STA457 Project

Xing Yu Wang

2025-03-29

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

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

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

```r
library(astsa)
```

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

```r
library(tseries)
```

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

```r
library(mgcv)
library(vars)
```

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

## Warning: package 'strucchange' was built under R version 4.3.3

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

## Warning: package 'sandwich' was built under R version 4.3.3

## Warning: package 'urca' was built under R version 4.3.3

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

1. EDA

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

1.1 Clean Data

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

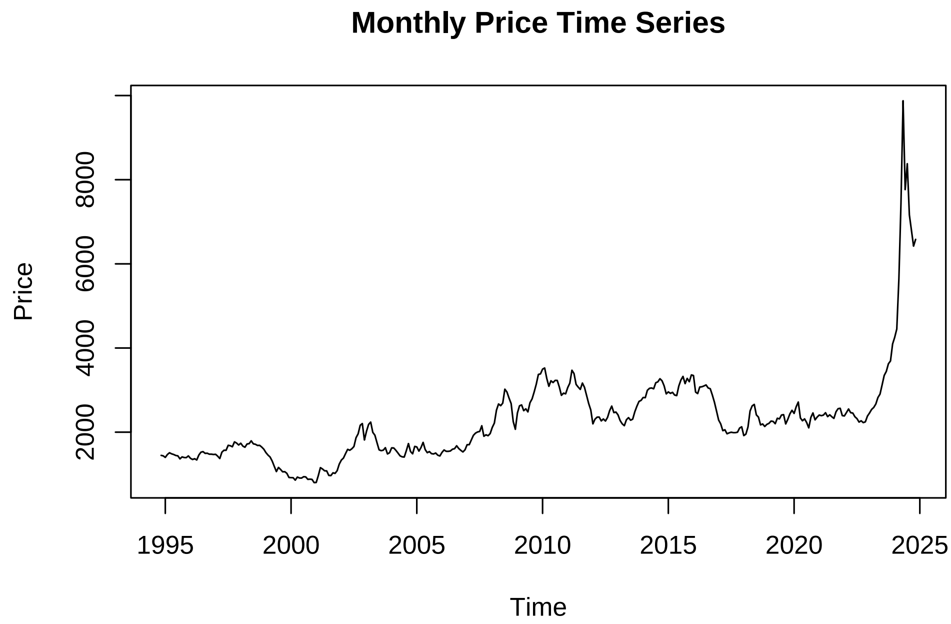
```r
price$Date <- as.Date(price$Date, format="%d/%m/%Y")
price$Daily_Price <- as.numeric(gsub(",", "", price$Daily_Price))
price_month <- price |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(month_Price = mean(Daily_Price, na.rm = TRUE)) |> ungroup()
```

```r
summary(price)
```

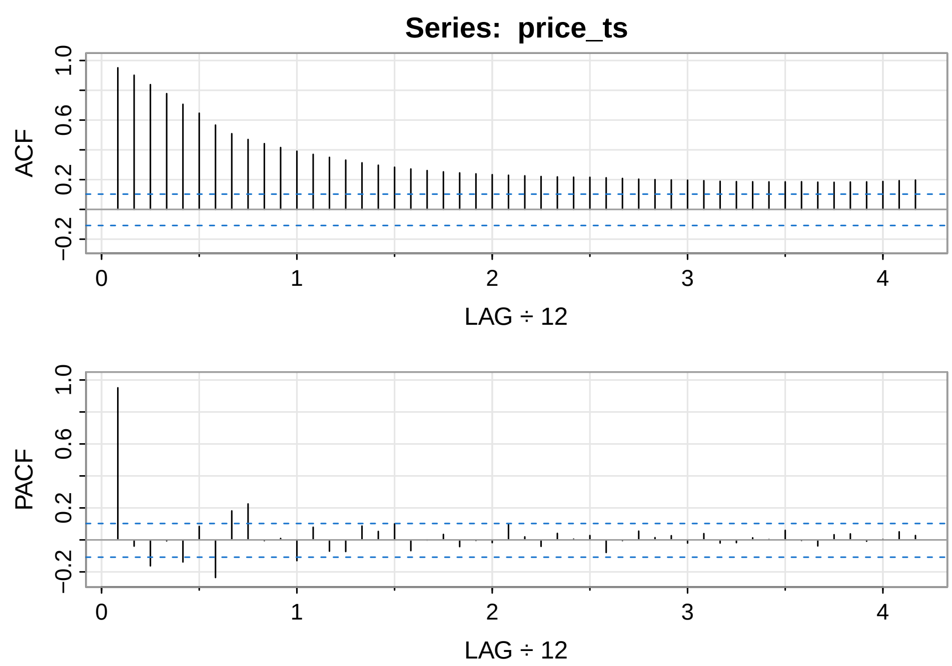
```
##       Date             Daily_Price
##  Min.   :1994-10-03   Min.   :  774.1
##  1st Qu.:2002-05-16   1st Qu.: 1557.8
##  Median :2009-12-17   Median : 2202.0
##  Mean   :2009-12-17   Mean   : 2350.1
##  3rd Qu.:2017-07-24   3rd Qu.: 2738.1
##  Max.   :2025-02-27   Max.   :11984.7
```

```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")
```




```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
```

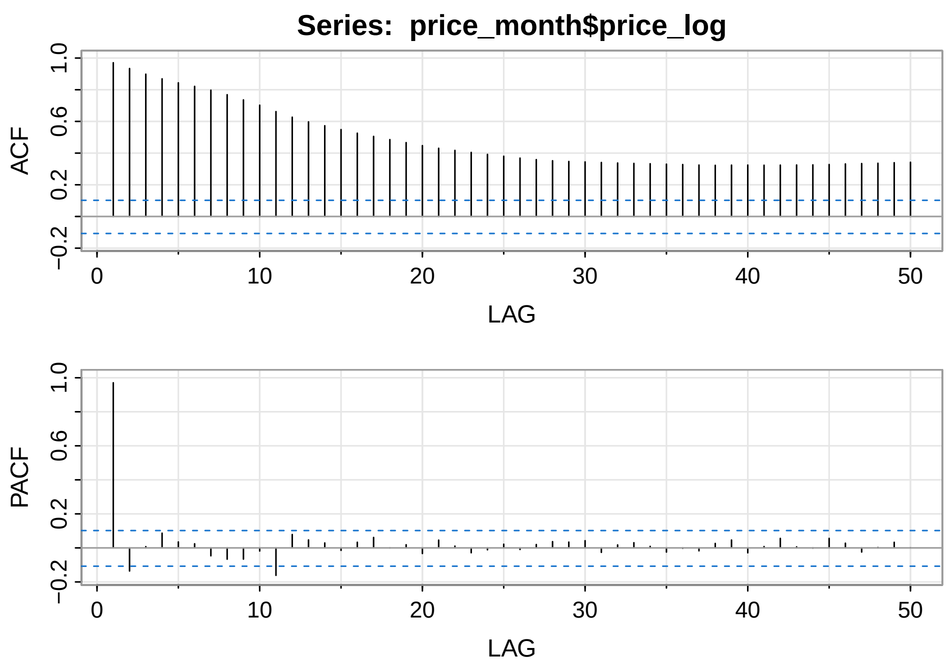
```
ndiffs(price_ts)
```

```
## [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
```

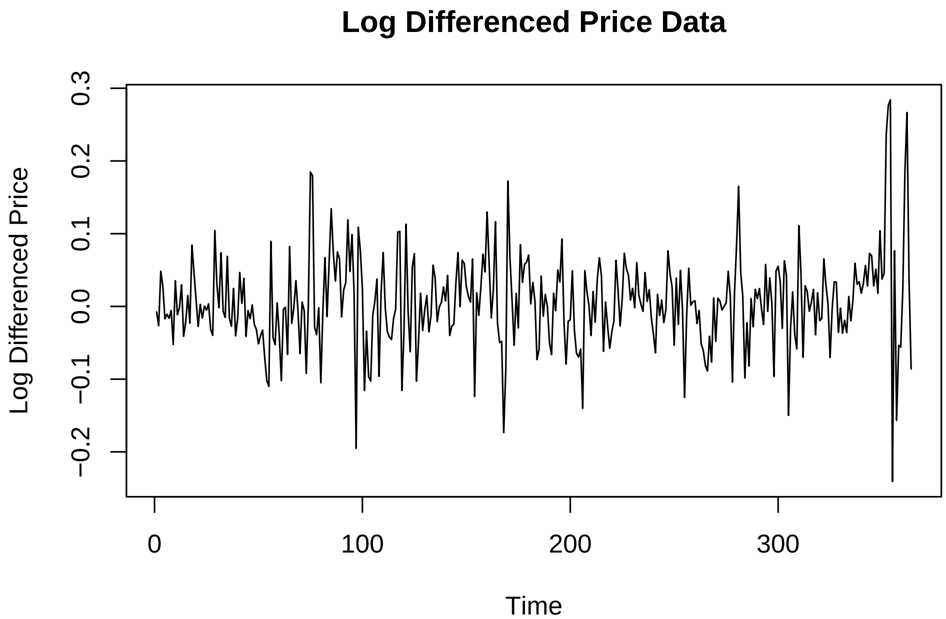
```
acf2(price_month$price_log, 50)
```




```
##      [,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.

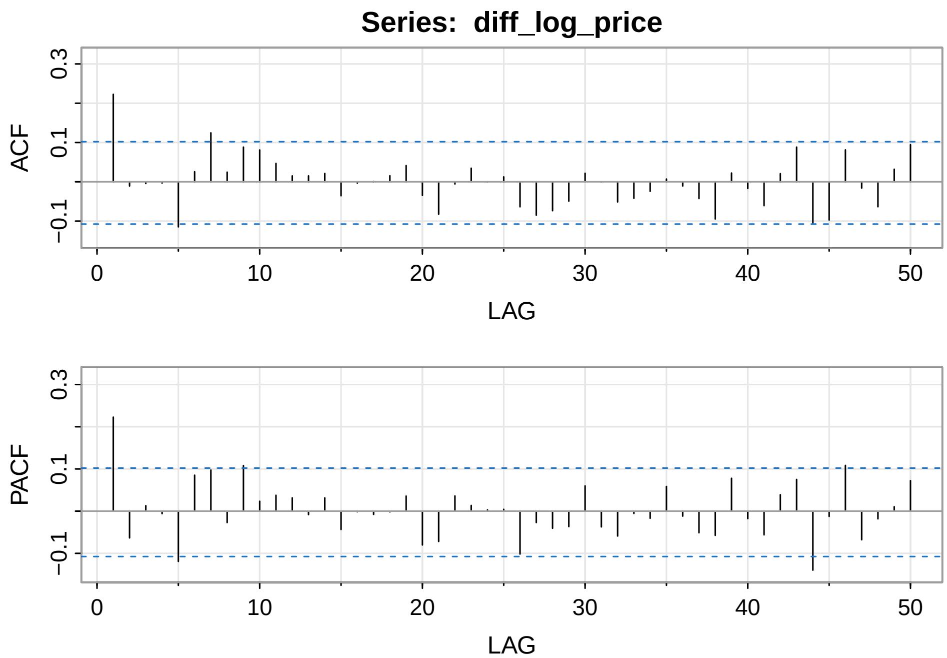
```r
diff_log_price = diff(price_month$price_log)
ts.plot(diff_log_price, main = "Log Differenced Price Data", ylab = "Log Differenced Price")
```




```r
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
```

```r
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

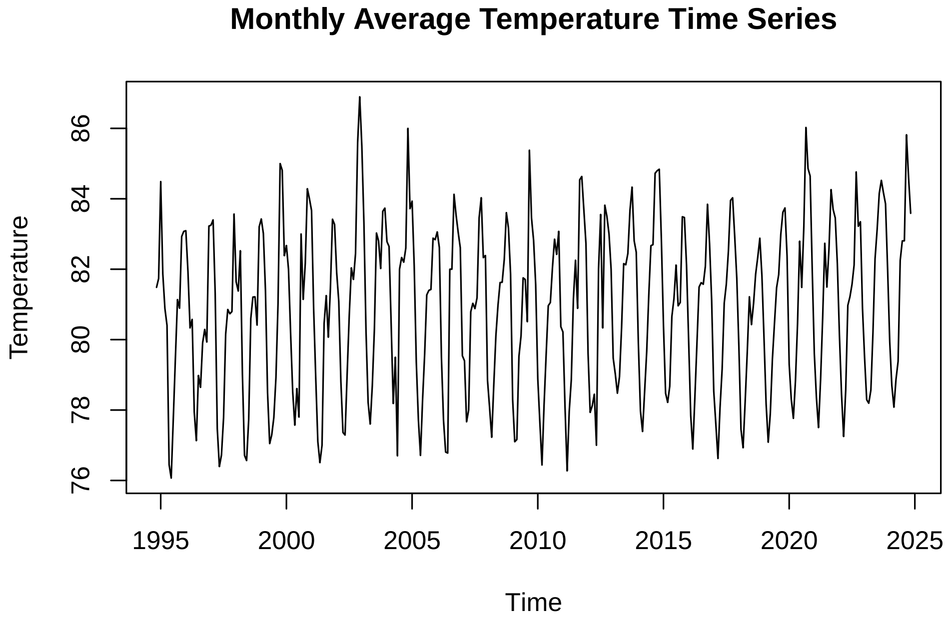
```
weather$Date <- as.Date(weather$Date)
weather$Avg_Temp <- as.numeric(gsub("", "", weather$Avg_Temp))
weather_month <- weather |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(Avg_Temp = mean(Avg_Temp, na.rm = TRUE)) |> ungroup()
```

```
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

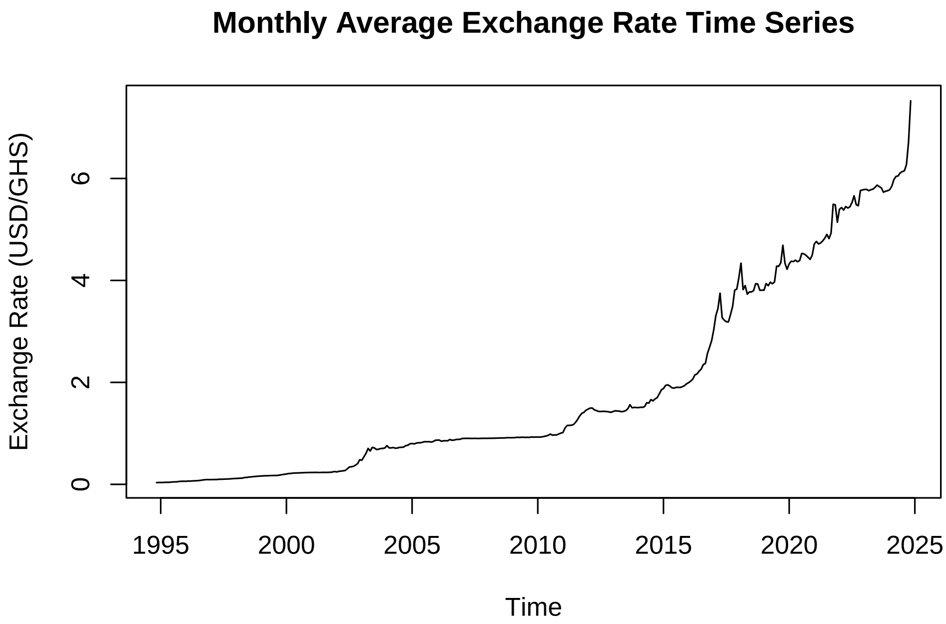
```r
exchangerate$Date <- as.Date(exchangerate$Date)
exchangerate$exchange_rate <- as.numeric(gsub("", "", exchangerate$exchange_rate))
rate_month <- exchangerate |> mutate(Time = floor_date(Date, "month")) |> group_by(Time) |>
  summarise(exchange_rate = mean(exchange_rate, na.rm = TRUE)) |> ungroup()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

```r
summary(exchangerate)
```

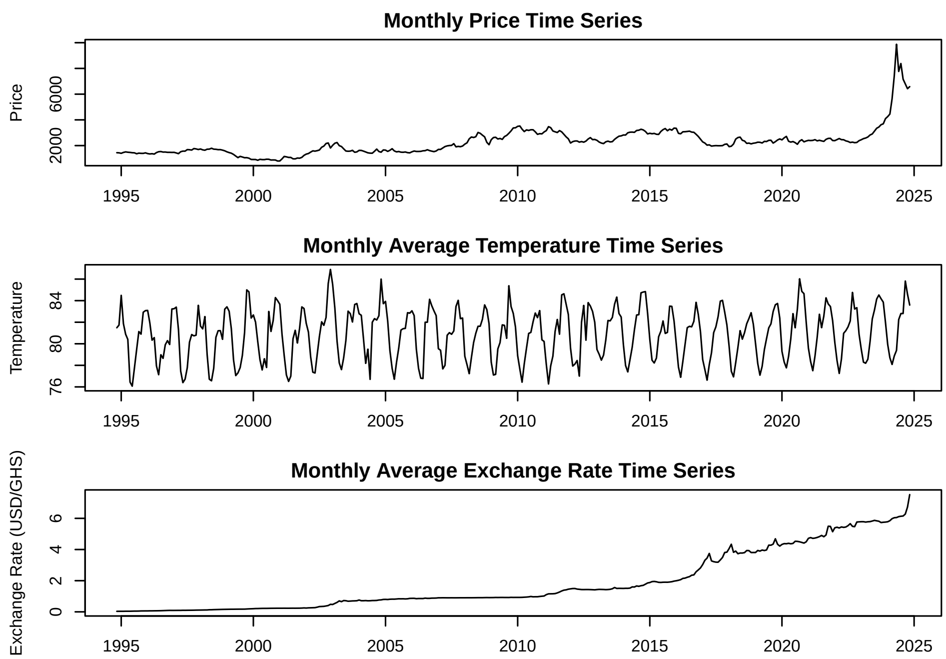
```
##       Date            exchange_rate
##  Min.   :1992-03-01   Min.   : 0.0338
##  1st Qu.:2000-06-01   1st Qu.: 0.5400
##  Median :2008-09-01   Median : 1.1595
##  Mean   :2008-08-31   Mean   : 2.8314
##  3rd Qu.:2016-12-01   3rd Qu.: 4.2805
##  Max.   :2025-03-01   Max.   :16.2500
```

```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=
```

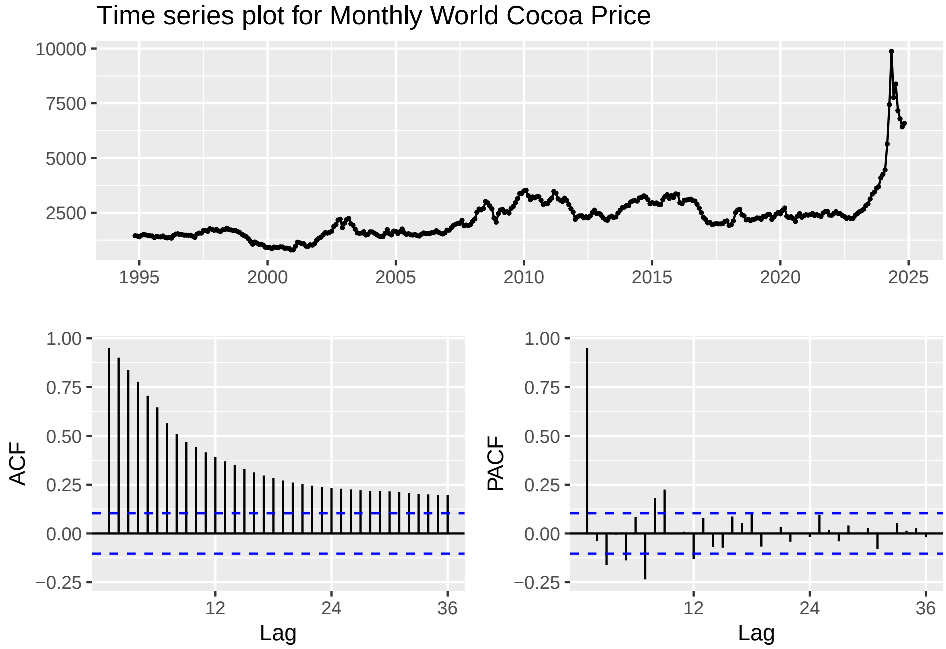



```r
par(mfrow=c(3,1), mar = c(3, 4, 2, 2))
# price
plot(price_ts, main="Monthly Price Time Series", ylab="Price", xlab="Time")
#temperature
ts.plot(weather_ts, main="Monthly Average Temperature Time Series", ylab="Temperature", xlab="Time")
# exchange rate
ts.plot(rate_ts, main="Monthly Average Exchange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

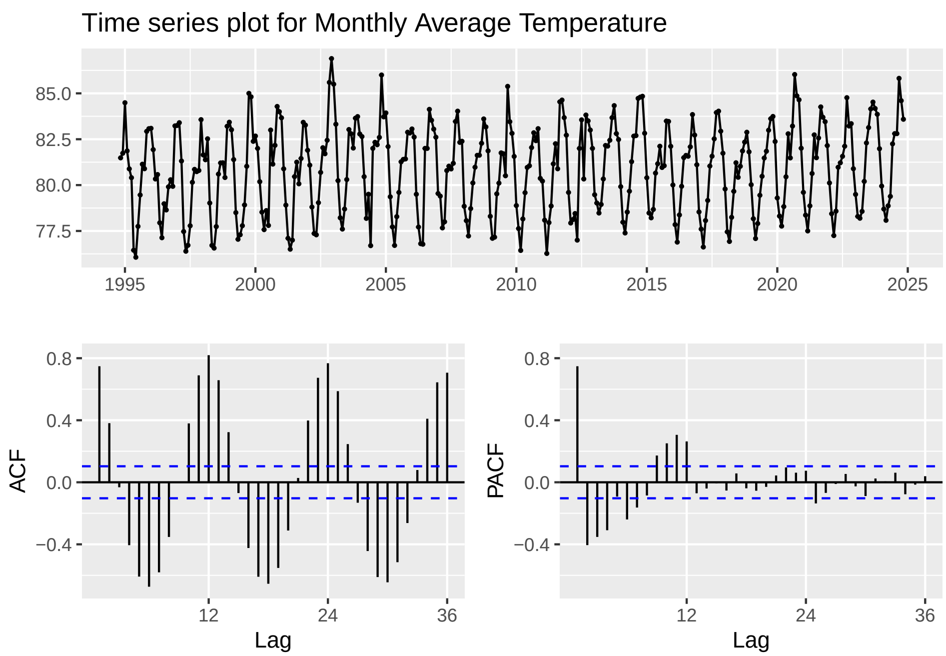


1.4 Time series plots for data

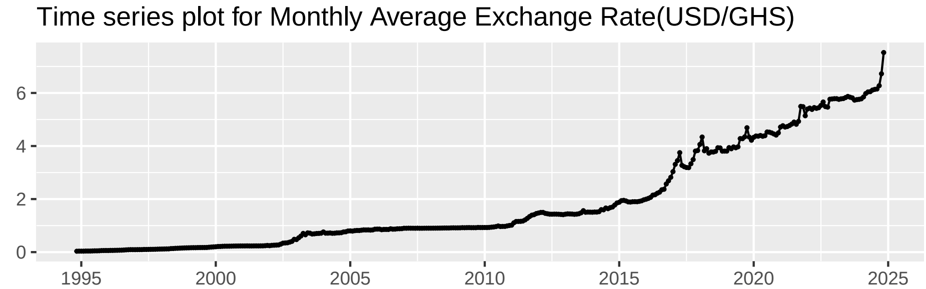
```
ggtsdisplay(price_ts, main="Time series plot for Monthly World Cocoa Price")
```

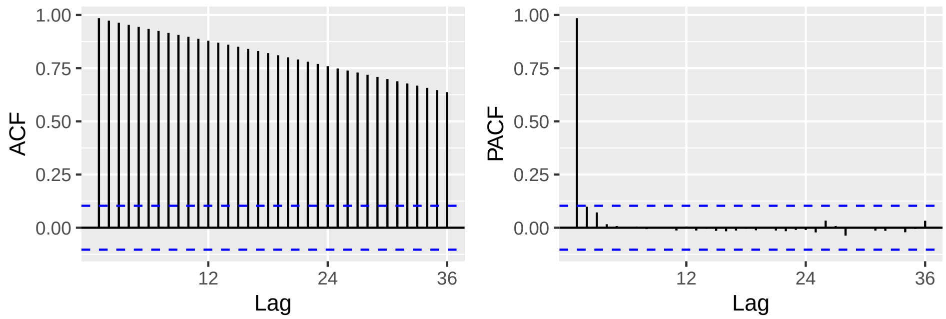



```
ggtsdisplay(weather_ts, main="Time series plot for Monthly Average Temperature")
```

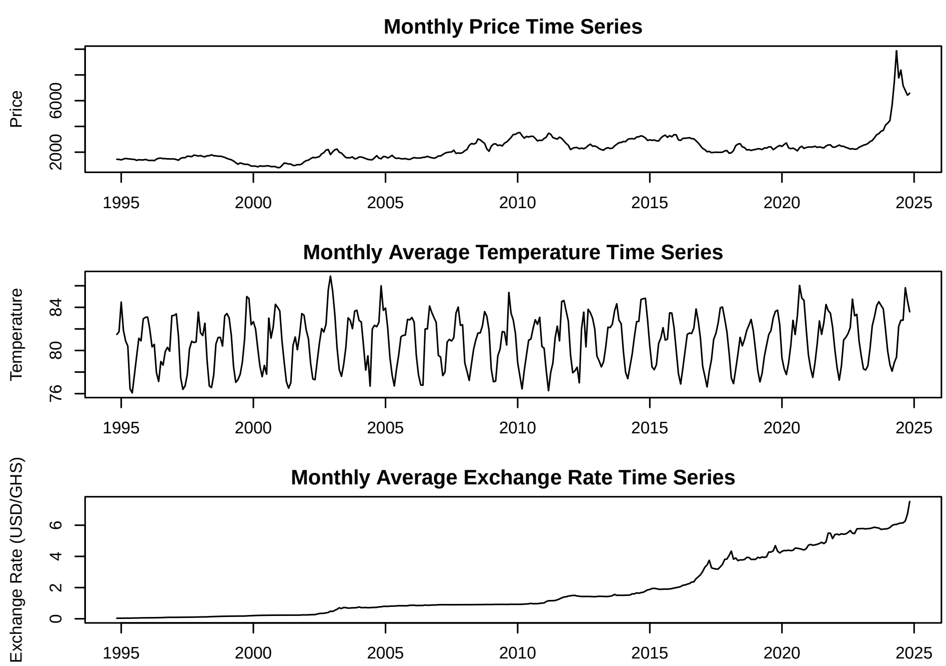



```
ggtsdisplay(rate_ts, main="Time series plot for Monthly Average Exchange Rate(USD/GHS)")
```






```
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 and Split data

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

```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
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)
```

1.6 Stationarity check and Decomposition

```r
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
```

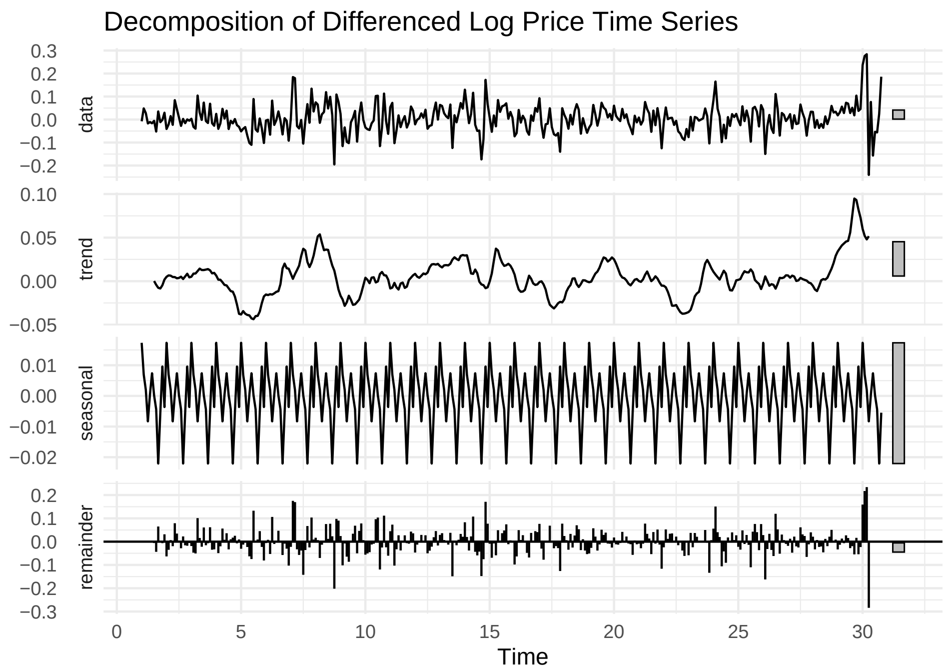
```r
adf.test(data$diff_log_price)
```

```
## 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()
```



2. Method

2.1 ETS Model

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

2.1.1 Fit Model

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

```
## ETS(A,N,N)
##
## Call:
## ets(y = data_train_ts)
##
##   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
```

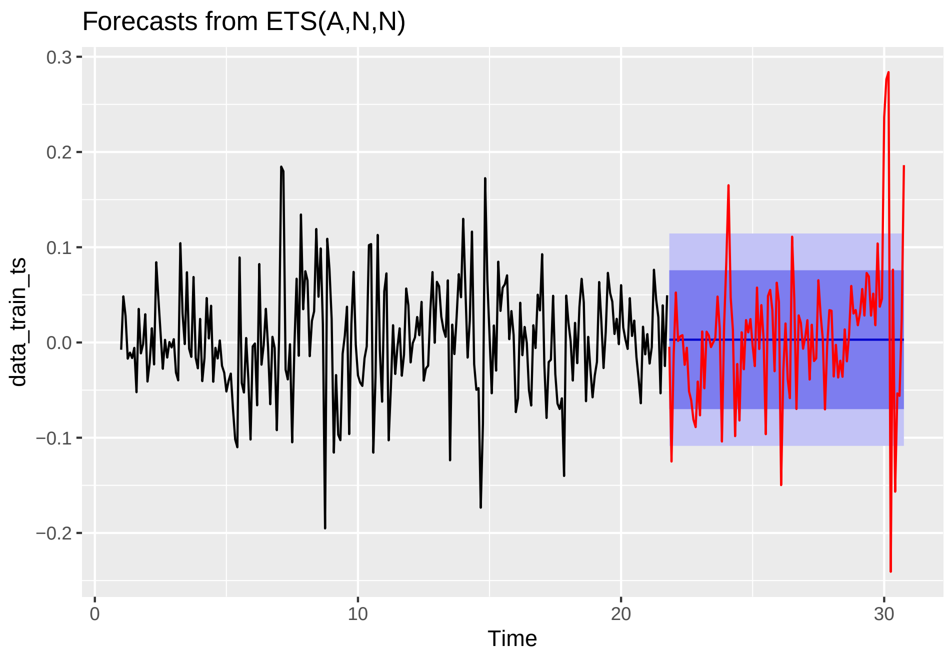
```r
summary(ets_zmodel)
```

```
## 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
```

ETS(A,N,N) is the best model.

2.1.2 Forecasting and Plotting

```r
# 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")
```

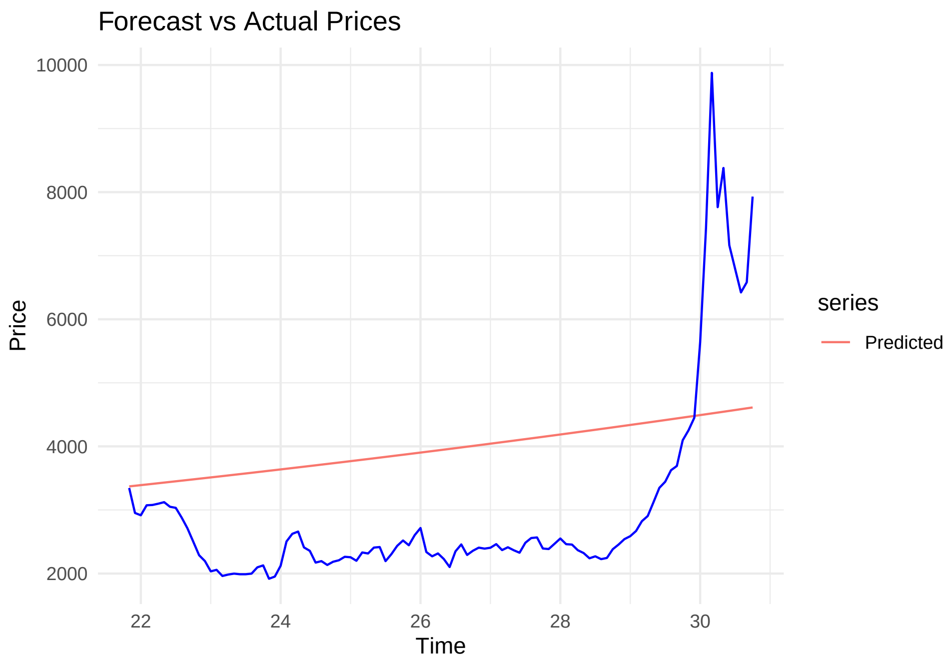


The red line is the observed actual values. The forecasted values are the central blue line within the blue shaded prediction intervals.

```
last_log_price <- tail(trainSet$log_price , 1)
forecasted_log_price <- cumsum(forecast_ets$mean) + last_log_price

# Convert back to actual price
forecasted_price <- exp(forecasted_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()
```



2.2 ARIMAX Model

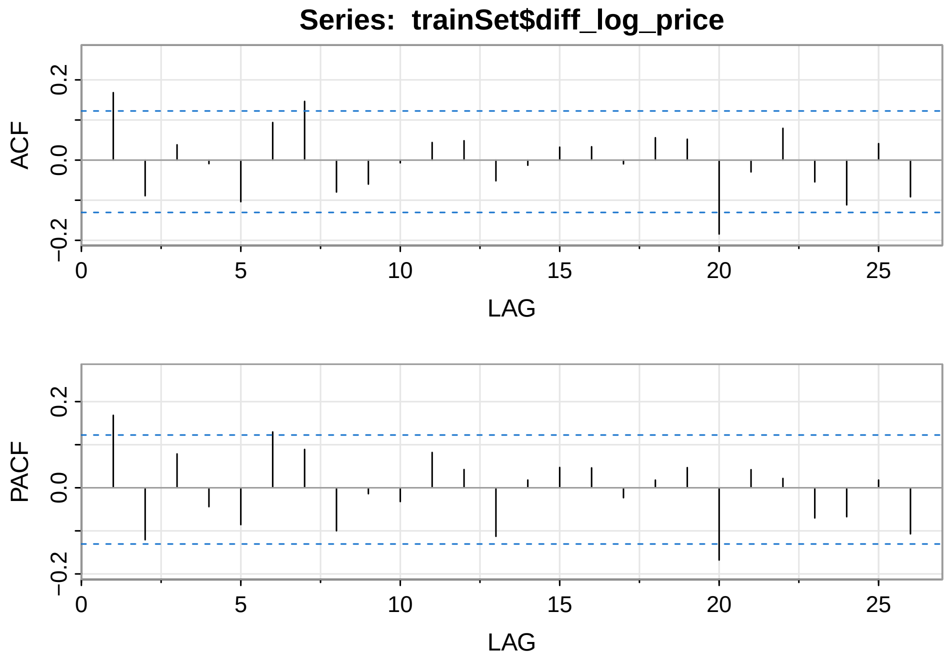
Recall that in Section 1.3.1, we have tested the acf and adf.test, and determined that we would be using the differenced price data. To fit the trainset, we evaluate p and q for ARIMA model.

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

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

```
##
##  Augmented Dickey-Fuller Test
##
## data:  trainSet$diff_log_price
## Dickey-Fuller = -5.015, Lag order = 6, p-value = 0.01
## 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
```

2.2.1 Fit ARIMAX Model

```r
xreg_matrix <- cbind(trainSet$Avg_Temp, trainSet$exchange_rate)

# Assign column names to xreg
colnames(xreg_matrix) <- c("Avg_Temp", "exchange_rate")

# Fit the SARIMA model with the named xreg
arimax_model <- Arima(trainSet$diff_log_price, order=c(1,0,1), xreg = xreg_matrix)

# Summary of the model
summary(arimax_model)
```

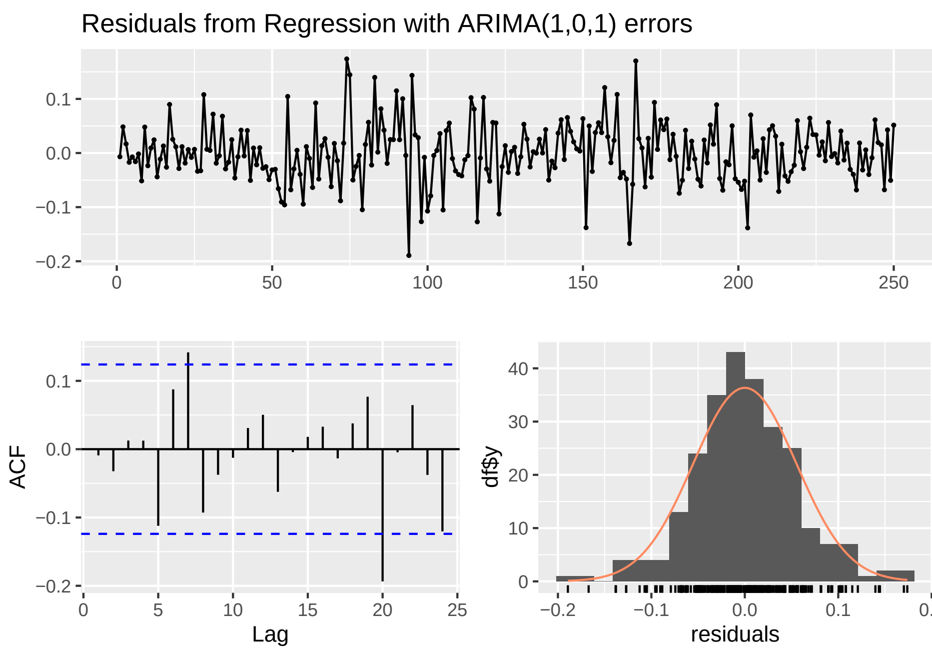
```
## Series: trainSet$diff_log_price
## Regression with ARIMA(1,0,1) errors
##
## Coefficients:
##           ar1     ma1  intercept  Avg_Temp  exchange_rate
##       -0.3267  0.5331    -0.0325    0.0004         0.0035
## s.e.   0.1996  0.1763     0.1384    0.0017         0.0044
##
## sigma^2 = 0.003118:  log likelihood = 369.07
## AIC=-726.14   AICc=-725.79   BIC=-705.01
```

```
##
## Training set error measures:
##                        ME       RMSE        MAE      MPE     MAPE      MASE
## Training set 1.154543e-05 0.05527999 0.04194407 130.5909 158.5406 0.7578074
##                      ACF1
## Training set -0.009038859
```

```r
AIC(arimax_model)
```

```
## [1] -726.1383
```

```r
checkresiduals(arimax_model)
```



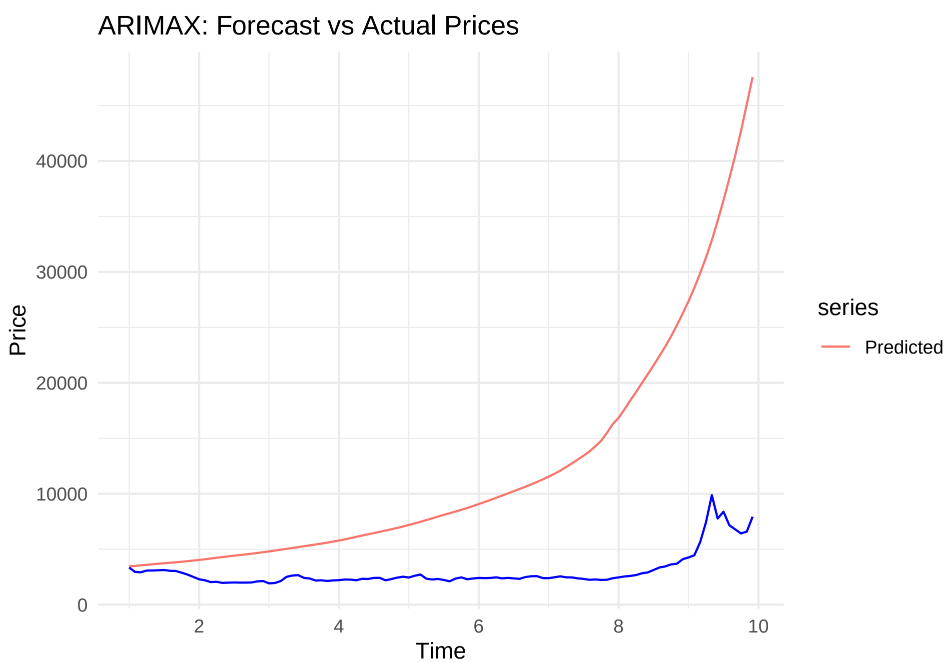

```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,0,1) errors
## Q* = 13.437, df = 8, p-value = 0.09766
##
## Model df: 2.   Total lags used: 10
```

Fail to reject H_0, hence residuals of this plot do not show significant autocorrelation.

2.2.2 Forecasting With ARIMAX Model

```r
# Create future xreg from test set
forecast_arimax_xreg <- cbind(testSet$Avg_Temp, testSet$exchange_rate)

# Ensure column names match the original xreg
colnames(forecast_arimax_xreg) <- c("Avg_Temp", "exchange_rate")

# Forecast using the ARIMAX model with xreg
forecast_arimax <- forecast(arimax_model, xreg = forecast_arimax_xreg)

# Assuming your model was trained on log_price, use the last log_price from the training set
```

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

# Convert the forecasted log prices back to actual prices
forecasted_price_arimax <- exp(cumsum(forecast_arimax$mean) + last_log_price)

# Get actual prices from the test set
actual_price_arimax <- exp(cumsum(testSet$diff_log_price) + last_log_price)

# Create time series objects with the correct start point and frequency
forecast_arimax_ts <- ts(forecasted_price_arimax, start = start(testSet$Time), frequency = 12)
actual_arimax_ts <- ts(actual_price_arimax, start = start(testSet$Time), frequency = 12)

# Plot forecast vs actual prices
autoplot(forecast_arimax_ts, series = "Predicted") +
  autolayer(actual_arimax_ts, series = "Actual", color = "blue") +
  ggtitle("ARIMAX: Forecast vs Actual Prices") +
  ylab("Price") +
  xlab("Time") +
  theme_minimal()
```



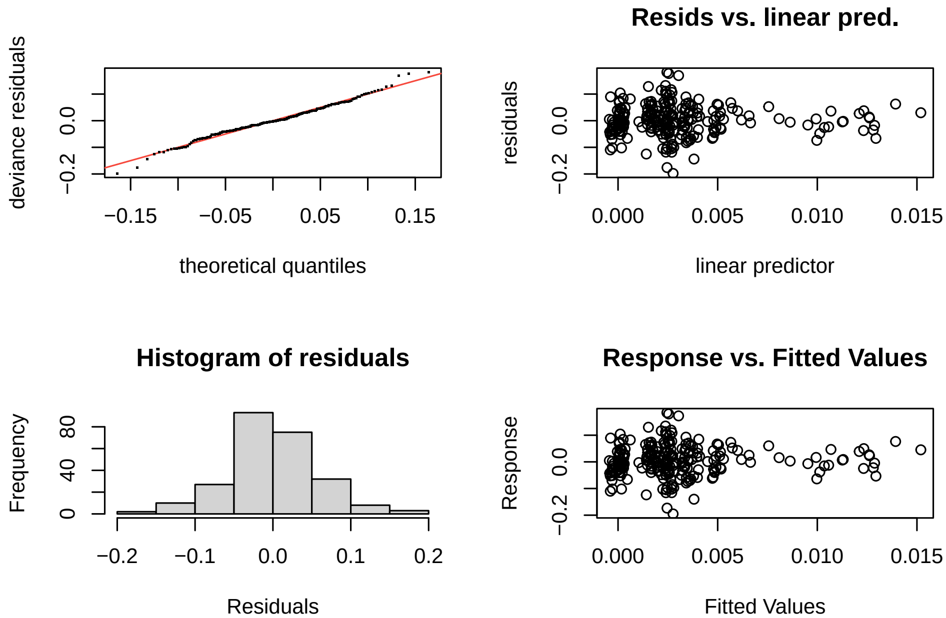
2.5 GAM Model

2.5.1 Fit Model

```r
# Uses simple smoothing splines for variables
trainSet$Time_num <- as.numeric(trainSet$Time)

gam_basic <- gam(diff_log_price ~ s(Time_num) + s(Avg_Temp) + s(exchange_rate),
                 data = trainSet, method = "REML")
```

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

2.5.1.1 Basic Model




```
##
## Method: REML   Optimizer: outer newton
## full convergence after 10 iterations.
## Gradient range [-0.0001493842,0.0002242223]
## (score -345.2659 & scale 0.003250926).
## Hessian positive definite, eigenvalue range [1.158838e-05,122.9998].
## Model rank =  28 / 28
##
## 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(Time_num)      9.00 1.00    0.84  <2e-16 ***
## s(Avg_Temp)      9.00 1.00    1.06    0.81
## s(exchange_rate) 9.00 1.14    0.90    0.02 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Fit GAM model with smooth trend, random effects, and seasonality
trainSet$dateInt = as.integer(trainSet$Time)
trainSet$monthFac <- factor(format(trainSet$Time, "%m"))
gam_complex <- gam(diff_log_price ~ s(dateInt, k = 100) + s(monthFac, bs = "re") +
                     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", optimizer = "efs")
```

```r
summary(gam_basic)
```

2.5.1.2 Complex Model

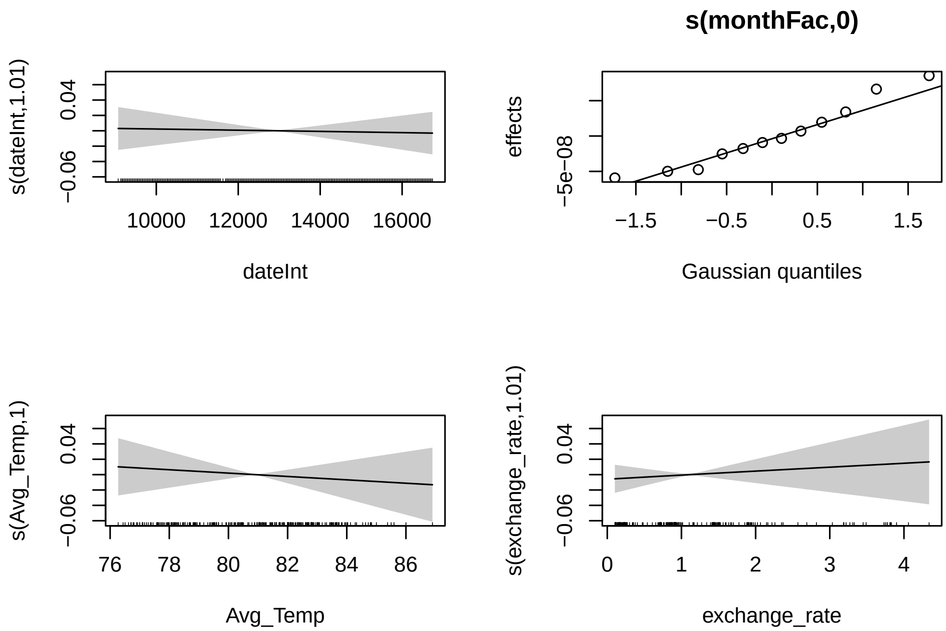
```
##
```

```
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(Time_num) + s(Avg_Temp) + s(exchange_rate)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.002929   0.003606   0.812    0.417
##
## Approximate significance of smooth terms:
##                    edf Ref.df     F p-value
## s(Time_num)      1.000  1.000 0.074   0.786
## s(Avg_Temp)      1.001  1.001 0.000   0.996
## s(exchange_rate) 1.142  1.276 0.266   0.695
##
## R-sq.(adj) =  -0.00863   Deviance explained = 0.41%
## -REML = -345.27  Scale est. = 0.0032509  n = 250
```

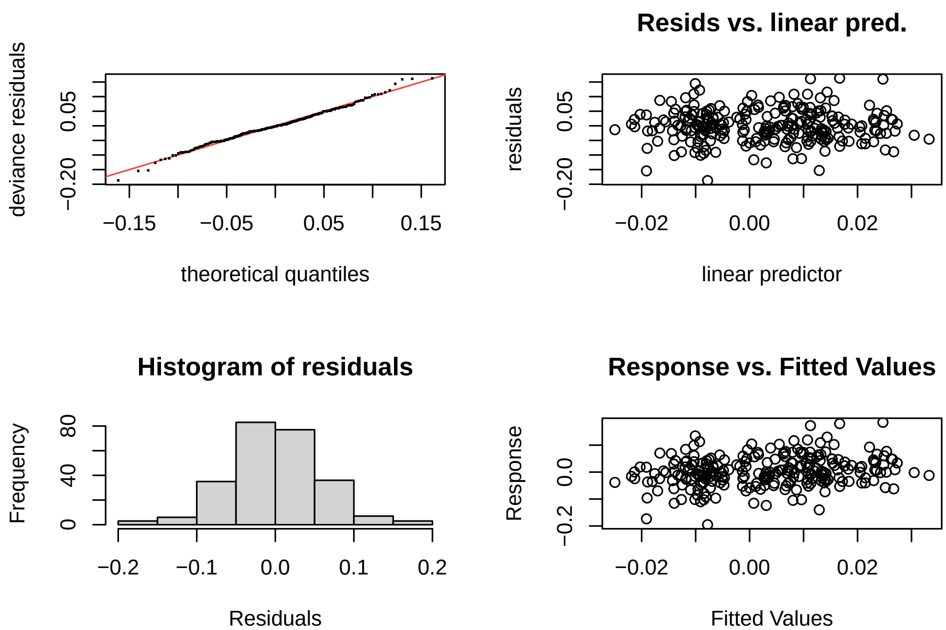
```r
summary(gam_complex)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## diff_log_price ~ s(dateInt, k = 100) + s(monthFac, bs = "re") +
##     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)             0.0029355  0.0035494   0.827   0.4090
## sinpi(dateInt/182.625)  0.0081148  0.0085424   0.950   0.3431
## cospi(dateInt/182.625)  0.0094069  0.0106187   0.886   0.3766
## sinpi(dateInt/91.3125) -0.0000812  0.0050924  -0.016   0.9873
## cospi(dateInt/91.3125)  0.0144616  0.0060586   2.387   0.0178 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                        edf Ref.df     F p-value
## s(dateInt)       1.006e+00  1.013 0.048   0.842
## s(monthFac)      4.392e-05 11.000 0.000   0.809
## s(Avg_Temp)      1.000e+00  1.000 0.297   0.586
## s(exchange_rate) 1.013e+00  1.027 0.351   0.561
##
## R-sq.(adj) =  0.0234   Deviance explained = 5.09%
## -REML = -334.76  Scale est. = 0.0031478  n = 250
```

```r
plot(gam_complex, pages = 1, shade = TRUE)
```




```r
gam.check(gam_complex)
```




```
##
## Method: REML   Optimizer: efs
## $iter
## [1] 31
##
## $score.hist
##  [1] -332.7868 -333.5308 -334.0373 -334.3219 -334.4896 -334.5939 -334.6572
##  [8] -334.6941 -334.7155 -334.7283 -334.7364 -334.7418 -334.7455 -334.7481
## [15] -334.7501 -334.7515 -334.7526 -334.7535 -334.7542 -334.7548 -334.7553
## [22] -334.7556 -334.7559 -334.7562 -334.7564 -334.7566 -334.7568 -334.7569
## [29] -334.7570 -334.7571 -334.7572
##
```

```
## $conv
## [1] "full convergence"
##
## 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(dateInt)       9.90e+01 1.01e+00    0.86    0.02 *
## s(monthFac)      1.20e+01 4.39e-05      NA      NA
## s(Avg_Temp)      9.00e+00 1.00e+00    1.06    0.80
## s(exchange_rate) 9.00e+00 1.01e+00    0.89    0.03 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```