```r
# Load required libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(ggplot2)
library(astsa)
require(lubridate)
require(zoo)
```

```
## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.4.2

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
require(ggplot2)
require(knitr)
```

```
## Loading required package: knitr
```

```r
require(dplyr)
require(MASS)
```

```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
require(glmnet)
```

```
## Loading required package: glmnet

## Warning: package 'glmnet' was built under R version 4.4.2
```

```
## Loading required package: Matrix
## Loaded glmnet 4.1-8
```

```r
require(L1pack)
```

```
## Loading required package: L1pack
## Warning: package 'L1pack' was built under R version 4.4.1
## Loading required package: fastmatrix
## Warning: package 'fastmatrix' was built under R version 4.4.1
##
## Attaching package: 'fastmatrix'
## The following object is masked from 'package:Matrix':
##
##     lu
```

```r
require(tidyr)
```

```
## Loading required package: tidyr
##
## Attaching package: 'tidyr'
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```r
require(car)
```

```
## Loading required package: car
## Warning: package 'car' was built under R version 4.4.1
## Loading required package: carData
## Warning: package 'carData' was built under R version 4.4.1
##
## Attaching package: 'car'
## The following object is masked from 'package:dplyr':
##
##     recode
```

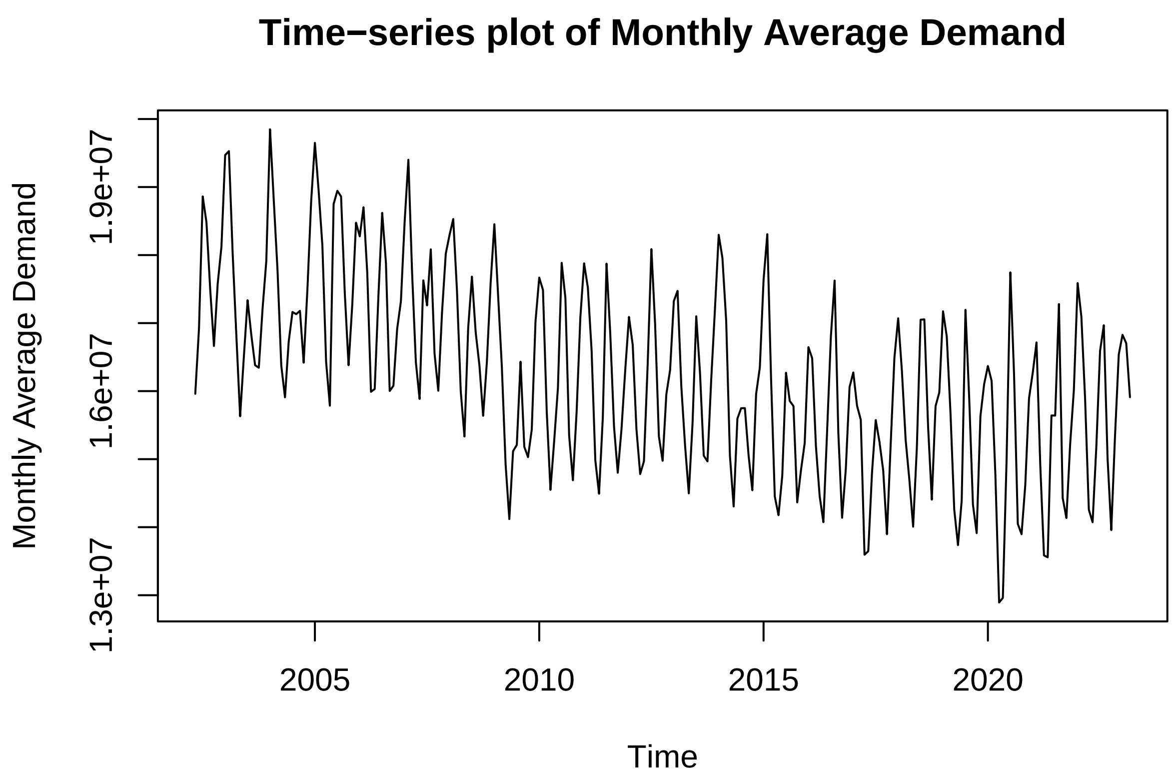
```r
# Read the data (replace 'Data_Group12.csv' with the correct file path)
data <- read.csv('Data_Group12.csv')

# Convert 'date' to Date format
data <- data %>%
  mutate(
    date = as.Date(date, format = "%Y-%m-%d"),
    year = year(date),
    month = month(date)
  ) %>%
  filter(!(year == 2023 & month == 4))

# Calculate average hourly demand per month
monthly_avg_demand <- data %>%
```

```r
  group_by(year, month) %>%
  summarise(avg_demand = mean(hourly_demand), avg_price = mean(hourly_average_price)) %>%
  ungroup() %>%
  mutate(month_year = as.yearmon(paste(year, month, sep = "-")),
         month=as.factor(month),
         Set = ifelse(row_number() > n() - 12, "Test", "Train"))
```

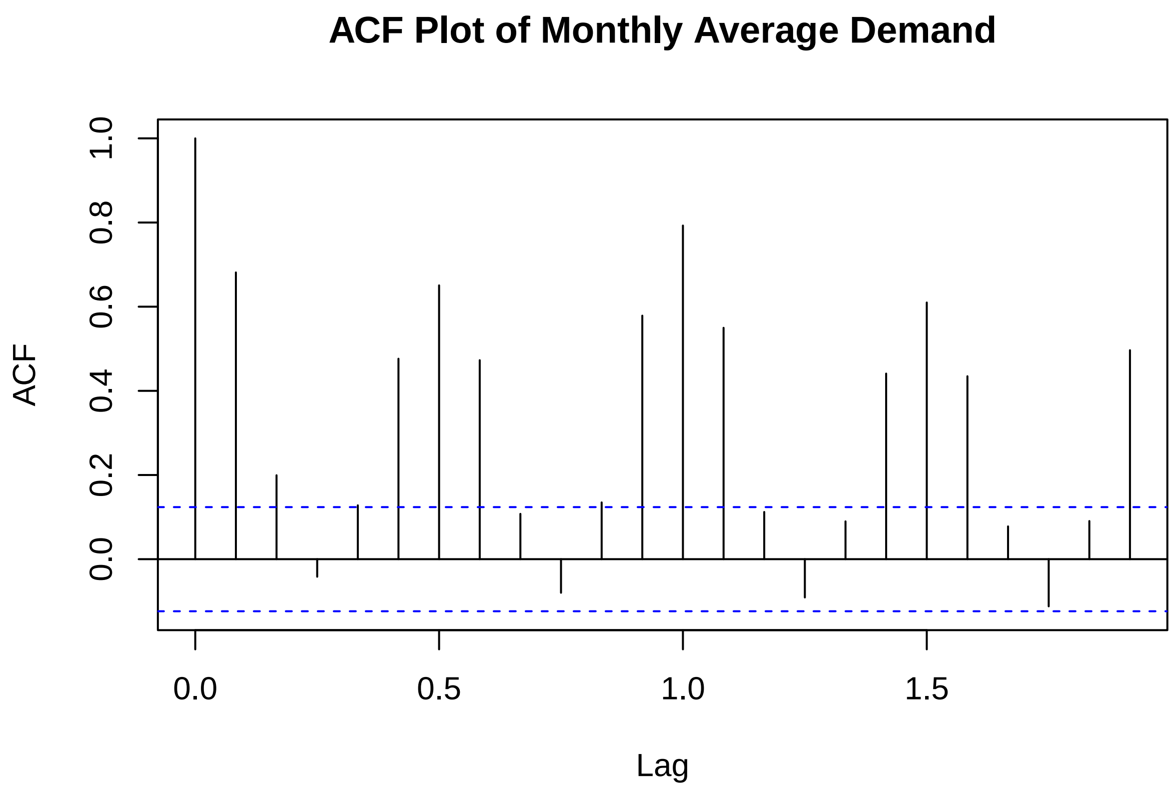
```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```r
# Create a Time Series object
monthly_avg_demandTS <- ts(monthly_avg_demand$avg_demand,
               start = c(2002, 5), frequency = 12)
```

```r
# Plot the time series of monthly average demand
plot(monthly_avg_demandTS, main = "Time-series plot of Monthly Average Demand",
     ylab = "Monthly Average Demand", xlab = "Time")
```



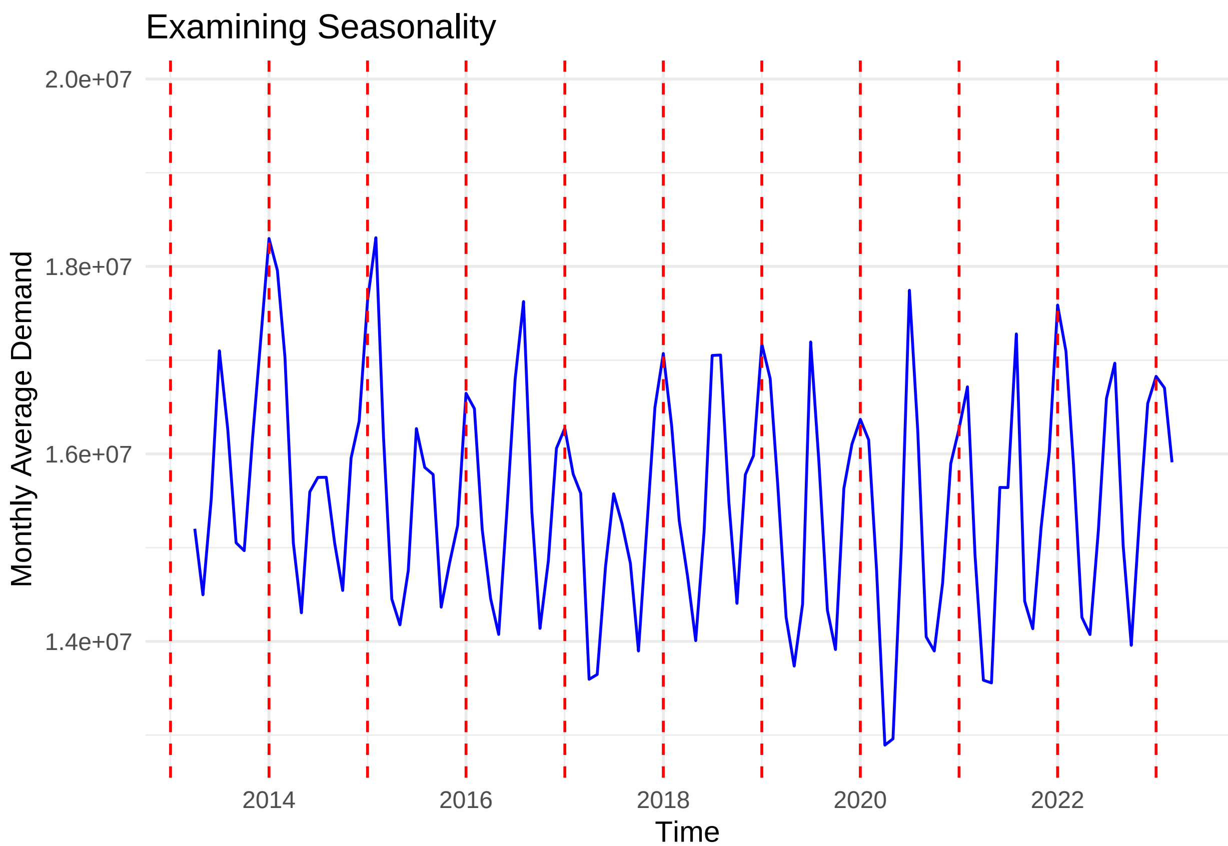

```r
acf(monthly_avg_demandTS, main = "ACF Plot of Monthly Average Demand")
```




```r
# Define the range of years to zoom in
start_date <- as.Date("2013-04-01") # Start of zoom
end_date <- as.Date("2023-3-31") # End of zoom
# Load required libraries
library(ggplot2)
# Create a data frame from the time series for ggplot
data_df <- data.frame(
Date = seq(as.Date("2002-05-01"),
by = "month",
length.out = length(monthly_avg_demandTS)),
Demand = as.numeric(monthly_avg_demandTS)
)
# Define yearly cycle boundaries
cycle_boundaries <- data.frame(
  Date = seq(as.Date("2003-01-01"),
             by = "1 year",
             length.out = length(seq(2003, 2023))),
  Label = paste("Cycle", seq(2003, 2023))
)

# Plot the time series with 1-year cycles
ggplot(data_df, aes(x = Date, y = Demand)) +
  geom_line(color = "blue") +
  geom_vline(data = cycle_boundaries, aes(xintercept = as.numeric(Date)),
             linetype = "dashed", color = "red") +
  labs(title = "Examining Seasonality", x = "Time",
```

```r
    y = "Monthly Average Demand") + xlim(start_date, end_date) +
  theme_minimal()
```

```
## Warning: Removed 131 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



1. We do not see an obvious trend (Could be small linear one).
2. There is seasonality which is evident from the peridiocity in the time series and the ACF plot. The period, d = 6 or 12 based on our observation from the ACF plot.
3. Variance seems to be constant.

Hence, we conclude that the series is **not** stationary due to the presence of seasonality, since it indicates non-stationarity due to non-constant mean.

```r
# Perform the Fligner-Killeen test to check homogeneity of variances
indx <- factor(rep(1:12, each = 21, length.out = 251))
fligner.test(monthly_avg_demandTS, indx)
```

```
##
##  Fligner-Killeen test of homogeneity of variances
##
## data:  monthly_avg_demandTS and indx
## Fligner-Killeen:med chi-squared = 3.436, df = 11, p-value = 0.9836
```

The p-value is greater than 0.05, so we have no evidence against the homogeneity of variances. We can conclude that we have constant variance, so there is no need to stabilize it.

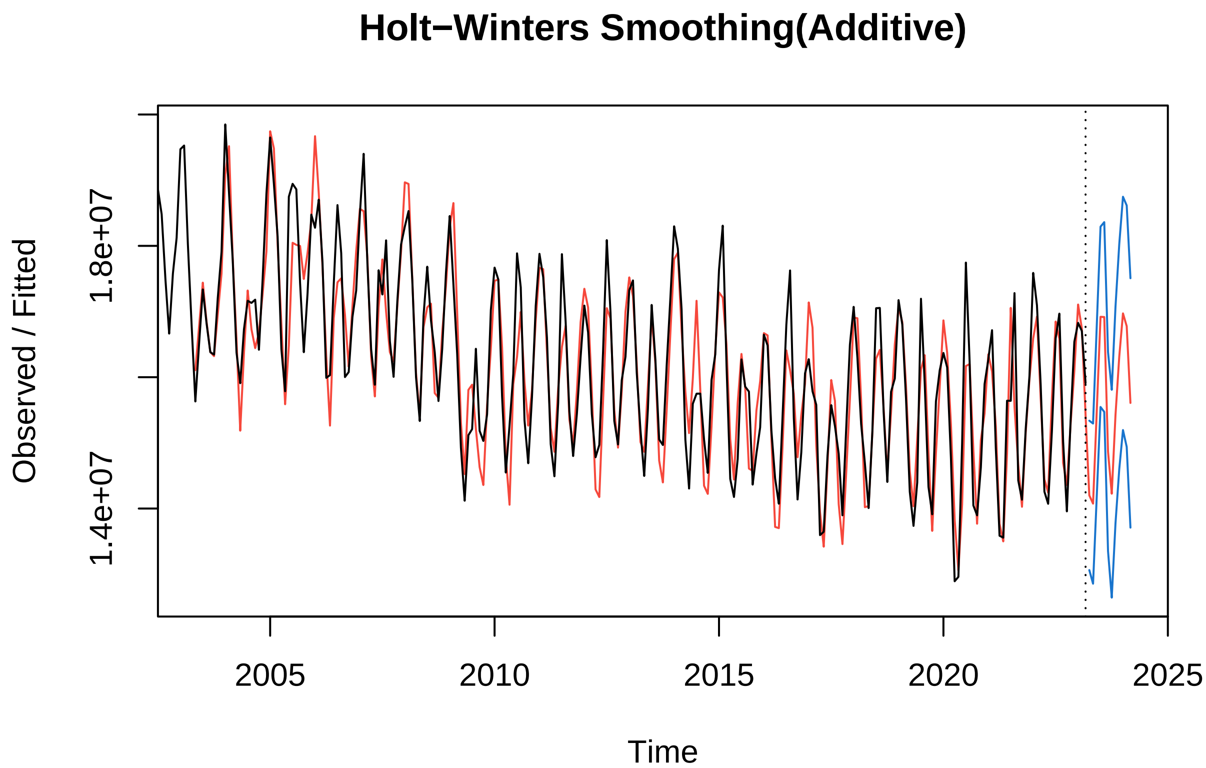
```r
# Train-test split
train <- window(monthly_avg_demandTS, end = c(2022, 3))
test <- window(monthly_avg_demandTS, start = c(2022, 4))

calculate_apse <- function(fitted_model, test) {
  test_predict <- predict(fitted_model, n.ahead = length(test))
  apse <- mean((test - test_predict)^2)
  return(apse)
}

es <- HoltWinters(train, gamma = FALSE , beta = FALSE)
double.es <- HoltWinters(train, gamma = F)
hw.additive <- HoltWinters(train, seasonal = "additive")
hw.multiplicative <- HoltWinters(train, seasonal = "multiplicative")

es_apse <- calculate_apse(es, test)
double.es_apse <- calculate_apse(double.es, test)
hw.additive_apse <- calculate_apse(hw.additive, test)
hw.multiplicative_apse <- calculate_apse(hw.multiplicative, test)

model_apse_results <- data.frame(Model = c("Exponential Smoothing",
                                           "Double Exponential Smoothing",
                                           "Additive Holt-Winters",
                                           "Multiplicative Holt-Winters"),
                        APSE = c(es_apse, double.es_apse, hw.additive_apse,
                                 hw.multiplicative_apse))
print(model_apse_results)
```

```
##                          Model         APSE
## 1        Exponential Smoothing 1.236721e+12
## 2 Double Exponential Smoothing 1.244853e+12
## 3        Additive Holt-Winters 1.396673e+11
## 4  Multiplicative Holt-Winters 1.604688e+11
```

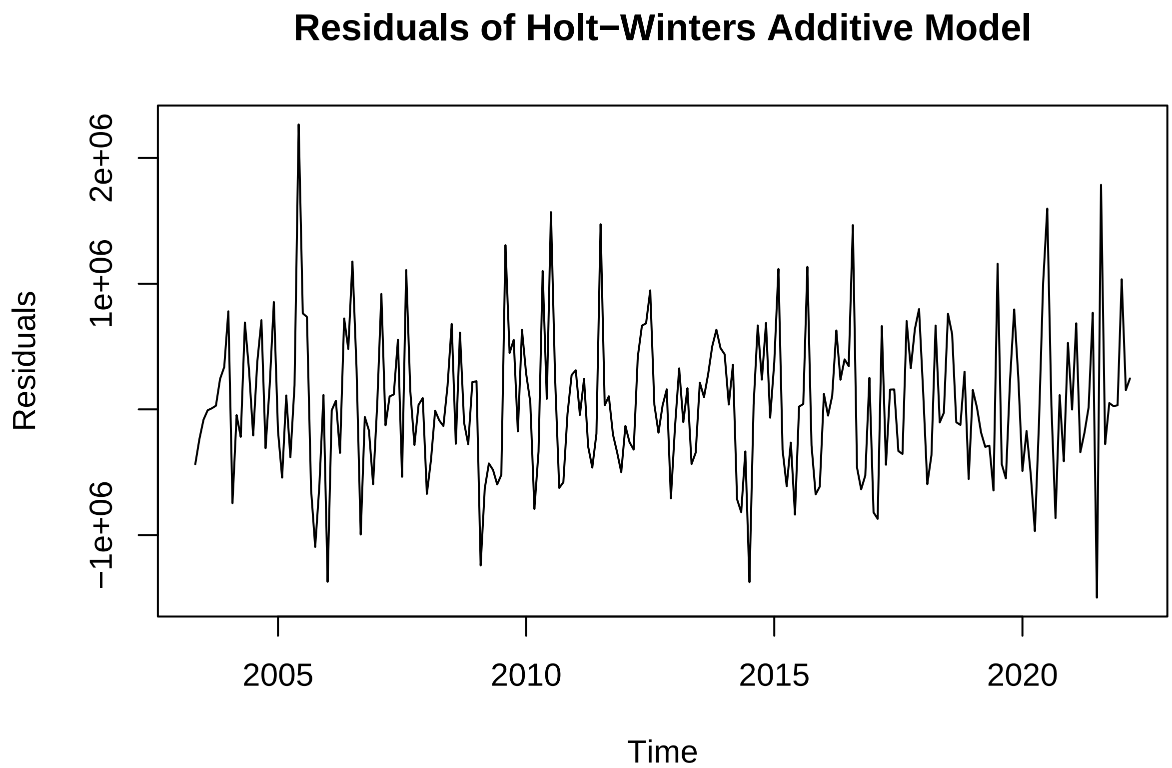
```r
best_model <- model_apse_results[which.min(model_apse_results$APSE), ]
cat("The best model based on APSE is:", best_model$Model, "with APSE:",
    best_model$APSE)
```

```
## The best model based on APSE is: Additive Holt-Winters with APSE: 139667294919
```

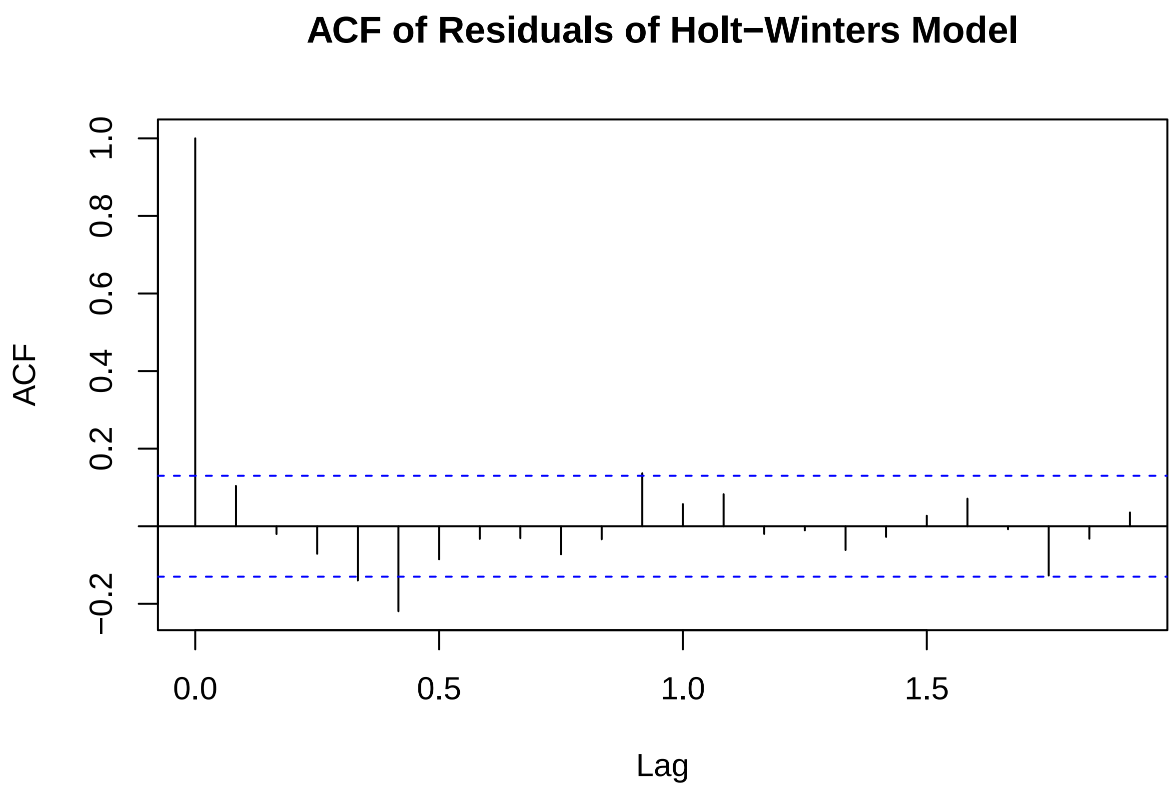
```r
# Best Smoothing model
hw.additive_full = HoltWinters(monthly_avg_demandTS, seasonal ="additive")
pred.additive = predict(hw.additive_full, n.ahead=12,
                        prediction.interval = TRUE)
plot(hw.additive_full, pred.additive,
     main="Holt-Winters Smoothing(Additive)")
```




```
#getting residuals of hw
hw_residuals <- residuals(hw.multiplicative)
plot(hw_residuals, main = "Residuals of Holt-Winters Additive Model",
     ylab = "Residuals", xlab = "Time")
```



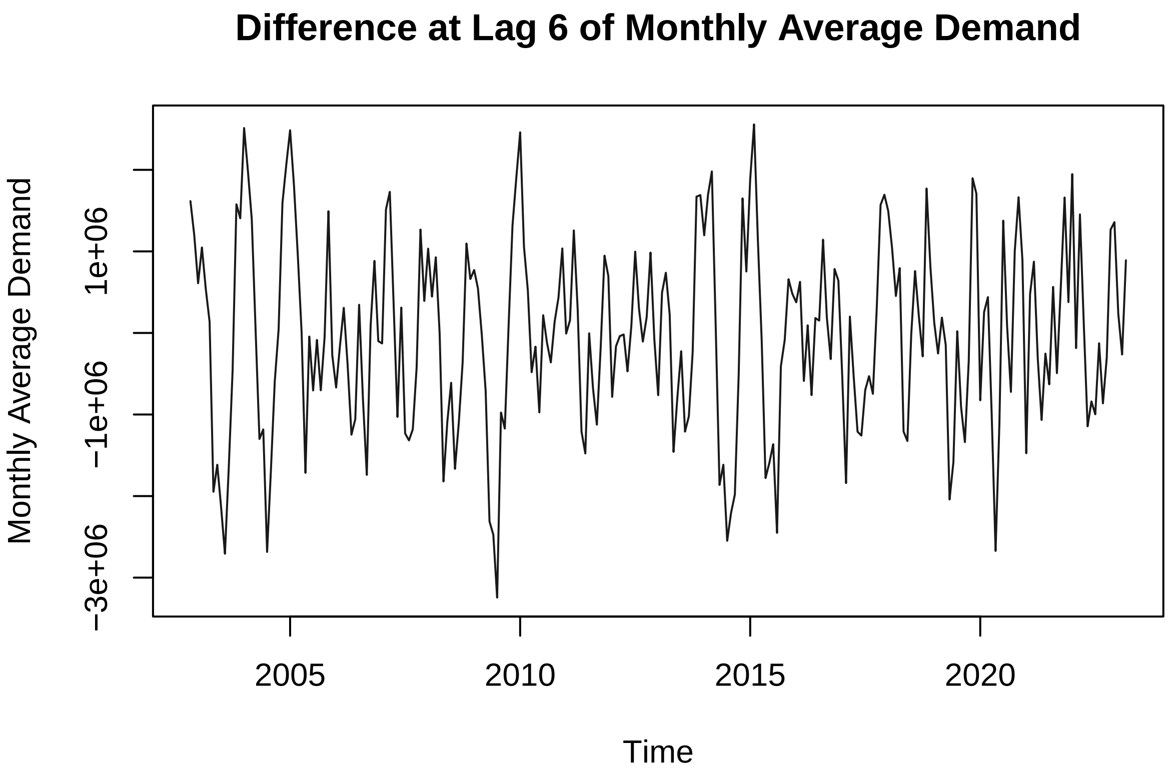

```r
acf(hw_residuals, main = "ACF of Residuals of Holt-Winters Model")
```



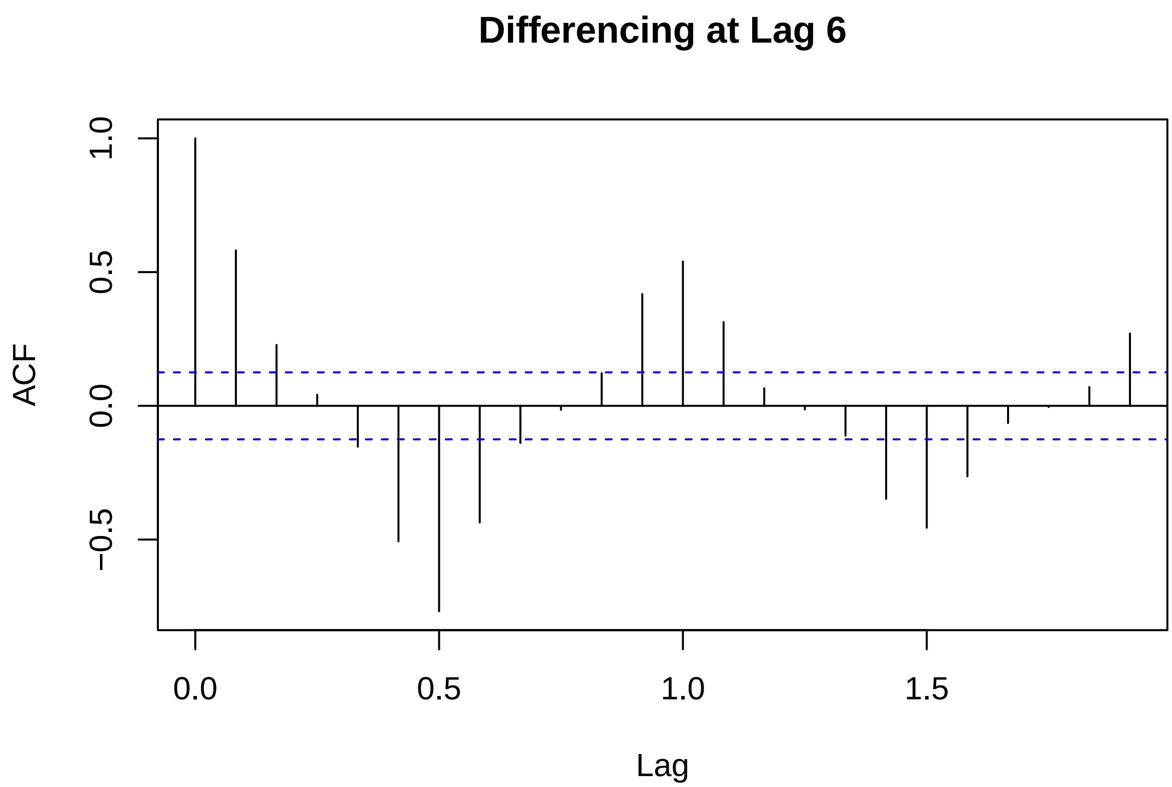

```r
hw_SSE <- c(es$SSE,double.es$SSE,hw.additive$SSE,hw.multiplicative$SSE)
which.min(hw_SSE)
```

```
## [1] 4
```

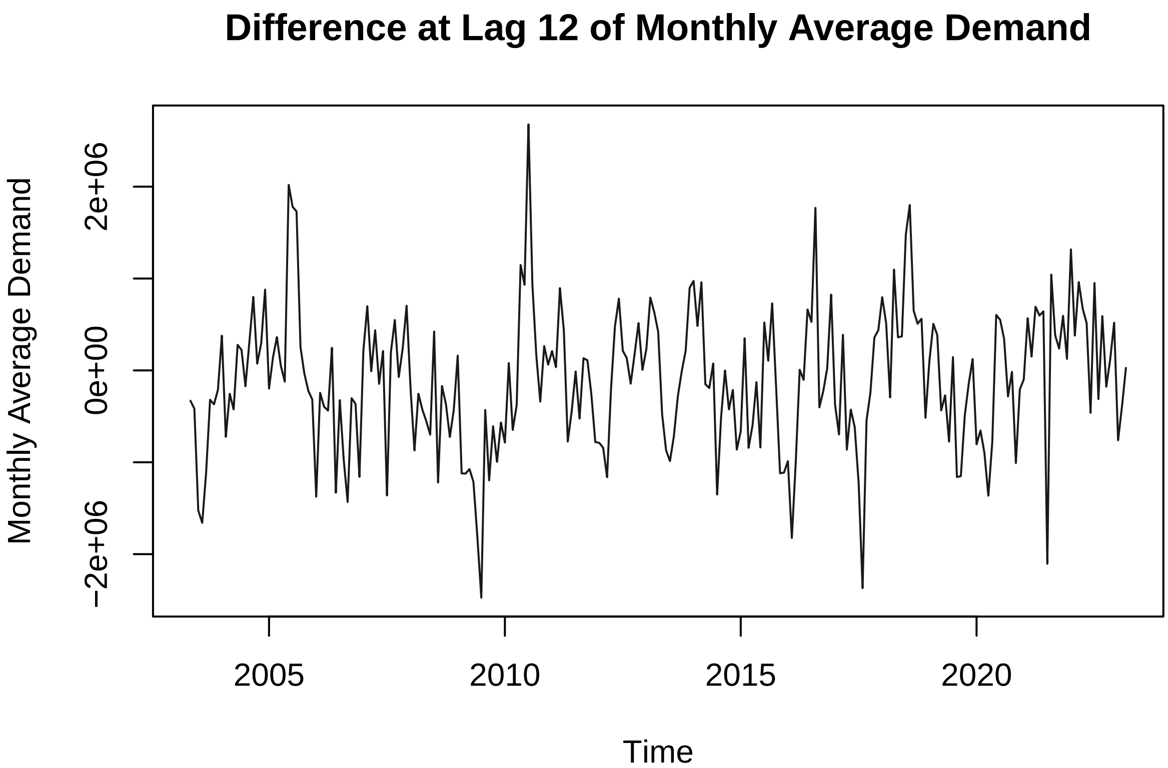
```r
# one time differencing in lag 6
Diff.data_lag6 = diff(monthly_avg_demandTS, lag=6)
plot(Diff.data_lag6, main="Difference at Lag 6 of Monthly Average Demand",
     ylab = "Monthly Average Demand", xlab = "Time",
     pch=16  ,col=adjustcolor("black" , 0.9))
```



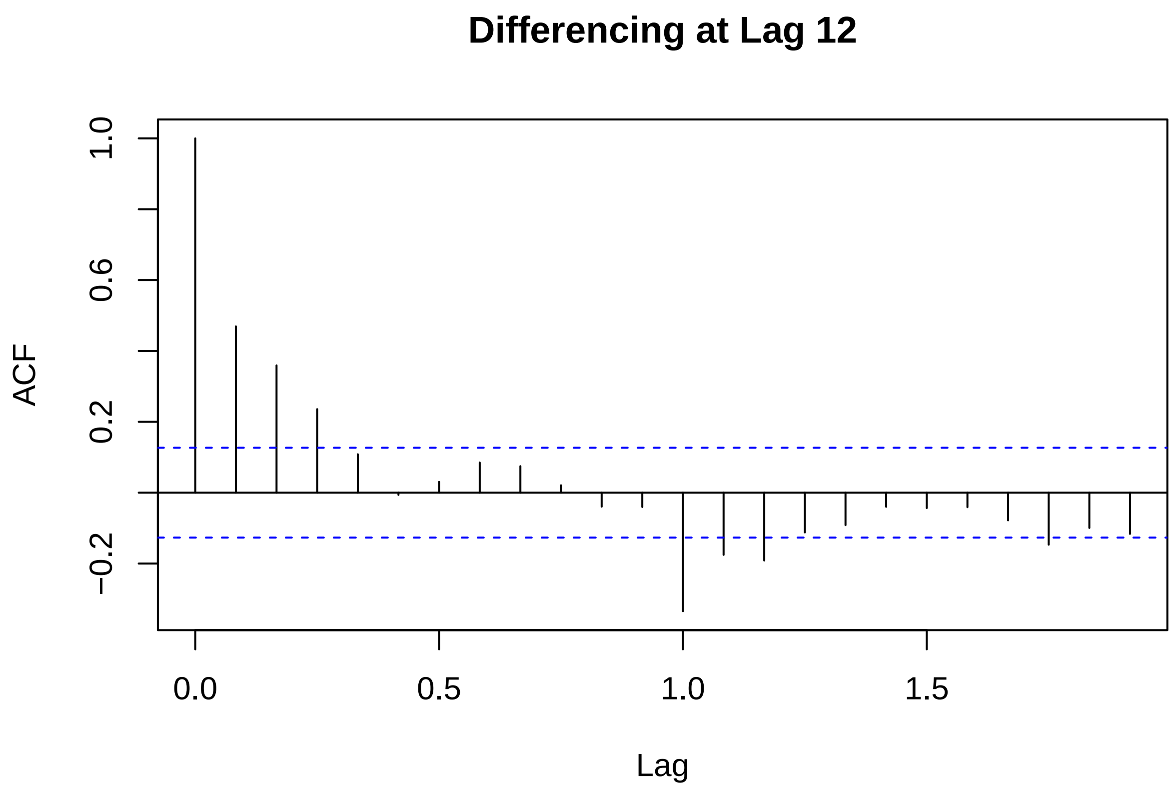

```r
acf(Diff.data_lag6, main="Differencing at Lag 6")
```



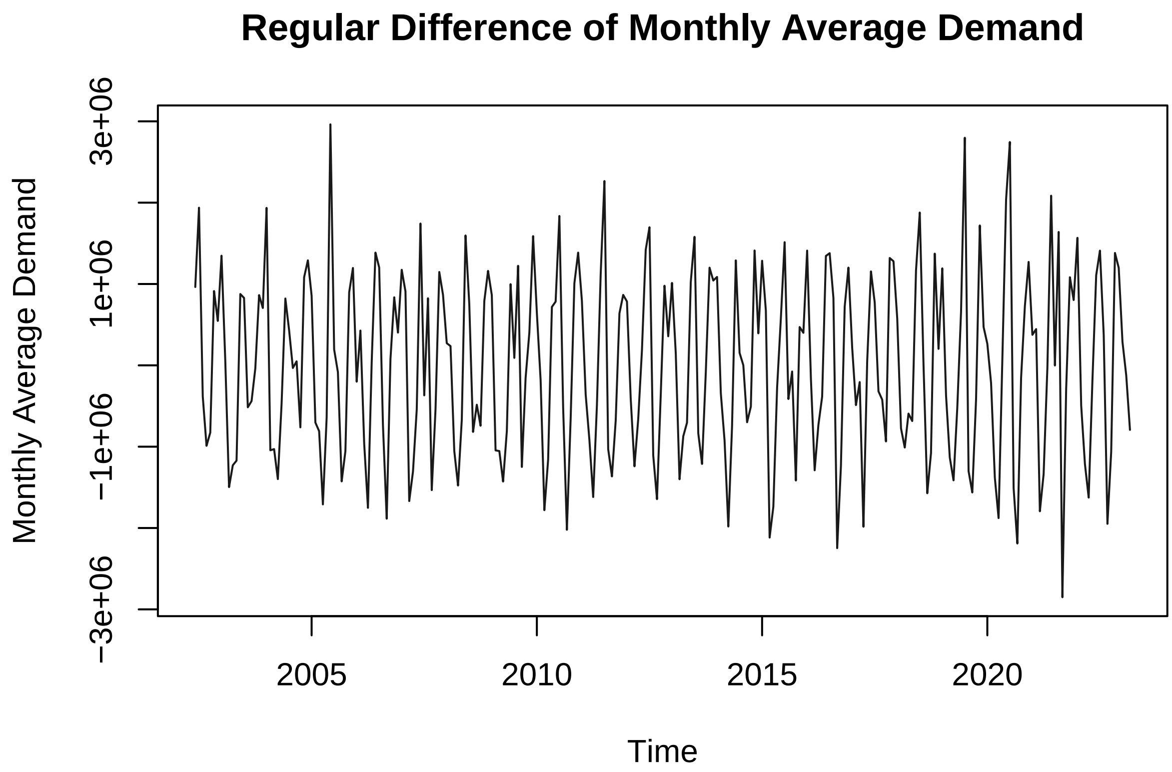

```
# one time differencing in lag 12
Diff.data_lag12 = diff(monthly_avg_demandTS, lag=12)
plot(Diff.data_lag12,
     main="Difference at Lag 12 of Monthly Average Demand",
     ylab = "Monthly Average Demand", xlab = "Time",
     pch=16  ,col=adjustcolor("black" , 0.9))
```



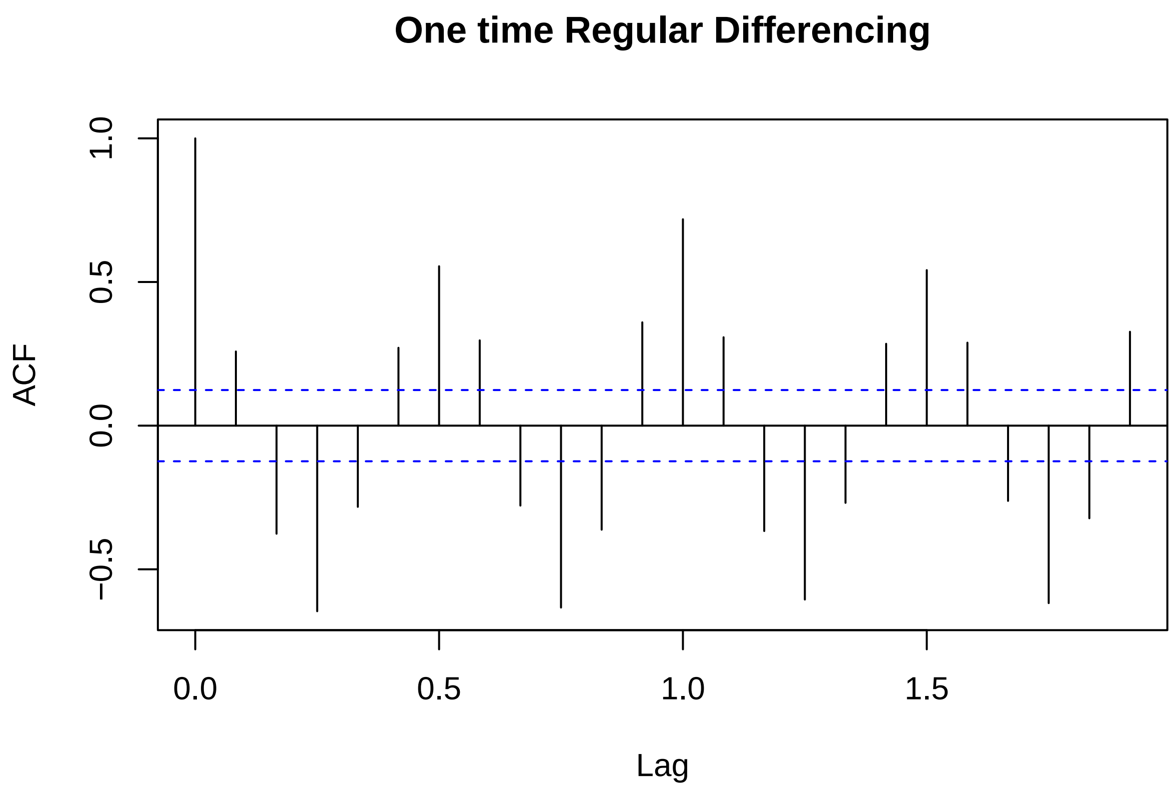

```
acf(Diff.data_lag12, main="Differencing at Lag 12")
```



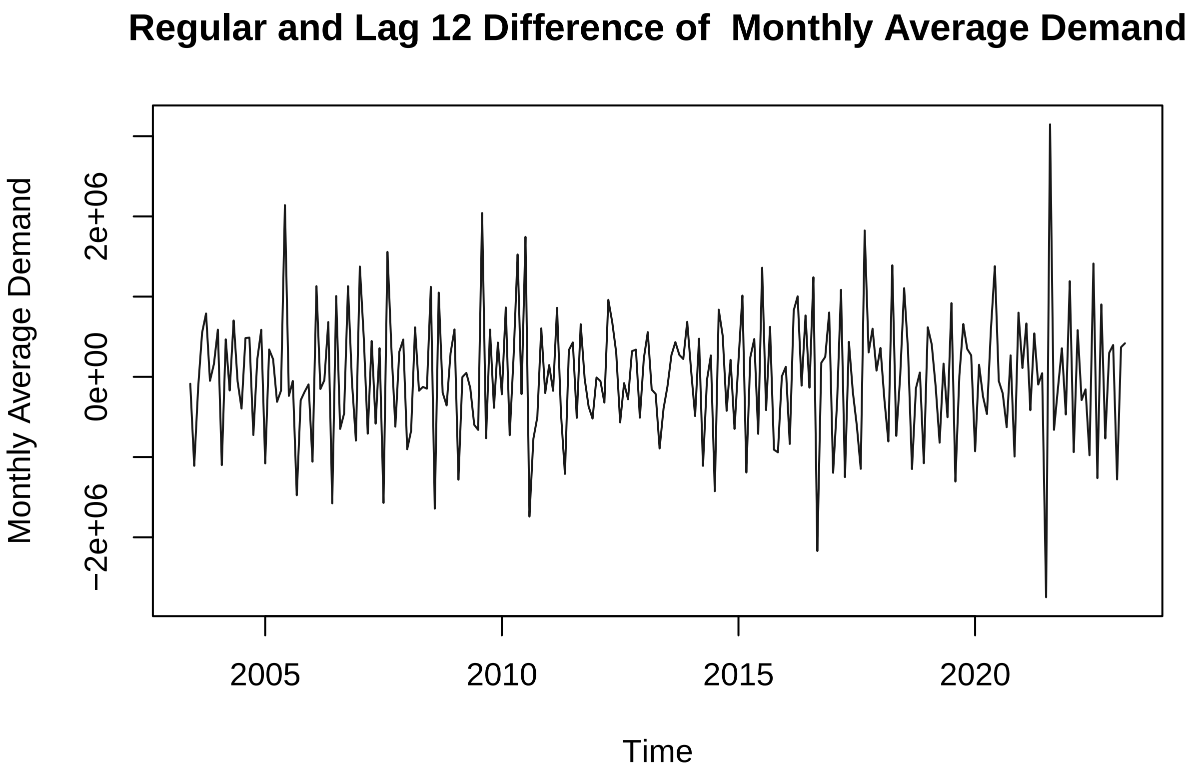

```r
# one time differencing in regular
Diff.data_reg = diff(monthly_avg_demandTS)
plot(Diff.data_reg, main="Regular Difference of Monthly Average Demand",
     ylab = "Monthly Average Demand", xlab = "Time",
     pch=16  ,col=adjustcolor("black" , 0.9))
```



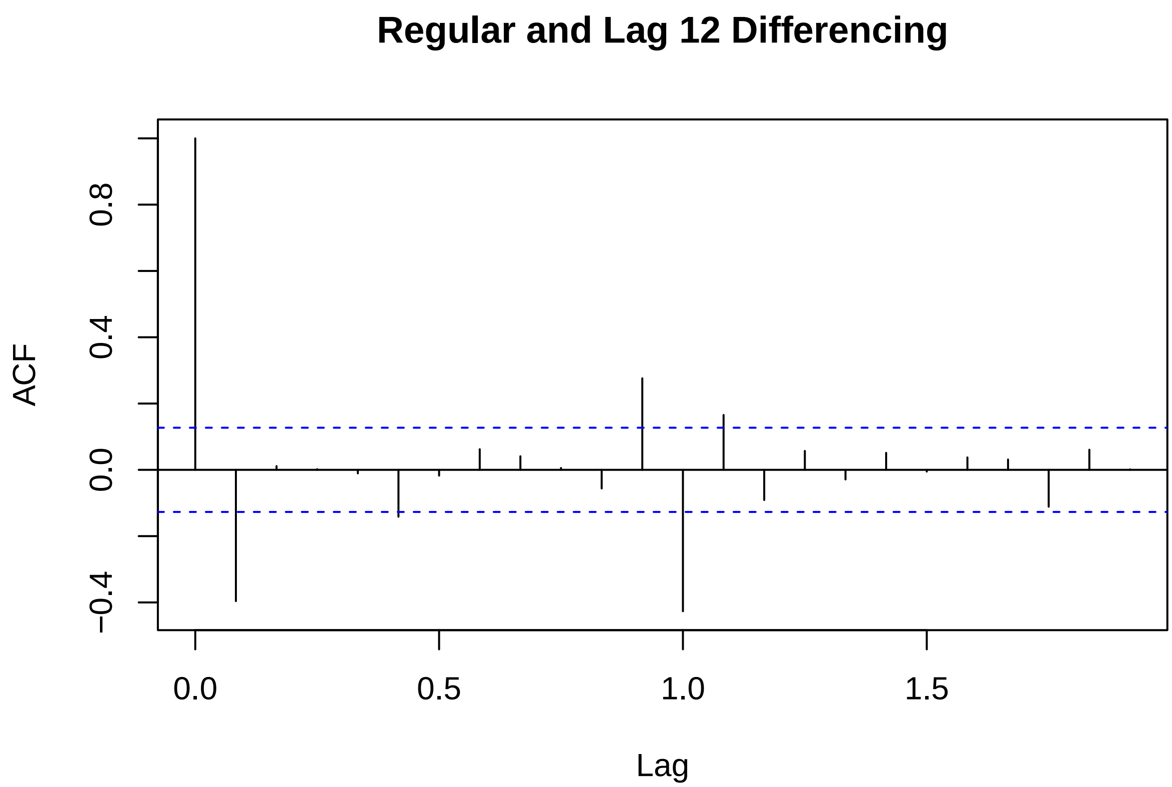

```
acf(Diff.data_reg, main="One time Regular Differencing")
```




```r
# one time differencing in lag 12 and regular
Diff.data_reg_lag12 = diff(diff(monthly_avg_demandTS, lag=12))
plot(Diff.data_reg_lag12,
     main="Regular and Lag 12 Difference of  Monthly Average Demand",
     ylab = "Monthly Average Demand", xlab = "Time",
     pch=16  ,col=adjustcolor("black" , 0.9))
```

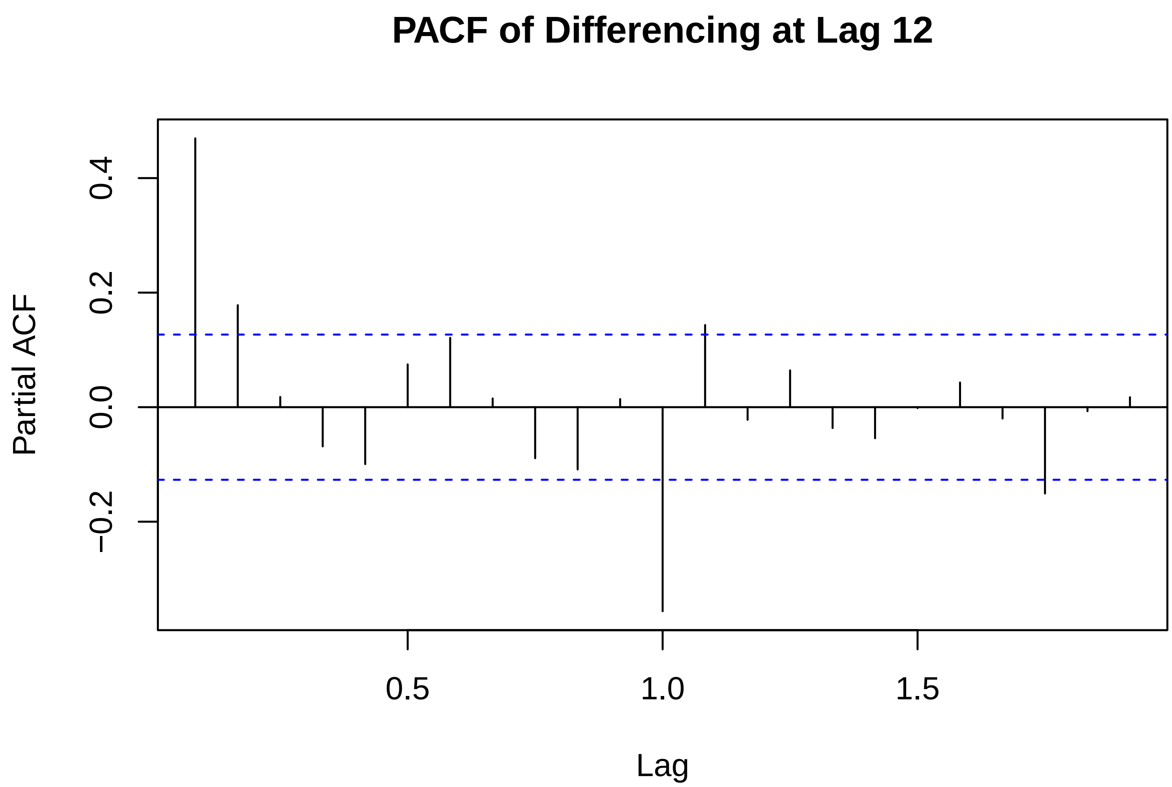



```
acf(Diff.data_reg_lag12, main="Regular and Lag 12 Differencing")
```



Plotting the PACF:

```
pacf(Diff.data_lag12, main="PACF of Differencing at Lag 12")
```



From this, we see that differencing at lag 12 results in stationary data. So we fit SARMA models to this. One could propose the following models for Holt-Winters Residuals (SHOULD BE DONE WITHOUT TEST SET)

One can propose the following model for differenced data:

We see exponential decay in the ACF, and it cuts off at lag 3, with seasonal correlation. We see exponential decay in the PACF and it cuts off at lag 1, also with seasonal correlation. Here are the proposed models:

SARIMA(0,0,0) x (1,1,1)_12 SARIMA(1,0,0) x (1,1,1)_12 SARIMA(2,0,0) x (1,1,1)_12 SARIMA(3,0,0) x (1,1,1)_12

SARIMA(0,0,1) x (1,1,1)_12 SARIMA(1,0,1) x (1,1,1)_12 SARIMA(2,0,1) x (1,1,1)_12 SARIMA(3,0,1) x (1,1,1)_12

Also iterate over all combinations of 0 and 1 for the seasonal ARIMA part.

These are extra no need to fit if no time: SARIMA(1,0,0) x (1,1,0)_12 SARIMA(2,0,0) x (1,1,0)_12 SARIMA(3,0,0) x (1,1,0)_12

SARIMA(1,0,0) x (0,1,1)_12 SARIMA(2,0,0) x (0,1,1)_12 SARIMA(3,0,0) x (0,1,1)_12

```r
## Fitting Models of form SARIMA(i,0,j)x(1,1,1)_12
SARIMA_APSE <- matrix(,nrow = 2,ncol = 4)
for(i in 0:3){
  for(j in 0:1){
    forecast <- sarima.for(train,plot = FALSE,
                        p = i,d = 0,q = j,
                        P = 1, D= 1, Q=1,
                        S=12,
```

```r
                       n.ahead = 12)
    SARIMA_APSE[j+1,i+1] = mean((forecast$pred - test)^2)
  }
}

## Fitting Models of form SARIMA(i,0,j)x(0,1,1)_12
SARIMA_APSE_P <- matrix(,nrow = 2,ncol = 4)
for(i in 0:3){
  for(j in 0:1){
    forecast <- sarima.for(train,plot = FALSE,
                       p = i,d = 0,q = j,
                       P = 0, D= 1, Q=1,
                       S=12,
                       n.ahead = 12)
    SARIMA_APSE_P[j+1,i+1] = mean((forecast$pred - test)^2)
  }
}

## Fitting Models of form SARIMA(i,0,j)x(1,1,0)_12
SARIMA_APSE_Q <- matrix(,nrow = 2,ncol = 4)
for(i in 0:3){
  for(j in 0:1){
    forecast <- sarima.for(train,plot = FALSE,
                       p = i,d = 0,q = j,
                       P = 1, D= 1, Q=0,
                       S=12,
                       n.ahead = 12)
    SARIMA_APSE_Q[j+1,i+1] = mean((forecast$pred - test)^2)
  }
}

## Fitting Models of form SARIMA(i,0,j)x(0,1,0)_12
SARIMA_APSE_PQ <- matrix(,nrow = 2,ncol = 4)
for(i in 0:3){
  for(j in 0:1){
    forecast <- sarima.for(train,plot = FALSE,
                       p = i,d = 0,q = j,
                       P = 0, D= 1, Q=0,
                       S=12,
                       n.ahead = 12)
    SARIMA_APSE_PQ[j+1,i+1] = mean((forecast$pred - test)^2)
  }
}
best_SARIMA_apse <- c(min(SARIMA_APSE),
                      min(SARIMA_APSE_P),
                      min(SARIMA_APSE_Q),
                      min(SARIMA_APSE_PQ))

best_SARIMA_apse
```

```
## [1] 406948402985 337094285190 188356686838 226677793928
```

```r
hw.additive_apse > best_SARIMA_apse
```

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

Not really good, but tried some Q = 2 ones. We fit these because the ACF at high lags showed another seasonal correlation.

```r
## Fitting Models of form SARIMA(i,0,j)x(1,1,2)_12
SARIMA_APSE_Q2 <- matrix(,nrow = 2,ncol = 4)
for(i in 0:3){
  for(j in 0:1){
    forecast <- sarima.for(train,plot = FALSE,
                           p = i,d = 0,q = j,
                           P = 0, D= 1, Q=2,
                           S=12,
                           n.ahead = 12)
    SARIMA_APSE_Q2[j+1,i+1] = mean((forecast$pred - test)^2)
  }
}
```

From the code chunk above, we see that the best model on the test set is of the form SARIMA(p,0,q)x(1,1,0). Extracting the best model:

```r
which(SARIMA_APSE_Q == min(SARIMA_APSE_Q), arr.ind = TRUE)
```

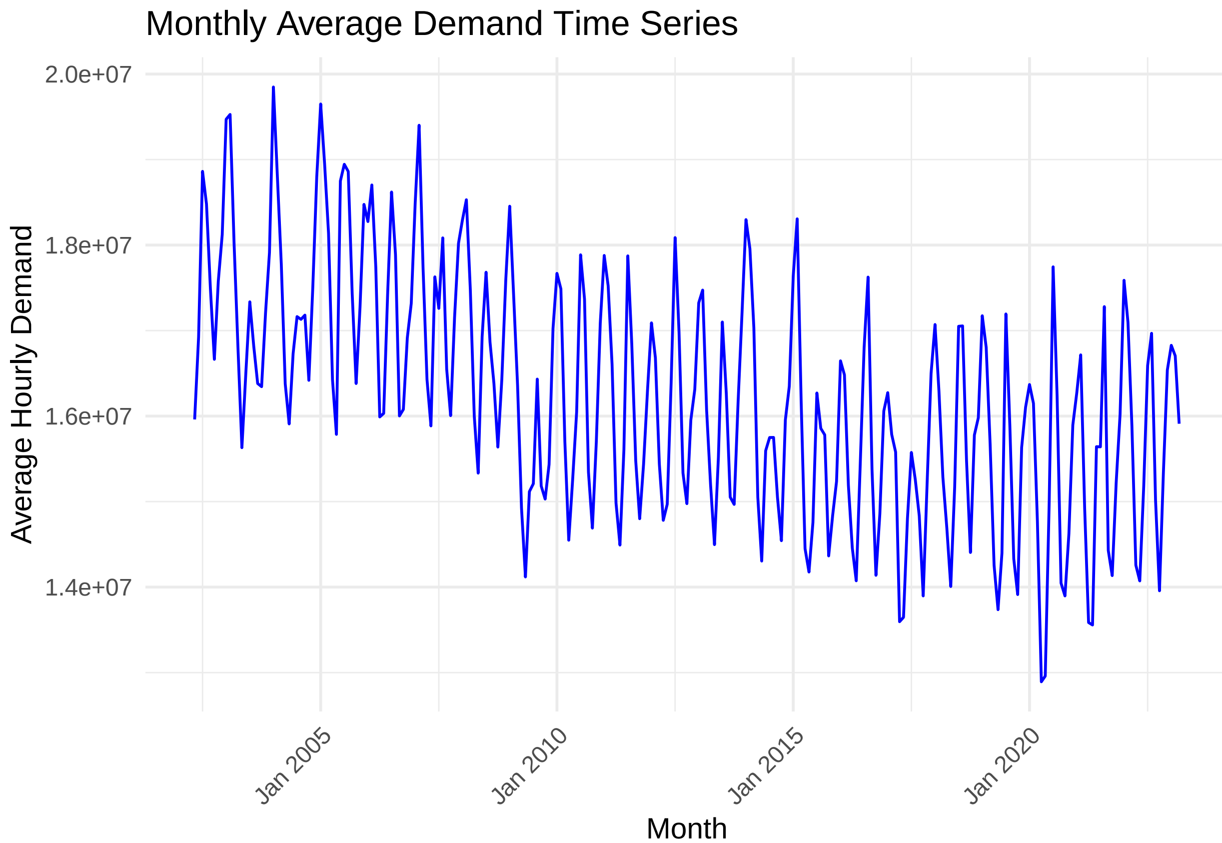
```
##      row col
## [1,]   1   4
```

And so the best model on the test set is SARIMA(3,0,0)x(1,1,0)_12.

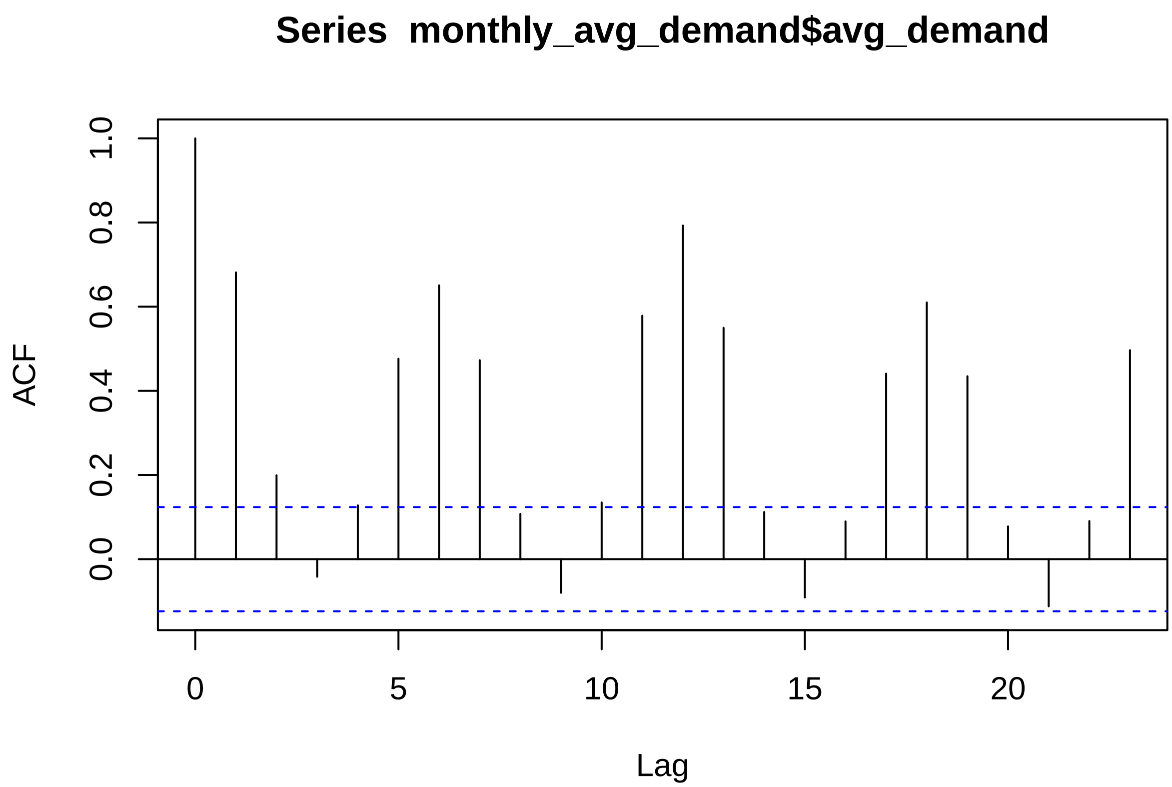
plotting the data and its acf

```r
# Plot the time series of monthly average demand
ggplot(monthly_avg_demand, aes(x = month_year, y = avg_demand)) +
  geom_line(color = "blue") +
  labs(title = "Monthly Average Demand Time Series",
       x = "Month",
       y = "Average Hourly Demand") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

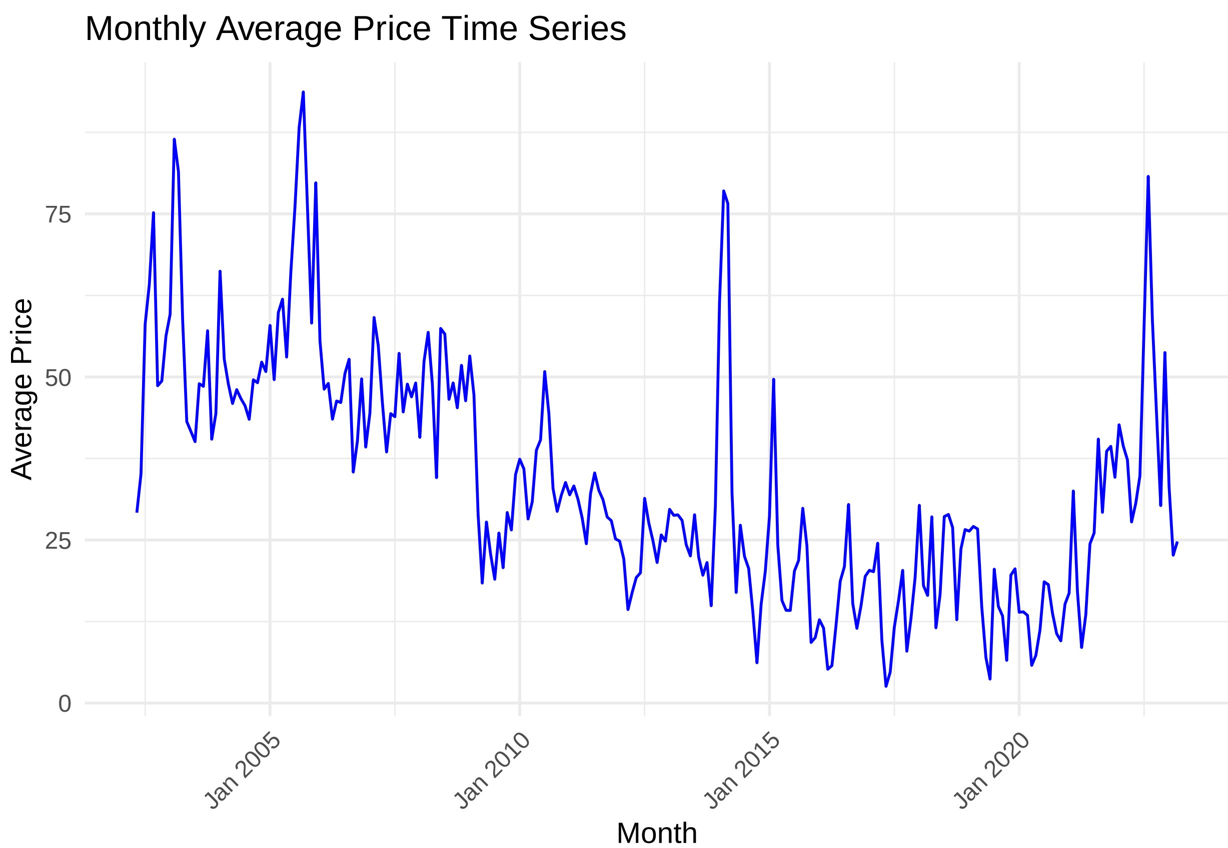
```
## Warning: The `trans` argument of `continuous_scale()` is deprecated as of ggplot2 3.5.0.
## i Please use the `transform` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



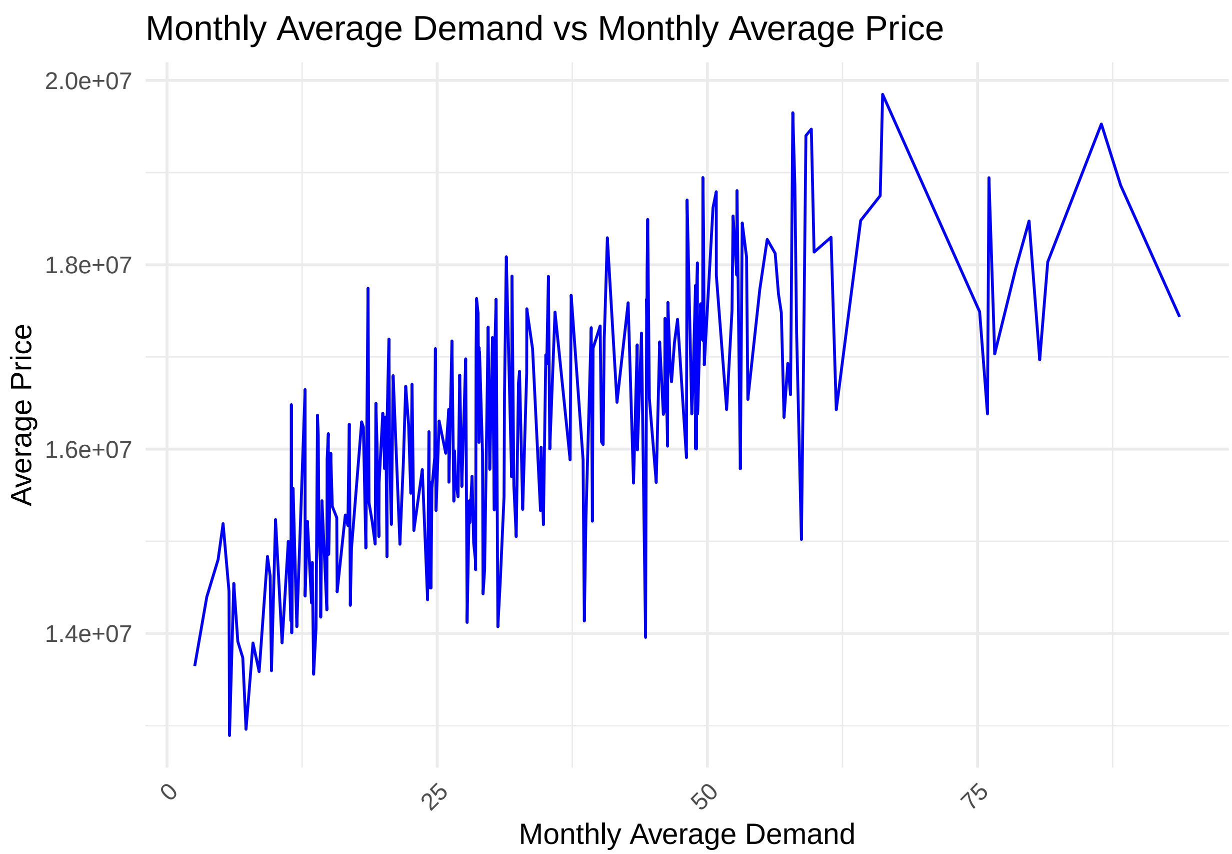

```
acf(monthly_avg_demand$avg_demand)
```



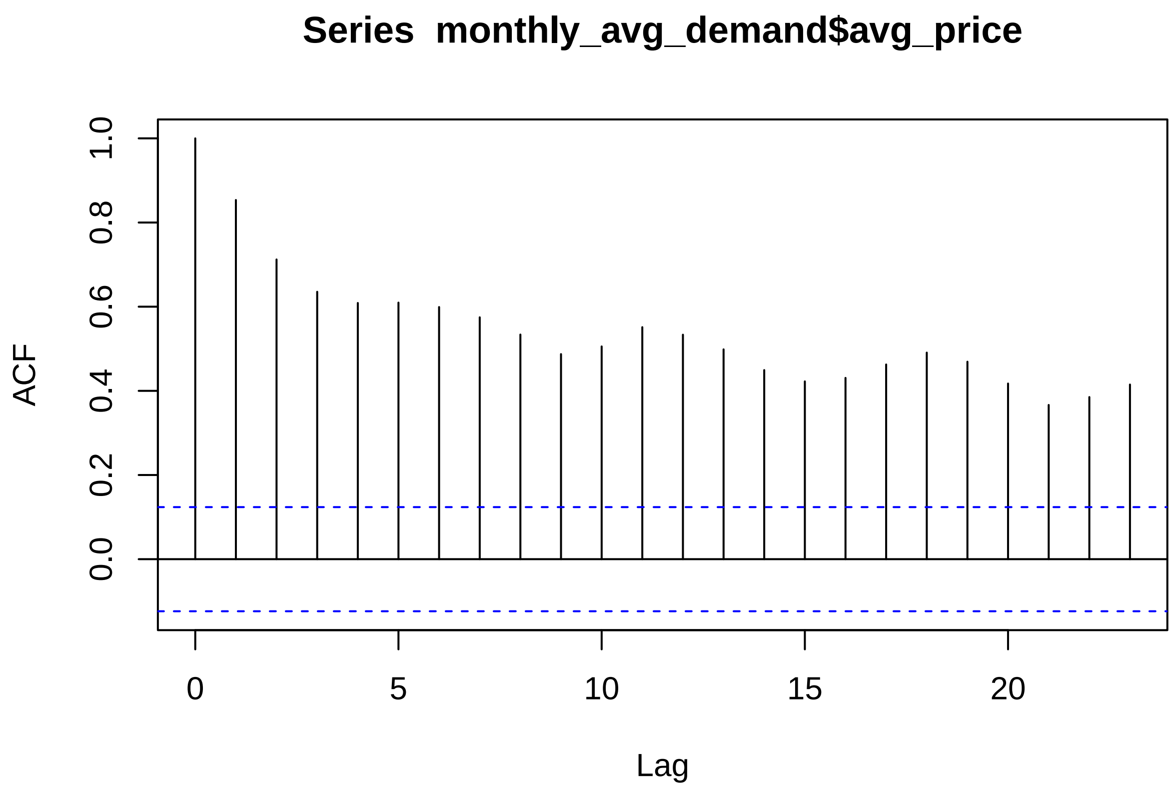

```
ggplot(monthly_avg_demand, aes(x = month_year, y = avg_price)) +
  geom_line(color = "blue") +
  labs(title = "Monthly Average Price Time Series",
       x = "Month",
       y = "Average Price") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```r
ggplot(monthly_avg_demand, aes(x = avg_price, y = avg_demand)) +
  geom_line(color = "blue") +
  labs(title = "Monthly Average Demand vs Monthly Average Price",
       x = "Monthly Average Demand",
       y = "Average Price") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

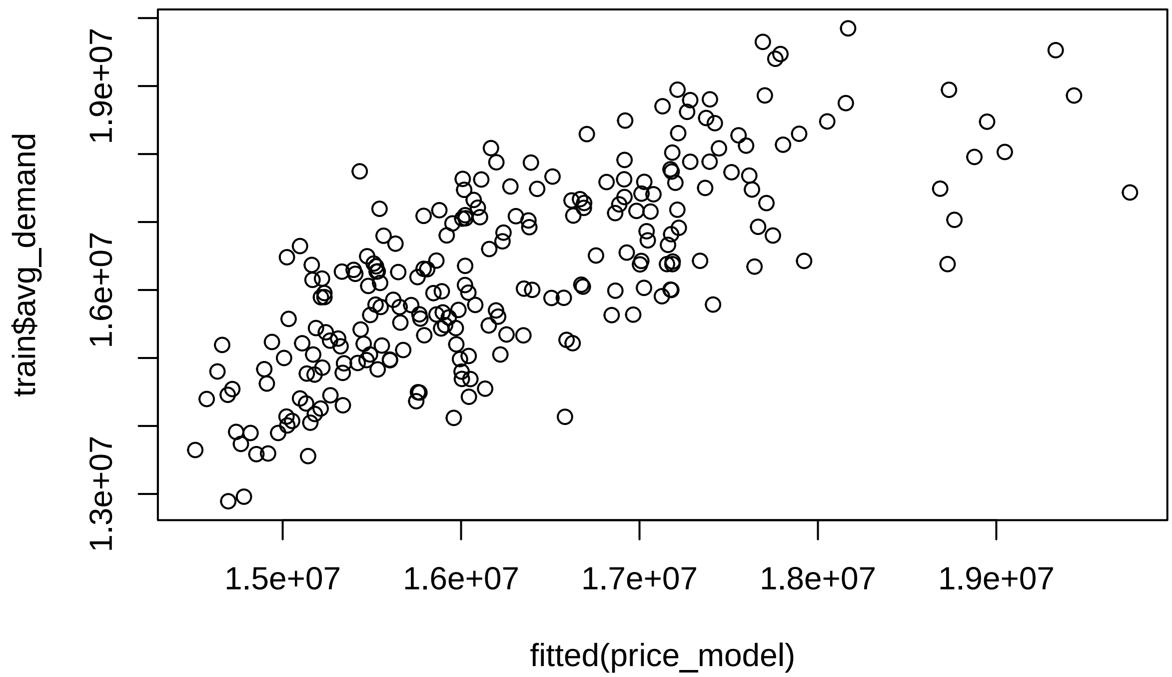



```r
#checking time correlation with price
acf(monthly_avg_demand$avg_price)
```

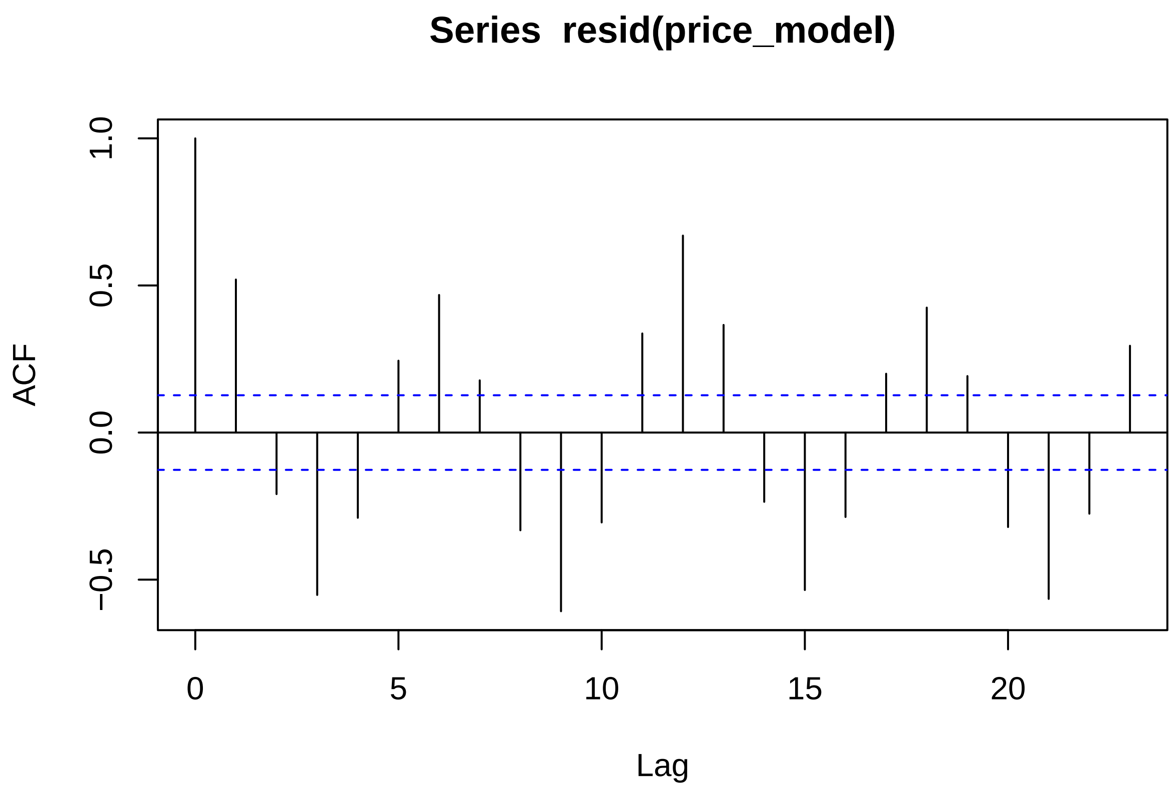


fitting regression models

```r
#fit against price
train = monthly_avg_demand[monthly_avg_demand$Set == "Train",]
test = monthly_avg_demand[monthly_avg_demand$Set == "Test",]

price_model <- lm(train$avg_demand~train$avg_price)

plot(fitted(price_model),train$avg_demand)
```




```
acf(resid(price_model))
```



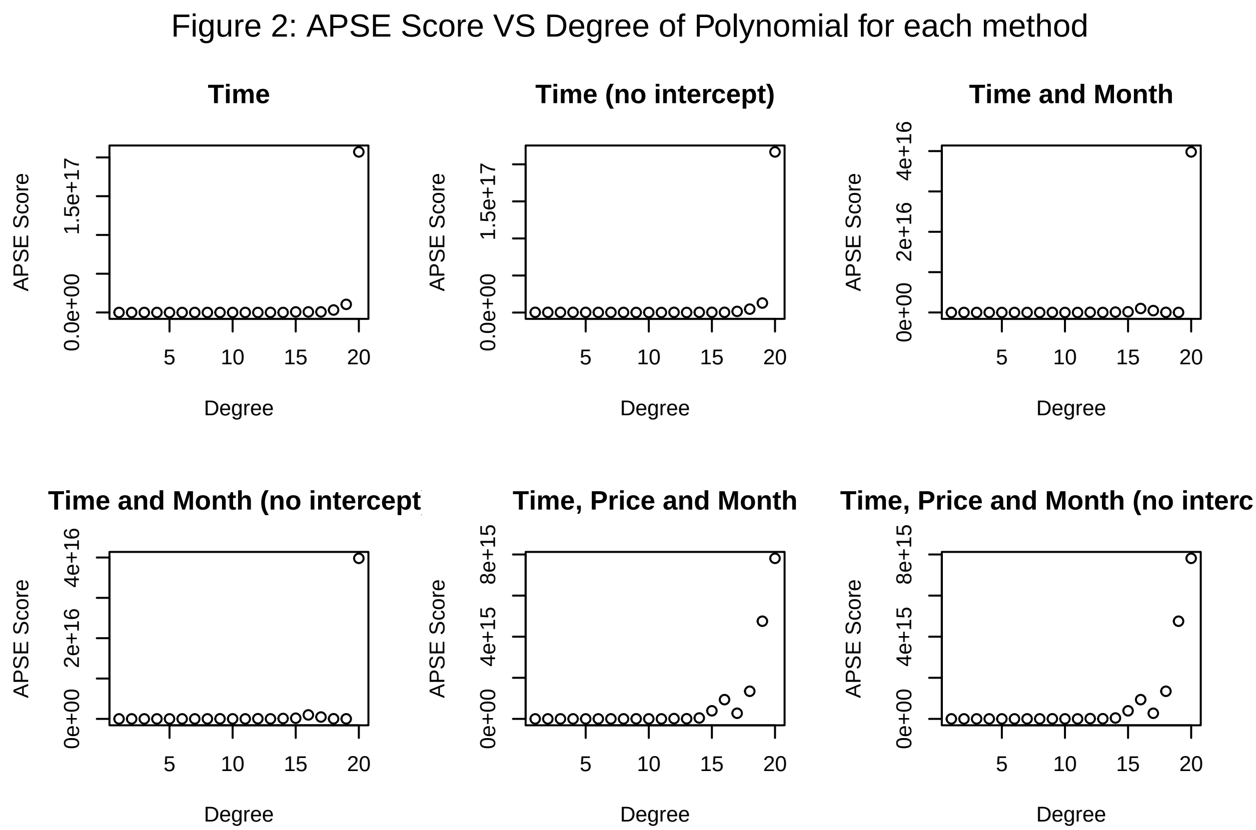
Regular regression

```
#initialize as a time series object
tim <- ts(monthly_avg_demand,start = c(2002, 5),frequency = 12)


tim1 <- window(tim, end = c(2022, 3))

#get month as a factor ()
month <- as.factor(cycle(tim1))

#initialize degrees
p_vals<-c(1:20)


#dataframe to store the cv scores
APSE_scores <- setNames(data.frame(
  replicate(length(p_vals), numeric(6), simplify = FALSE)),
  paste0("p_", p_vals)
)
APSE_scores$method <- c('Time', 'Time and Month', 'Time, Price and Month',
                        'Time (no intercept)', 'Time and Month (no intercept)',
                        'Time, Price and Month (no intercept)')

#loop through the different degree,alpha, and method combinations
```

```r
for (p_val in p_vals) {
  train = monthly_avg_demand[monthly_avg_demand$Set == "Train",]
  time = paste0("poly_", 1:p_val)
  test = monthly_avg_demand[monthly_avg_demand$Set == "Test",]
  price <- train$avg_price
  train_avg_demand <- train$avg_demand
  test_avg_demand <- test$avg_demand

  i = 1
  for (method in c('Time', 'Time and Month', 'Time, Price and Month')) {
    if (method == 'Time') {

      # With intercept
      model_ <- lm(train_avg_demand ~ poly(time(tim1),p_val))
      APSE_ <- mean((predict(model_, newdata =
                              data.frame(tim1=time(window(tim, start = c(2022, 4)))))
                    - test_avg_demand)^2)
      APSE_scores[i, p_val] <- APSE_

      # Without intercept
      model_ <- lm(train_avg_demand ~ poly(time(tim1),p_val)-1)
      APSE_ <- mean((predict(model_, newdata =
                              data.frame(tim1=time(window(tim, start = c(2022, 4)))))
                    - test_avg_demand)^2)
      APSE_scores[i + 3, p_val] <- APSE_



    } else if (method == 'Time and Month') {

      # With intercept
      model_ <- lm(train_avg_demand ~ poly(time(tim1),p_val) +  month)
      APSE_ <- mean((predict(model_, newdata =
                              data.frame(tim1=time(window(tim, start = c(2022, 4))),
                                         month=as.factor(cycle(window(tim, start = c(2022, 4))))))
                    - test_avg_demand)^2)
      APSE_scores[i, p_val] <- APSE_

      # Without intercept
      model_ <- lm(train_avg_demand ~ poly(time(tim1),p_val) + month -1)
      APSE_ <- mean((predict(model_, newdata =
                              data.frame(tim1=time(window(tim, start = c(2022, 4))),
                                         month=as.factor(cycle(window(tim, start = c(2022, 4))))))
                    - test_avg_demand)^2)
      APSE_scores[i + 3, p_val] <- APSE_

    } else if (method == 'Time, Price and Month') {

      # With intercept
      model_ <- lm(train_avg_demand ~ poly(time(tim1),p_val) +  month + price)
      APSE_ <- mean((predict(model_, newdata =
                              data.frame(tim1=time(window(tim, start = c(2022, 4))),
                                         month=as.factor(cycle(window(tim, start = c(2022, 4)))), price
```

```
                    - test_avg_demand)^2)
      APSE_scores[i, p_val] <- APSE_

      # Without intercept
      model_ <- lm(train_avg_demand ~ poly(time(tim1),p_val) + month + price -1)
      APSE_ <- mean((predict(model_, newdata =
                              data.frame(tim1=time(window(tim, start = c(2022, 4))),
                                         month=as.factor(cycle(window(tim, start = c(2022, 4)))), price
                    - test_avg_demand)^2)
      APSE_scores[i + 3, p_val] <- APSE_
    }

    i = i + 1
  }
}
```

```
#plot the APSE of each combination
par(mfrow = c(2,3),oma = c(0, 0, 2, 0))
for(i in 1:3){
  plot(x = 1:20, y = APSE_scores[i, 1:(ncol(APSE_scores) - 1)],
       main = APSE_scores$method[i],
       xlab = "Degree",
       ylab = "APSE Score")
  plot(x = 1:20, y = APSE_scores[i+3, 1:(ncol(APSE_scores) - 1)],
       main = APSE_scores$method[i+3],
       xlab = "Degree",
       ylab = "APSE Score")
}
mtext("Figure 2: APSE Score VS Degree of Polynomial for each method", outer = TRUE, cex = 1, line = 0)
```




```r
long_APSE_scores <- APSE_scores %>%
  pivot_longer(
    cols = -method,
    names_to = "p_value",
    values_to = "score"
  )

# Find the minimum score for each method
best_p_df <- long_APSE_scores %>%
  group_by(method) %>%
  summarize(
    best_p = p_value[which.min(score)],
    min_score = min(score)
  )

best_p_df
```

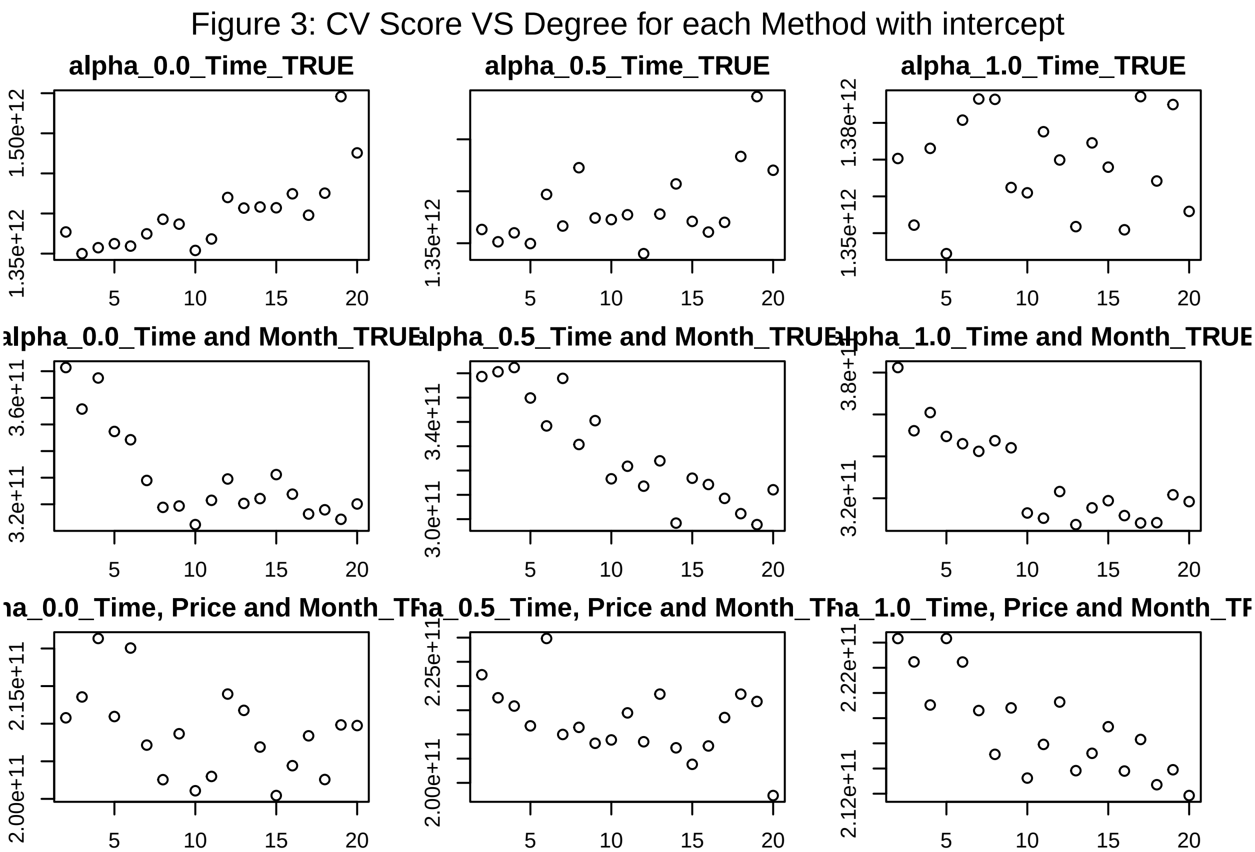
```
## # A tibble: 6 x 3
##   method                              best_p min_score
##   <chr>                               <chr>      <dbl>
## 1 Time                                p_3      1.11e12
## 2 Time (no intercept)                 p_13     9.94e13
## 3 Time and Month                      p_3      9.72e10
## 4 Time and Month (no intercept)       p_3      9.72e10
## 5 Time, Price and Month               p_5      2.18e11
## 6 Time, Price and Month (no intercept) p_5      2.18e11
```

Regularization

```
set.seed(443)

#initialize degrees
p_vals<-c(2:20)

#get month as a factor ()
month <- as.factor(cycle(tim1))

#initialize as a time series object
tim <- ts(monthly_avg_demand,start = c(2002, 5),frequency = 12)

#initialize orthogonal polynomials based on month
monthly_avg_demand <- cbind(monthly_avg_demand, poly(time(tim), 20))
colnames(monthly_avg_demand)[(ncol(monthly_avg_demand)- 19):ncol(monthly_avg_demand)] <- paste0("poly_"



#initialize the different alpha values.
a_vals<-c(0,0.5,1)

Log.Lambda.Seq = c(c(-15, -10, -5, -2, -1, -0.5), seq(0, 10,
    by = 0.1))
Lambda.Seq = exp(Log.Lambda.Seq)


#dataframe to store the cv scores
methods <- c('Time', 'Time and Month', 'Time, Price and Month')

# Create a dataframe to store the CV scores
method_combinations <- expand.grid(alpha = a_vals, method = methods, intercept = c(TRUE, FALSE))
CV_scores <- setNames(data.frame(
  replicate(length(p_vals), numeric(nrow(method_combinations)), simplify = FALSE)),
  paste0("p_", p_vals)
)
CV_scores$combination <- apply(method_combinations, 1, function(row) {
  paste("alpha", row['alpha'], row['method'], row['intercept'], sep = "_")
})


# dataframe to store the optimal lambdas
optim_lambdas <- setNames(data.frame(
  replicate(length(p_vals), numeric(nrow(method_combinations)), simplify = FALSE)),
  paste0("p_", p_vals)
)
optim_lambdas$combination <- CV_scores$combination


#define training data
train <- monthly_avg_demand[monthly_avg_demand$Set == "Train",]
y_train <- train$avg_demand
avg_price <- train$avg_price
```

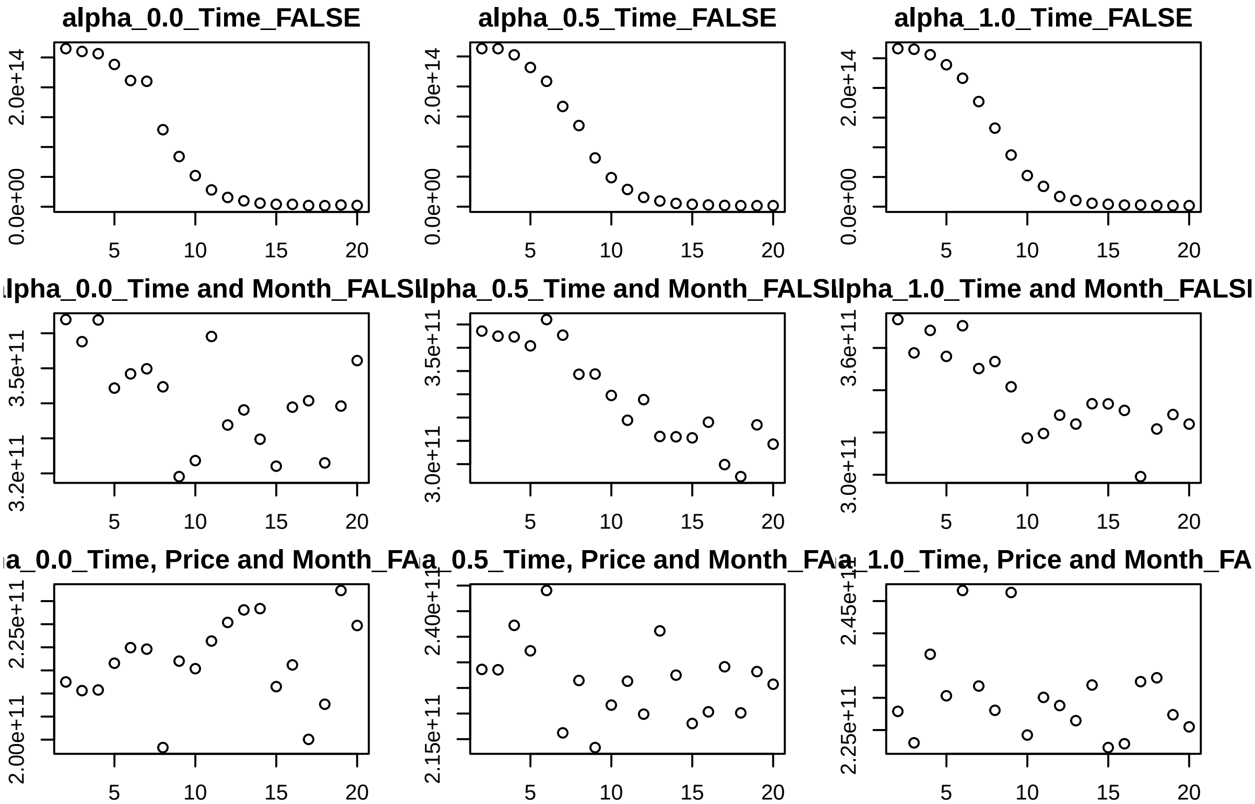
```r
# do cross validation and loop through the different degree, alpha, and method combinations

for (p_val in p_vals){
  i=0
  for (a_val in a_vals){
    train_time = paste0("poly_", 1:p_val)
    X_train = as.matrix(train[, train_time])

    j=0
    for (method in methods) {
      if (method == 'Time and Month'){
        X_train = cbind(X_train,model.matrix(~month-1))

      } else if (method == 'Time, Price and Month'){
        X_train = cbind(X_train,model.matrix(~month-1),avg_price)
      }

      #with intercept
      CV <- cv.glmnet(X_train, y_train, alpha = a_val, lambda = Lambda.Seq,
                      nfolds = 10, standardize = TRUE)

      # Store CV score and optimal lambda
      CV_scores[1+i+j, p_val-1] <- CV$cvm[CV$index[2]]
      optim_lambdas[1+i+j, p_val-1] <- CV$lambda.1se

      #without
      CV1 <- cv.glmnet(X_train, y_train, alpha = a_val, lambda = Lambda.Seq,
                      nfolds = 10, standardize = TRUE,intercept=FALSE)

      # Store CV score and optimal lambda
      CV_scores[10+i+j, p_val-1] <- CV1$cvm[CV1$index[2]]
      optim_lambdas[10+i+j, p_val-1] <- CV1$lambda.1se

      j <- j+3
    }
    i <- i+1
  }
}
```

```r
#plot the CV scores from the previous step
par(mfrow = c(3,3),mar=c(2,2,2,2),oma=c(0,0,2,0))
for(i in 1:9){
  plot(x = 2:20, y = CV_scores[i, 1:(ncol(CV_scores) - 1)],
       main = CV_scores$combination[i],
       xlab = "Method",
       ylab = "CV Score at 1se")
}
mtext("Figure 3: CV Score VS Degree for each Method with intercept",
      outer = TRUE, cex = 1, line = 0)
```




```r
par(mfrow = c(3,3),mar=c(2,2,2,2))
for(i in 10:18){
  plot(x = 2:20, y = CV_scores[i, 1:(ncol(CV_scores) - 1)],
       main = CV_scores$combination[i],
       xlab = "Method",
       ylab = "CV Score at 1se")
}
mtext("Figure 3: CV Score VS Degreefor Method without Intercept", outer = TRUE,
      cex = 1, line = 0)
```

Figure 3: CV Score VS Degreefor Method without Intercept




```r
best_p <- c()
for(i in 1:18){
  best_p[i] = which.min(CV_scores[i,1:(ncol(CV_scores) - 1)])+1
}


# the best p for each corresponding alpha is:

for (i in 1:18){
  print(paste(CV_scores$combination[i],best_p[i],sep=": "))
}
```

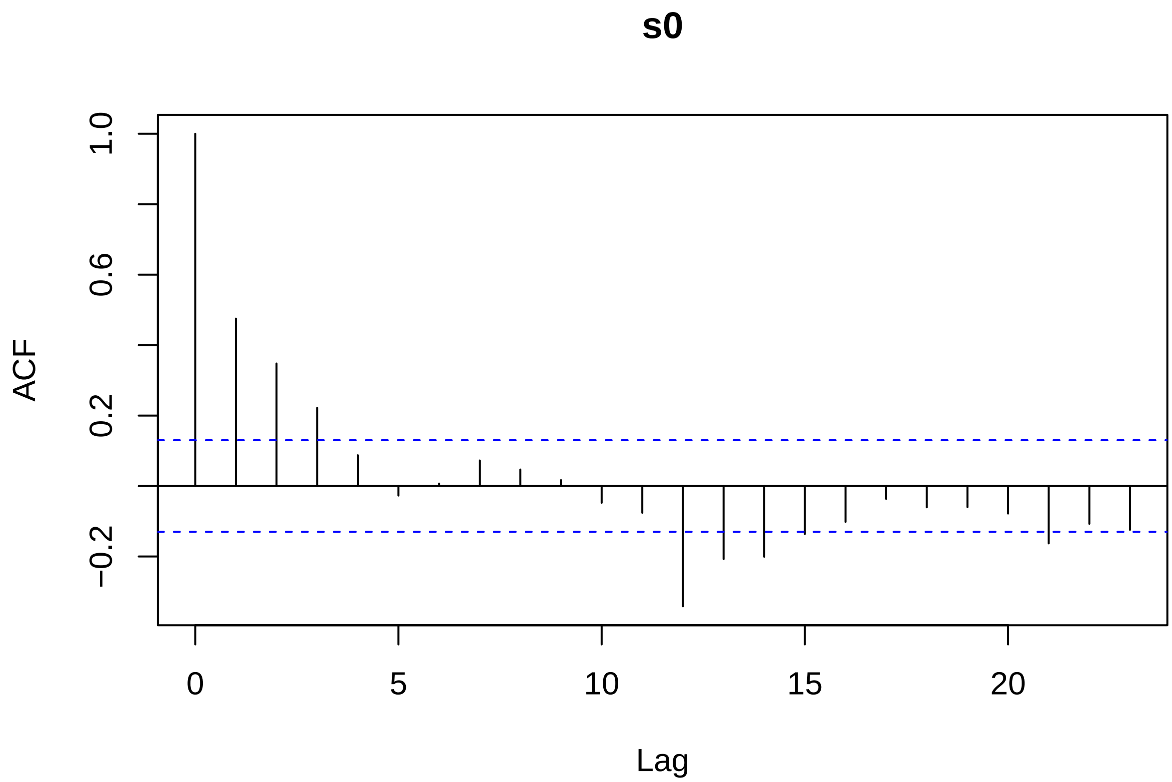
```
## [1] "alpha_0.0_Time_TRUE: 3"
## [1] "alpha_0.5_Time_TRUE: 12"
## [1] "alpha_1.0_Time_TRUE: 5"
## [1] "alpha_0.0_Time and Month_TRUE: 10"
## [1] "alpha_0.5_Time and Month_TRUE: 19"
## [1] "alpha_1.0_Time and Month_TRUE: 13"
## [1] "alpha_0.0_Time, Price and Month_TRUE: 15"
## [1] "alpha_0.5_Time, Price and Month_TRUE: 20"
## [1] "alpha_1.0_Time, Price and Month_TRUE: 20"
## [1] "alpha_0.0_Time_FALSE: 18"
## [1] "alpha_0.5_Time_FALSE: 19"
## [1] "alpha_1.0_Time_FALSE: 19"
## [1] "alpha_0.0_Time and Month_FALSE: 9"
## [1] "alpha_0.5_Time and Month_FALSE: 18"
```

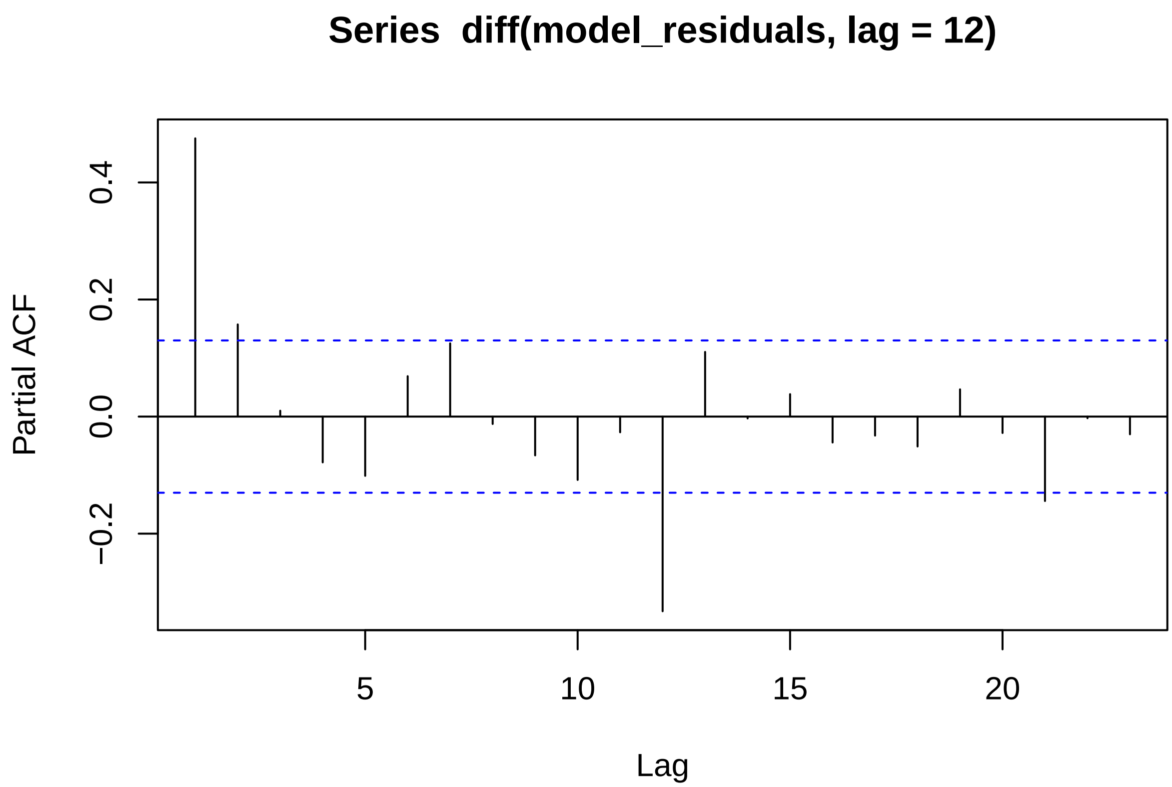
```
## [1] "alpha_1.0_Time and Month_FALSE: 17"
## [1] "alpha_0.0_Time, Price and Month_FALSE: 8"
## [1] "alpha_0.5_Time, Price and Month_FALSE: 9"
## [1] "alpha_1.0_Time, Price and Month_FALSE: 15"
```

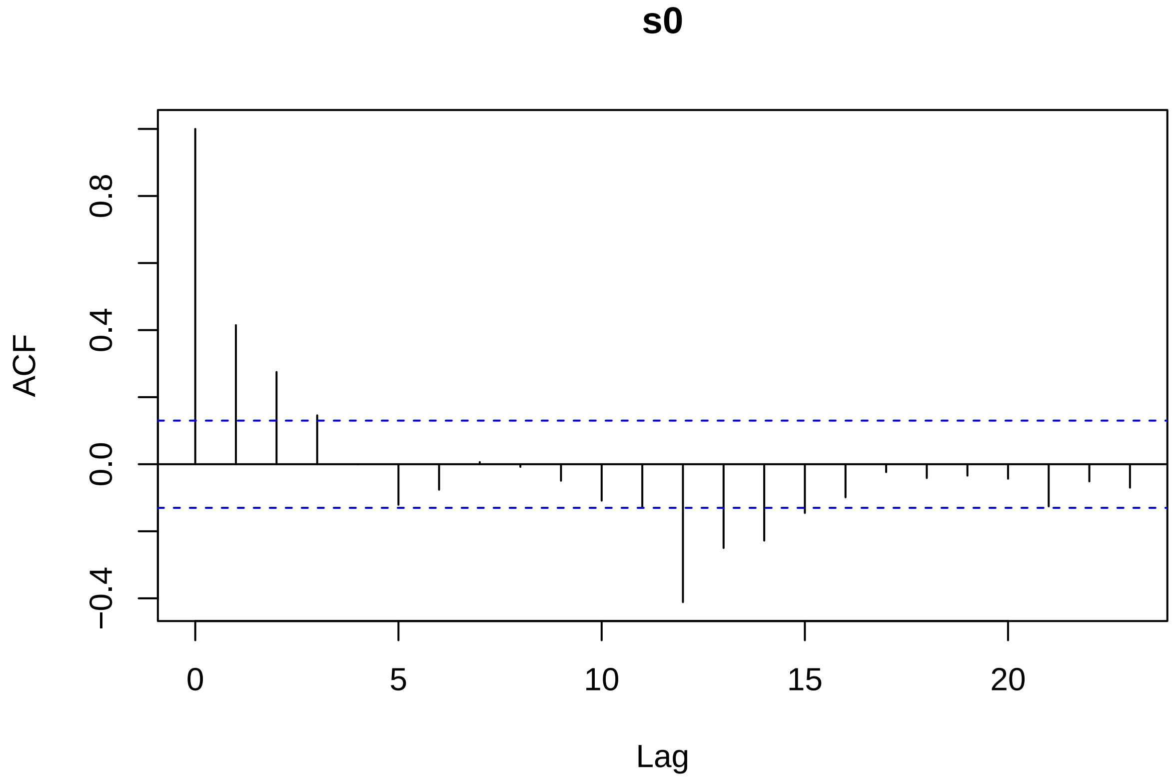
```r
# test on the data

#create new APSE dataframe to store the regularization APSE results
APSE_score1 <- setNames(data.frame(numeric(nrow(method_combinations))),'APSE')
APSE_score1$combination <- CV_scores$combination

#another APSE dataframe to check later for SARIMA
APSE_score2 <-setNames(as.data.frame(matrix(0, ncol = 8, nrow = 9))
                       ,c('APSE_10_11','APSE_01_11','APSE_11_11','APSE_20_11',
                          'APSE_21_11','APSE_22_11', 'APSE_02_11','APSE_12_11'))
APSE_score2$combination <- CV_scores$combination[1:9]

train = monthly_avg_demand[monthly_avg_demand$Set == "Train",]
y_train = train$avg_demand

test <- monthly_avg_demand[monthly_avg_demand$Set == "Test",]
y_test <- test$avg_demand

i=0
k=1
for (a_val in a_vals){


  j=0
  for (method in methods) {


    p <- best_p[k]

    train_time = paste0("poly_", 1:p)
    X_train = as.matrix(train[, train_time])

    test_time <- paste0("poly_", 1:p)
    X_test <- as.matrix(test[, test_time])


    #for no intercept
    p1 <- best_p[k+9]

    train_time1 = paste0("poly_", 1:p1)
    X_train1 = as.matrix(train[, train_time])

    test_time1 <- paste0("poly_", 1:p1)
    X_test1 <- as.matrix(test[, test_time])


    #get month as a factor ()
    month <- as.factor(cycle(tim1))
    avg_price <- train$avg_price
```

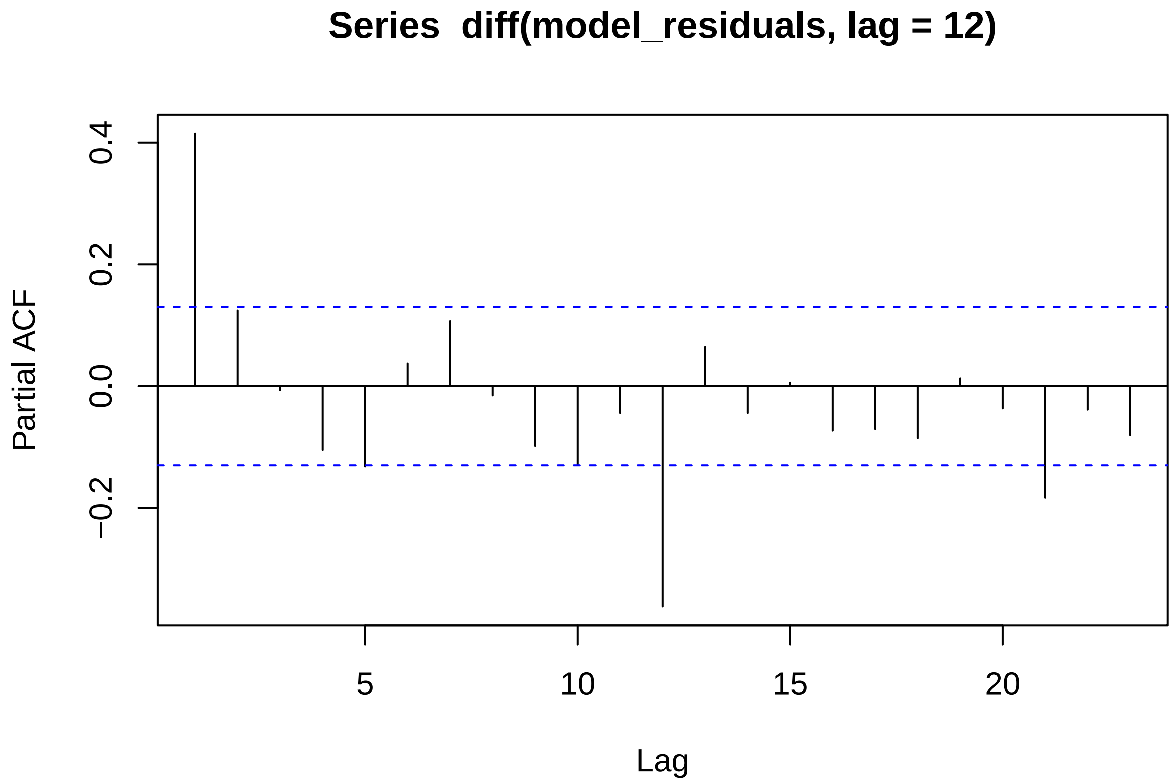
```r
  if (method == 'Time and Month'){
    X_train = cbind(X_train,model.matrix(~month-1))
    X_train1 = cbind(X_train1,model.matrix(~month-1))

    #redefine test variables with same name just in case
    month <- as.factor(cycle(window(tim, start = c(2022, 4)))) #month from test
    X_test = cbind(X_test,model.matrix(~month-1))
    X_test1 = cbind(X_test1,model.matrix(~month-1))

  } else if (method == 'Time, Price and Month'){
    X_train = cbind(X_train,model.matrix(~month-1),avg_price)
    X_train1 = cbind(X_train1,model.matrix(~month-1),avg_price)

    #redefine test variables with same name just in case
    month <- as.factor(cycle(window(tim, start = c(2022, 4)))) #month from test
    avg_price <- test$avg_price #avg_price from test
    X_test = cbind(X_test,model.matrix(~month-1),avg_price)
    X_test1 = cbind(X_test1,model.matrix(~month-1),avg_price)
  }

  #with intercept
  model <- glmnet(X_train, y_train, alpha = a_val,
                  lambda = optim_lambdas[1+i+j,p-1],
                  standardize = TRUE)

  APSE_score1[1+i+j,1] = mean((predict(model, newx =X_test,
                      type="response") - y_test)^2)

  model_residuals <- predict(model,newx = X_train)- y_train
  acf(diff(model_residuals, lag = 12))
  pacf(diff(model_residuals, lag = 12))

  ## p = 1, q = 0
  sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                            p = 1,q=0,d=0, P = 1, Q= 1,D = 1, S= 12)
  APSE_score2[1+i+j,1] =  mean((predict(model, newx =X_test,
                      type="response")+sarima_part$pred - y_test)^2)

  ## p = 0, q = 1
  sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                            p = 0,q=1,d=0, P = 1, Q= 1,D = 1, S= 12)
  APSE_score2[1+i+j,2] =  mean((predict(model, newx =X_test,
                      type="response")+sarima_part$pred - y_test)^2)

  ## p = 1, q = 1
  sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                            p = 1,q=1,d=0, P = 1, Q= 1,D = 1, S= 12)
  APSE_score2[1+i+j,3] =  mean((predict(model, newx =X_test,
                      type="response")+sarima_part$pred - y_test)^2)

  ## p = 2 q = 0
  sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                            p = 2,q=0,d=0, P = 1, Q= 1,D = 1, S= 12)
```

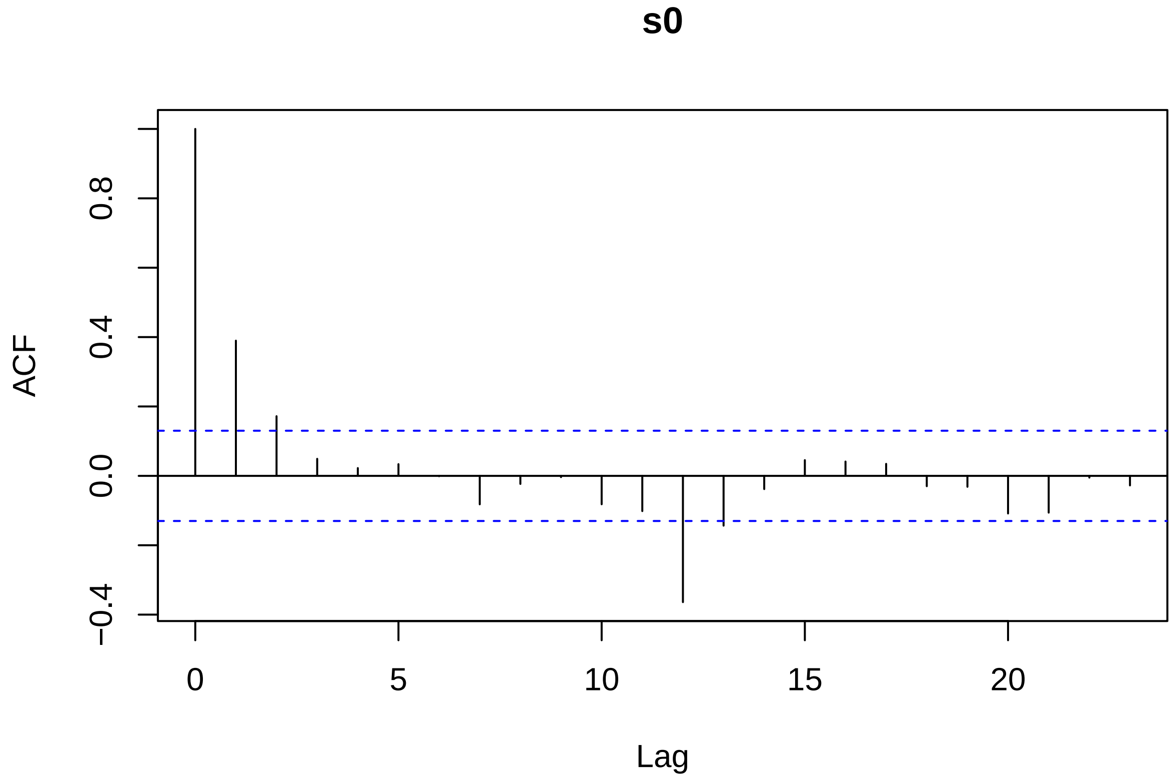
```r
    APSE_score2[1+i+j,4] =  mean((predict(model, newx =X_test,
                        type="response")+sarima_part$pred - y_test)^2)

    ## p = 2 q = 1
    sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                              p = 2,q=1,d=0, P = 1, Q= 1,D = 1, S= 12)
    APSE_score2[1+i+j,5] =  mean((predict(model, newx =X_test,
                        type="response")+sarima_part$pred - y_test)^2)

    ## p = 2 q = 2
    sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                              p = 2,q=2,d=0, P = 1, Q= 1,D = 1, S= 12)
    APSE_score2[1+i+j,6] =  mean((predict(model, newx =X_test,
                        type="response")+sarima_part$pred - y_test)^2)

    ## p = 0 q = 2
    sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                              p = 0,q=2,d=0, P = 1, Q= 1,D = 1, S= 12)
    APSE_score2[1+i+j,7] =  mean((predict(model, newx =X_test,
                        type="response")+sarima_part$pred - y_test)^2)

    ## p = 1 q = 2
    sarima_part <- sarima.for(model_residuals,n.ahead = 12,plot = FALSE,
                              p = 1,q=2,d=0, P = 1, Q= 1,D = 1, S= 12)
    APSE_score2[1+i+j,8] =  mean((predict(model, newx =X_test,
                        type="response")+sarima_part$pred - y_test)^2)

    #without intercept
    model1 <- glmnet(X_train1, y_train, alpha = a_val,
                    lambda = optim_lambdas[10+i+j,p-1],
                    standardize = TRUE, intercept =FALSE)

    APSE_score1[10+i+j,1] = mean((predict(model1, newx =X_test1,
                        type="response") - y_test)^2)

    j <- j+3
    k <- k+1
  }
  i <- i+1
}
```


s0
1.0
0.6
0.2
−0.2
ACF
0
5
10
15
20
Lag

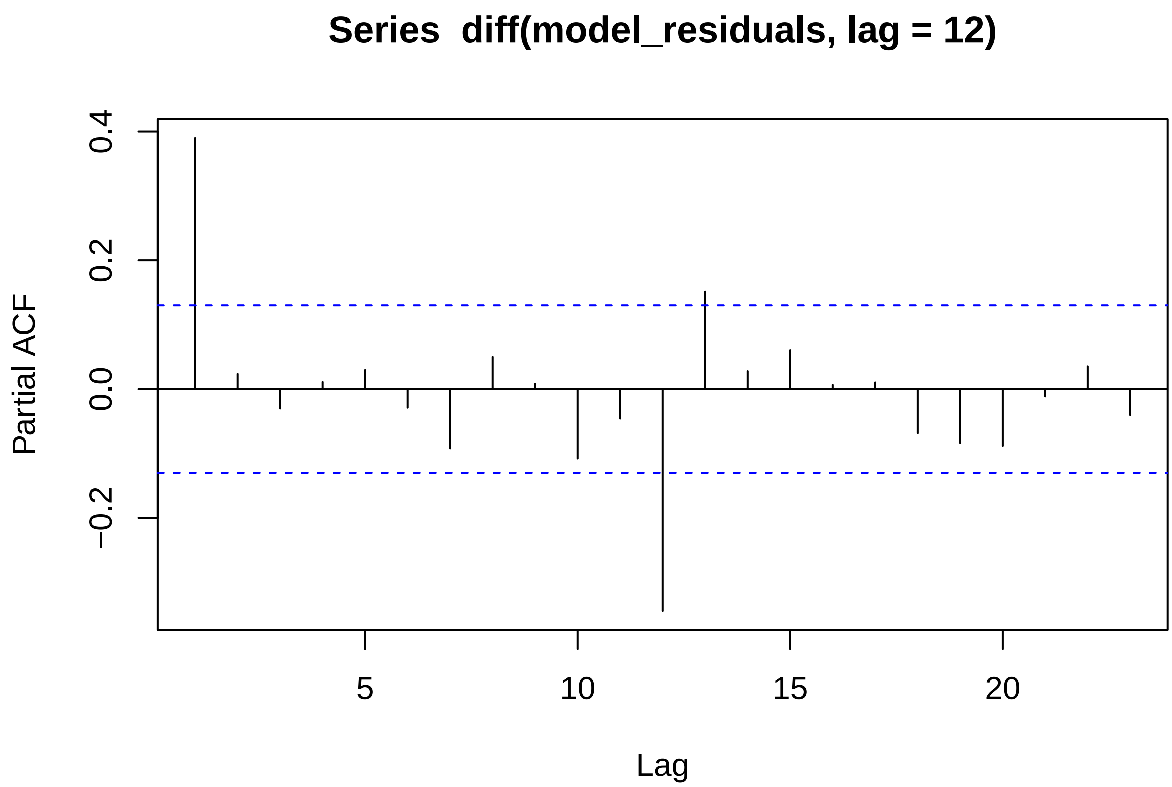

Series diff(model_residuals, lag = 12)
Partial ACF
0.4
0.2
0.0
−0.2
5
10
15
20
Lag

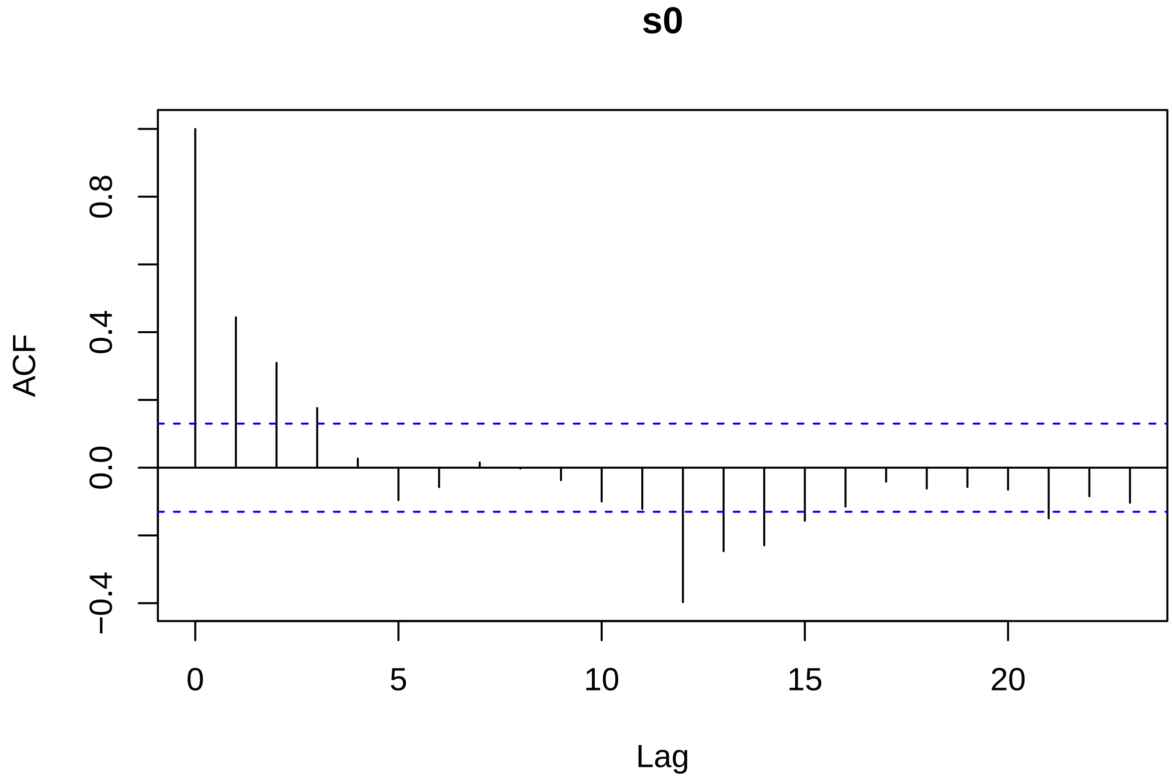

s0
ACF
0.8
0.4
0.0
−0.4
0
5
10
15
20
Lag

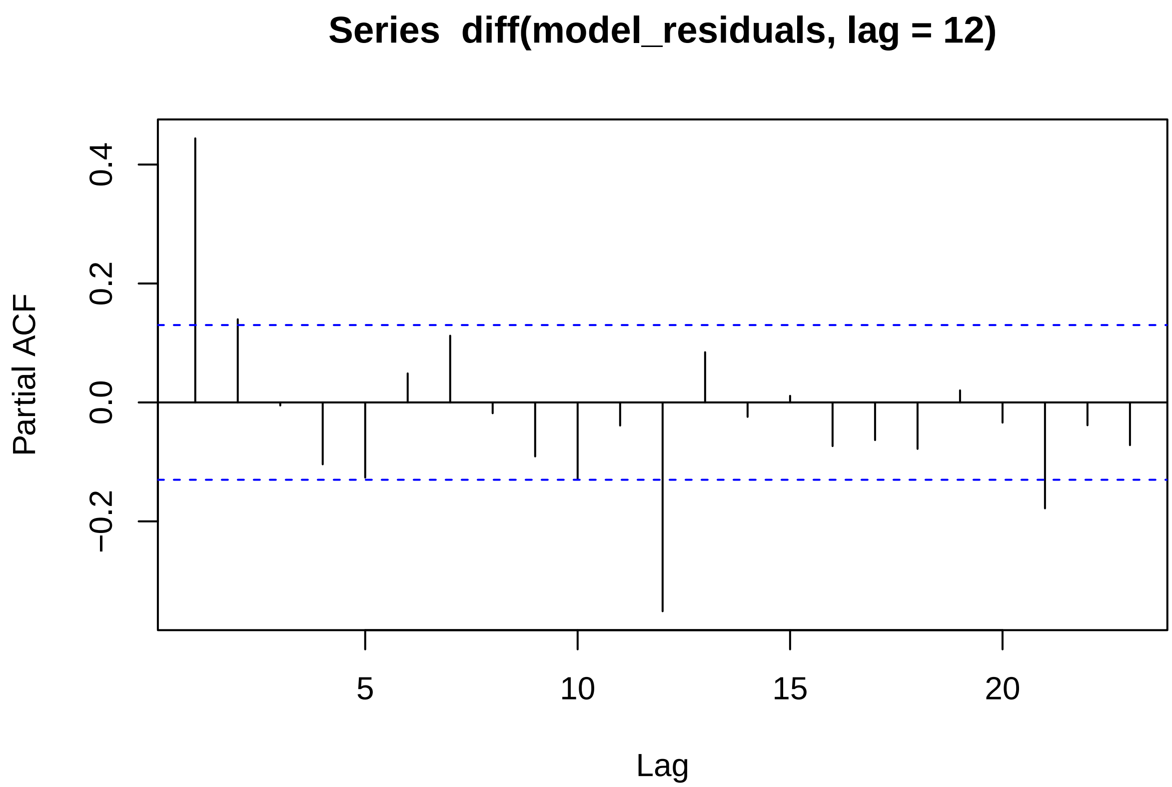

Series diff(model_residuals, lag = 12)
Partial ACF
0.4
0.2
0.0
−0.2
5
10
15
20
Lag

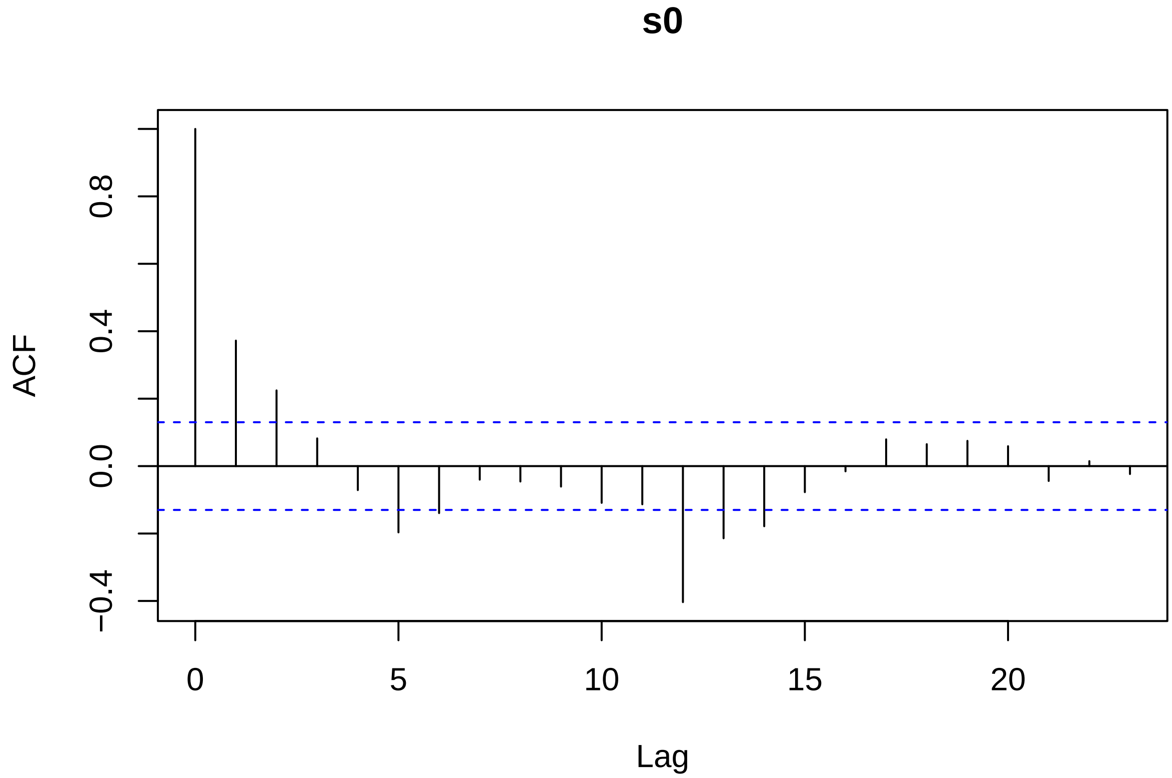


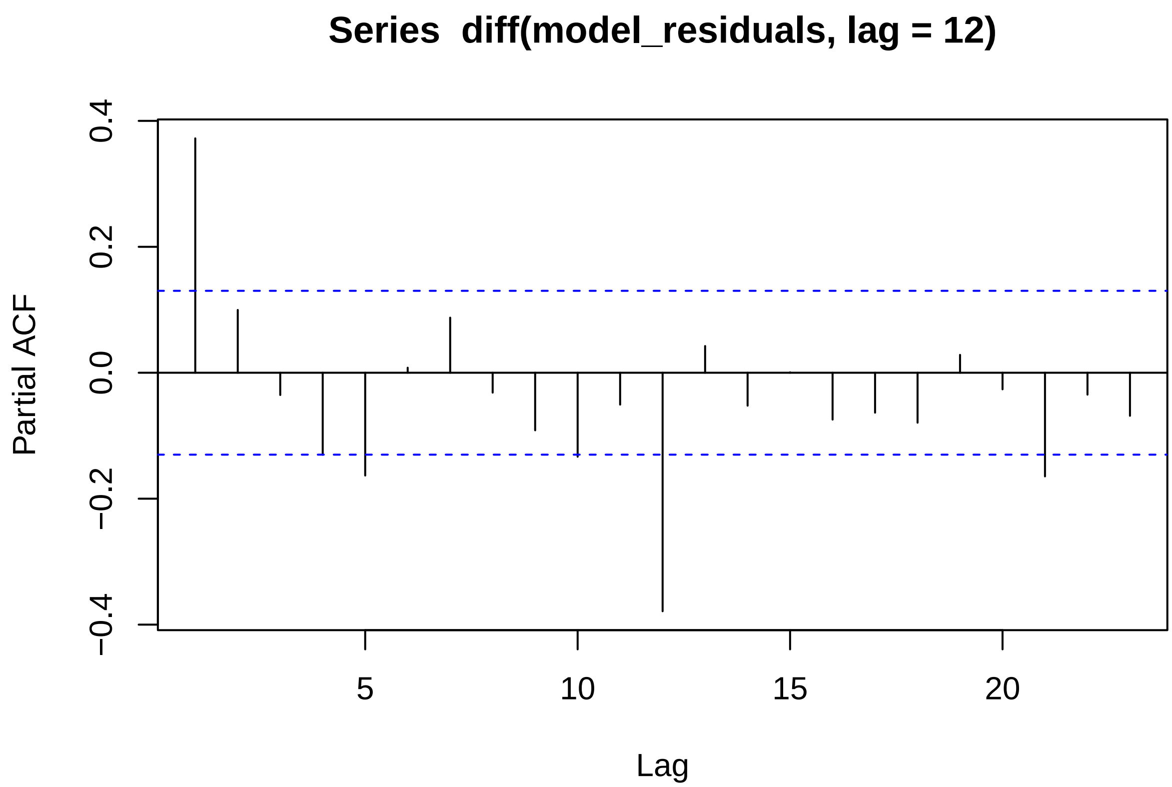
s0
ACF
Lag
0.8
0.4
0.0
−0.4
0
5
10
15
20

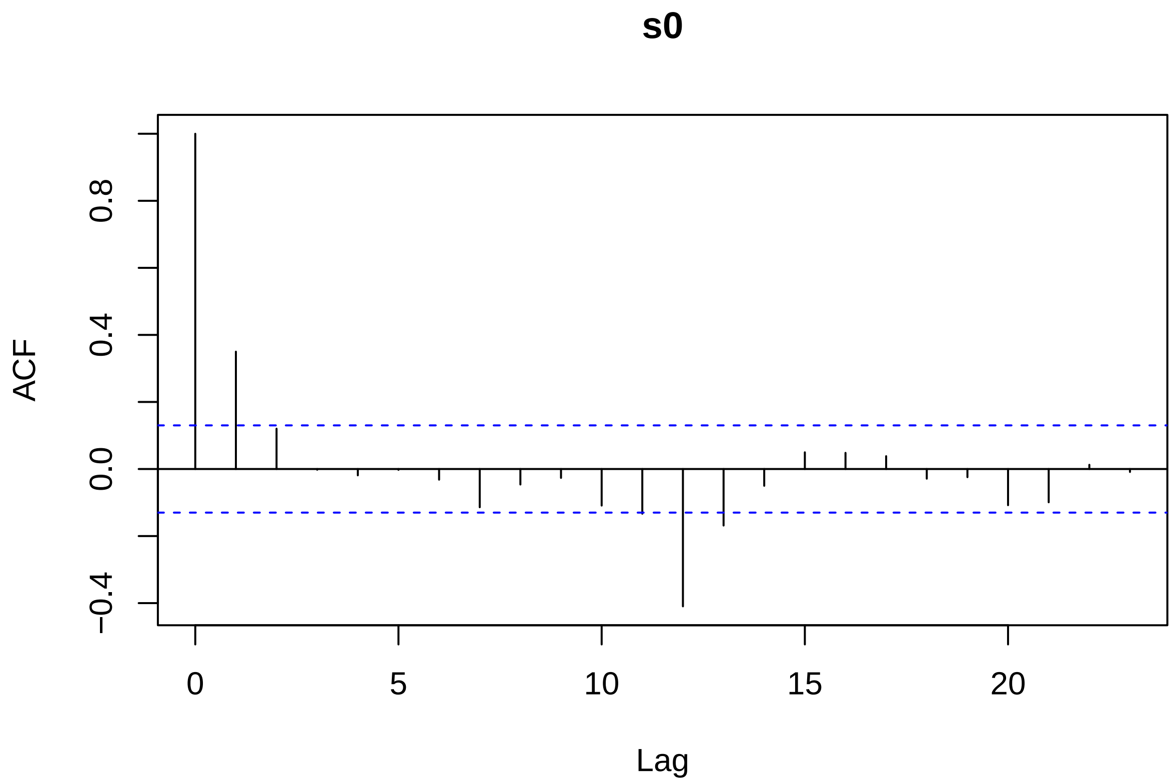

Series diff(model_residuals, lag = 12)
Partial ACF
0.4
0.2
0.0
−0.2
5
10
15
20
Lag

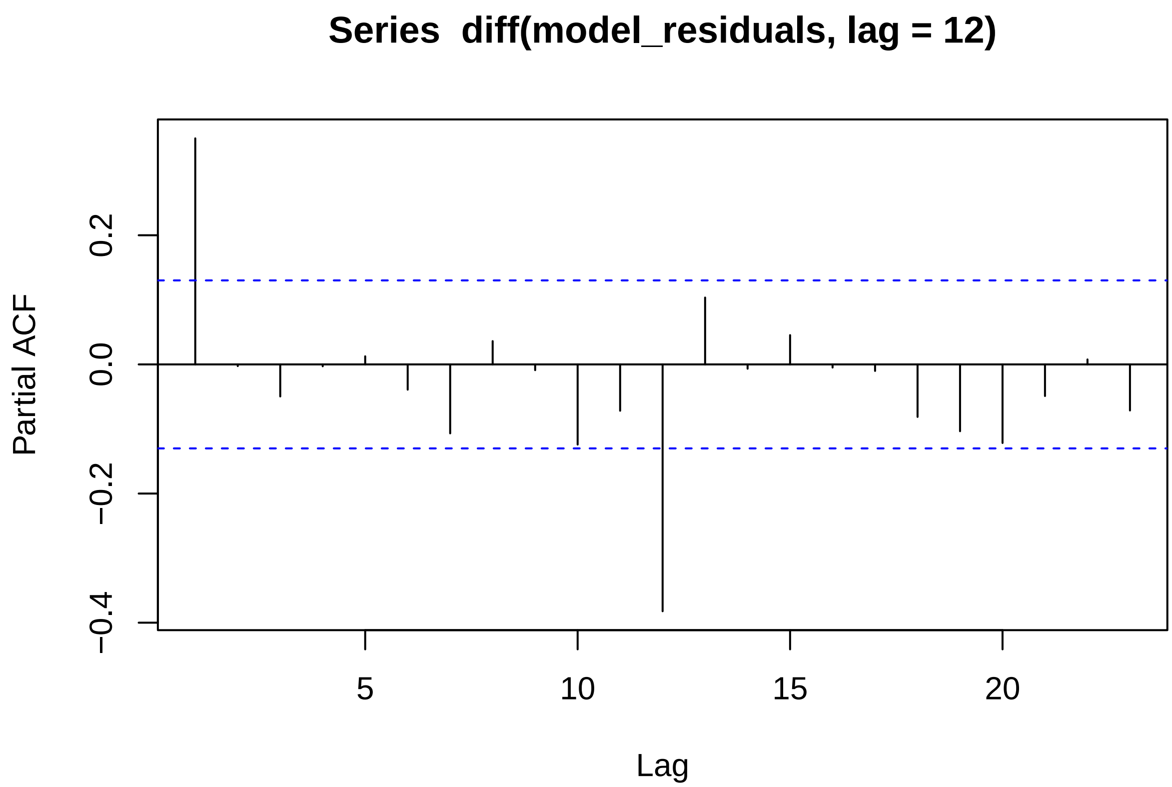

s0
ACF
0.8
0.4
0.0
−0.4
0
5
10
15
20
Lag

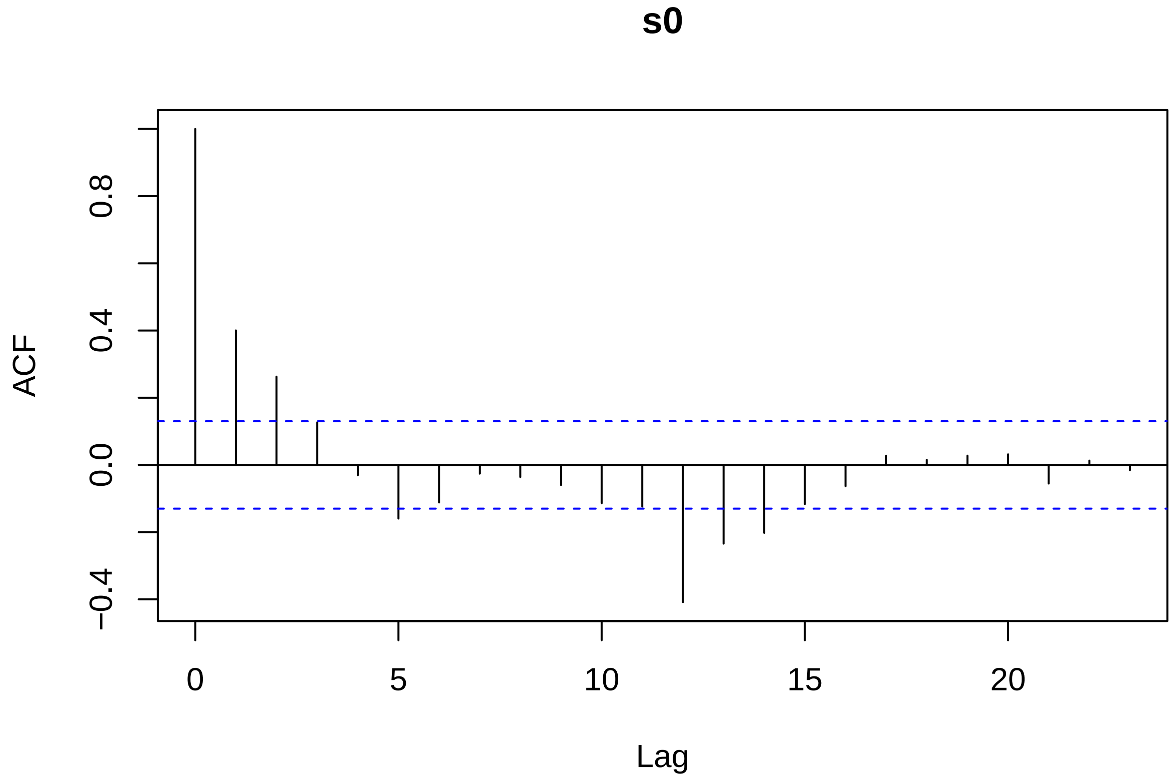

Series diff(model_residuals, lag = 12)
Partial ACF
0.4
0.2
0.0
−0.2
5
10
15
20
Lag

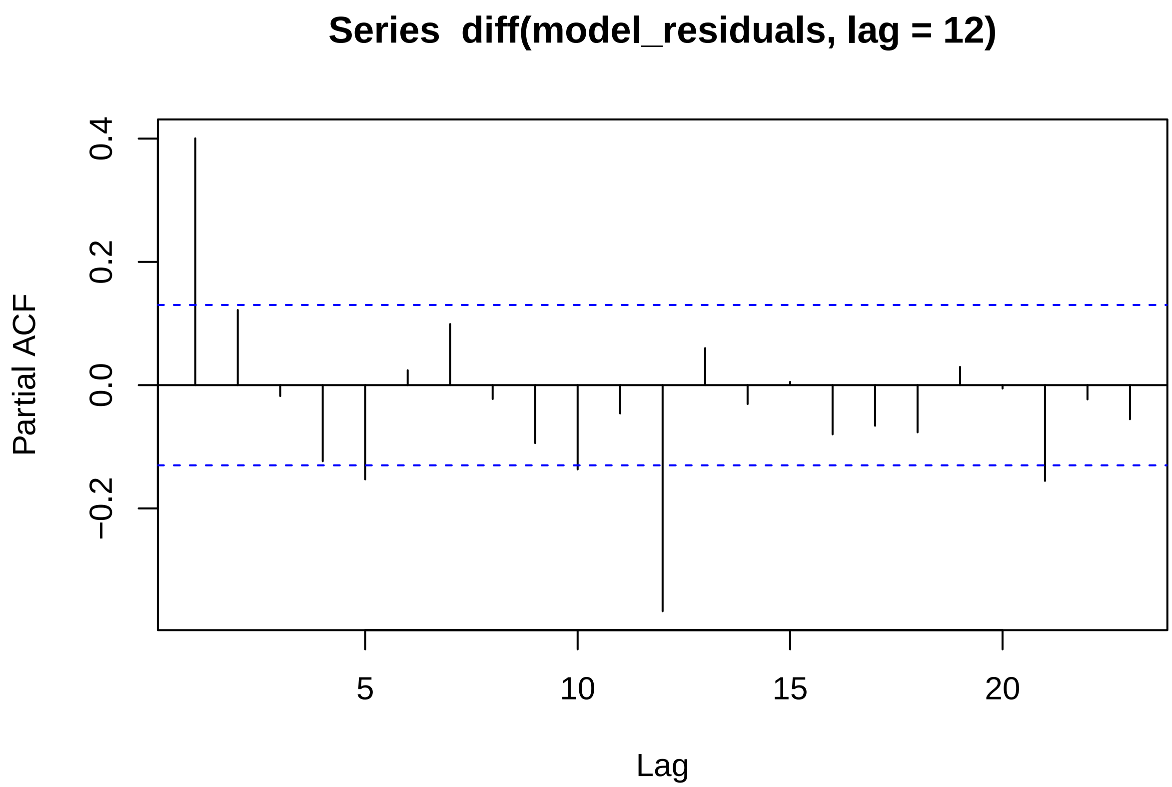

s0
ACF
0.8
0.4
0.0
−0.4
0
5
10
15
20
Lag

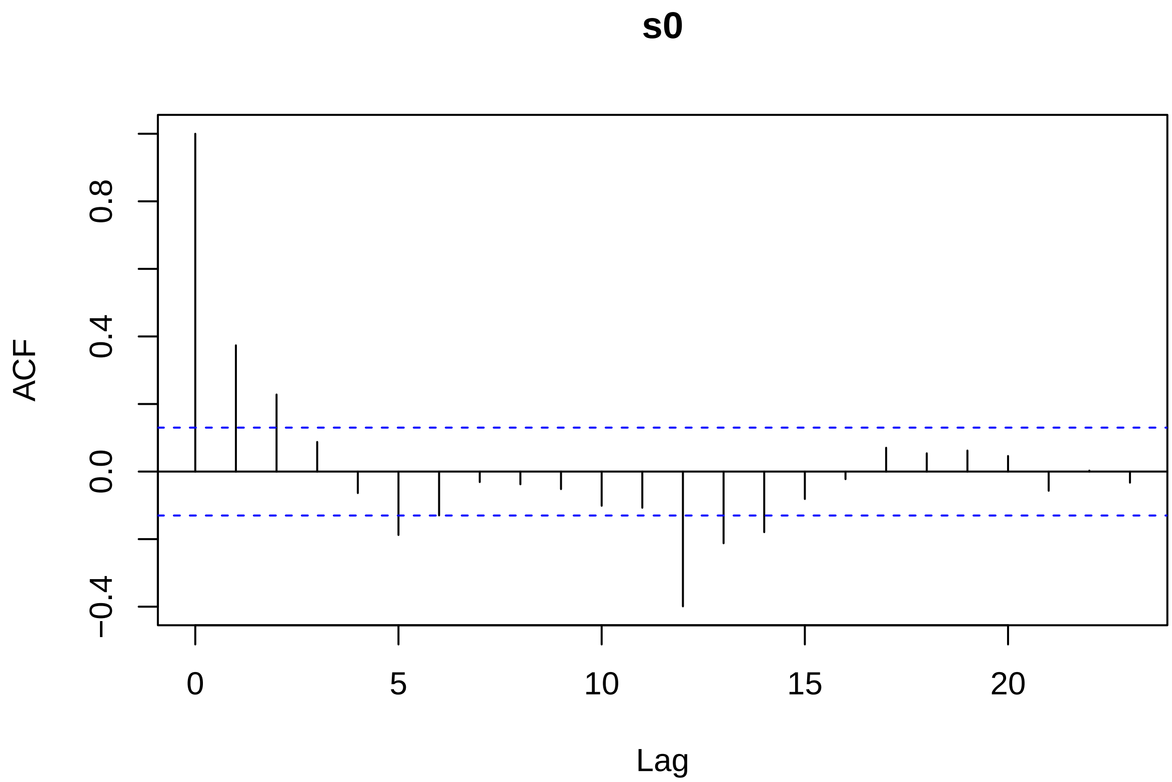

Series diff(model_residuals, lag = 12)
Partial ACF
0.4
0.2
0.0
−0.2
−0.4
5
10
15
20
Lag

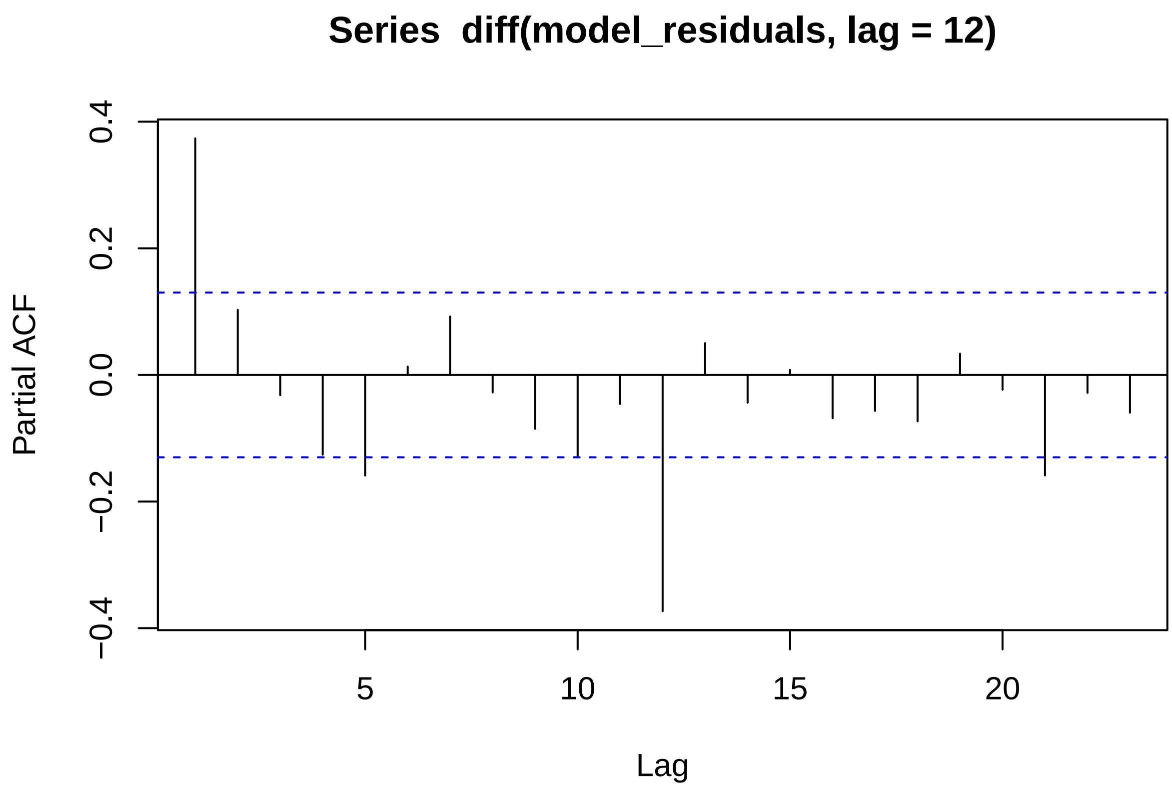

s0
ACF
0.8
0.4
0.0
−0.4
0
5
10
15
20
Lag

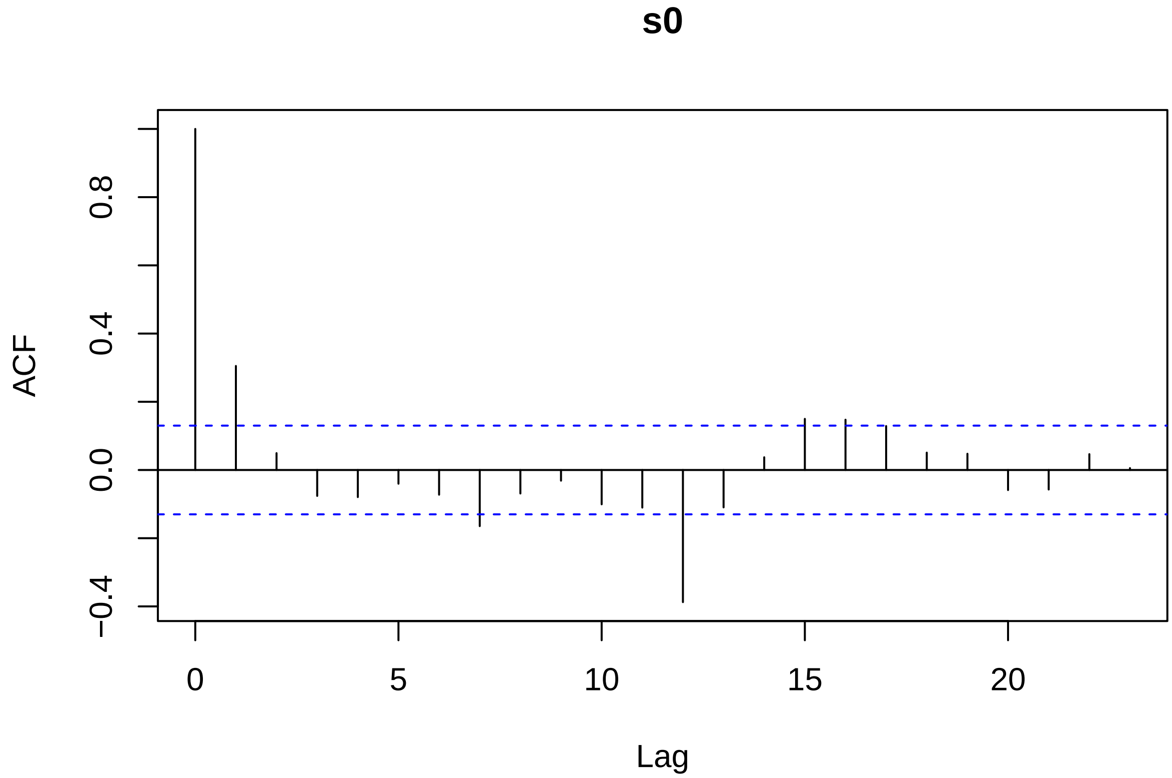

Series diff(model_residuals, lag = 12)
Partial ACF
0.2
0.0
−0.2
−0.4
5
10
15
20
Lag

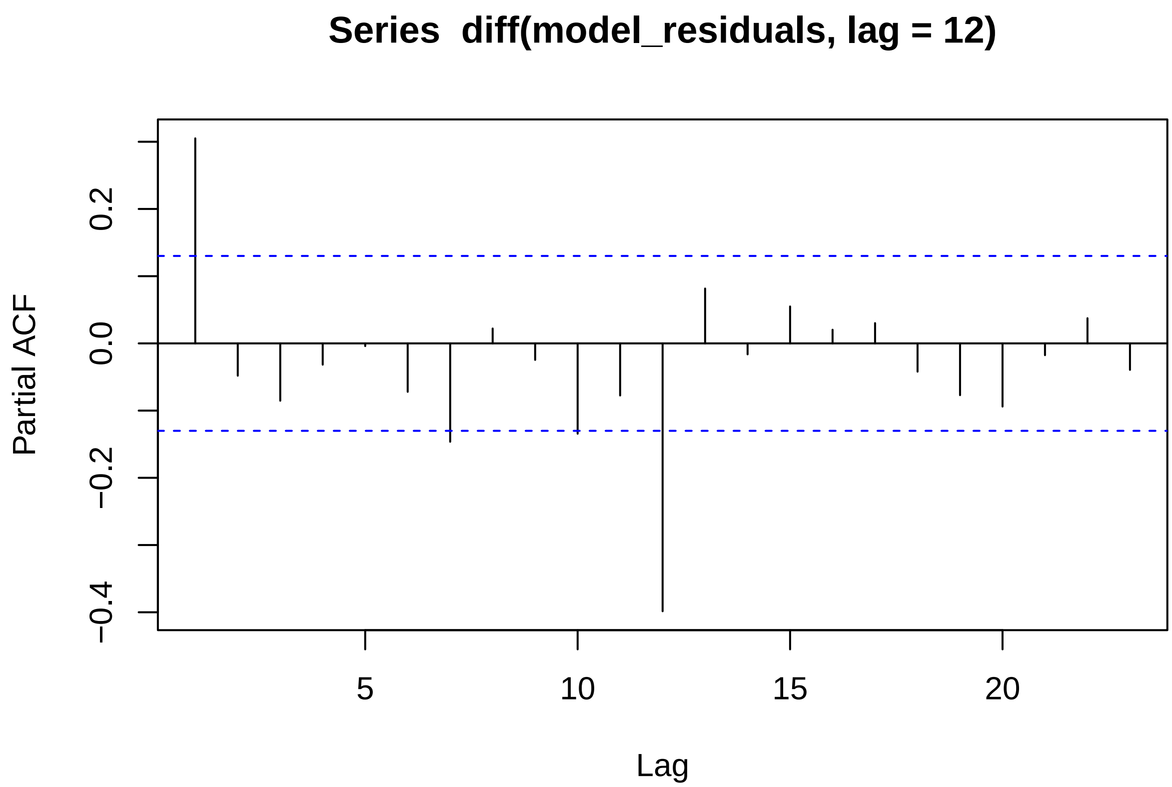

s0
ACF
0.8
0.4
0.0
−0.4
0
5
10
15
20
Lag


Series diff(model_residuals, lag = 12)
Partial ACF
0.4
0.2
0.0
−0.2
5
10
15
20
Lag


s0
ACF
0.8
0.4
0.0
−0.4
0
5
10
15
20
Lag


Series diff(model_residuals, lag = 12)
Partial ACF
0.4
0.2
0.0
−0.2
−0.4
5
10
15
20
Lag


s0
ACF
0.8
0.4
0.0
−0.4
0
5
10
15
20
Lag




```
which(APSE_score2[,-9] == min(APSE_score2[,-9]), arr.ind = TRUE)
```

```
##      row col
## [1,]   5   6
```

So the best model is fitting SARIMA (0,0,1)x(1,1,1)_12 to the elastic net model with time and month and $\alpha = 0$.

```
print(paste(APSE_score1[which.min(APSE_score1$APSE),],best_p[which.min(APSE_score1$APSE)],optim_lambdas
```

```
## [1] "155009840923.835  19  22026.4657948067"
## [2] "alpha_0.5_Time and Month_TRUE  19  22026.4657948067"
```

```
# notice that the best model from Regularization is worse than that of regular regression based on APSE
min(APSE_scores[, -ncol(APSE_scores)])<min(APSE_score1$APSE)
```

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

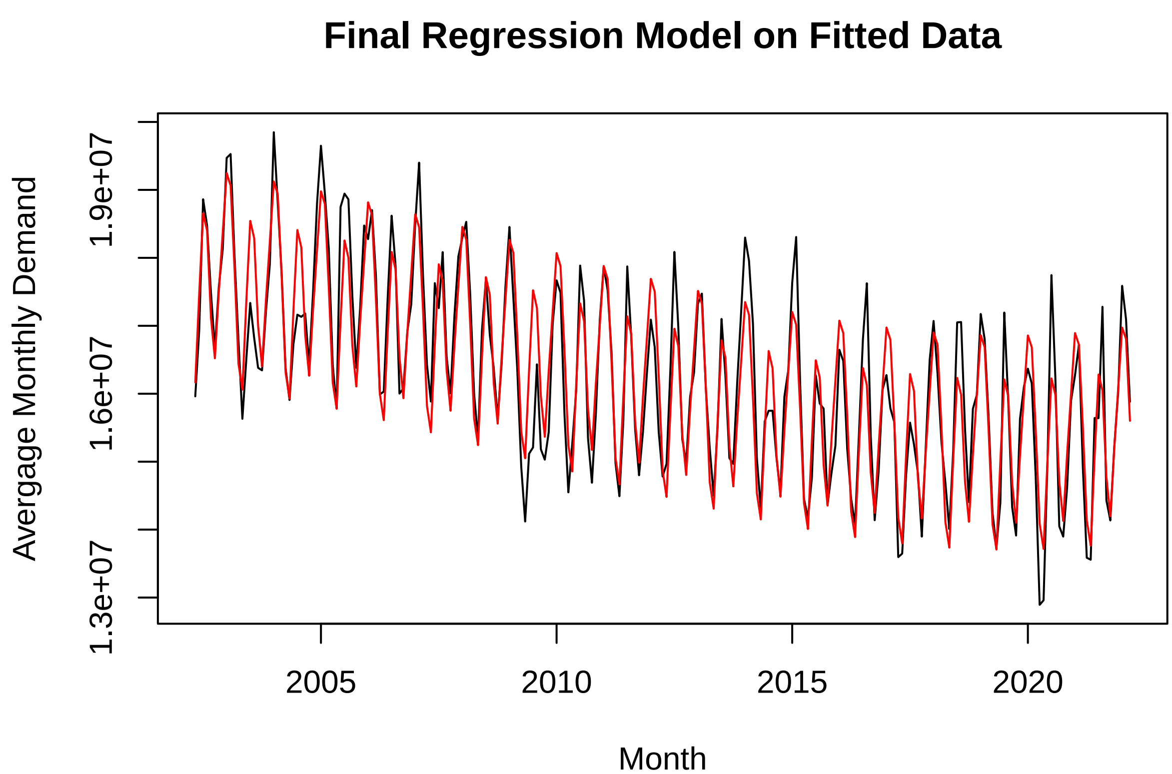
```
#look for the degree and method for the min regular regression
min_index_reg <- which(APSE_scores[, -ncol(APSE_scores)] == min(APSE_scores[, -ncol(APSE_scores)]), arr
min_index_reg
```

```
##      row col
## [1,]   2   3
```

The Best model is regular regression against time and month with degree 3 and with intercept.

Fit Final Model Against Training data

```
#plot the final model against the training data
best_deg1 <- min_index_reg[2]

month <- as.factor(cycle(tim1))

reg_model1 <- lm(train_avg_demand ~ poly(time(tim1),best_deg1) +  month)

fitted1 <- predict(reg_model1,newx =time(tim1),type="response")

plot(y = y_train,x = time(tim1),
     main = 'Final Regression Model on Fitted Data',
     xlab = "Month",
     ylab = "Avergage Monthly Demand", type = 'l')
  points(y = fitted1, x=time(tim1), type = 'l',col = "red")
```

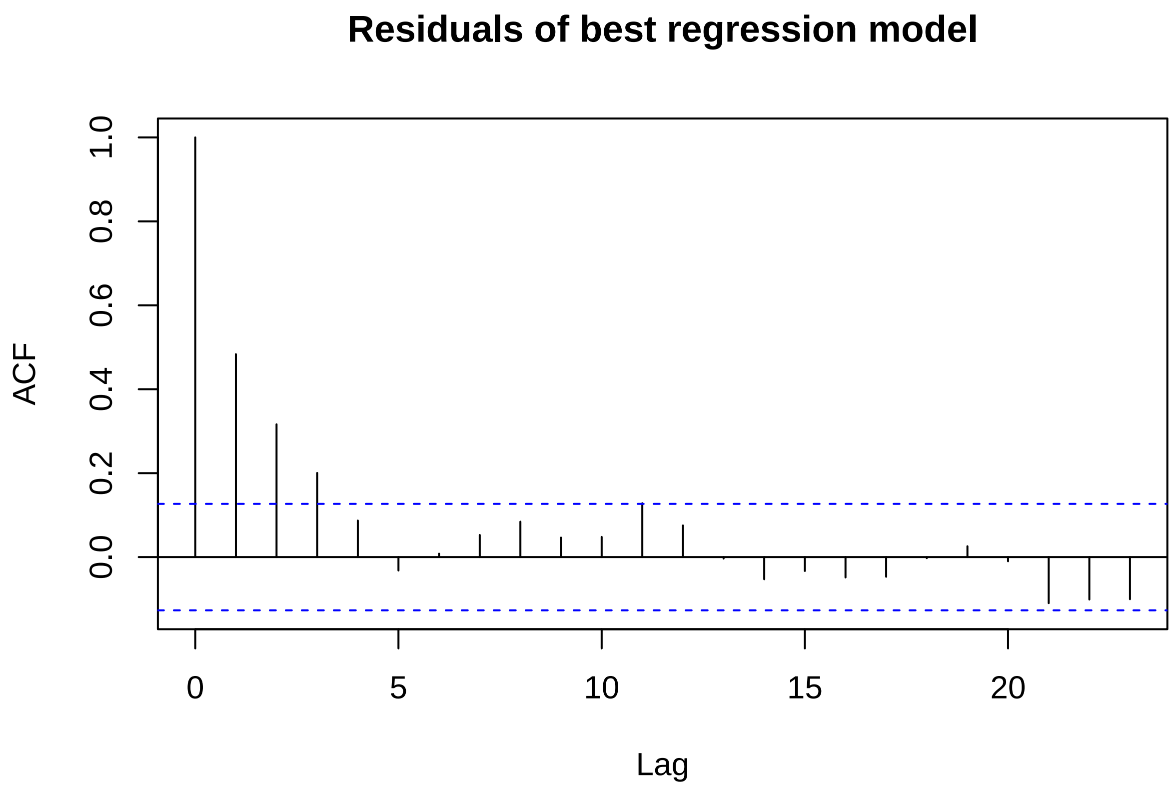



```
month <- as.factor(cycle(window(tim, start = c(2022, 4))))

mean((predict(reg_model1, newdata =
                              data.frame(tim1=time(window(tim, start = c(2022, 4))),
                                         model.matrix(~month-1)))
                    - test_avg_demand)^2)
```

```
## [1] 97227915187
```

ACF of residuals on training

```
acf((y_train-fitted1),main="Residuals of best regression model")
```



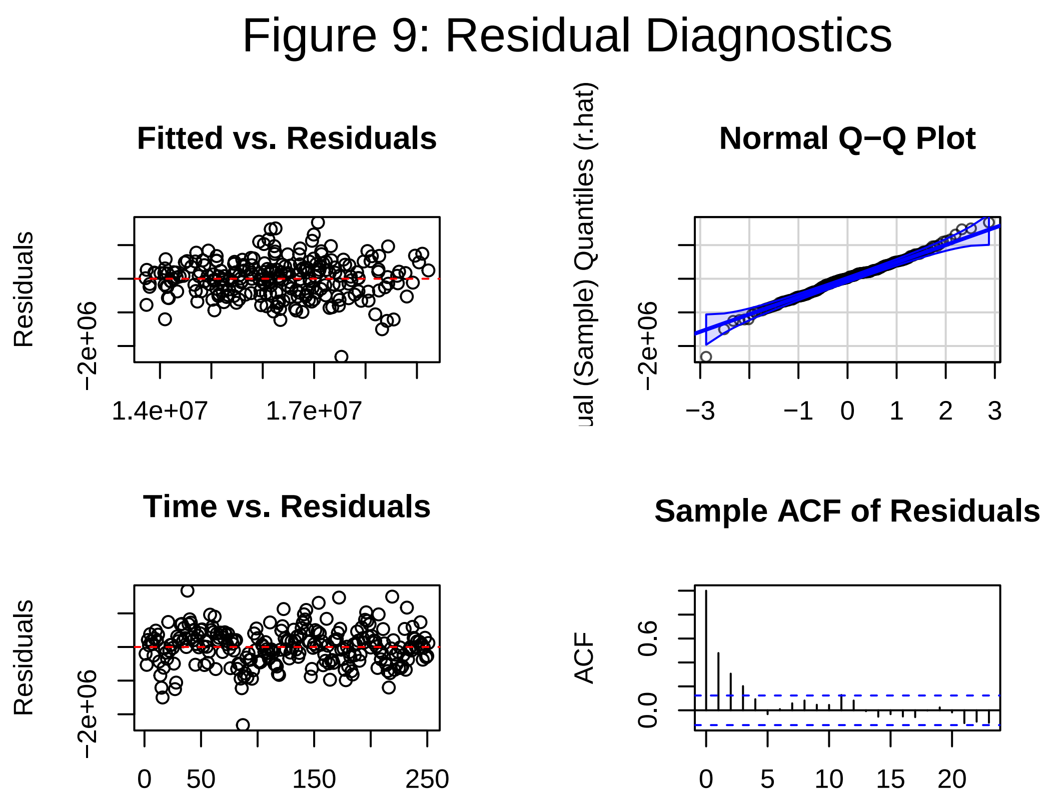
We can see that the residuals seem stationary.

Predicting the future using the best regression model (next year)

```
tim2 <- time(ts(start = 2002 + 5/12, end=2024+3/12,frequency = 12))
X <- poly(tim2, best_deg1)
colnames(X) <- paste0("poly_", 1:best_deg1)


month_f <- as.factor(cycle(window(tim2,end=2023+3/12)))
X_f <- X[1:251,]
y_f <- monthly_avg_demand$avg_demand




reg_model_f <- lm(y_f~X_f+month_f)

fitted <- predict(reg_model_f,newx = X_f,type="response")

#to ensure same name
X_f <- X[252:263,]
```

```r
month_f <- as.factor(cycle(window(tim2, start=2023+4/12)))

prediction1 <- predict(reg_model_f, newdata = data.frame(X_f, model.matrix(~month_f-1)), type = "respons
```

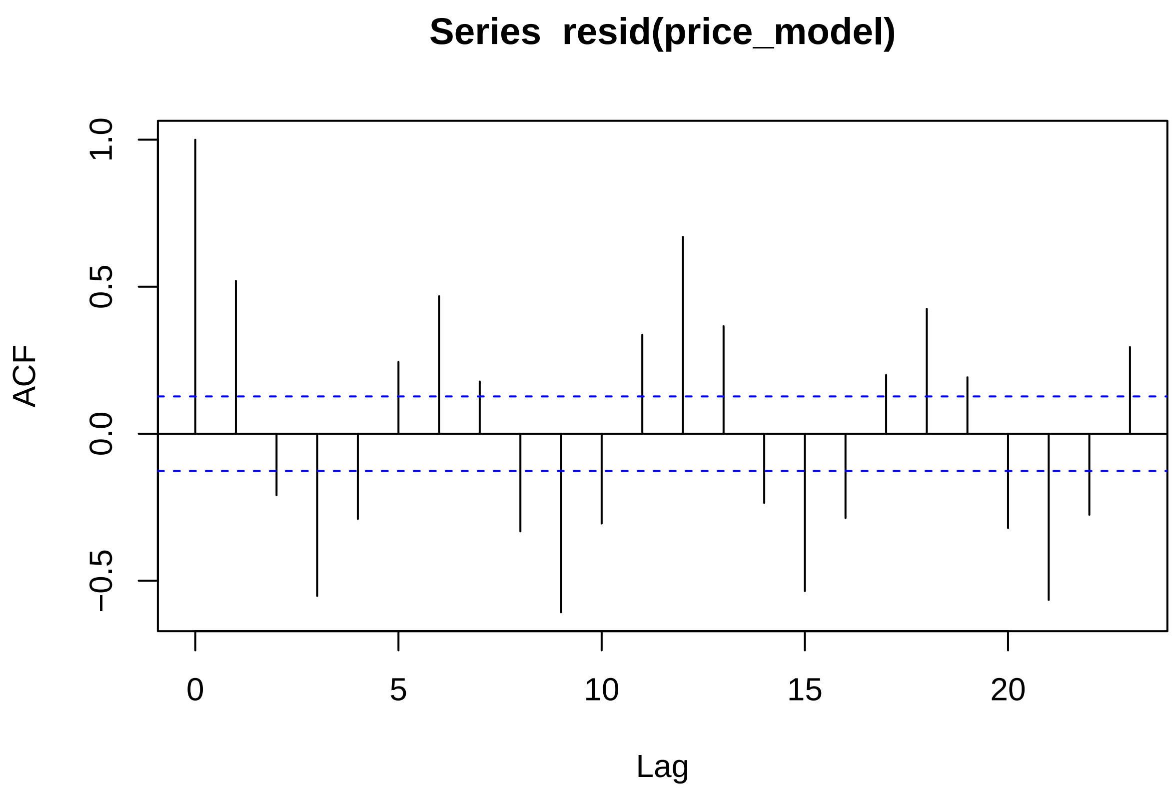
```r
# checking normality assumptions

residuals_f <- monthly_avg_demand$avg_demand-fitted


# Diagnostic plots for reg2 model
par(mfrow = c(2, 2),mar=c(2,4,5,4),oma=c(1,2,3,2)) # Dividing the plotting page into 4 panels
# Plot of fitted values vs residuals
plot(fitted, residuals_f, xlab = "Fitted", ylab = "Residuals",
main = "Fitted vs. Residuals")
abline(h = 0, lty = 2, col = "red") # plotting a horizontal line at 0
# QQ-Plot of residuals:
car::qqPlot(residuals_f, col = adjustcolor("black", 0.7), xlab = "Theoretical Quantiles (Normal)",
ylab = "Residual (Sample) Quantiles (r.hat)", main = "Normal Q-Q Plot",
id = FALSE)
# Plot of residuals versus time
plot(residuals_f, xlab = "Time", ylab = "Residuals", main = "Time vs. Residuals")
abline(h = 0, lty = 2, col = "red") # Plotting a horizontal line at 0
# Sample ACF plot of residuals:
acf(residuals_f, main = "Sample ACF of Residuals")
mtext("Figure 9: Residual Diagnostics", outer = TRUE, cex = 1.5)
```



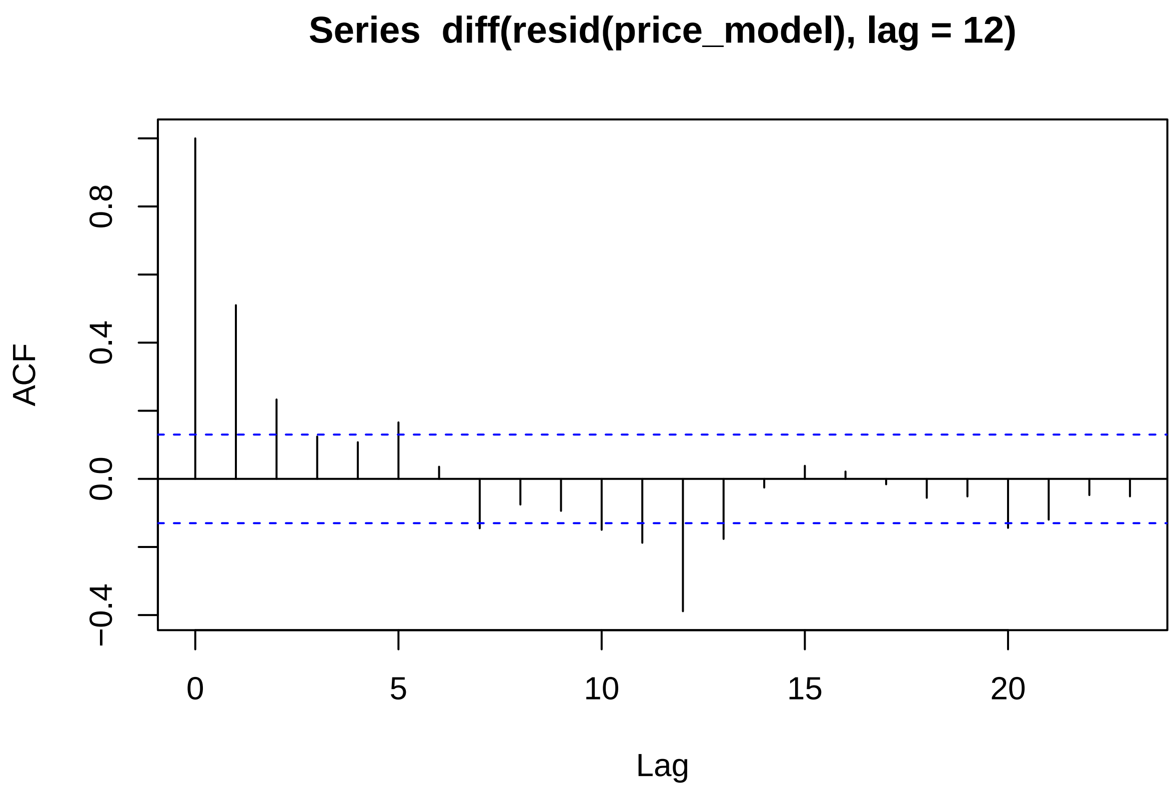
Residuals from the best linear regression model seem normal and stationary.

SARIMA Models on linear model with just price residuals

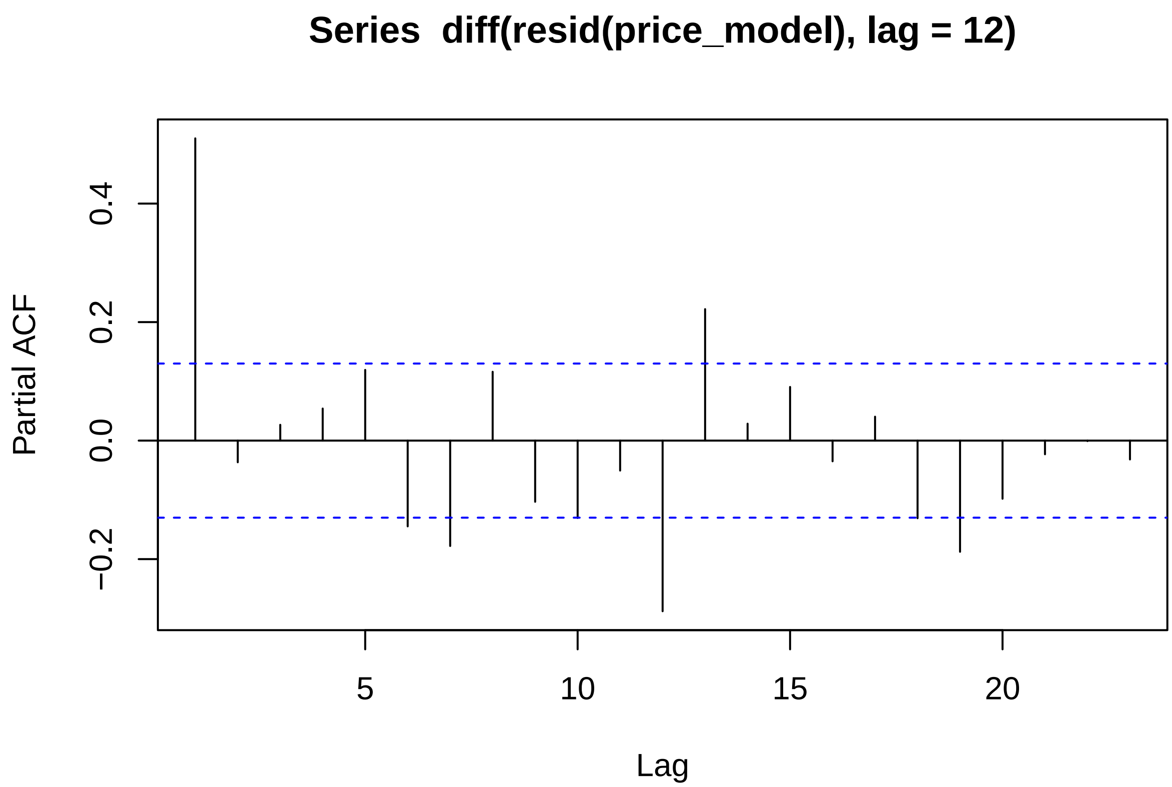
```
acf(resid(price_model))
```




```
acf(diff(resid(price_model),lag = 12))
```




```r
pacf(diff(resid(price_model),lag = 12))
```

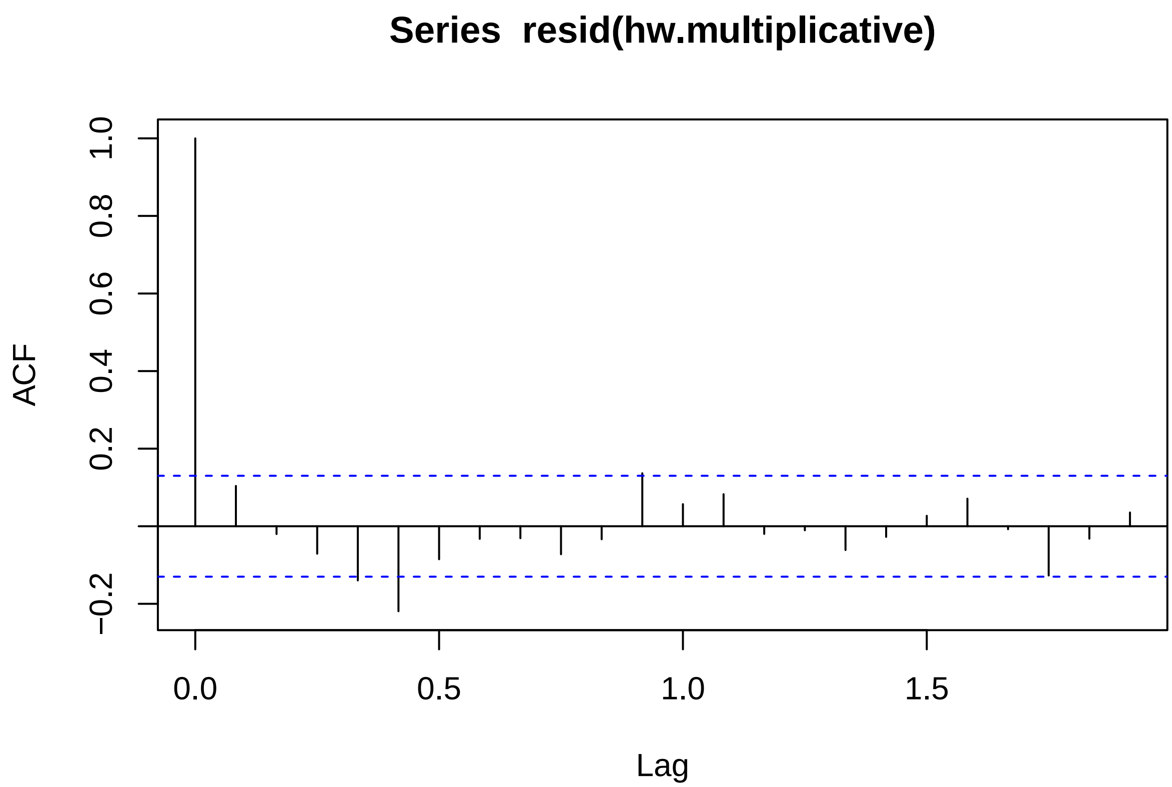



```
forecast <- sarima.for(resid(price_model), n.ahead = 12, plot = FALSE,
           p = 2,d = 0, q = 1, P = 1, D = 1, Q = 1, S = 12)

APSE_lm_sarima <- mean((forecast$pred + cbind(rep(1,12),test$avg_price) %*% price_model$coefficients - t
APSE_lm_sarima
```

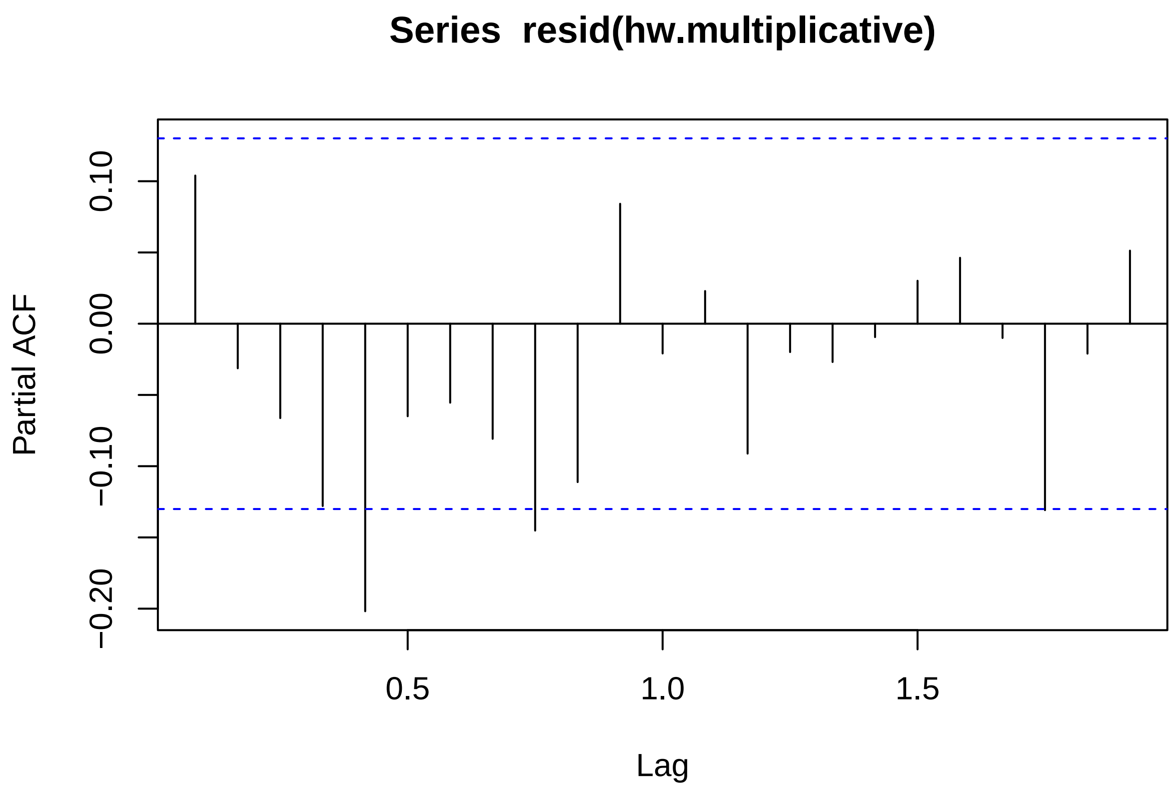
```
## [1] 1.604847e+12
```

SARIMA Models on HW-Residuals

```
acf(resid(hw.multiplicative))
```



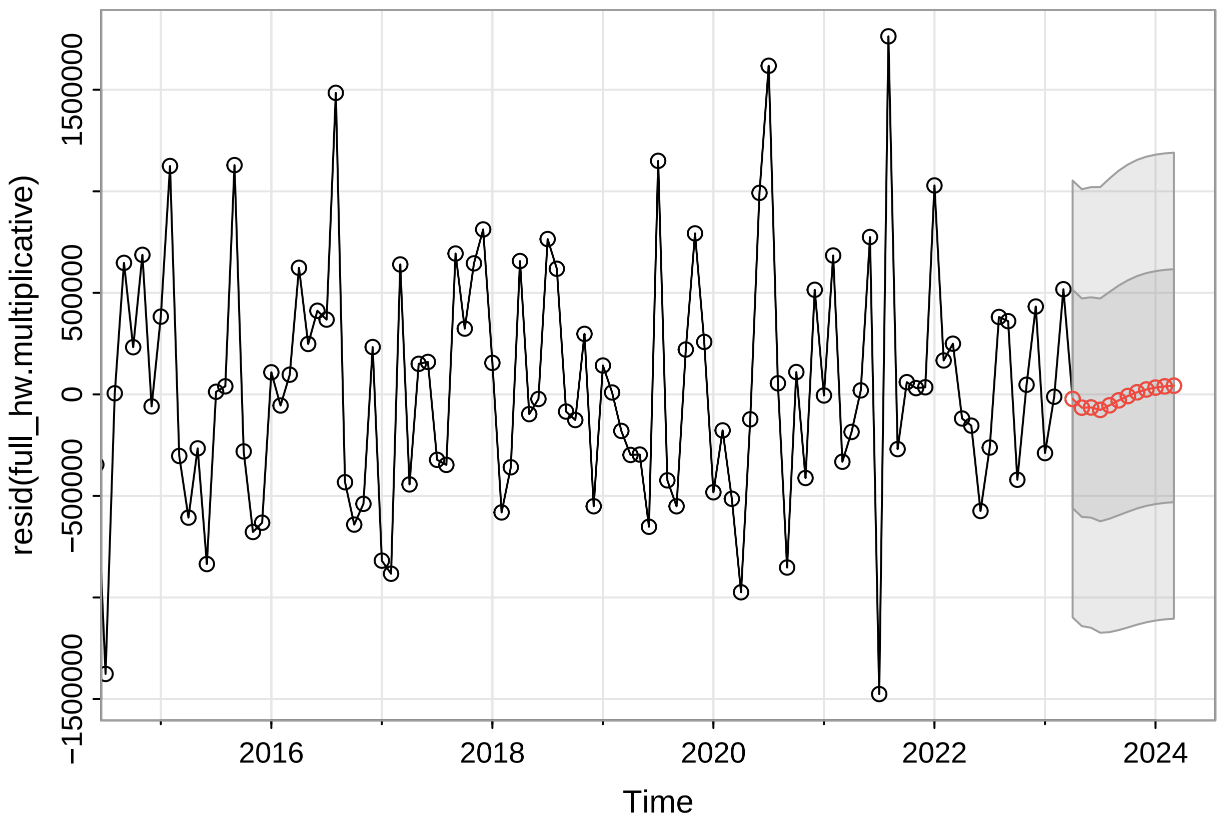

```
pacf(resid(hw.multiplicative))
```




```
forecast <- sarima.for(resid(hw.multiplicative),n.ahead = 12,
                       p = 4,d = 0,q = 2,P= 0,Q=0,D=0, plot = FALSE)
actual_forecast <- forecast$pred + predict(hw.multiplicative,n.ahead = 12)
APSE_hw_sarima <- mean((actual_forecast-test$avg_demand)^2)
APSE_hw_sarima
```

```
## [1] 115129287334
```

```
full_hw.multiplicative <- HoltWinters(monthly_avg_demandTS,
                                      seasonal = "multiplicative")
full_hw_forecast <- predict(full_hw.multiplicative,n.ahead = 12,
                            prediction.interval = TRUE)
full_hw <- fitted(full_hw.multiplicative)[,1]


full_sarima_hw_forecast <- sarima.for(resid(full_hw.multiplicative),
                                      n.ahead = 12,
                       p = 4,d = 0,q = 2,P= 0,Q=0,D=0, plot = TRUE)
```

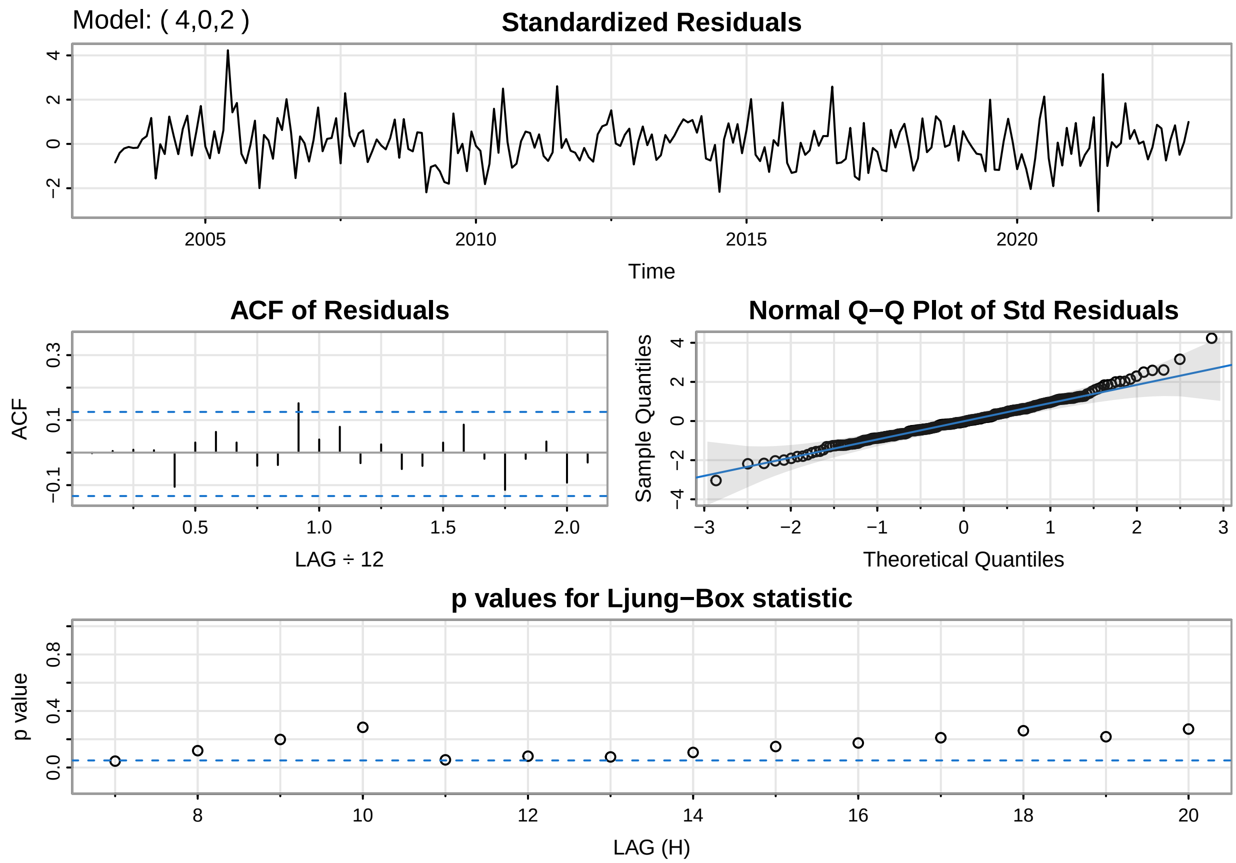



```r
full_sarima_hw <- sarima(resid(full_hw.multiplicative),
                         p = 4,d = 0,q = 2,P= 0,Q=0,D=0)
```

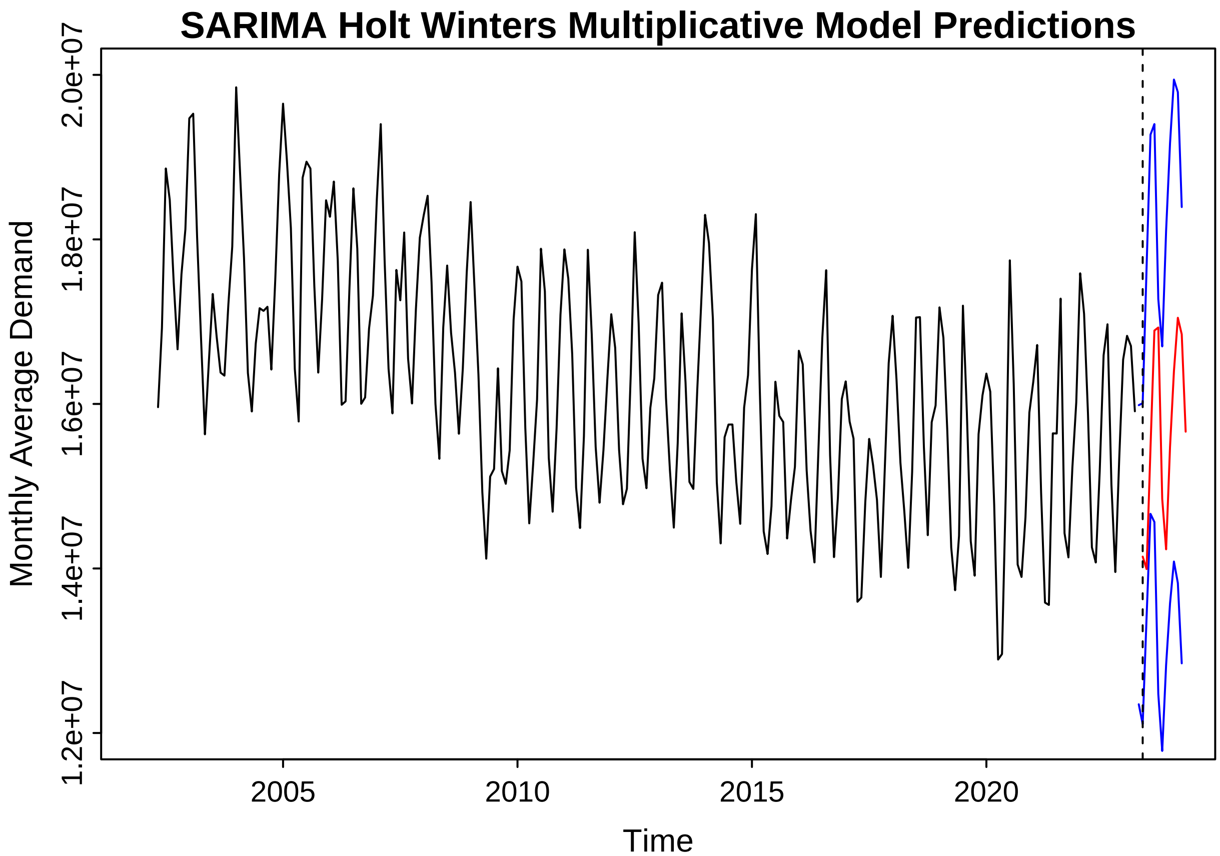
```
## initial  value 13.267122
## iter   2 value 13.259908
## iter   3 value 13.250461
## iter   4 value 13.249658
## iter   5 value 13.248058
## iter   6 value 13.243280
## iter   7 value 13.240770
## iter   8 value 13.238214
## iter   9 value 13.232557
## iter  10 value 13.222692
## iter  11 value 13.222002
## iter  12 value 13.219430
## iter  13 value 13.218237
## iter  14 value 13.217894
## iter  15 value 13.217320
## iter  16 value 13.217206
## iter  17 value 13.217164
## iter  18 value 13.217137
## iter  19 value 13.217017
## iter  20 value 13.216808
## iter  21 value 13.216323
## iter  22 value 13.215868
## iter  23 value 13.215721
```

```
## iter  24 value 13.215574
## iter  25 value 13.215420
## iter  26 value 13.215386
## iter  27 value 13.215312
## iter  28 value 13.215240
## iter  29 value 13.214976
## iter  30 value 13.214420
## iter  31 value 13.214024
## iter  32 value 13.213856
## iter  33 value 13.213719
## iter  34 value 13.213624
## iter  35 value 13.213619
## iter  36 value 13.213619
## iter  36 value 13.213618
## final  value 13.213618
## converged
## initial  value 13.207908
## iter   2 value 13.207877
## iter   3 value 13.207847
## iter   4 value 13.207772
## iter   5 value 13.207688
## iter   6 value 13.207523
## iter   7 value 13.204912
## iter   8 value 13.203193
## iter   9 value 13.201628
## iter  10 value 13.201553
## iter  11 value 13.201382
## iter  12 value 13.201310
## iter  13 value 13.201304
## iter  14 value 13.201303
## iter  15 value 13.201297
## iter  16 value 13.201285
## iter  17 value 13.201261
## iter  18 value 13.201230
## iter  19 value 13.201201
## iter  20 value 13.201190
## iter  21 value 13.201190
## iter  21 value 13.201190
## iter  21 value 13.201190
## final  value 13.201190
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##         Estimate        SE t.value p.value
## ar1       0.9343    0.4952  1.8865  0.0605
## ar2      -0.0695    0.5022 -0.1385  0.8900
## ar3      -0.0296    0.1099 -0.2696  0.7877
## ar4      -0.0601    0.0737 -0.8154  0.4157
## ma1      -0.9332    0.4931 -1.8928  0.0596
## ma2      -0.0668    0.4929 -0.1354  0.8924
## xmean 42965.9749 2314.5447 18.5635  0.0000
##
## sigma^2 estimated as 288123087897 on 232 degrees of freedom
```

```
##
## AIC = 29.3072  AICc = 29.30923  BIC = 29.42357
##
```




```r
future.forecast <- full_hw_forecast[,1] + full_sarima_hw_forecast$pred

plot(y=monthly_avg_demand$avg_demand,x=c(time(tim)),type='l',
     xlab="Time", ylab="Monthly Average Demand",
     main="SARIMA Holt Winters Multiplicative Model Predictions",
      xlim = c(2002,2024), ylim = c(1.2*10^7,2*10^7))


lower <- -1.96*full_sarima_hw_forecast$se + full_hw_forecast[,3]
upper <- +1.96*full_sarima_hw_forecast$se + full_hw_forecast[,2]


lines(lower,col='blue',lty=1)
lines(upper,col='blue',lty=1)
points(y=future.forecast,
       x=window(tim2,start=2023+4/12), type='l',col='red')

abline(v=2023+4/12,lty=2)
```




```r
#find the model with the least APSE

final_apse <- data.frame(
  SARIMA_HW = APSE_hw_sarima,
  Regression = min(APSE_scores[, -ncol(APSE_scores)]),
  Smoothing = min(model_apse_results$APSE),
  SARIMA_NET = min(APSE_score2[,-ncol(APSE_score2)]),
  SARIMA = min(SARIMA_APSE,SARIMA_APSE_PQ,SARIMA_APSE_Q,
               SARIMA_APSE_Q2,SARIMA_APSE_P),
  SARIMA_Price = APSE_lm_sarima
)

final_apse[which.min(final_apse)]
```

```
##    Regression
## 1 97227915187
```

Plotting the Best SARIMA Model

```r
All_SARIMA_APSE <- data.frame(
  SARIMA_HW = APSE_hw_sarima,
  SARIMA_NET = min(APSE_score2[,-ncol(APSE_score2)]),
  SARIMA = min(SARIMA_APSE,SARIMA_APSE_PQ,SARIMA_APSE_Q,
               SARIMA_APSE_Q2,SARIMA_APSE_P),
  SARIMA_Price = APSE_lm_sarima
)
```

```r
All_SARIMA_APSE[which.min(All_SARIMA_APSE)]
```

```
##      SARIMA_HW
## 1 115129287334
```