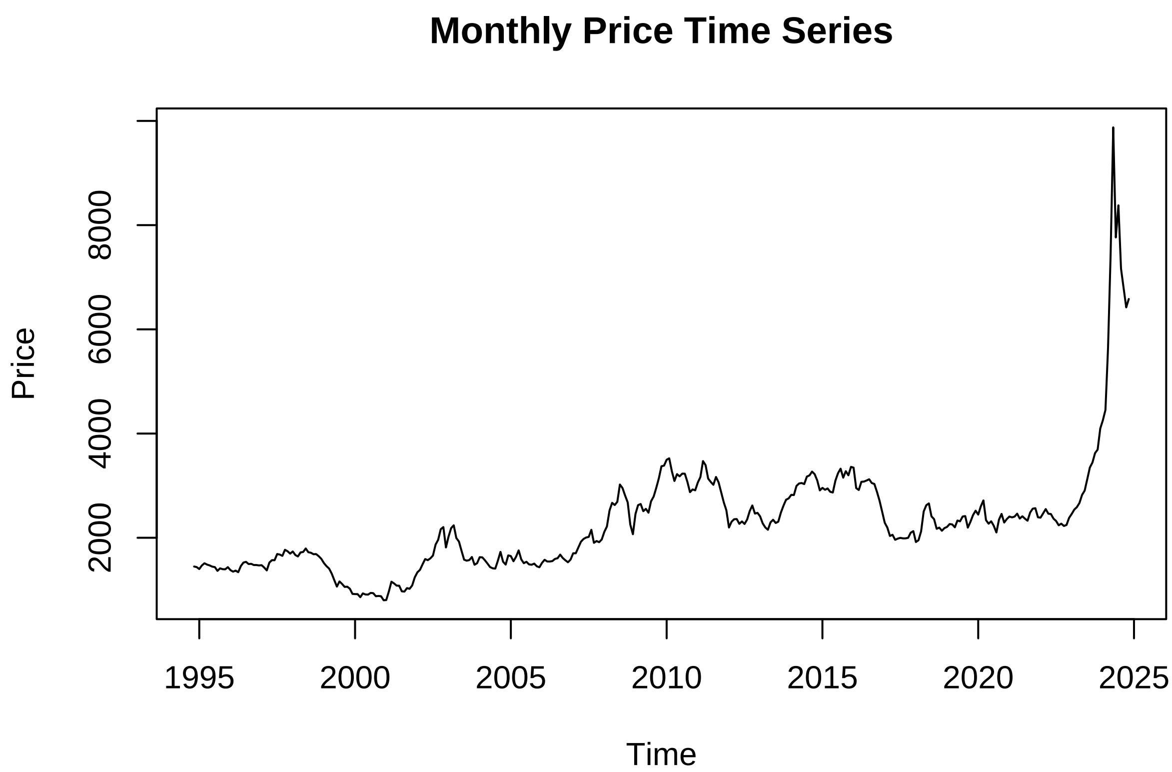
```r
price$Date <- as.Date(price$Date, format="%d/%m/%Y")
price$Daily_Price <- as.numeric(gsub(",", "", price$Daily_Price))
price_month <- price |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(month_Price = mean(Daily_Price, na.rm = TRUE)) |> ungroup()
```

```r
summary(price)
```

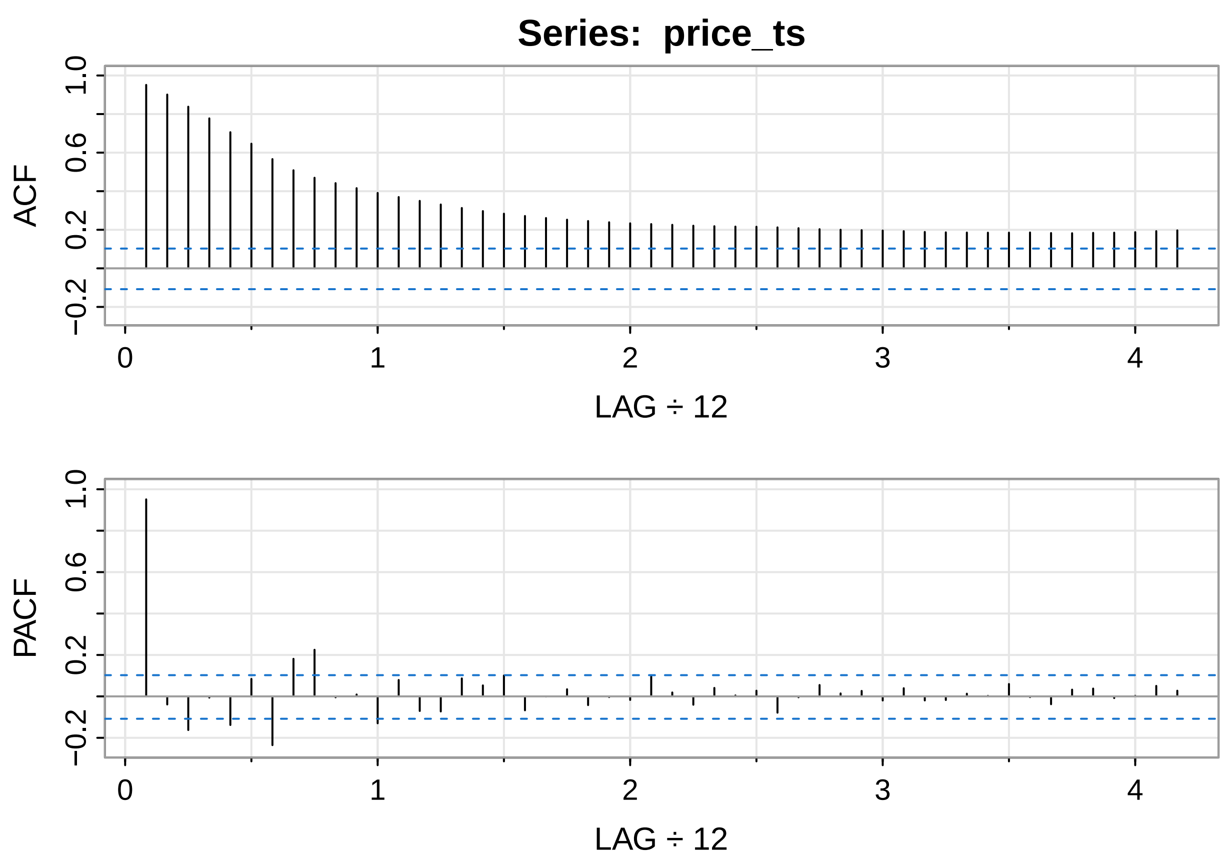
```
##       Date                Daily_Price
##  Min.   :1994-10-03   Min.   :  774.1
##  1st Qu.:2002-05-16   1st Qu.: 1557.8
##  Median :2009-12-17   Median : 2202.0
##  Mean   :2009-12-17   Mean   : 2350.1
##  3rd Qu.:2017-07-24   3rd Qu.: 2738.1
##  Max.   :2025-02-27   Max.   :11984.7
```

```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")
```




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




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

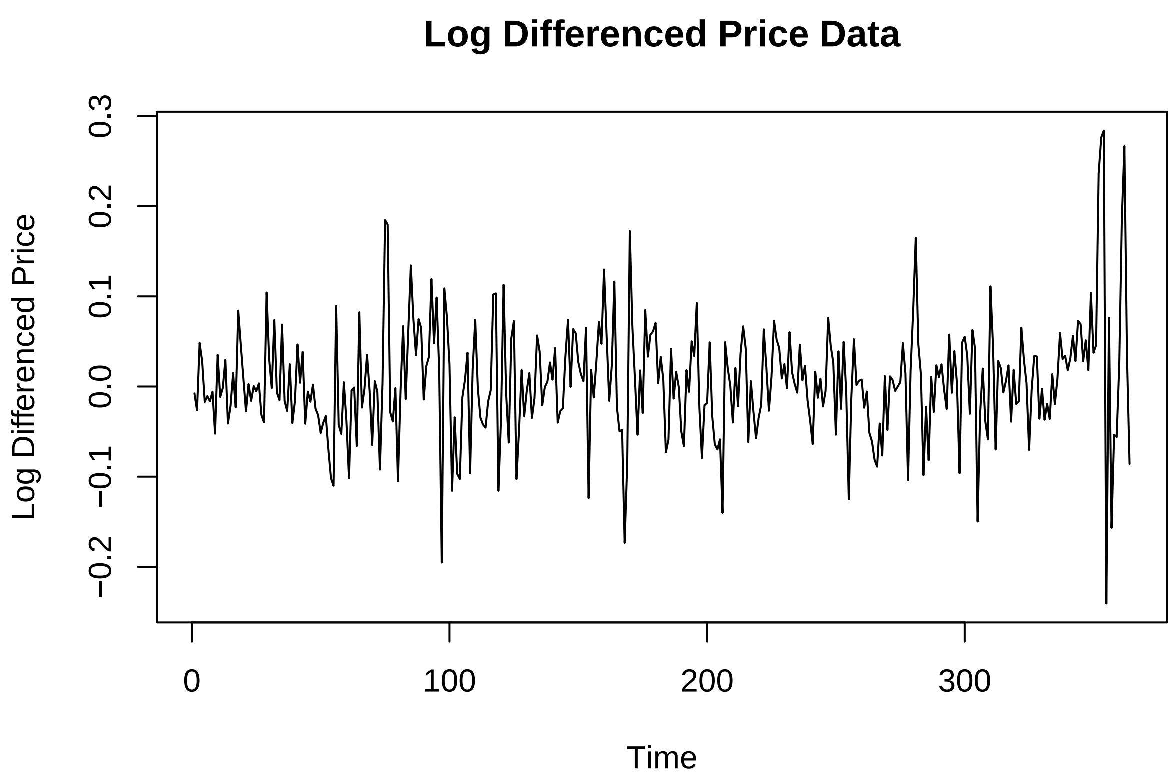
```
## ACF  0.95  0.90  0.84  0.78  0.71 0.65  0.57 0.51 0.47  0.44  0.42  0.39  0.37
## PACF 0.95 -0.04 -0.16 -0.01 -0.14 0.08 -0.24 0.18 0.23  0.00  0.01 -0.13  0.08
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.35  0.33  0.31  0.30  0.28  0.27  0.26  0.25  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)
diff_log_price = diff(price_month$price_log)
```

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



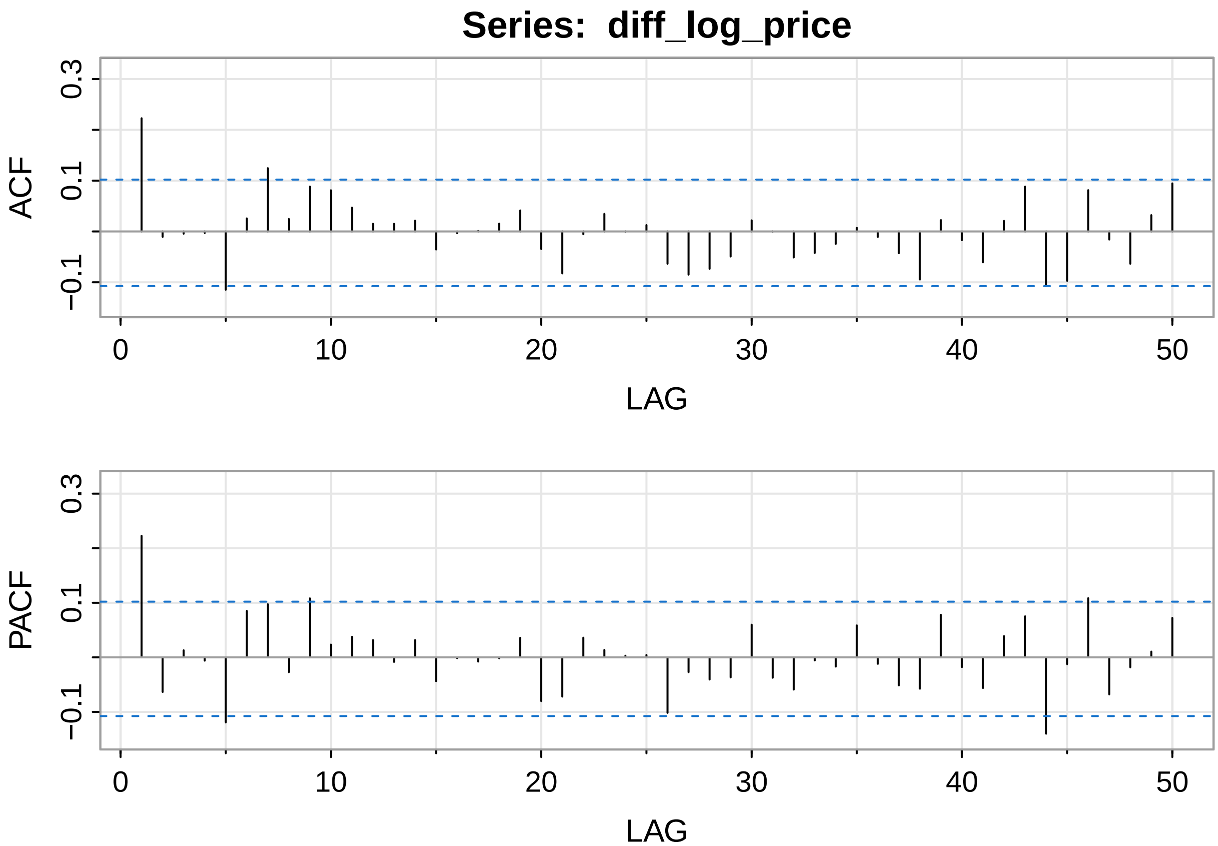

```
adf.test(diff_log_price)
```

```
## 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

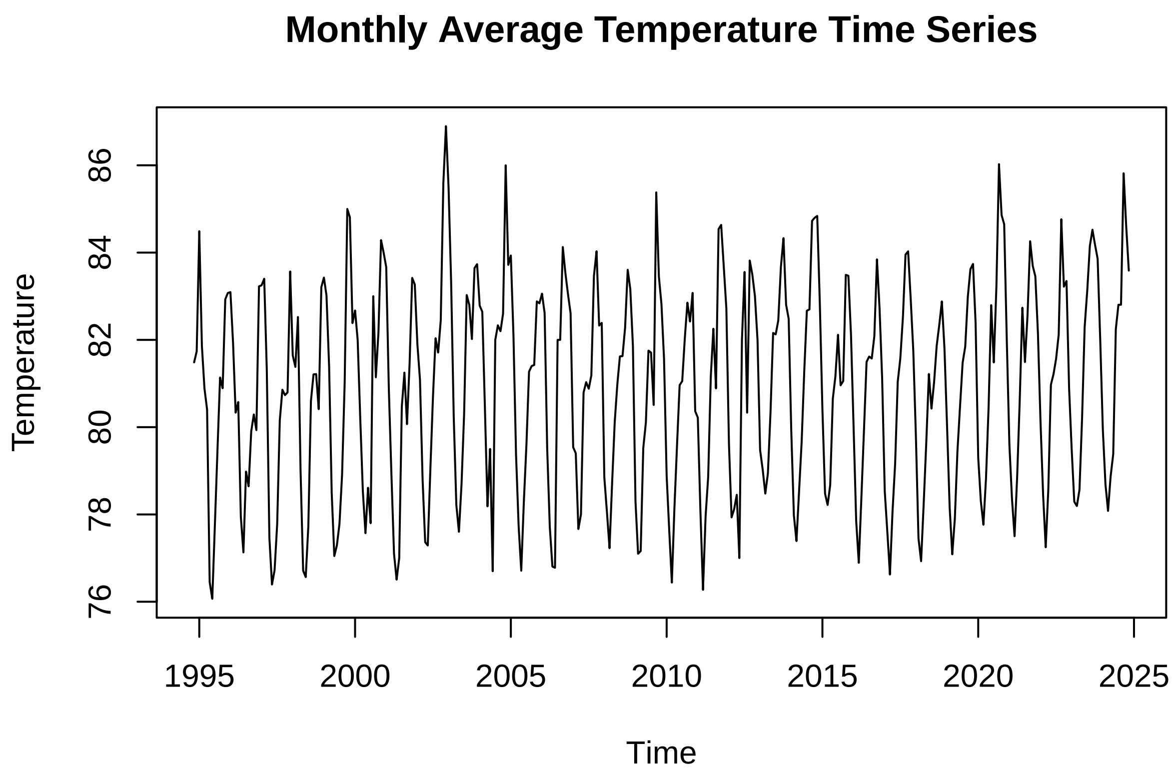
```
weather$Date <- as.Date(weather$Date)
weather$Avg_Temp <- as.numeric(gsub("", "", weather$Avg_Temp))
weather_month <- weather |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(Avg_Temp = mean(Avg_Temp, na.rm = TRUE)) |> ungroup()
```

```r
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")
```



exchange Data

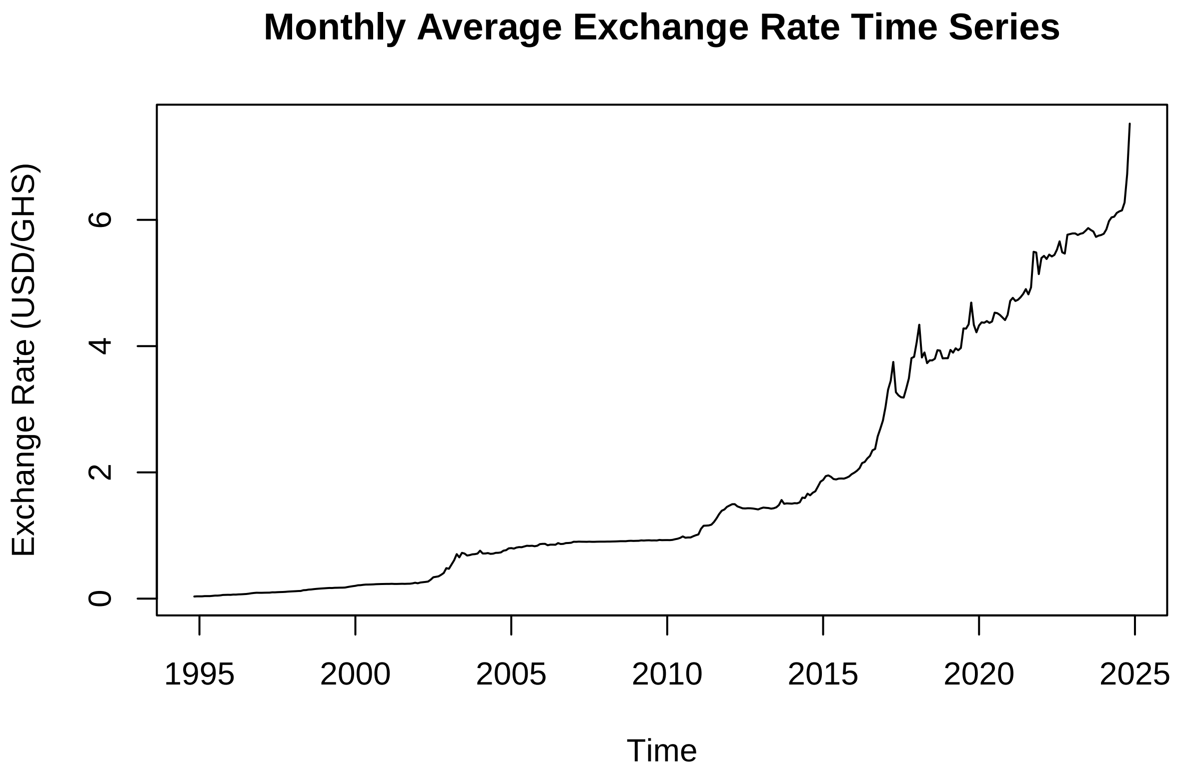
```r
exchangerate$Date <- as.Date(exchangerate$Date)
exchangerate$exchange_rate <- as.numeric(gsub("", "", exchangerate$exchange_rate))
rate_month <- exchangerate |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(exchange_rate = mean(exchange_rate, na.rm = TRUE)) |> ungroup()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

```r
summary(exchangerate)
```

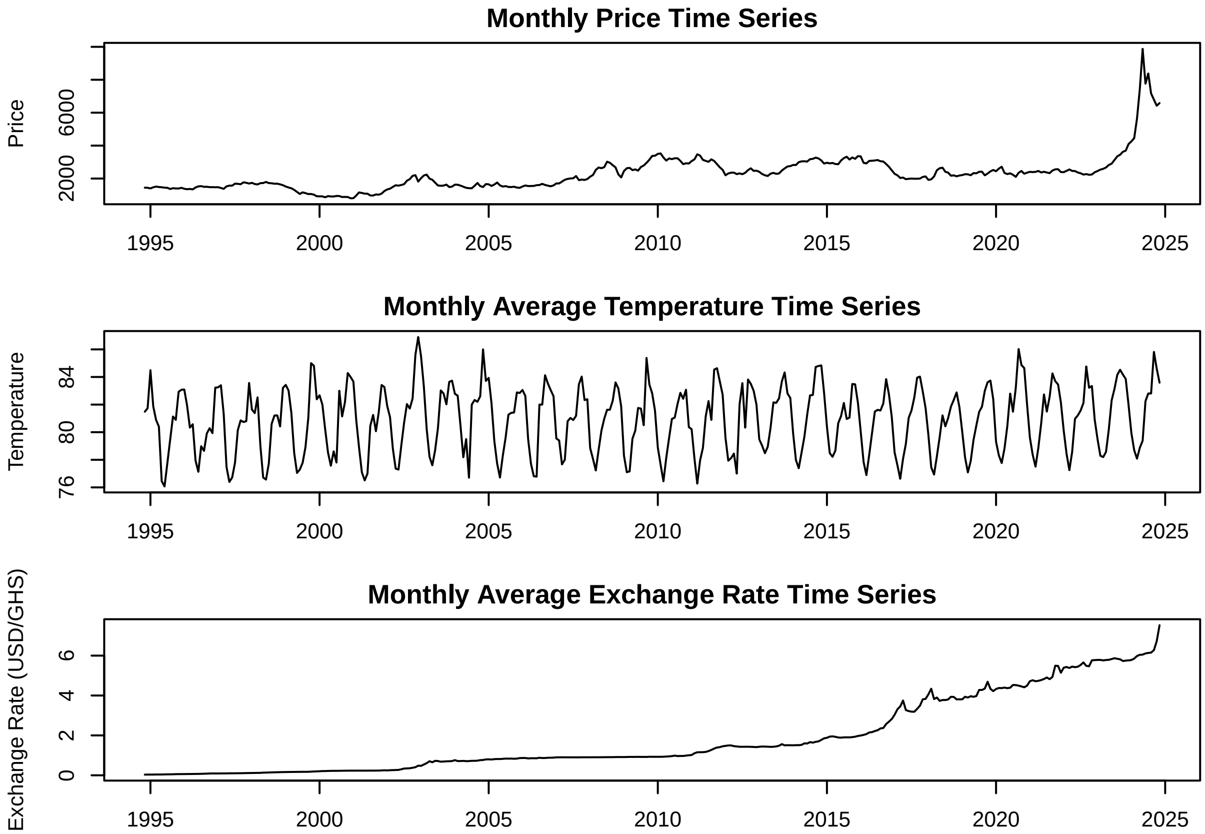
```
##       Date            exchange_rate    
##  Min.   :1992-03-01   Min.   : 0.0338  
##  1st Qu.:2000-06-01   1st Qu.: 0.5400  
##  Median :2008-09-01   Median : 1.1595  
##  Mean   :2008-08-31   Mean   : 2.8314  
##  3rd Qu.:2016-12-01   3rd Qu.: 4.2805  
##  Max.   :2025-03-01   Max.   :16.2500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

```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=
```



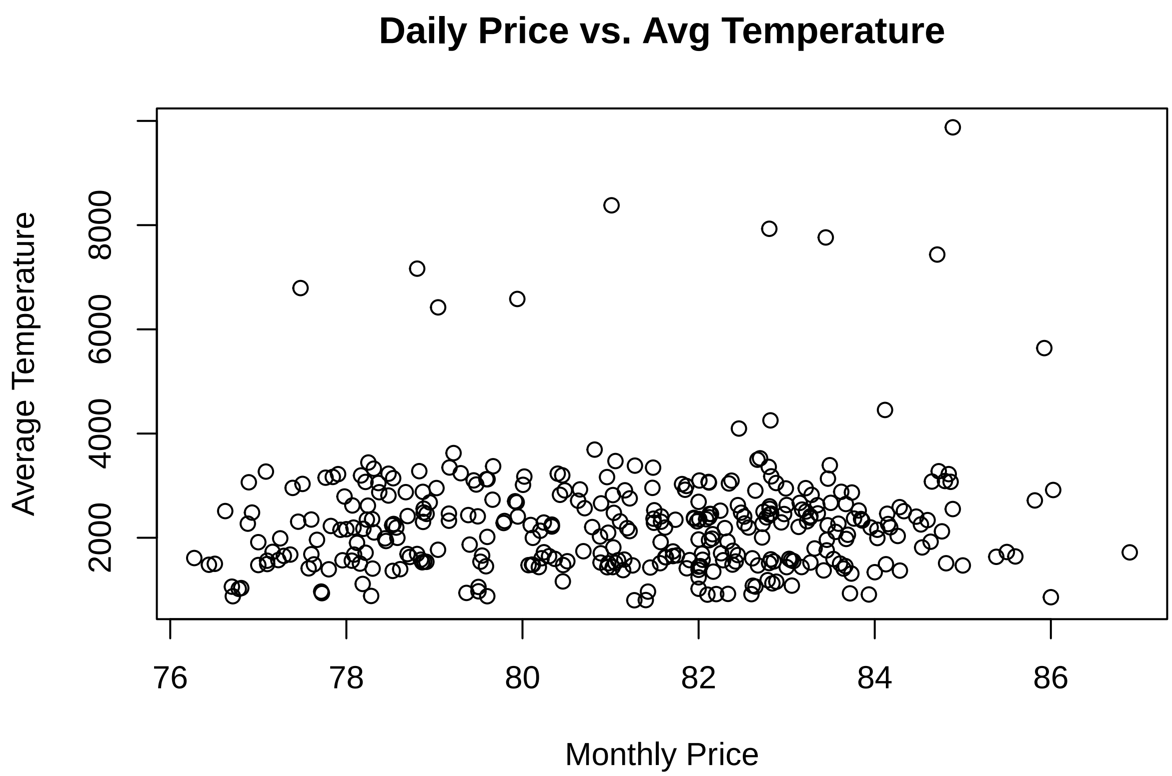

```r
par(mfrow=c(3,1), mar = c(3, 4, 2, 2))
# price
plot(price_ts, main="Monthly Price Time Series", ylab="Price", xlab="Time")
#temperature
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")
# exchange rate
ts.plot(rate_ts, main="Monthly Average Exchange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



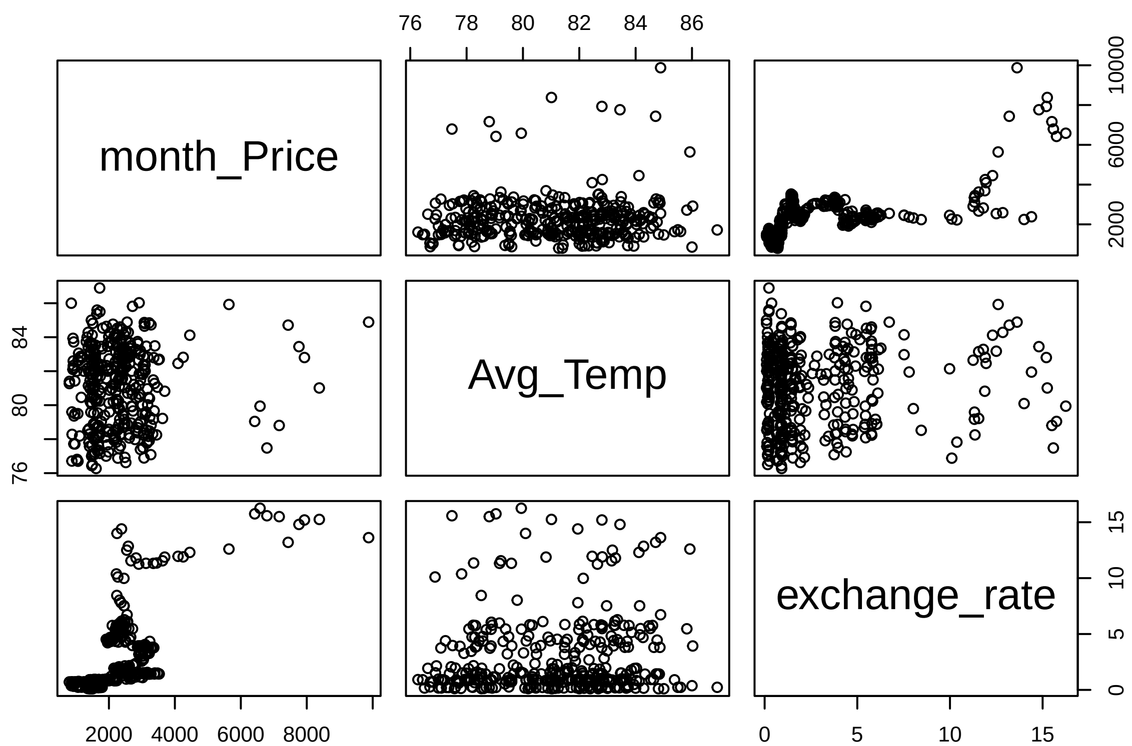
Combine and Split data

```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 =
                         c(NA, diff(price_month$price_log))) |> drop_na()
data <- data |> 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")])
```




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

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

2. Method

2.1 ETS Model

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

2.1.1 Fit Model

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

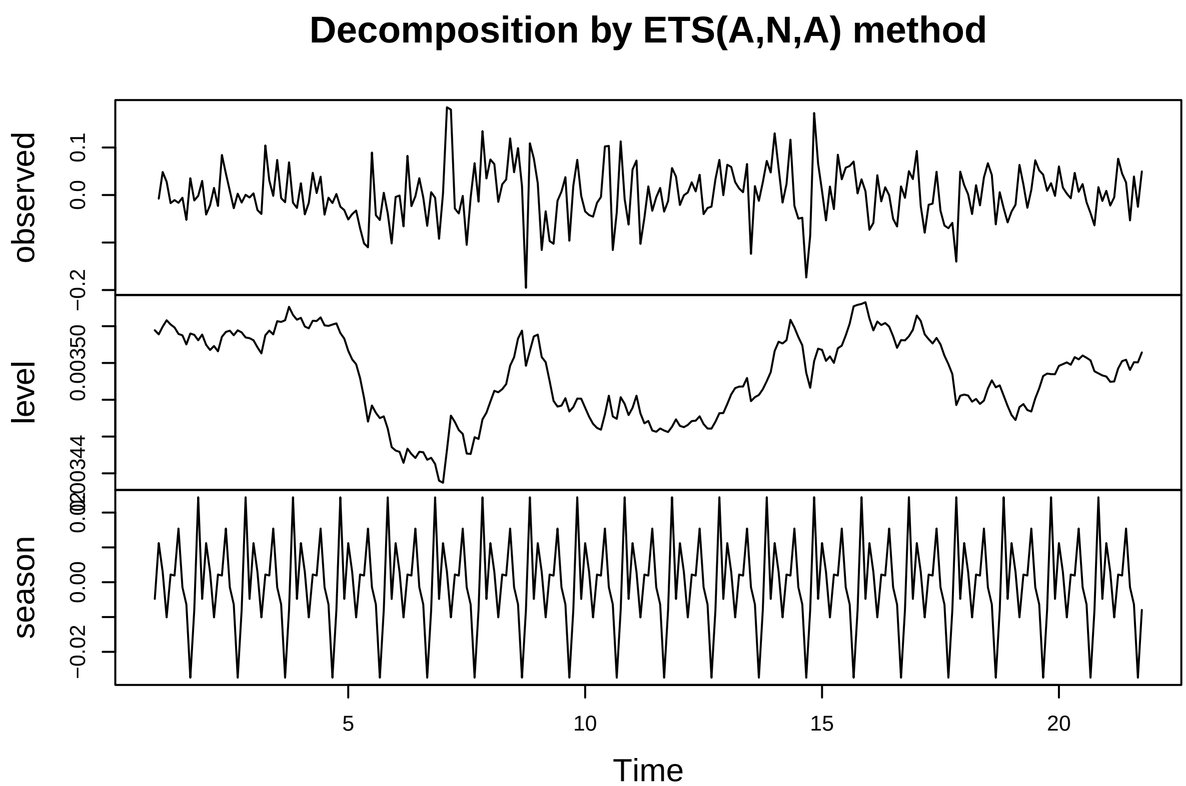
```
## ETS(A,N,A)
##
## Call:
##  ets(y = data_train_ts, model = "ANA")
##
##   Smoothing parameters:
##     alpha = 1e-04
##     gamma = 1e-04
##
##   Initial states:
##     l = 0.0035
##     s = -0.0048 0.0244 -0.008 -0.0274 -0.0064 -0.0014
##            0.0154 0.0019 0.0022 -0.0101 0.0029 0.0112
##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

```r
summary(ets_zmodel)
```

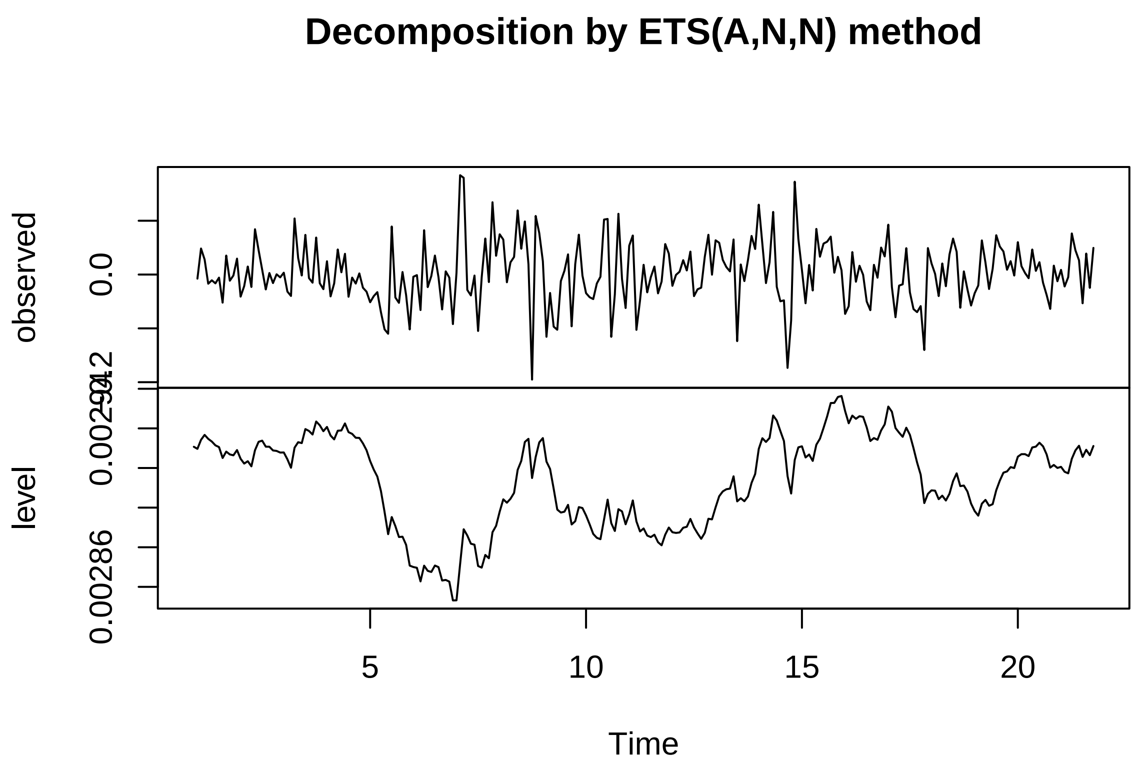
```
## ETS(A,N,N)
##
## Call:
##  ets(y = data_train_ts, model = "ZZZ")
##
##   Smoothing parameters:
##     alpha = 1e-04
##
##   Initial states:
##     l = 0.0029
##
##   sigma:  0.0569
##
```

```
##       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)
```



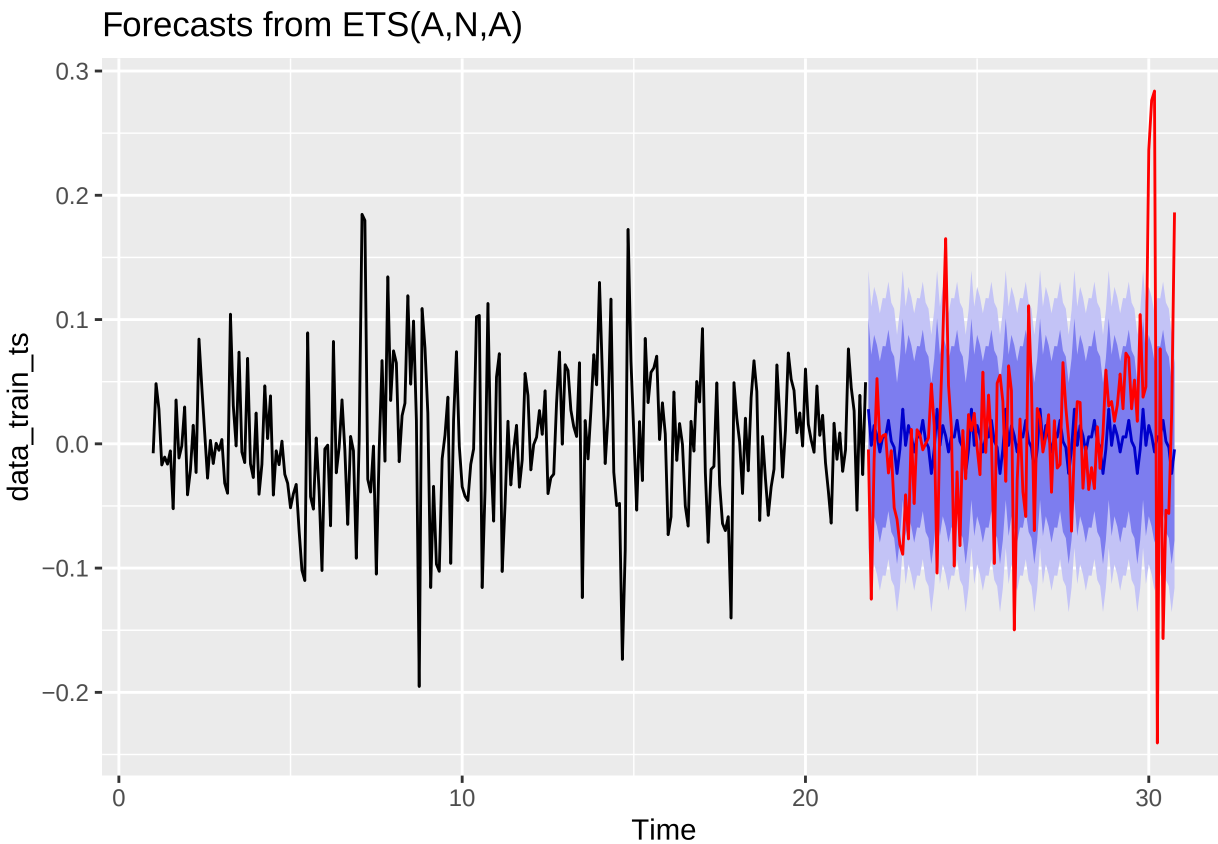

```r
plot(ets_zmodel)
```



«««<

HEAD ### 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")
```

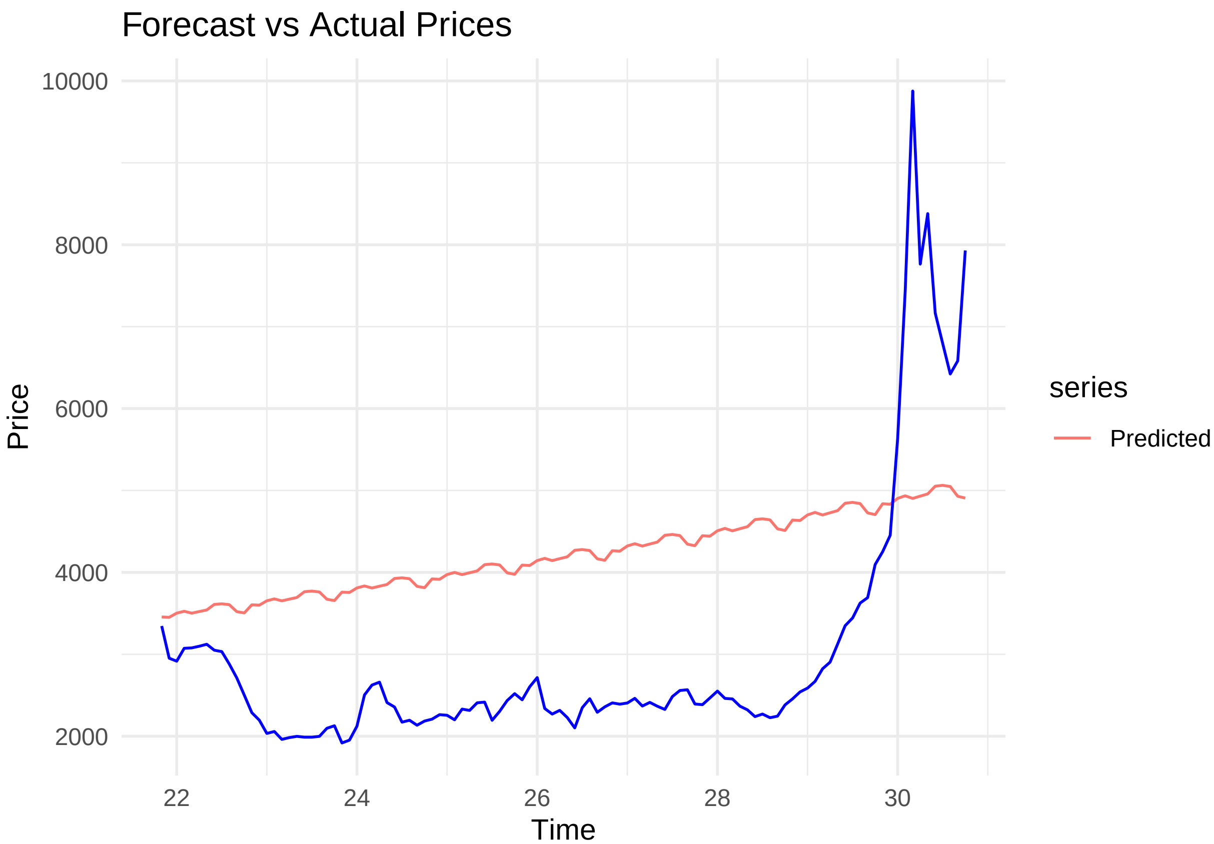


The red line is the observed actual values. The forecasted values are the central blue line within the blue shaded prediction intervals.

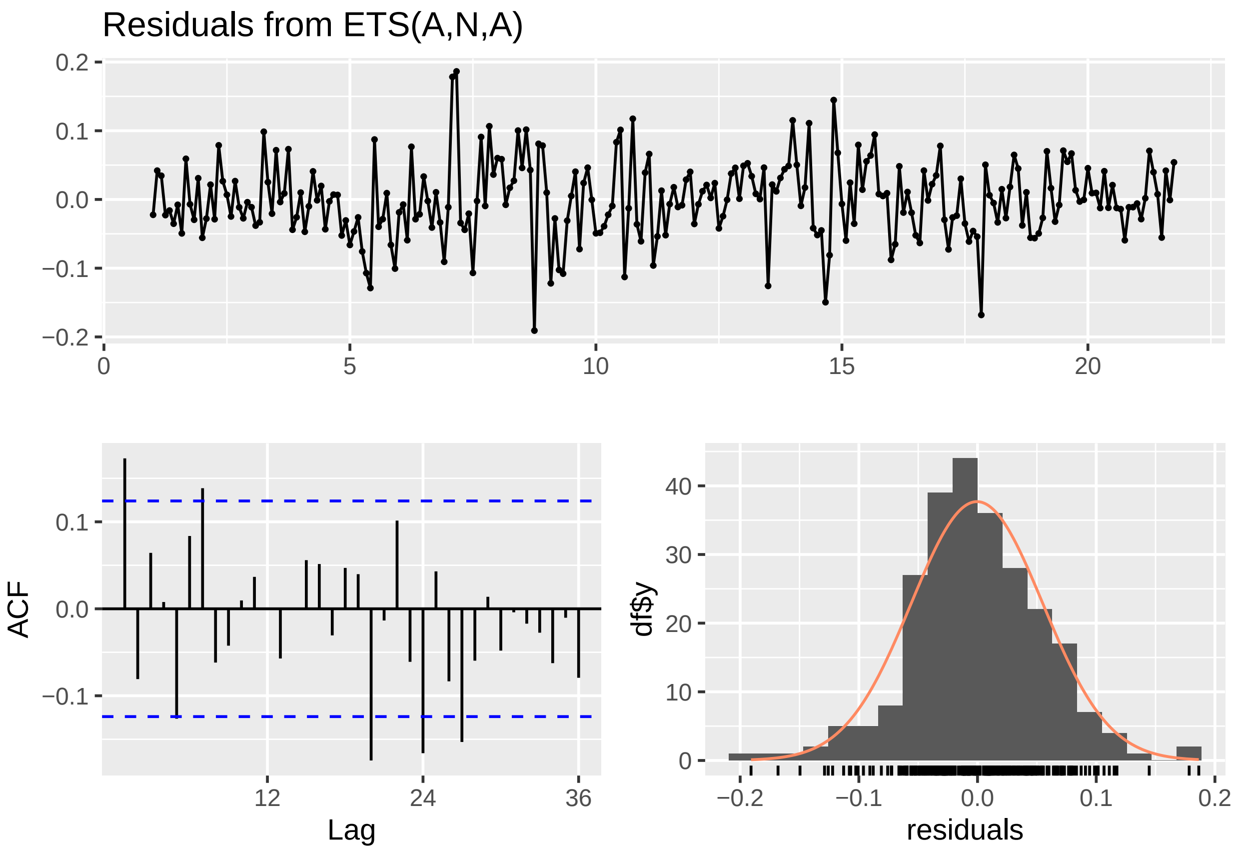
```
last_log_price <- tail(trainSet$log_price, 1)

# 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()
```

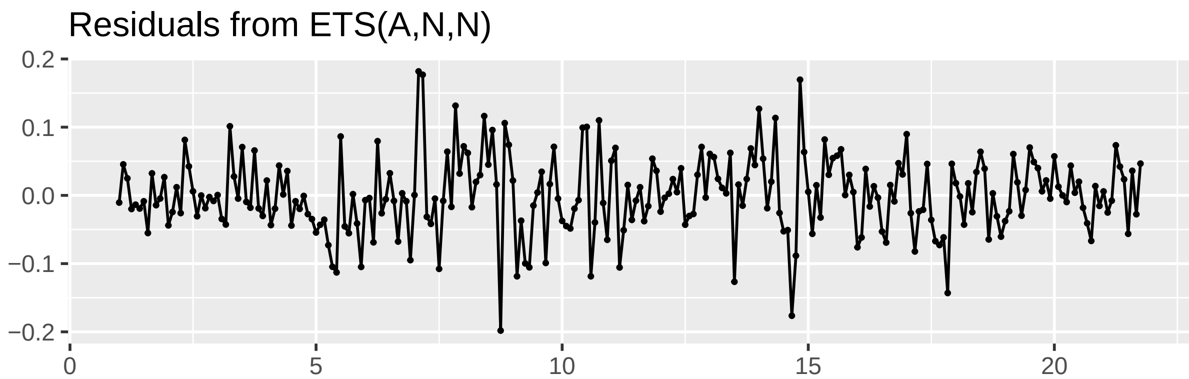


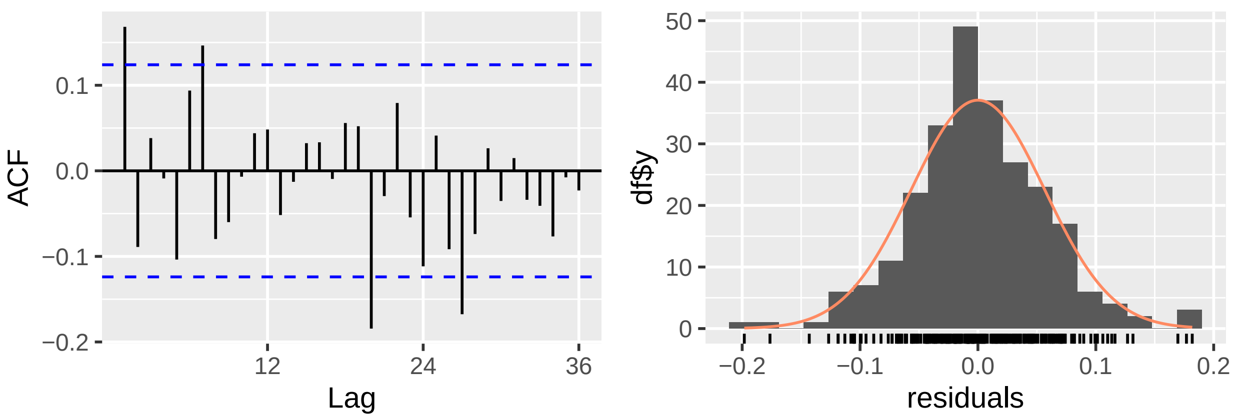

```
checkresiduals(ets_model)
```




```
##
##  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
```

```
sqrt(mean((actual_price - forecasted_price)^2))
```

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

======= ## 2.2 ARIMA Model »»»> c2884286efa75e995485ea3383a0f60ac38f463b

2.3 SARIMA Model

2.4 ARMAX Model

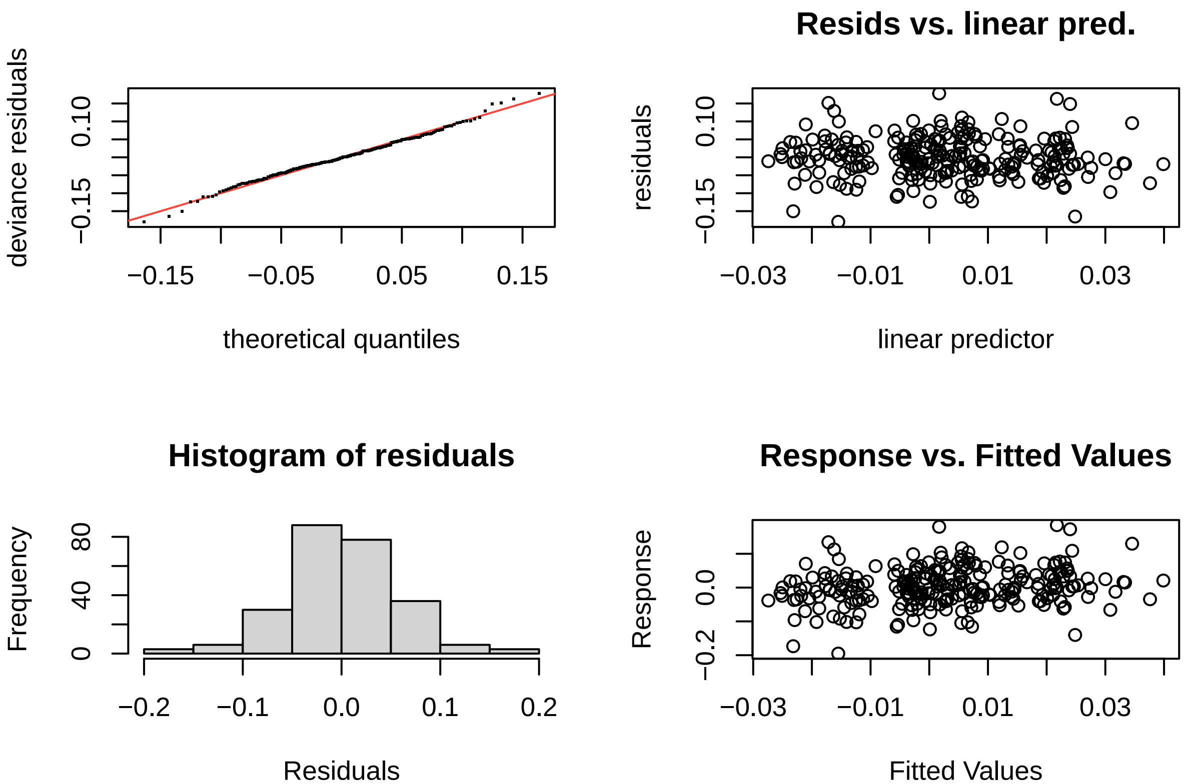
2.5 GAM Model

2.5.1 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) +
```

```r
                    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")
```

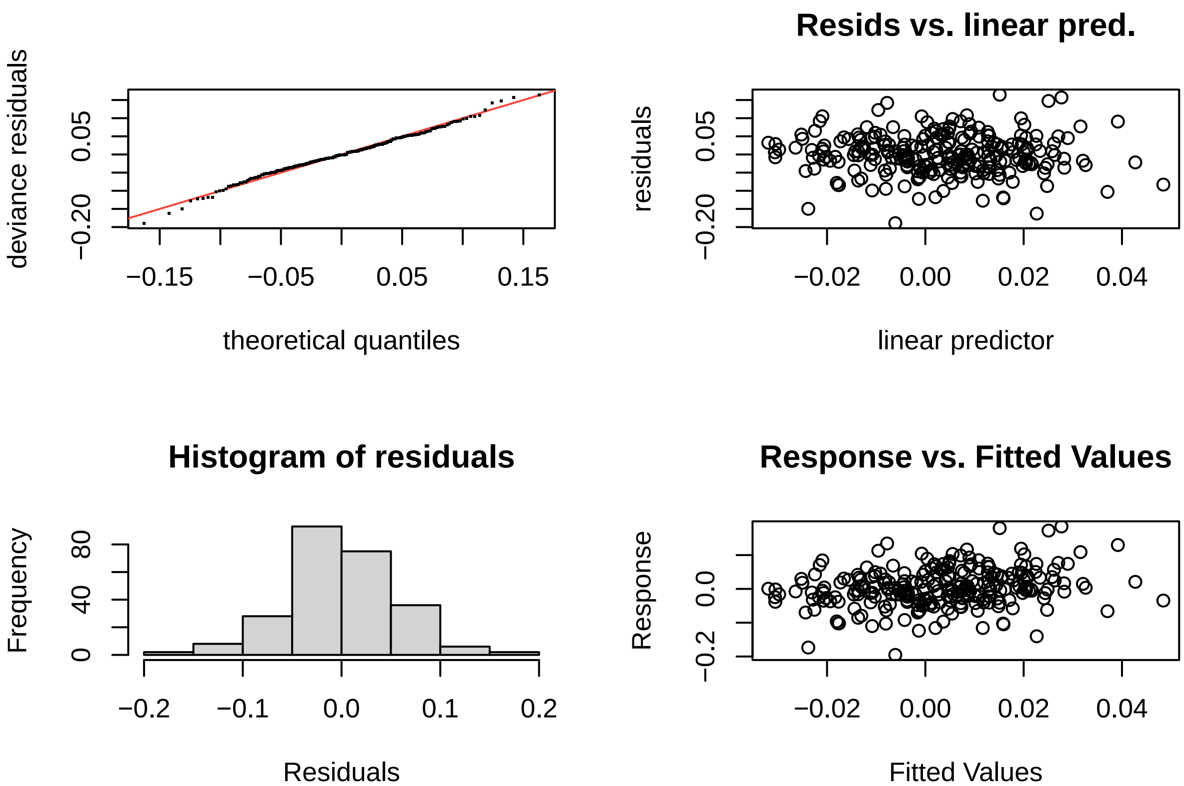
```r
gam.check(gam_model)
```




```
##
## 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
```

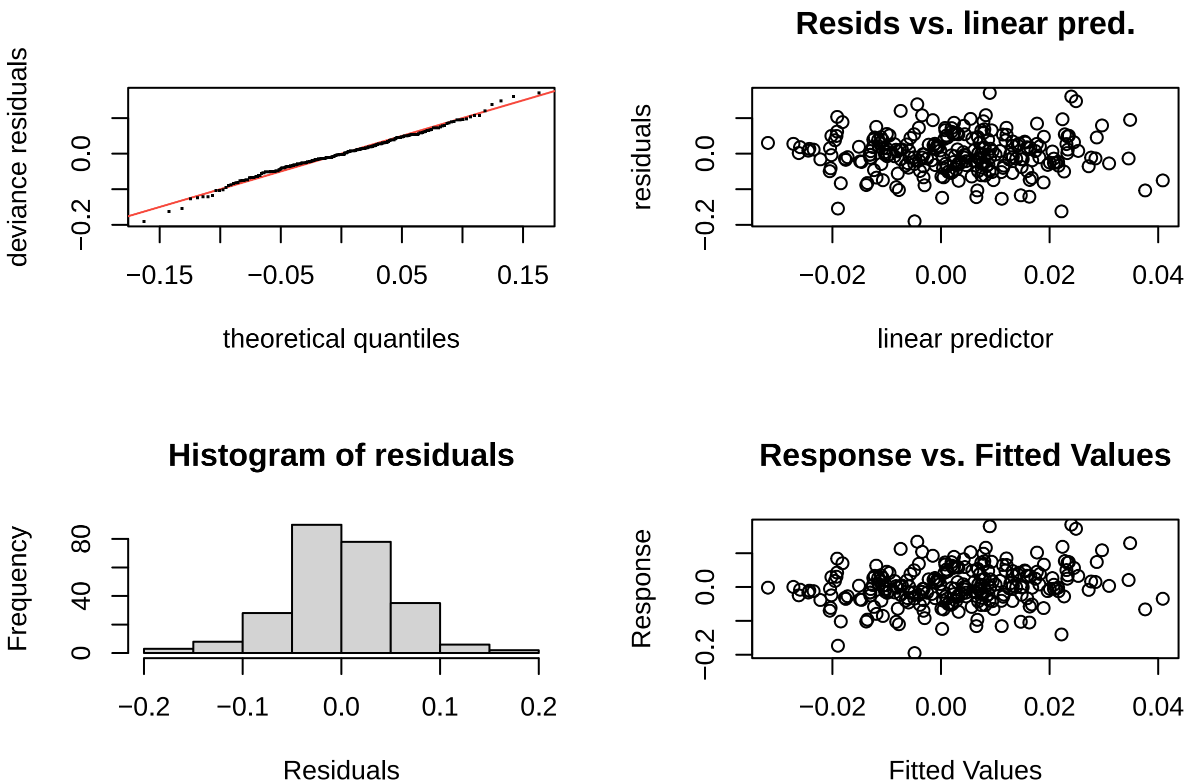
```
## s(as.numeric(Time)) 11.00  1.00    0.86   0.005 **
## s(Avg_Temp)           9.00  1.00    1.06   0.795
## s(exchange_rate)      9.00  1.00    0.87   0.020 *
## s(monthFac)          12.00  5.67      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```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   0.015 *
## s(Avg_Temp)             9.00  1.00    1.05   0.825
## 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    0.79
## s(log(exchange_rate)) 9.00e+00 1.00e+00    0.88    0.01 **
## s(monthFac)           1.20e+01 9.21e+00      NA      NA
## s(yday(Time))         8.00e+00 7.79e-05    1.20    1.00
## ---
## 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
```

2.5.2 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)
```

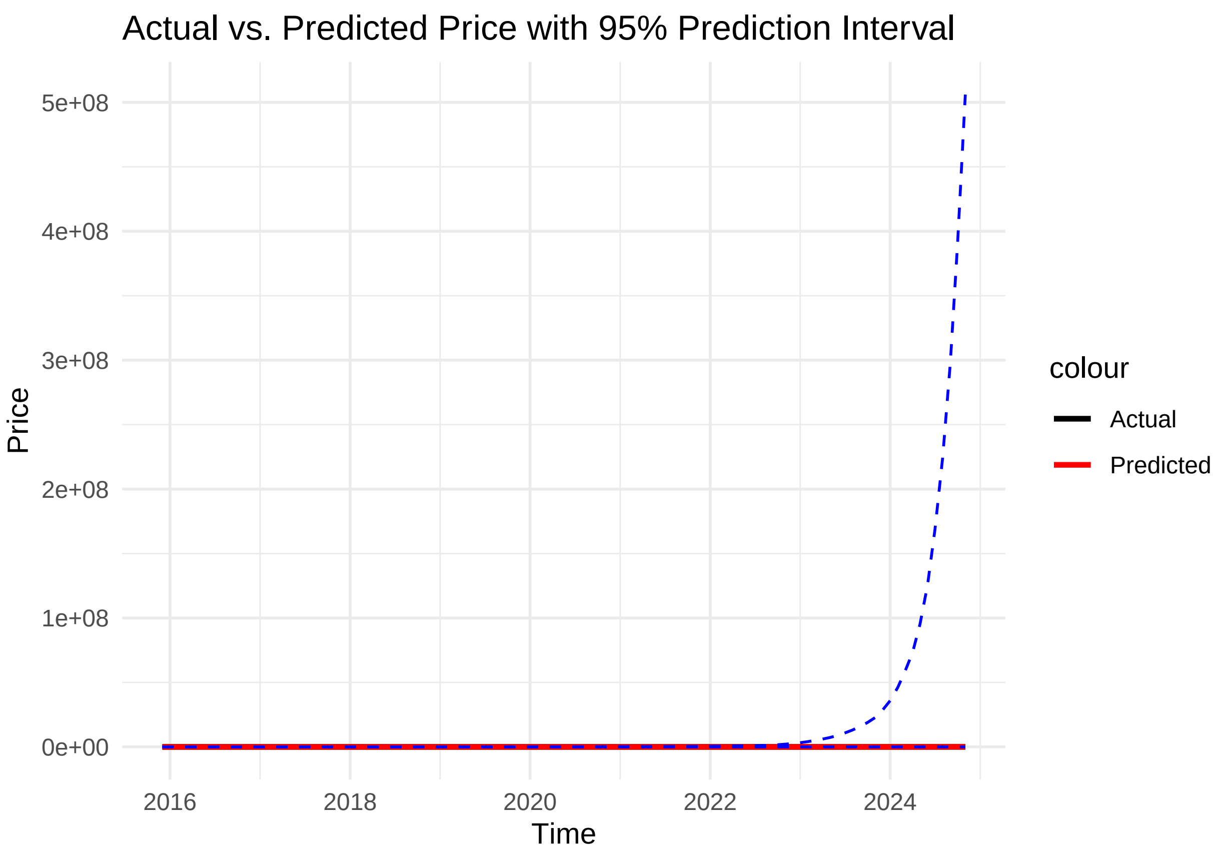
```r
testSet2$pred_price2 <- exp(testSet2$pred_log_price2)

# gam3
testSet2$pred_log3 <- predict(gam_model3, newdata = testSet2)
testSet2$pred_log_price3 <- last_log_price + cumsum(testSet2$pred_log3)
testSet2$pred_price3 <- exp(testSet2$pred_log_price3)
```

```r
## DO NOT KEEP

# gam1
preds <- predict(gam_model, newdata = testSet, se.fit = TRUE)

testSet$pred_log <- preds$fit
testSet$pred_log_upper <- preds$fit + 1.96 * preds$se.fit
testSet$pred_log_lower <- preds$fit - 1.96 * preds$se.fit

testSet$pred_log_price <- last_log_price + cumsum(testSet$pred_log)
testSet$pred_price <- exp(testSet$pred_log_price)

testSet$pred_log_price_upper <- last_log_price + cumsum(testSet$pred_log_upper)
testSet$pred_price_upper <- exp(testSet$pred_log_price_upper)

testSet$pred_log_price_lower <- last_log_price + cumsum(testSet$pred_log_lower)
testSet$pred_price_lower <- exp(testSet$pred_log_price_lower)

# gam2
preds2 <- predict(gam_model2, newdata = testSet, se.fit = TRUE)

testSet$pred_log2 <- preds2$fit
testSet$pred_log_upper2 <- preds2$fit + 1.96 * preds2$se.fit
testSet$pred_log_lower2 <- preds2$fit - 1.96 * preds2$se.fit

testSet$pred_log_price2 <- last_log_price + cumsum(testSet$pred_log2)
testSet$pred_price2 <- exp(testSet$pred_log_price2)

testSet$pred_log_price_upper2 <- last_log_price + cumsum(testSet$pred_log_upper2)
testSet$pred_price_upper2 <- exp(testSet$pred_log_price_upper2)

testSet$pred_log_price_lower2 <- last_log_price + cumsum(testSet$pred_log_lower2)
testSet$pred_price_lower2 <- exp(testSet$pred_log_price_lower2)

# gam3
preds3 <- predict(gam_model3, newdata = testSet, se.fit = TRUE)

testSet$pred_log3 <- preds3$fit
testSet$pred_log_upper3 <- preds3$fit + 1.96 * preds3$se.fit
testSet$pred_log_lower3 <- preds3$fit - 1.96 * preds3$se.fit

testSet$pred_log_price3 <- last_log_price + cumsum(testSet$pred_log3)
testSet$pred_price3 <- exp(testSet$pred_log_price3)

testSet$pred_log_price_upper3 <- last_log_price + cumsum(testSet$pred_log_upper3)
testSet$pred_price_upper3 <- exp(testSet$pred_log_price_upper3)
```

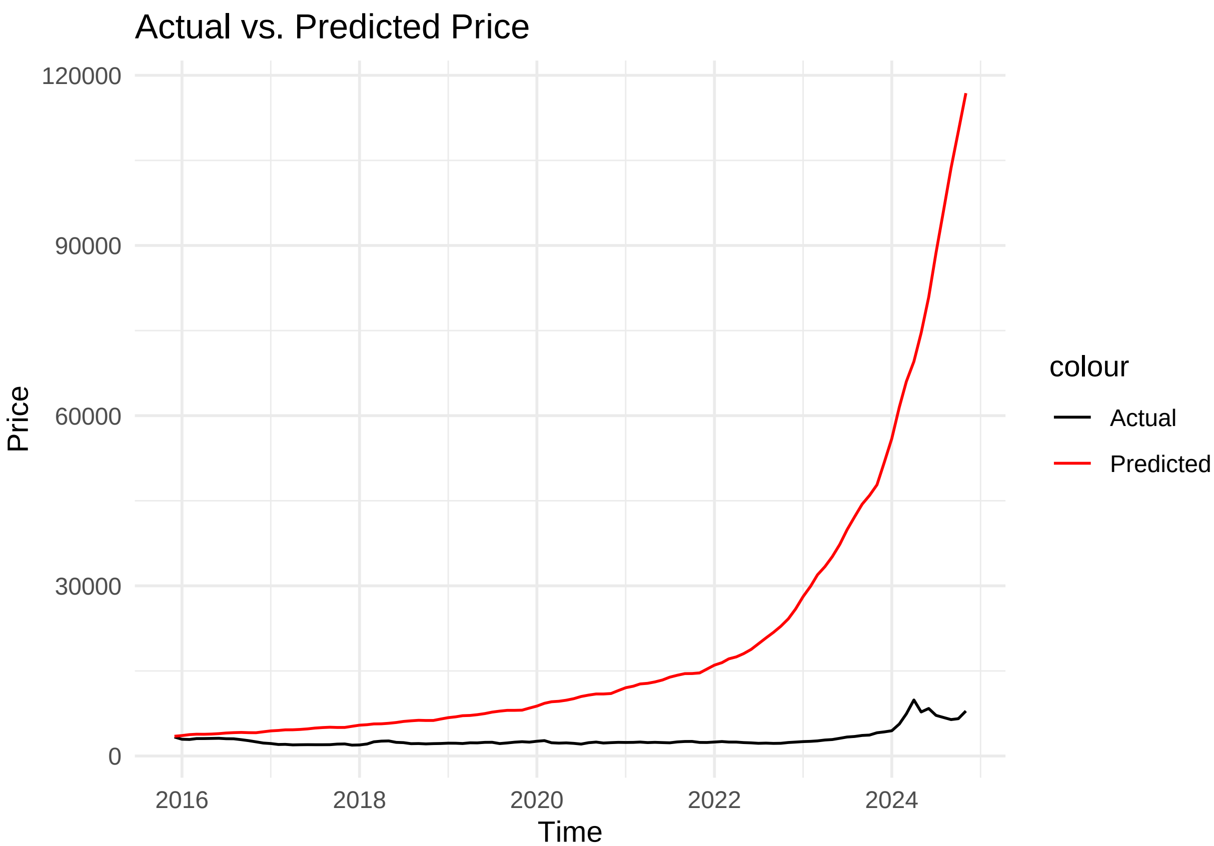
```r
testSet$pred_log_price_lower3 <- last_log_price + cumsum(testSet$pred_log_lower3)
testSet$pred_price_lower3 <- exp(testSet$pred_log_price_lower3)
```

```r
## DO NOT KEEP

ggplot(testSet, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual"), size = 1) +
  geom_line(aes(y = pred_price, color = "Predicted"), size = 1) +
  geom_line(aes(y = pred_price_upper), linetype = "dashed", color = "blue") +
  geom_line(aes(y = pred_price_lower), linetype = "dashed", color = "blue") +
  labs(title = "Actual vs. Predicted Price with 95% Prediction Interval",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red")) +
  theme_minimal()
```

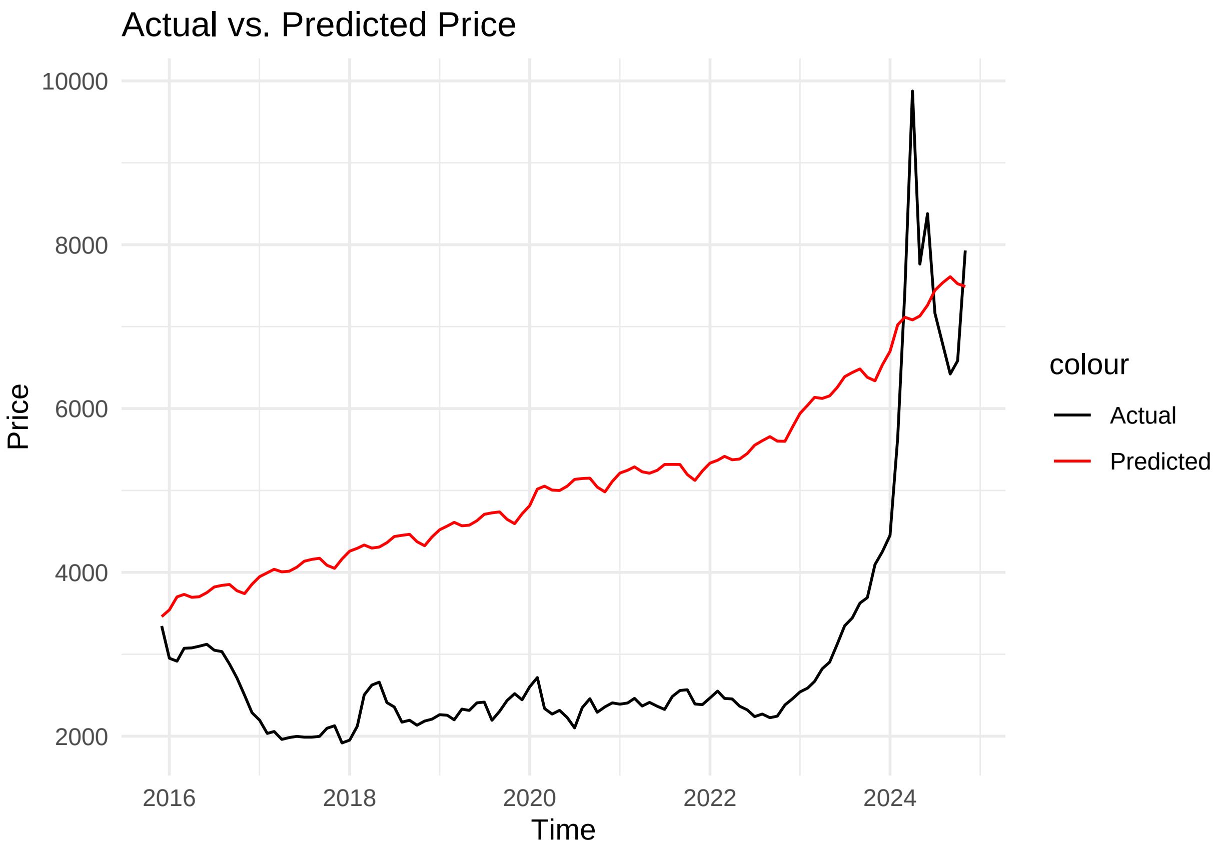
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



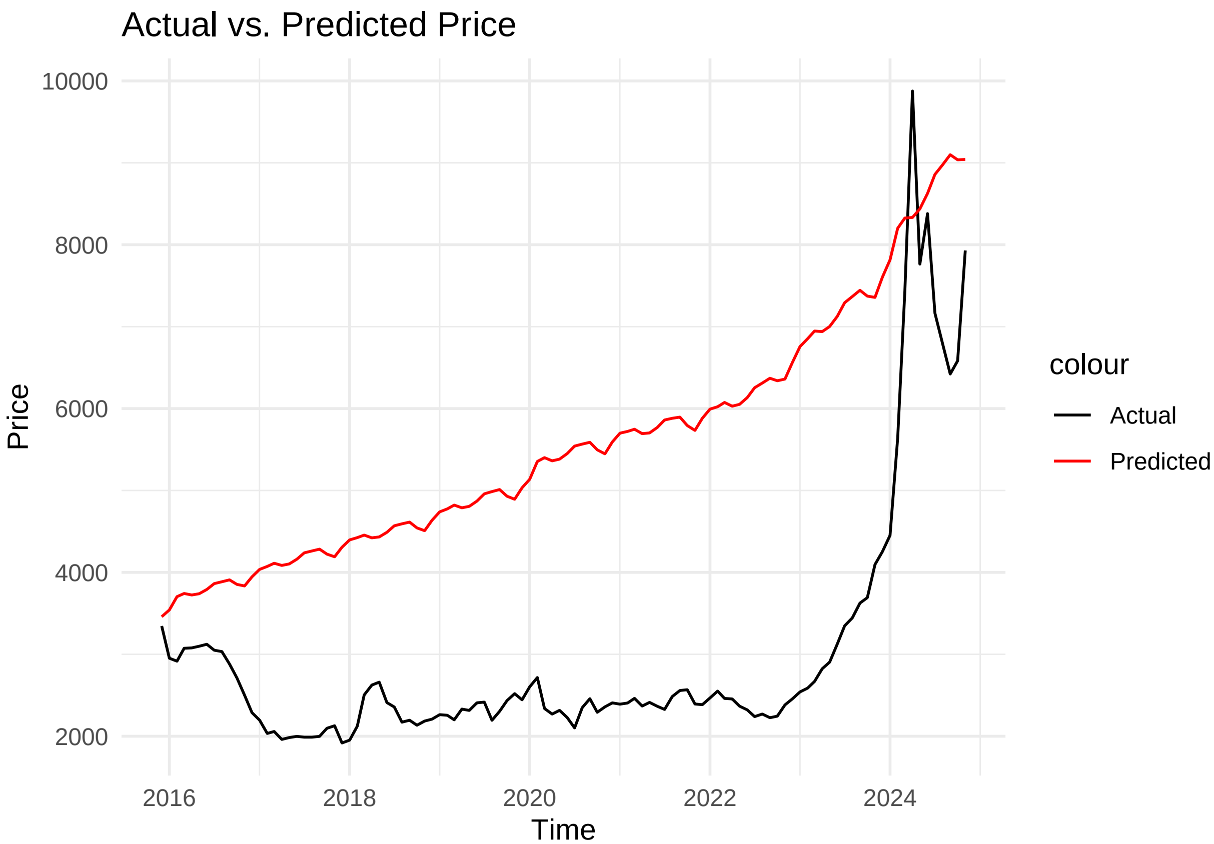

```r
ggplot(testSet, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price, color = "Predicted")) +
  labs(title = "Actual vs. Predicted Price",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red"))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




```r
ggplot(testSet2, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price2, color = "Predicted")) +
  labs(title = "Actual vs. Predicted Price",
       x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "Predicted" = "red")) +
  theme_minimal()
```




```r
ggplot(testSet2, aes(x = Time)) +
  geom_line(aes(y = month_Price, color = "Actual")) +
  geom_line(aes(y = pred_price3, color = "Predicted")) +
  labs(title = "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
```

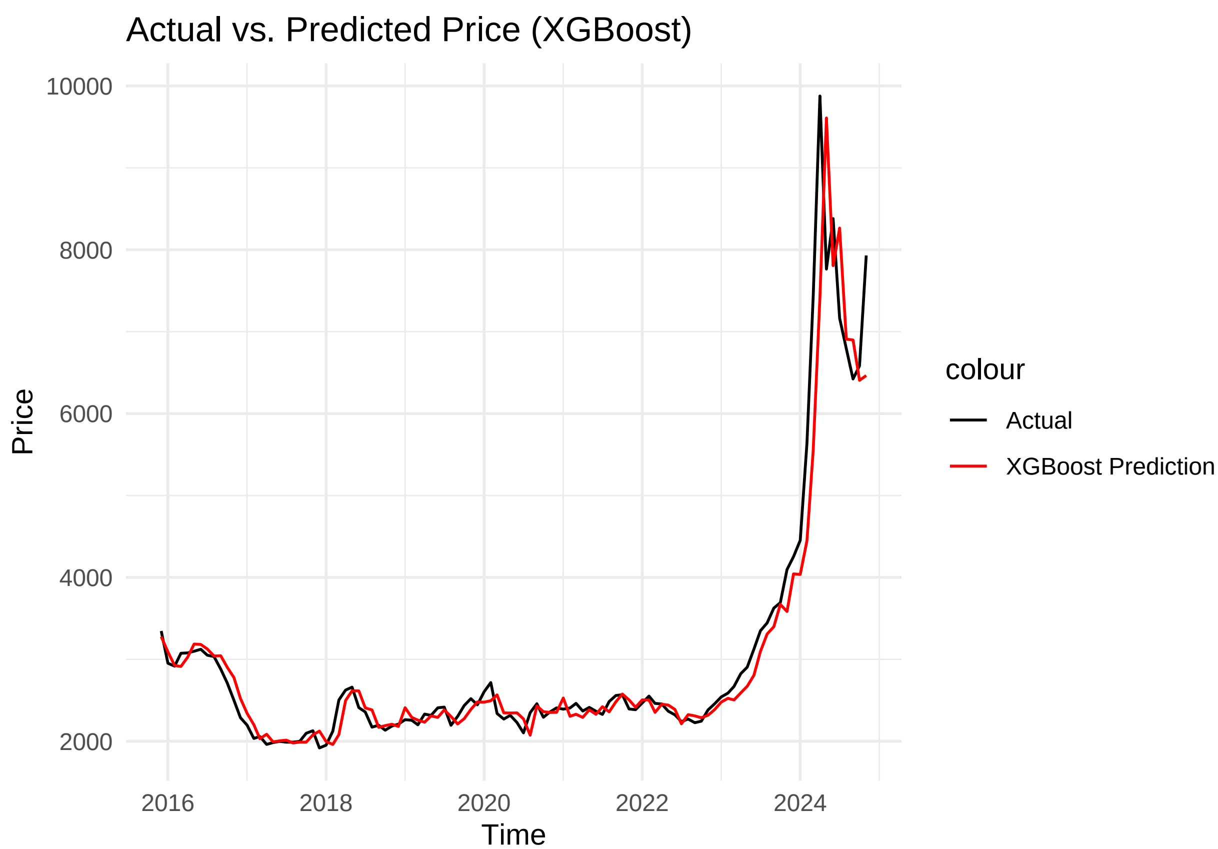
2.6 Walk-Forward Validation with XGBoost Model

2.6.1 Fit and Forecast

```r
ntest <- nrow(data) - cutoff
predictions <- c()
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
ggplot(xgb_walk_df, aes(x = Time)) + geom_line(aes(y = Actual, color = "Actual")) +
  geom_line(aes(y = Predicted, color = "XGBoost Prediction")) +
  labs(title = "Actual vs. Predicted Price (XGBoost)", x = "Time", y = "Price") +
  scale_color_manual(values = c("Actual" = "black", "XGBoost Prediction" = "red")) +
  theme_minimal()
```




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

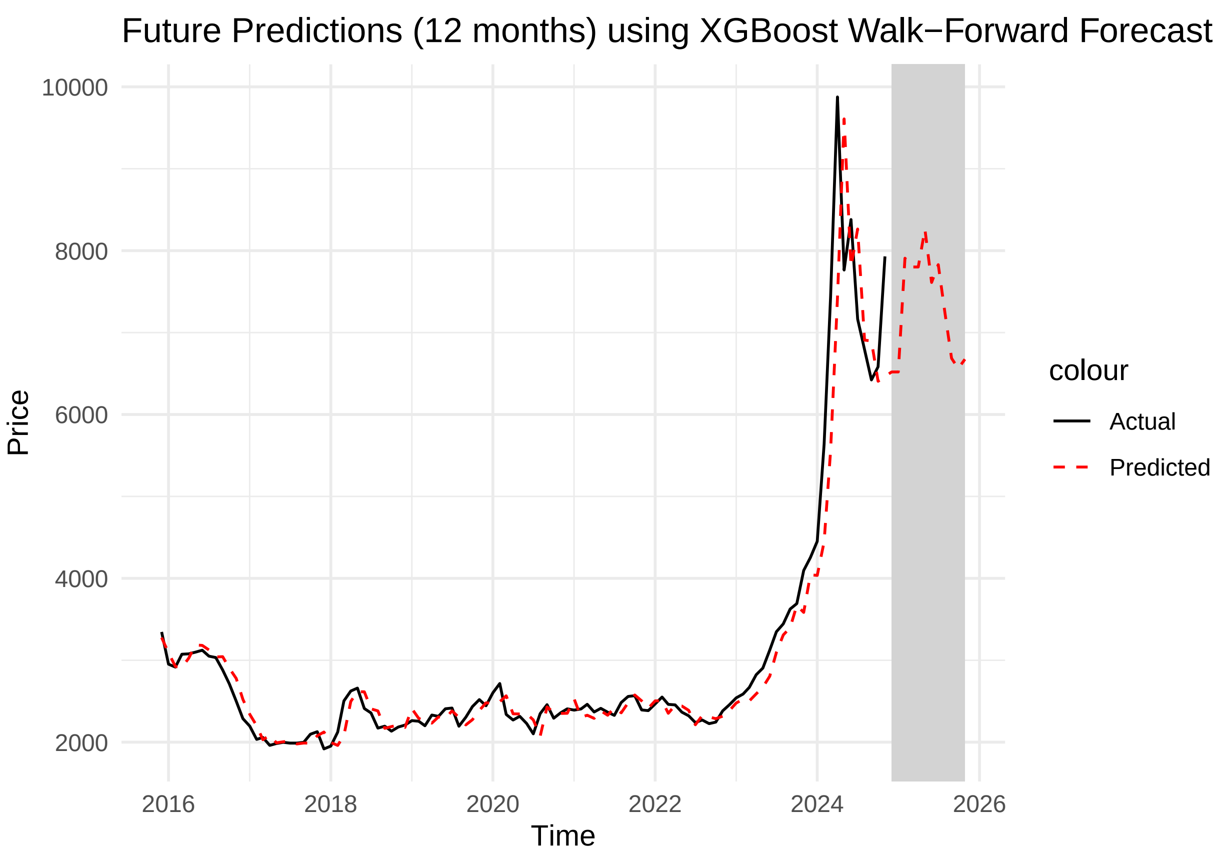
```
## [1] 435.0732
```

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
}

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

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

```
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                   
```