Project Report

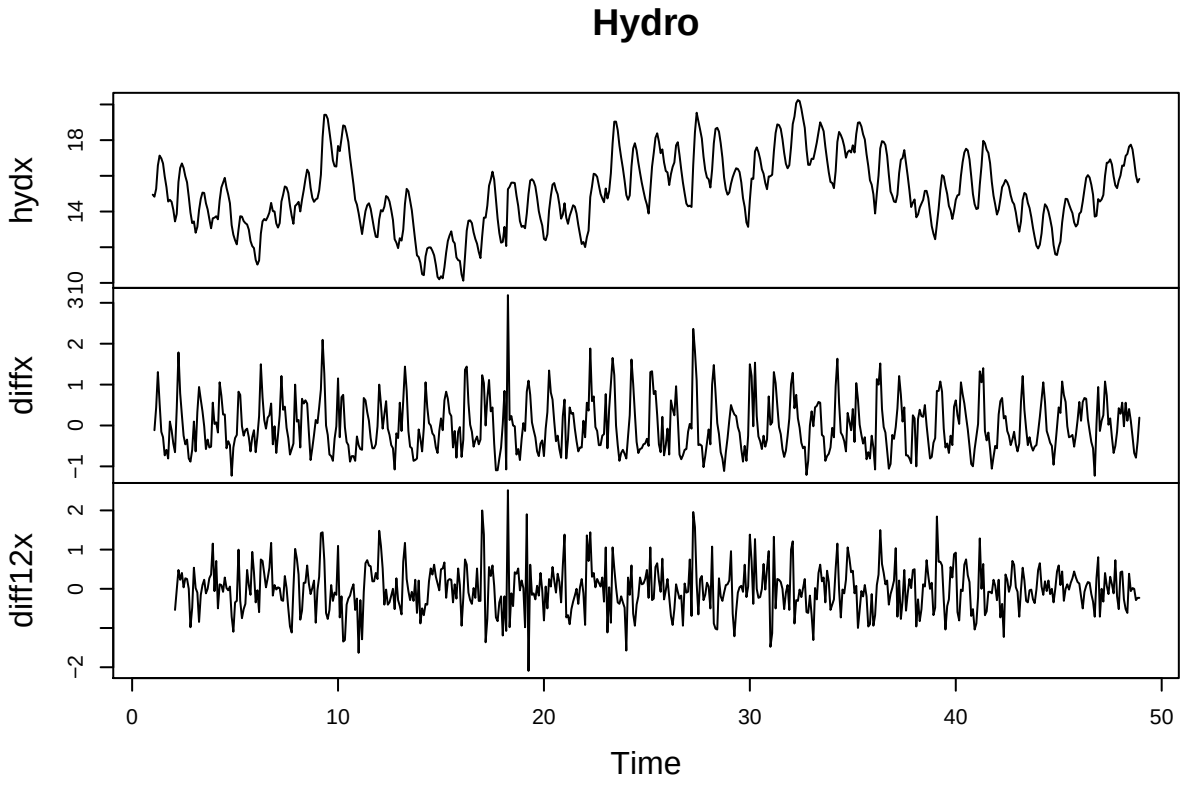
Donya Hamzeian 20852145

4/16/2020

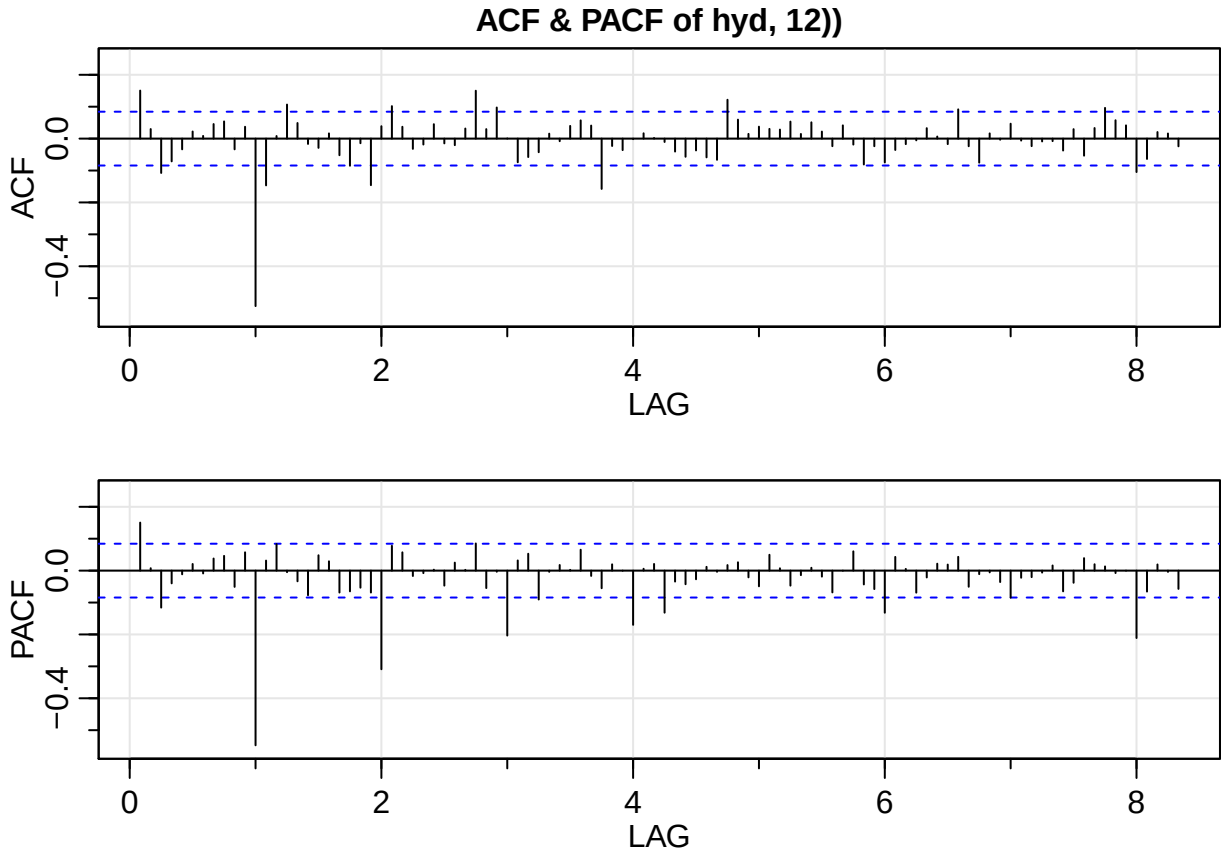
Scenario 1

For this scenario, I have taken lag 1 difference to detrend the series and then taken lag 12 difference of the already differenced series to remove seasonality. This data is a monthly data, so it makes sense for it to have a seasonality of 12. The reason why I did this is because as you can see in the plots below, the acf of the "hyd" shows seasonality or trend (or both) and the acf of the differenced series of "hyd" shows seasonality (because the series has already been detrended). Therefore, I think sarima model is a good choice for this data.

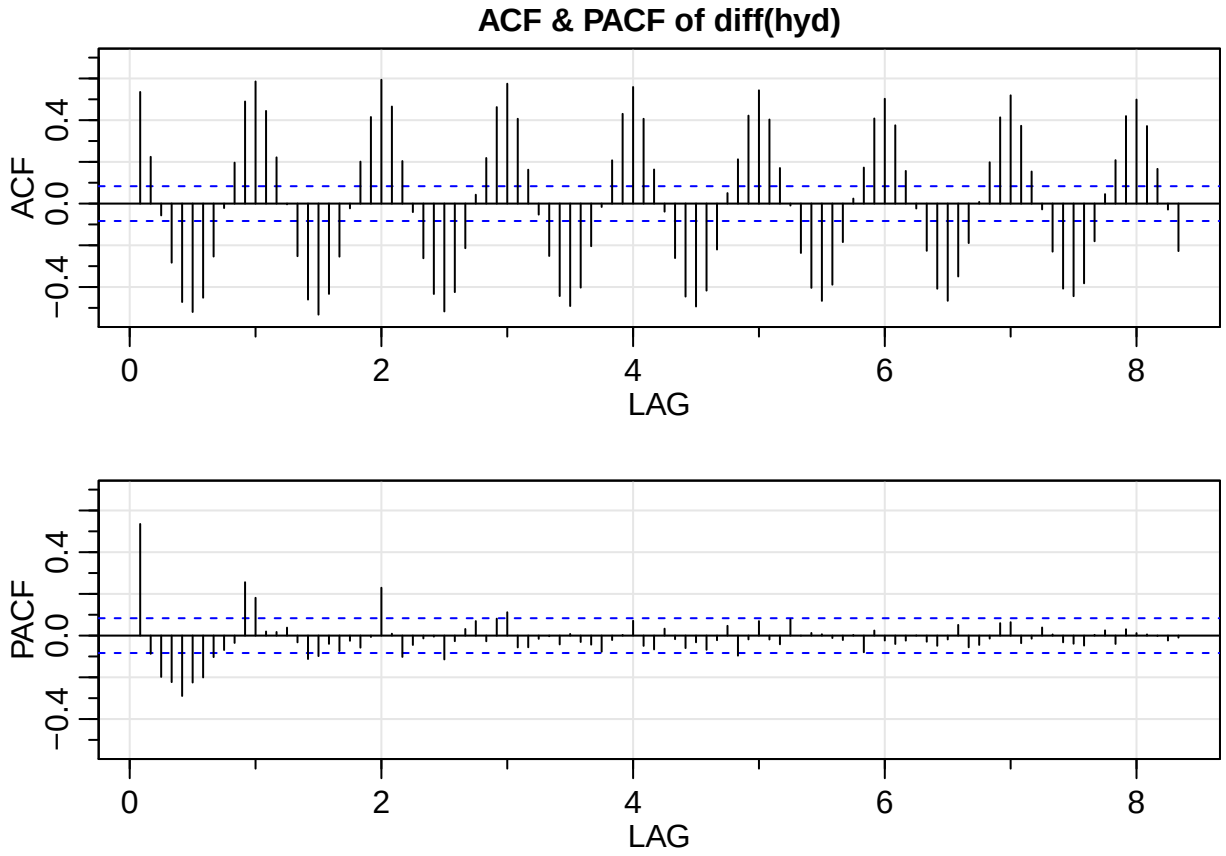
```
hyd = read.csv("hyd_post.txt")
hydx= ts(hyd$x, frequency = 12)
diffx= diff(hydx)
diff12x = diff(diffx, 12)

plot.ts(cbind(hydx, diffx, diff12x), main = "Hydro")
```



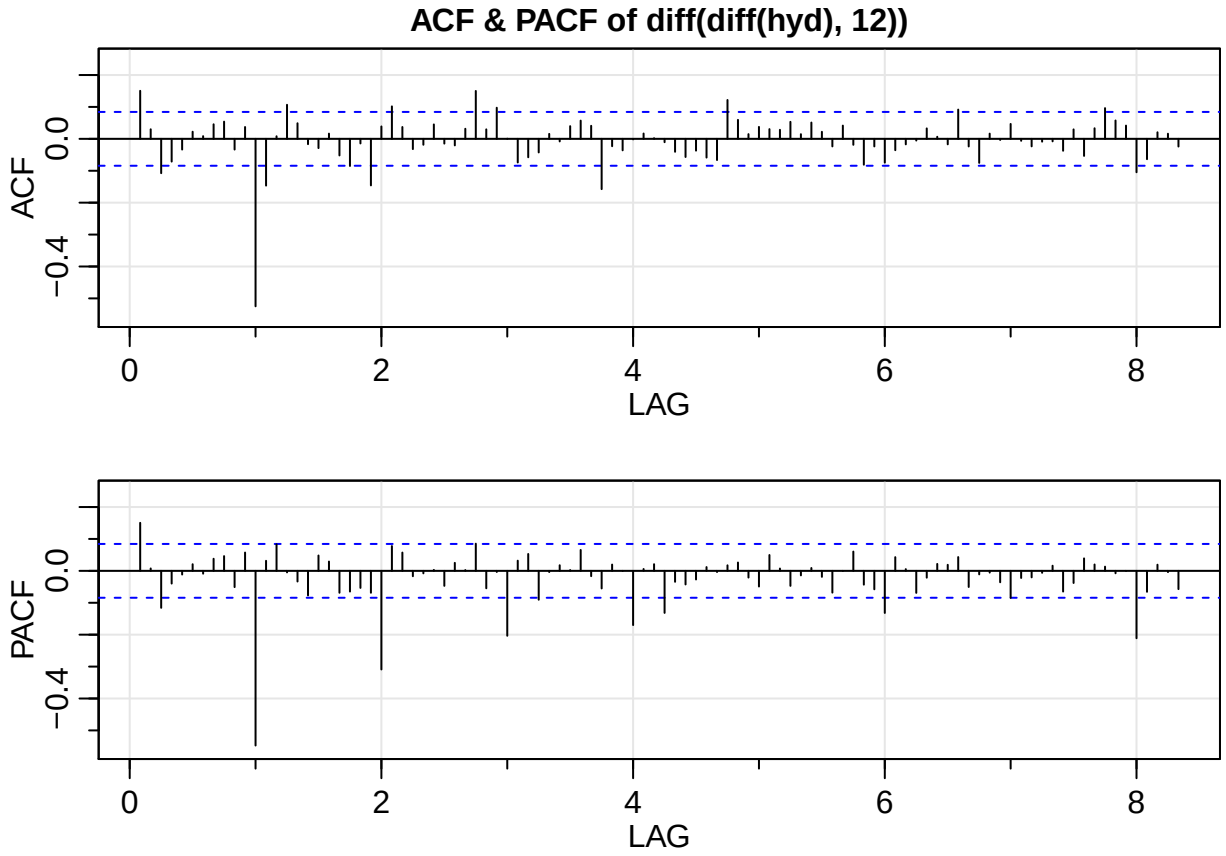
```
acf0 = acf2(diff12x, 100, plot= T, main = "ACF & PACF of hyd, 12))" )
```




```
acf1 = acf2(diffx, 100, plot= T, main = "ACF & PACF of diff(hyd)" )
```




```
acf2 = acf2(diff12x, 100, plot= T, main = "ACF & PACF of diff(diff(hyd), 12))" )
```




```
print("Dickey-Fuller test on the hyd series")
```

```
## [1] "Dickey-Fuller test on the hyd series"
```

```
adf.test(hydx)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  hydx
## Dickey-Fuller = -2.3118, Lag order = 8, p-value = 0.4463
## alternative hypothesis: stationary
```

```
print("Dickey-Fuller test on the diff(diff(hyd),12) series")
```

```
## [1] "Dickey-Fuller test on the diff(diff(hyd),12) series"
```

```
adf.test(diff12x)
```

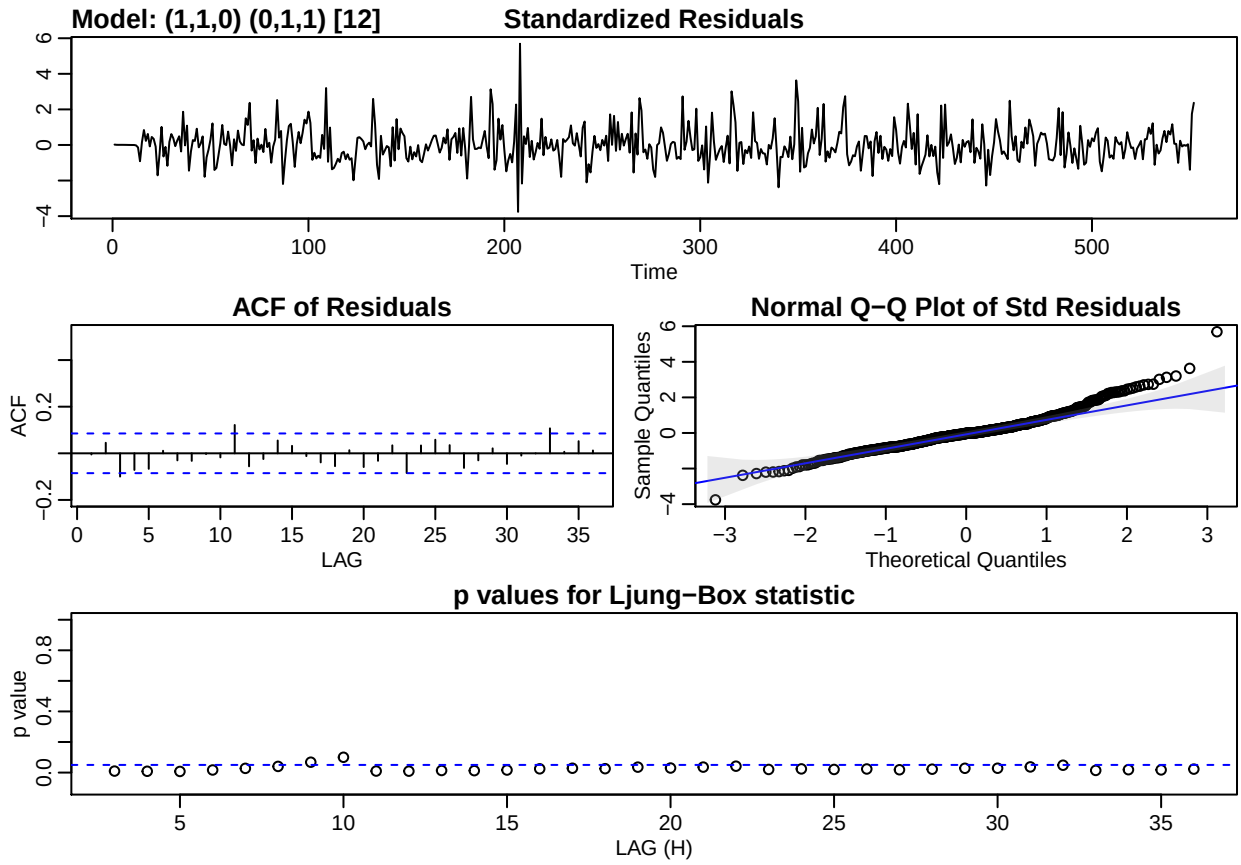
```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff12x
## Dickey-Fuller = -7.4156, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

Based on the acf of the three series, i.e. hyd, diff(hyd), diff(diff(hyd), 12), you can see that the diff(diff(hyd), 12) is quite similar to a stationary process, except for some few lags. Additionally, the Dickey-Fuller test on the hyd series and the diff(diff(hyd), 12) shows that hyd series was not stationary, but after twice differencing(with lag 1 and 12), it is stationary. Therefore, I decided the d, D, and S parameters of sarima to be 1, 1, and 12 respectively. Now, based on the ACF and PACF of diff(diff(hyd),12), we have to find p,q, P, and Q of the sarima model. The ACF cuts off at lag 1, but the PACF on the seasonality lags does not cut off at any specific lag and it trails off which shows MA for seasonality. Also, the PACF and ACF of the non-seasonality lags shows trail-off not cut-off at any specific lags, which is ARMA. However, the choice of p, q, P, and Q should also be based on MSE, i.e. these parameters should be selected in order to minimize the MSE on the test set. Therefore, I splitted my data into 2 parts- dtrain(the first 552 observations) and dtest(the last 24 observations)- in order to calculate MSE. I tested different values of p,q, P, and Q, and the least MSE as well as significant coefficients that I obtained on dtest is 8.92 with these parameters: 1,0, 0, 1.

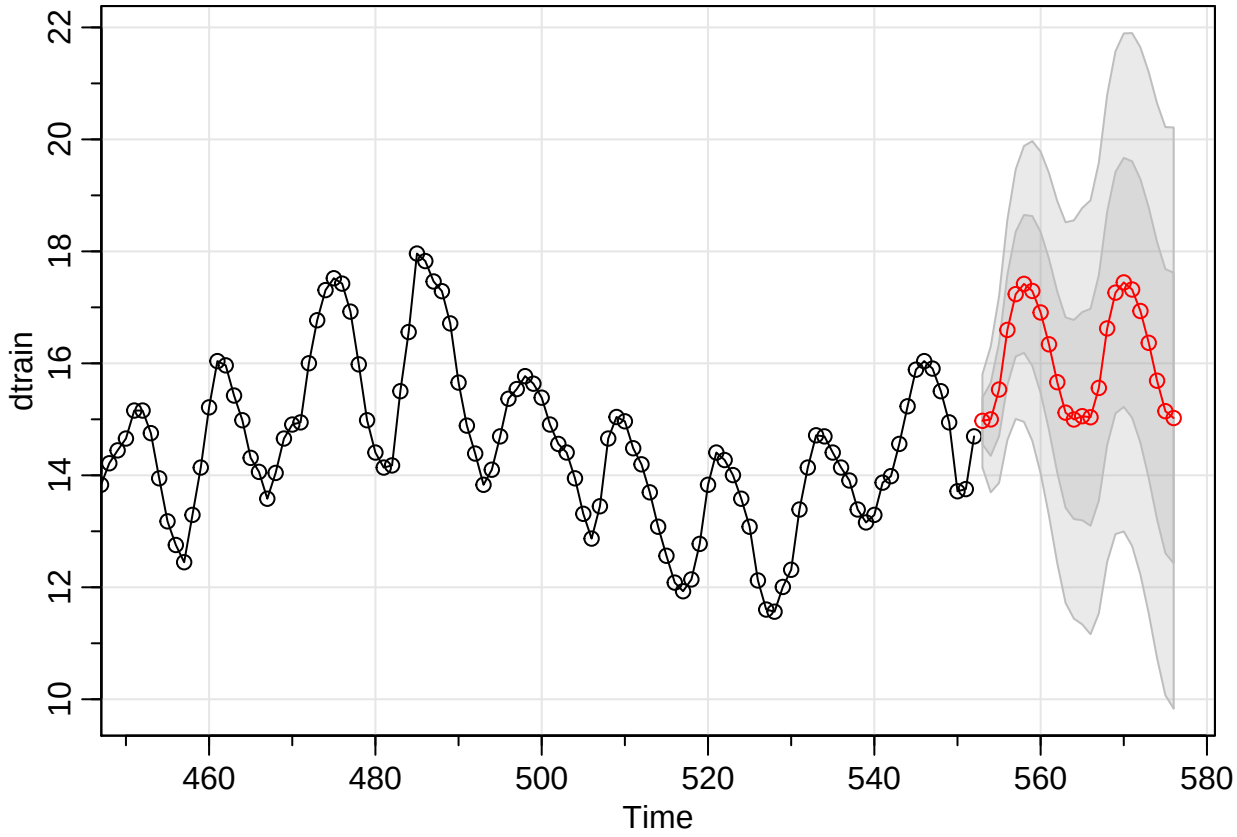
Evaluate the model on test data

```
ntrain= nrow(hyd)-24
dtrain = hyd[1:ntrain,"x" ]
dtest = hyd[(ntrain+1):nrow(hyd),"x" ]
sar= sarima(dtrain, 1,1,0,0,1,1,12 )
```

```
## initial  value -0.515325
## iter   2 value -0.750244
## iter   3 value -0.798175
## iter   4 value -0.830820
## iter   5 value -0.835429
## iter   6 value -0.835693
## iter   7 value -0.835875
## iter   8 value -0.835877
## iter   9 value -0.835877
## iter   9 value -0.835877
## iter   9 value -0.835877
## final  value -0.835877
## converged
## initial  value -0.842951
## iter   2 value -0.849553
## iter   3 value -0.849566
## iter   4 value -0.849600
## iter   5 value -0.849601
## iter   5 value -0.849601
## iter   5 value -0.849601
## final  value -0.849601
## converged
```




```
pred = sarima.for(dtrain, 24, 1,1,0,0,1,1,12 )
```



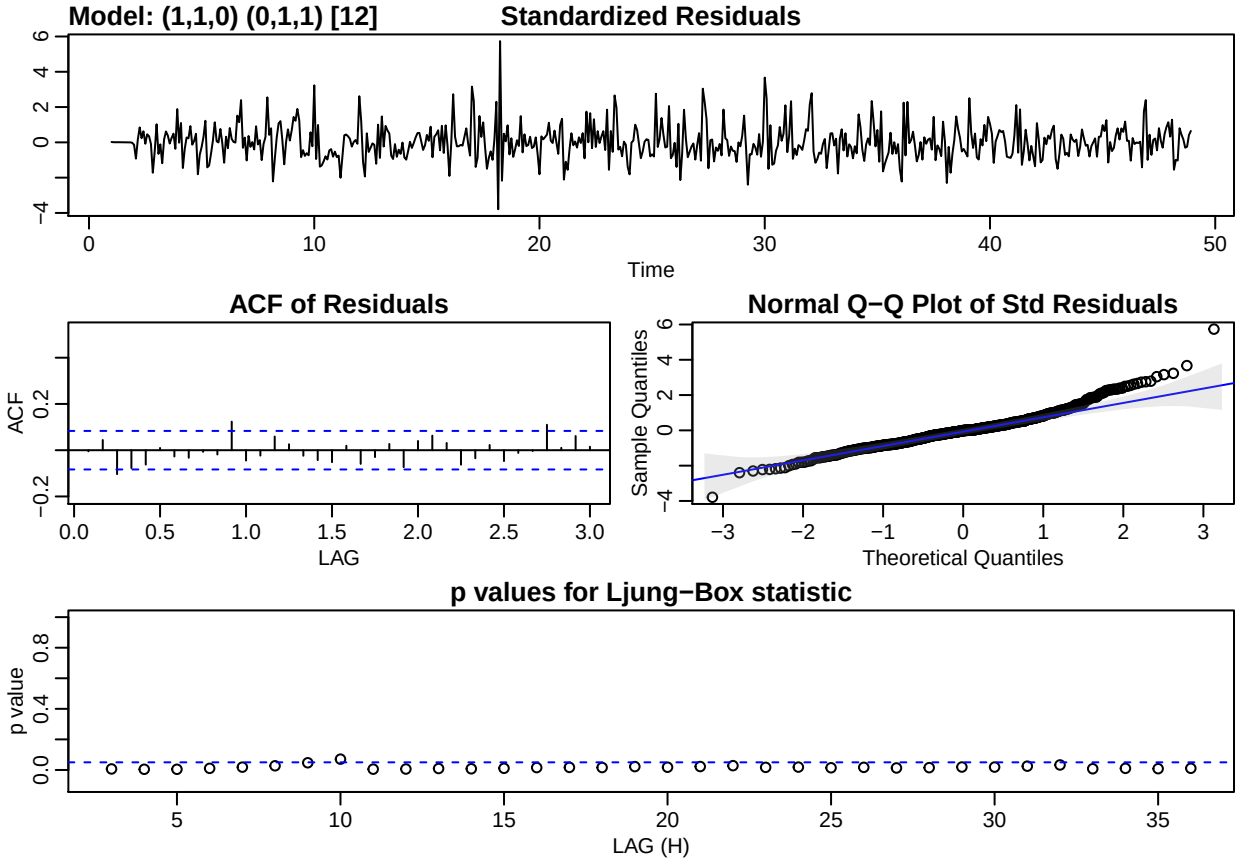

```
MSE_test = sum((pred$pred-dtest)^2)
```

Now that I have selected the best sarima model, I am going to fit this model on the whole series and use sarima.for function to get 24 steps ahead predictions.

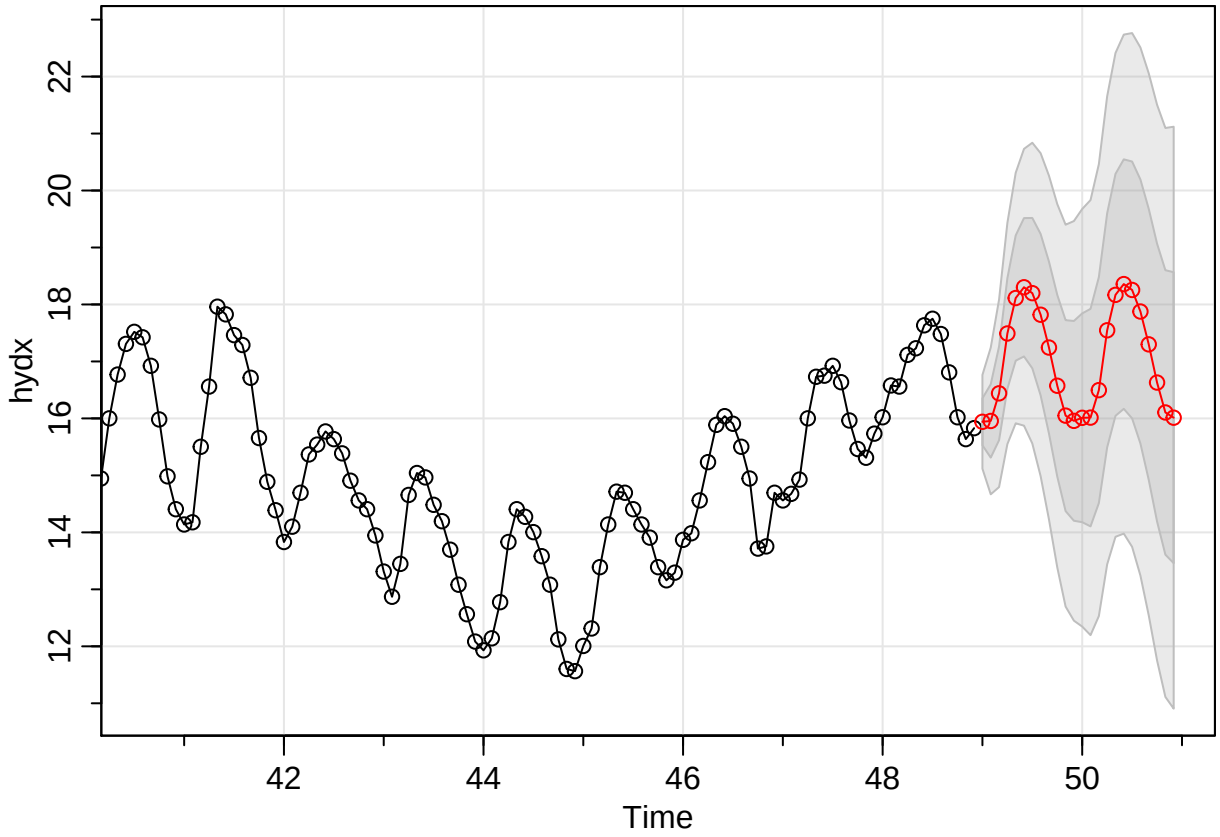
```
sar= sarima(hydx, 1,1,0,0,1,1,12 )
```

```
## initial  value -0.528908
## iter   2 value -0.762144
## iter   3 value -0.810147
## iter   4 value -0.840579
## iter   5 value -0.846108
## iter   6 value -0.846339
## iter   7 value -0.846828
## iter   8 value -0.846837
## iter   9 value -0.846837
## iter   9 value -0.846837
## iter   9 value -0.846837
## final  value -0.846837
## converged
## initial  value -0.854077
## iter   2 value -0.861004
## iter   3 value -0.861040
## iter   4 value -0.861079
## iter   5 value -0.861079
## iter   5 value -0.861079
```

```
## iter   5 value -0.861079
## final  value -0.861079
## converged
```



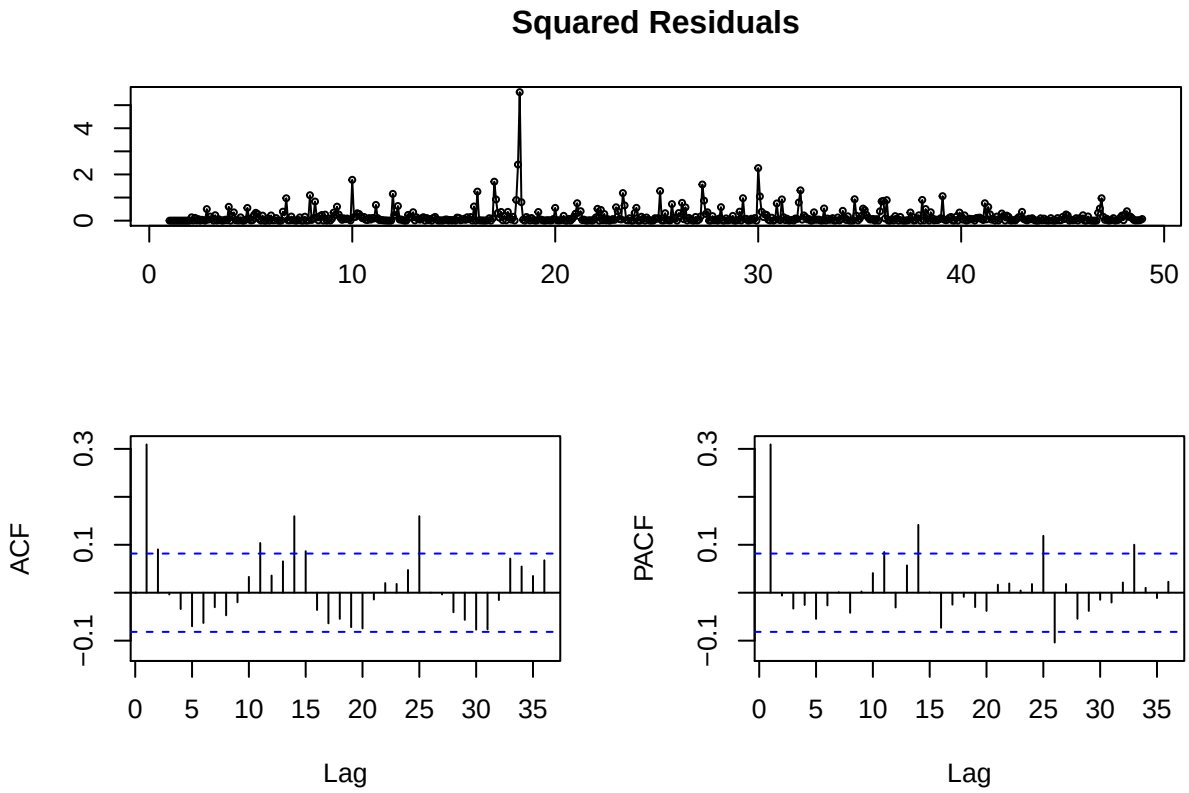

```
pred = sarima.for(hydx, 24, 1,1,0,0,1,1,12 )
```




```
write.csv(file = "hamzeian_Scenario1.csv", pred$pred, row.names = F)
```

The ACF of residuals looks stationary(except for few lags), but the Ljung-box test is not perfect because the p.values are not big. Then, I have plotted the acf of squared residuals to see if there is a serial correlation between them and the garch model is needed to fit on the residuals, but as you can see below, the acf of the squared residuals does not show serial correlation and also the Dickey-Fuller test shows that they are stationary. Therefore, there is no need to fit a garch model on the residuals.

```
tsdisplay(sar$fit$residuals^2 , main = 'Squared Residuals')
```




```
print("adf test on squared residuals")
```

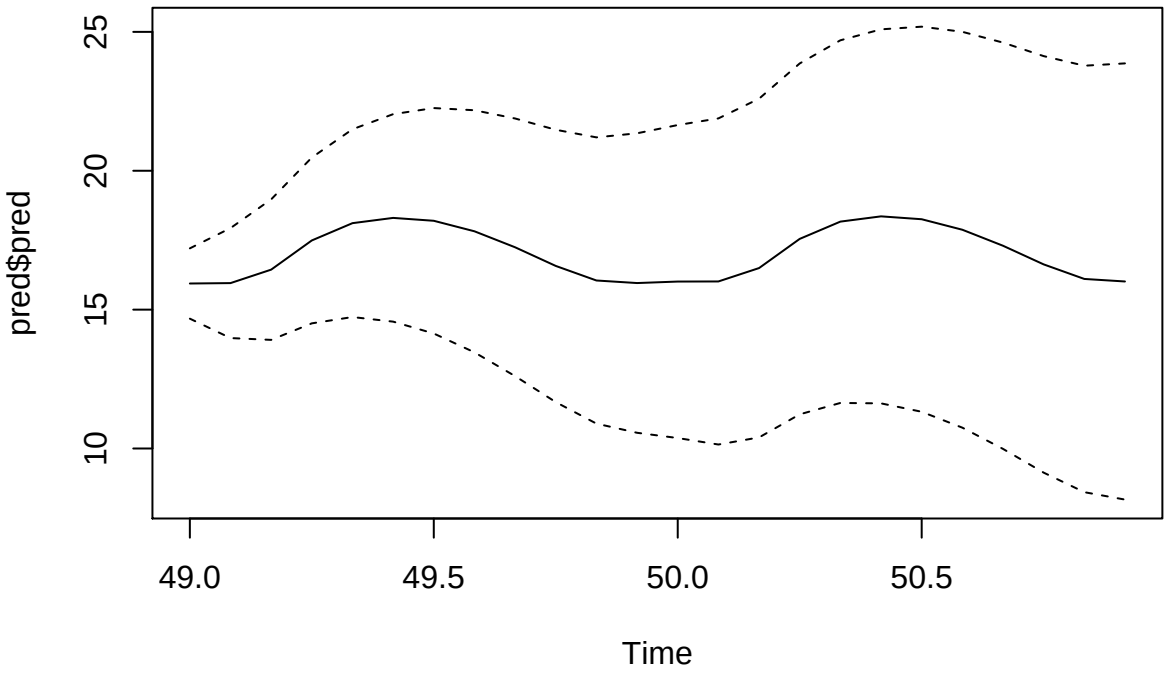
```
## [1] "adf test on squared residuals"
```

```
adf.test(sar$fit$residuals^2)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  sar$fit$residuals^2
## Dickey-Fuller = -8.4156, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

Now, I am going to build 95% confidence band for the predictions:

```
lower = pred$pred-3.075*pred$se
upper = pred$pred+3.075*pred$se
plot(pred$pred, ylim = c(min(lower), max(upper)))
lines(lower, lty= 2)
lines(upper, lty = 2)
```



Scenario 2

read data

```
stocks = data.frame(x = 1:150)
for(i in 1:40){
  filename = paste(paste("stock", i, sep = ""), ".txt", sep = "")
  stocks[[i+1]] = read.csv(filename)$x
}
```

For this scenario, I used garch(1,0) to the data and got the 10 step ahead predictions for standard errors and used the empirical distribution of the (standardized) residuals, noises, to calculate their 15% quantile to build the 15% quantile of the returns. My options for calculating these 15% quantiles were as follows. * 1) use 15% quantile of normal for all the stocks with a) garch(1,0) or b) garch(1,1) for predicting the standard errors * 2) use 15% quantile of the empirical distribution of the residuals from a) garch(1,0) model or b) garch(1,1) model. (Also, use the same garch model for predicting standard deviation)

- 3) for stocks that are normal use normal quantiles with a) garch(1,1) b) garch(1,0) and for the others use the empirical method with a)garch(1,1) and b) garch(1,0). In order to test whether a series is normal or not, I used the Jarque- Bera test and assumed the series is normal if the p.value was less than 0.05 and non-normal otherwise.

For comparing the above options, I used the first 140 entries of each of the stocks as the train and the rest as the test data and calculated the errors as given in the descriptions and chose the option with the least absolute error. The absolute errors for each option are: * 1 a) 0.005292785 , b) 0.005327665 * 2 a) 0.005246517, b) 0.005254587 * 3 a) 0.005312221, b) 0.005293085 Therefore, I chose 2a option which involves using garch(1,0) to predict the standard errors and use the 15% quantiles of the residuals for building the final quantiles.

```
vars = data.frame(x= 1:10)
n= 150
for( i in 2:41){
  model.garch = garchFit(~garch(1,0), data = stocks[[i]][1:n], cond.dist = "std", trace = F)
# jtest = jarque.bera.test(stocks[[i]])
# if( jtest$p.value>0.05){
q= quantile(residuals(model.garch, standardize = T), 0.15)
# }
# else{
# q= qnorm(0.15)
# }
VaR95_td =  mean(stocks[[i]][1:n])+ predict(model.garch, n.ahead = 10)$standardDeviation*q

vars[[i]] = VaR95_td
}
write.csv(file = "hamzeian_Scenario2.csv", vars[, 2:41], row.names = F )
```

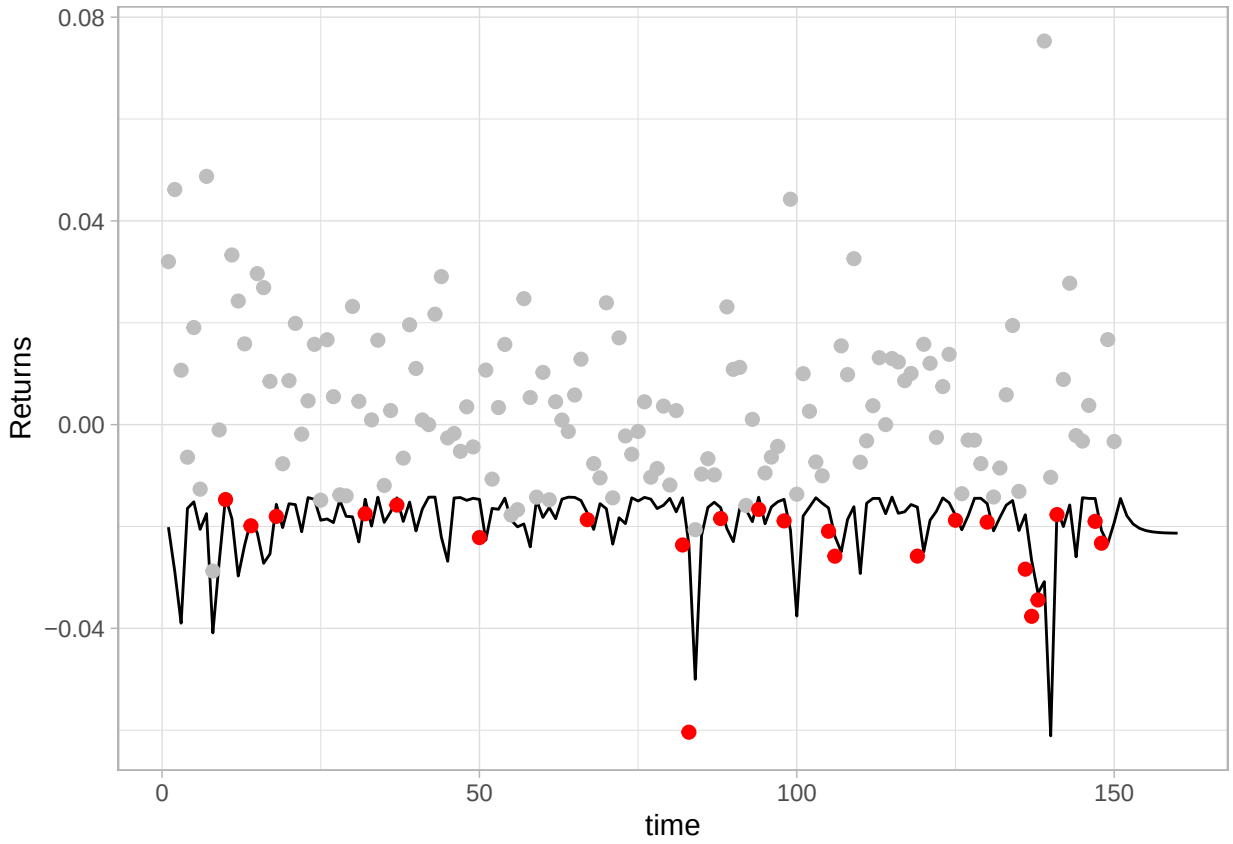
calculateError

This is the code that I used for calculating the error on the test set.

```
sum = 0
for (i in 2:41){
  for (j in 1:10){
    if(stocks[[i]][140+j]<= vars[[i]][j]){
      sum = sum+(stocks[[i]][140+j]-vars[[i]][j] )*(1-0.15)
    }
    else{
      sum = sum+(stocks[[i]][140+j]-vars[[i]][j] )*(0-0.15)
    }
  }
}
err = sum/400
```

In the below graph I fitted the garch(1,0) on the whole data and used it to get the estimated(not predicted) standard deviations and used the 15% quantile of the empirical distribution of the residuals to build the final quantiles. Finally, I also plotted the predicted 10-step ahead 15% quantiles in the tail of the plot.

```
  model.garch = garchFit(~garch(1,0), data = stocks[[2]][1:n], cond.dist = "std", trace = F)

  q= quantile(residuals(model.garch, standardize = T), 0.15)

VaR95_td =  mean(stocks[[2]][1:n])+ model.garch@sigma.t*q

qplot(y = c(VaR95_td, vars[[2]]) , x = 1:(n+10) , geom = 'line') +
    geom_point(aes(x = 1:n , y =stocks[[2]][1:n]  , color = as.factor(stocks[[2]][1:n] < VaR95_td)) , si
    labs(y = 'Returns', x= 'time' ) + theme_light() +
    theme(legend.position = 'none')
```




```
print("The percentage of returns that are below var")
```

```
## [1] "The percentage of returns that are below var"
```

```
sum(stocks[[2]] < VaR95_td)*100/150
```

```
## [1] 15.33333
```

Only 15.33% of the returns in the first series are below the 15% quantile, which is so promising.

Scenatio 3 & 4

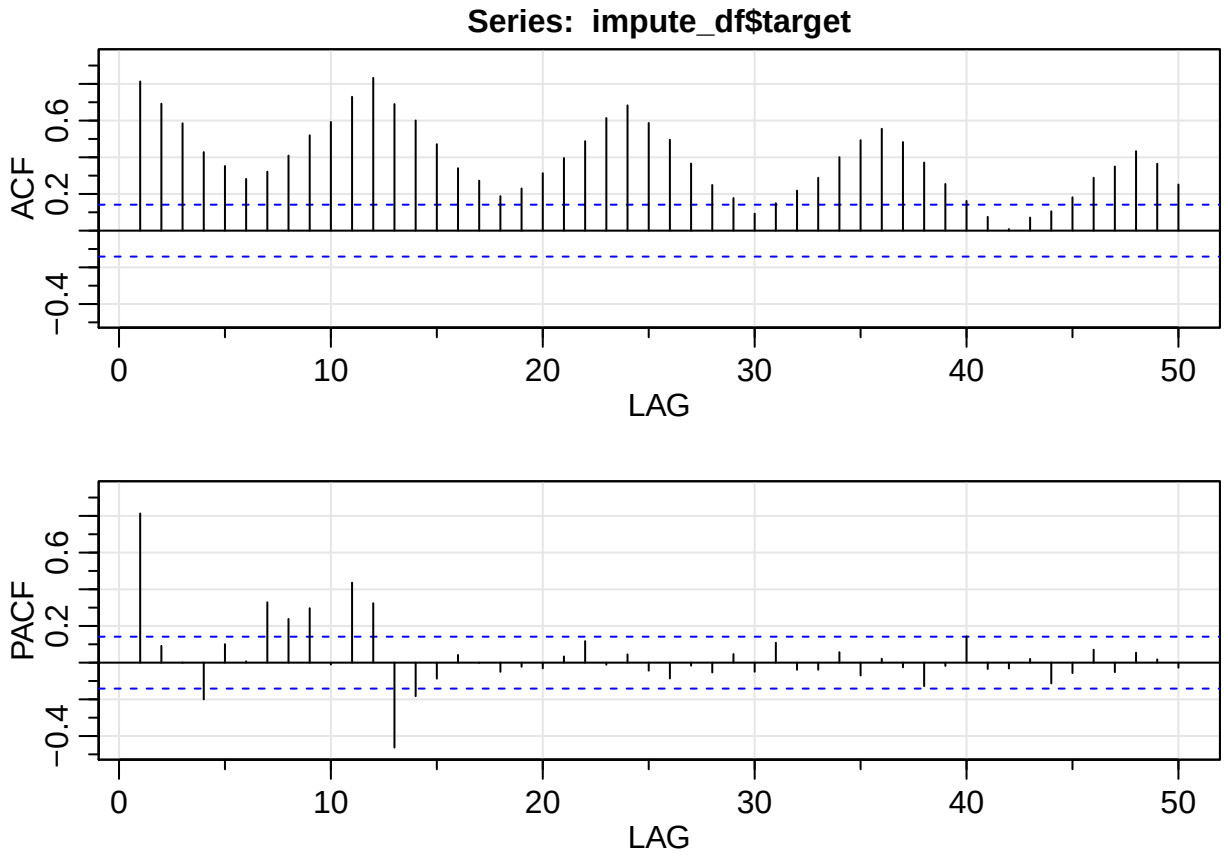
For this scenario, I merged all the series into one dataframe so that they are all correspoding to a single time vector.

```
prod_target = read.csv("prod_target.txt", stringsAsFactors = F)[, c("V1", "V2")]
colnames(prod_target)= c("time", "target" )

car_prod = read.csv("prod_1.txt", stringsAsFactors = F)[, c("V1", "V2")]
colnames(car_prod)= c("time", "car" )

steel_prod = read.csv("prod_2.txt", stringsAsFactors = F)[, c("V1", "V2")]
colnames(steel_prod)= c("time", "steel" )
```

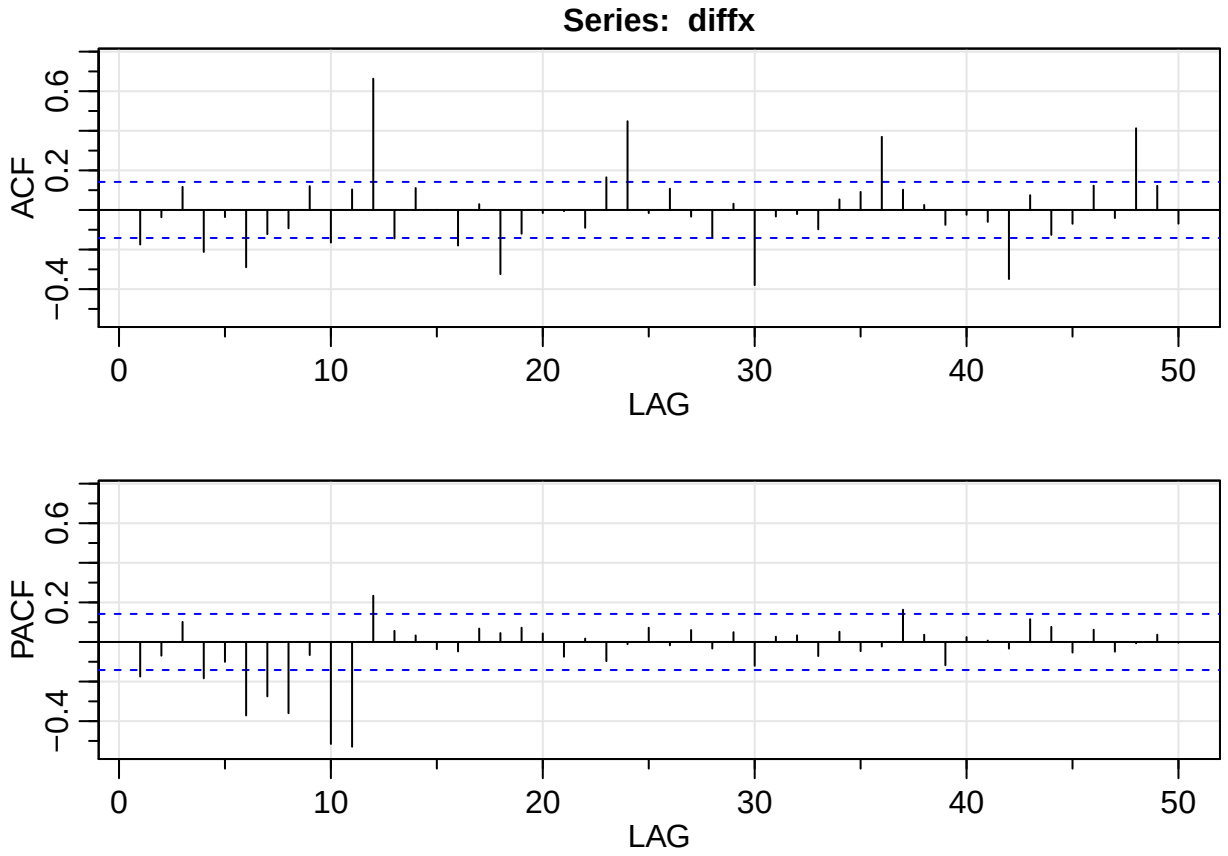
```
eng_1 = read.csv("eng_1.txt", stringsAsFactors = F)[, c("V1", "V2")]
colnames(eng_1)= c("time", "eng_1" )

eng_2 = read.csv("eng_2.txt", stringsAsFactors = F)[, c("V1", "V2")]
colnames(eng_2)= c("time", "eng_2" )

temp = read.csv("temp.txt", header = F, stringsAsFactors = F)
temp[1, ]= c("1", "1943", "10", "20.4")
temp[nchar(temp$V3)<2, ]$V3 = paste("0", temp[nchar(temp$V3)<2, ]$V3, sep = "")
temp$V1 = NULL
temp[, 2] = paste(temp$V2, temp$V3, sep = "-")
temp$V2 = NULL
colnames(temp)= c("time", "temp" )
temp$temp = as.numeric(temp$temp)

#merge
df = merge(merge(merge(merge(merge(prod_target, eng_1, by = "time") , eng_2, by = "time"), steel_prod, l
```

First, I want to impute the 30 missing values in `prod_target`. In order to do this, I want to know how this series is correlated to the other series. Maybe, I can use the past or future values of the exogenous series to impute y. In order to do this, I first fitted a sarima model to the series and obtained the residuals and looked at the cross correlations of the exogenous series with the residuals of the target series. I used the ccf plot for this task. For this part I used only the first 200 points, i.e. the data before the missing values of the target series. However, before fitting a sarima model I need ot use acf and pacf of the target, difference of the target, and etc to determine the p, d, q, P, D, and Q. The acf of the series shows trend and/or seasonality, so I differenced it. The acf of the differenced series again shows seasonality, so I differenced it with lag 12. Since it is a monthly data, it makes sense to have seasonality of 12. After twice differencing no trend or seasonality can be seen. Therefore, d=1, D=1, and S=12 seems good. By looking at the ACF and PACF of this new series, I realized that MA for non-seasonal and AR for seasonal part seems reasonable. By changing p,q, P, and Q I found p=0, q=1, P= 4 and Q=0 fitting well to the data. AIC was minimum as well as all coefficients being significant. Finally, I fit this model to the data and obtained the residuals

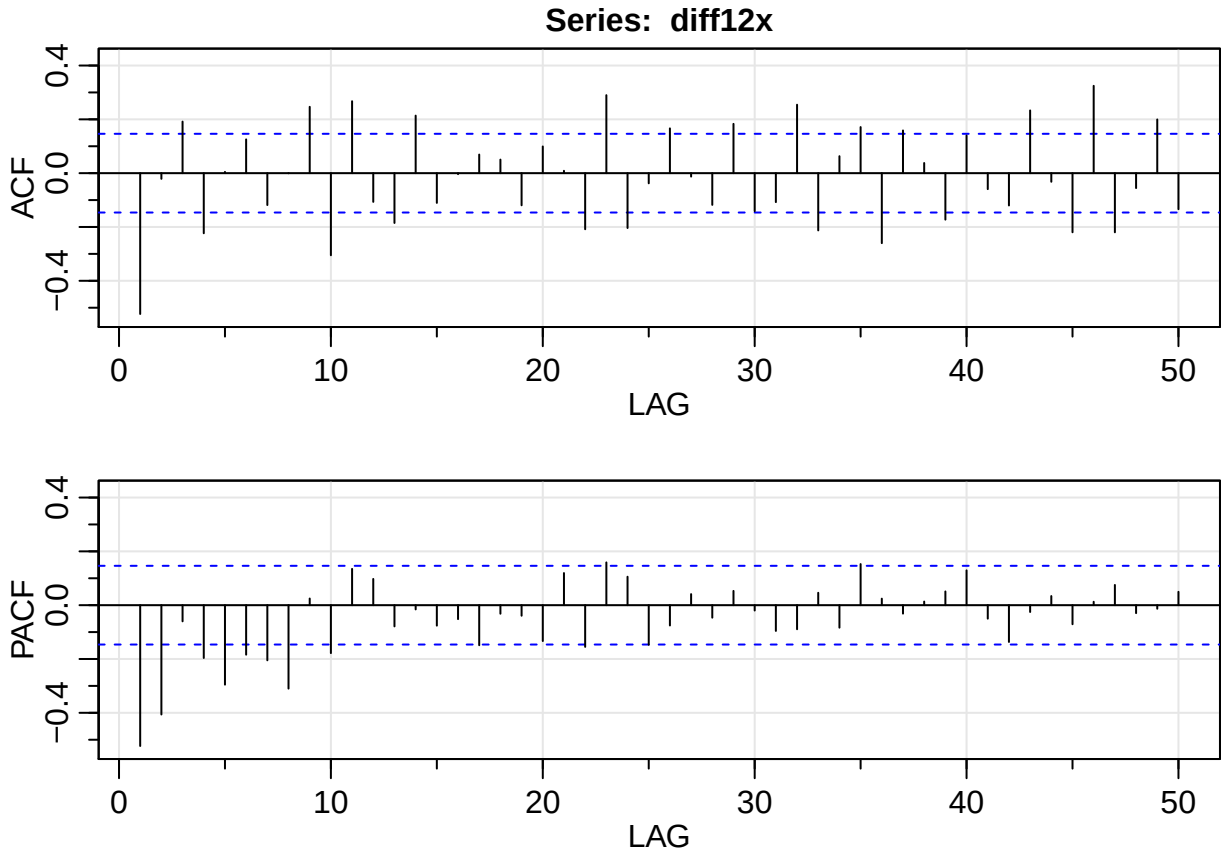
```
impute_df = df[1:200, ]
acf_0 = acf2(impute_df$target, 50, plot=T)
```




```
diffx = diff(impute_df$target)
diff_acf = acf2(diffx, 50, plot = T)
```



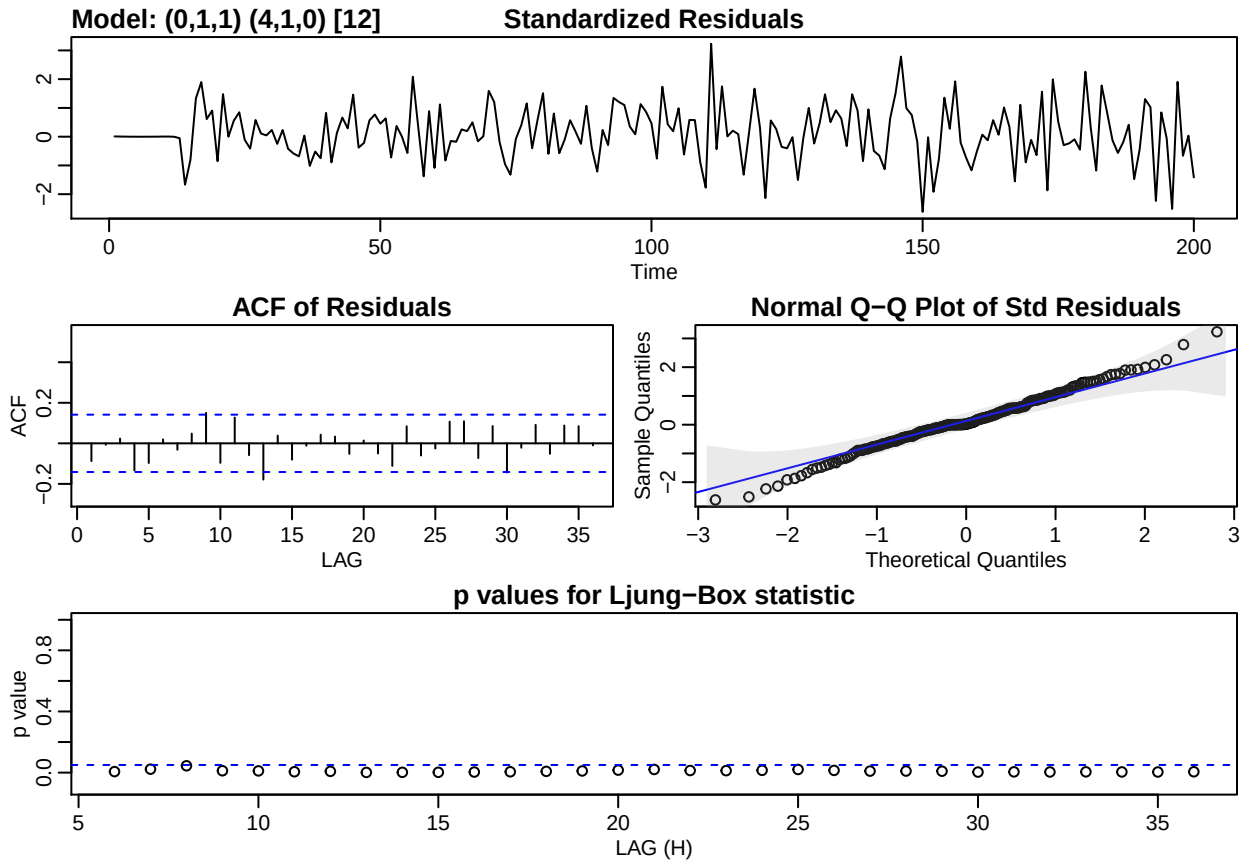

```
diff12x = diff(diffx, 12)
diff12_acf = acf2(diff12x, 50)
```




```
fit1 = sarima(impute_df$target, 0, 1, 1, 4, 1, 0, 12 )
```

```
## initial  value 2.343119
## iter   2 value 2.017417
## iter   3 value 1.914959
## iter   4 value 1.850074
## iter   5 value 1.832528
## iter   6 value 1.817796
## iter   7 value 1.797839
## iter   8 value 1.791738
## iter   9 value 1.789328
## iter  10 value 1.789016
## iter  11 value 1.788993
## iter  12 value 1.788991
## iter  13 value 1.788991
## iter  13 value 1.788991
## iter  13 value 1.788991
## final  value 1.788991
## converged
## initial  value 1.785357
## iter   2 value 1.783923
## iter   3 value 1.782056
## iter   4 value 1.780846
## iter   5 value 1.780005
## iter   6 value 1.779820
## iter   7 value 1.779796
```

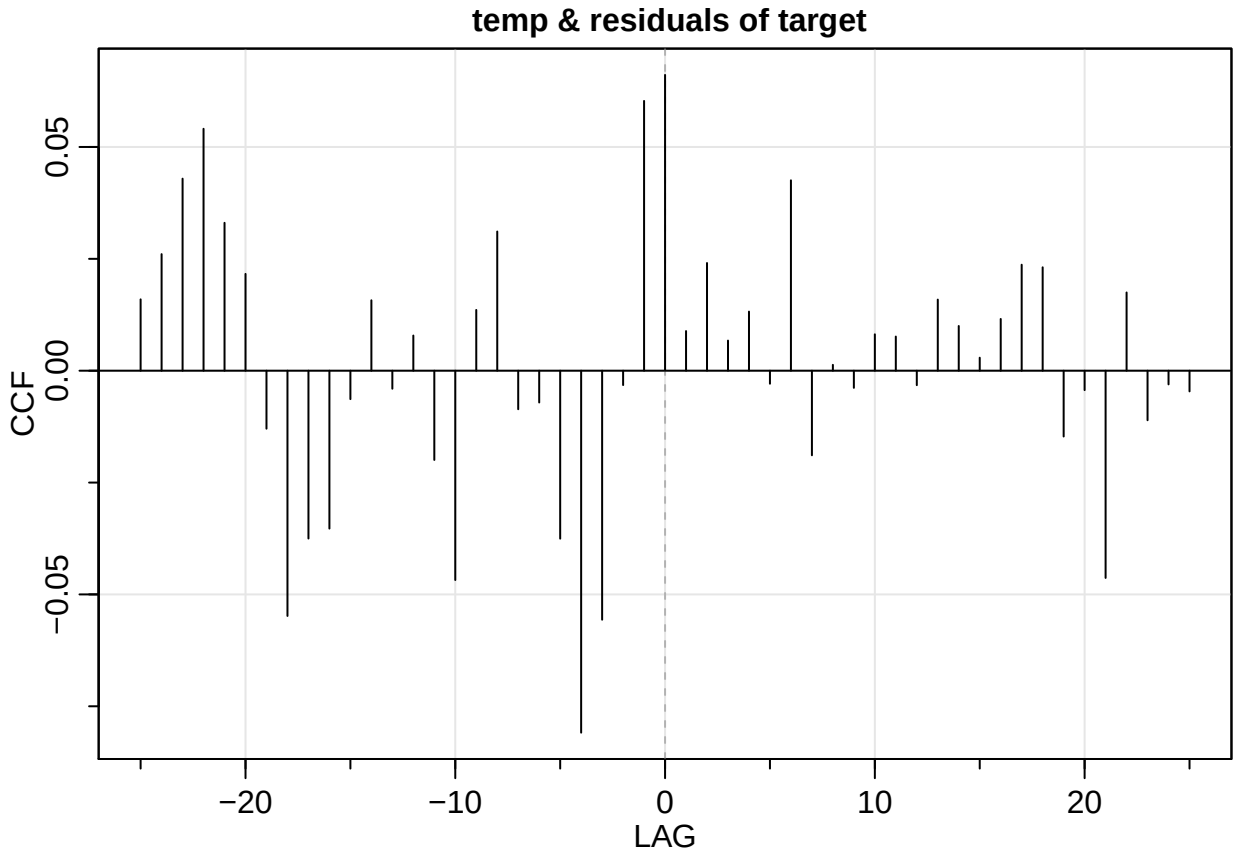
```
## iter   8 value 1.779793
## iter   9 value 1.779793
## iter   9 value 1.779793
## iter   9 value 1.779793
## final  value 1.779793
## converged
```



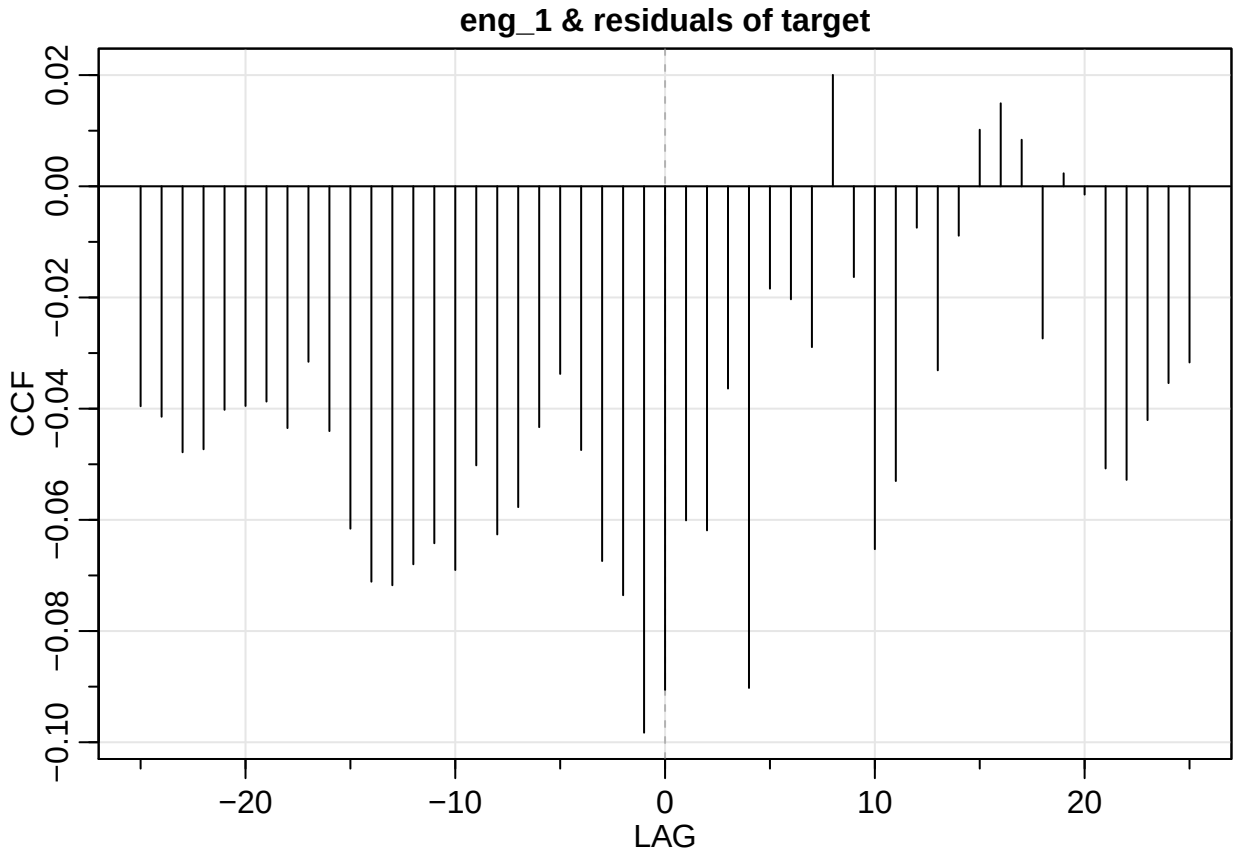

```
dtarget = resid(fit1$fit)
```

Below you can see the ccf plot of the residuals of the target series with the exogeneous series series

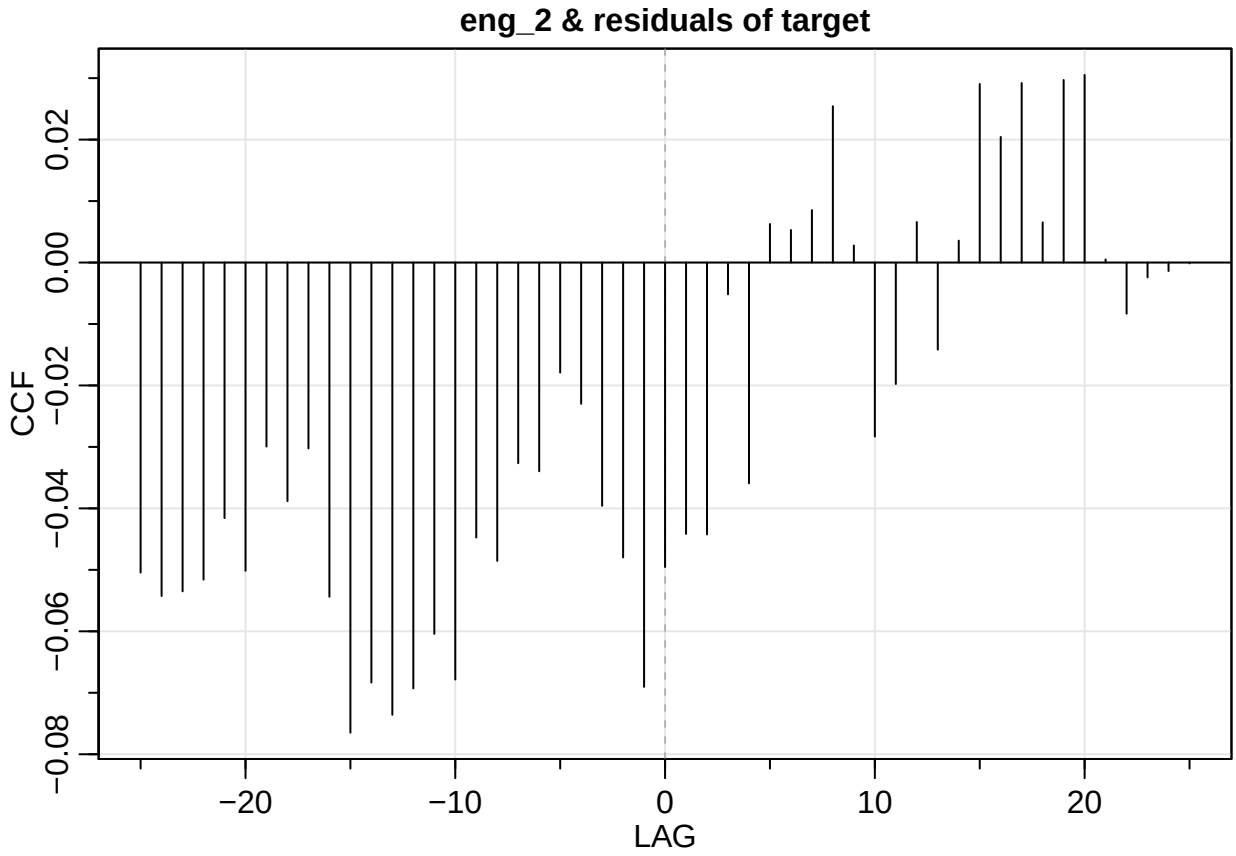
```
ccf2(impute_df$temp, dtarget, main = " temp & residuals of target")
```



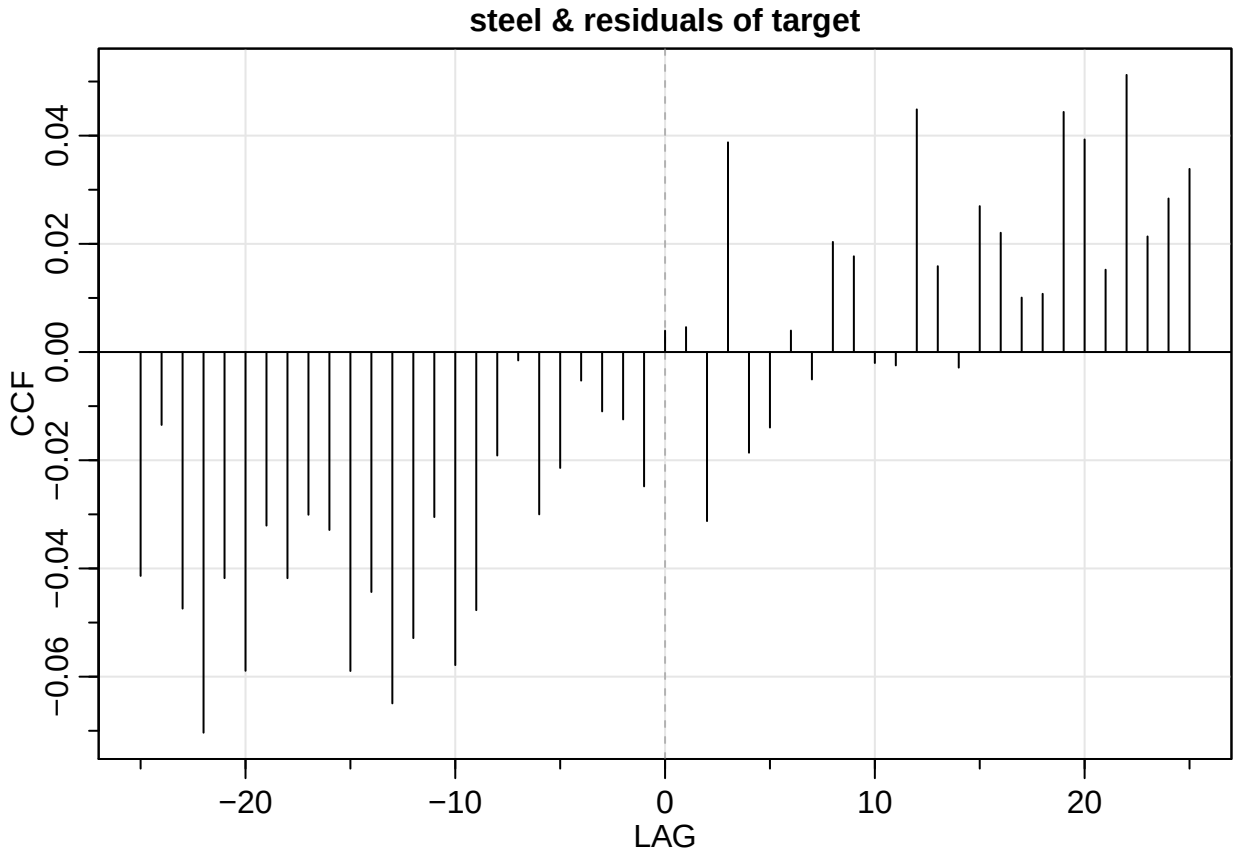

```
ccf2(impute_df$eng_1, dtarget, main = "eng_1 & residuals of target")
```



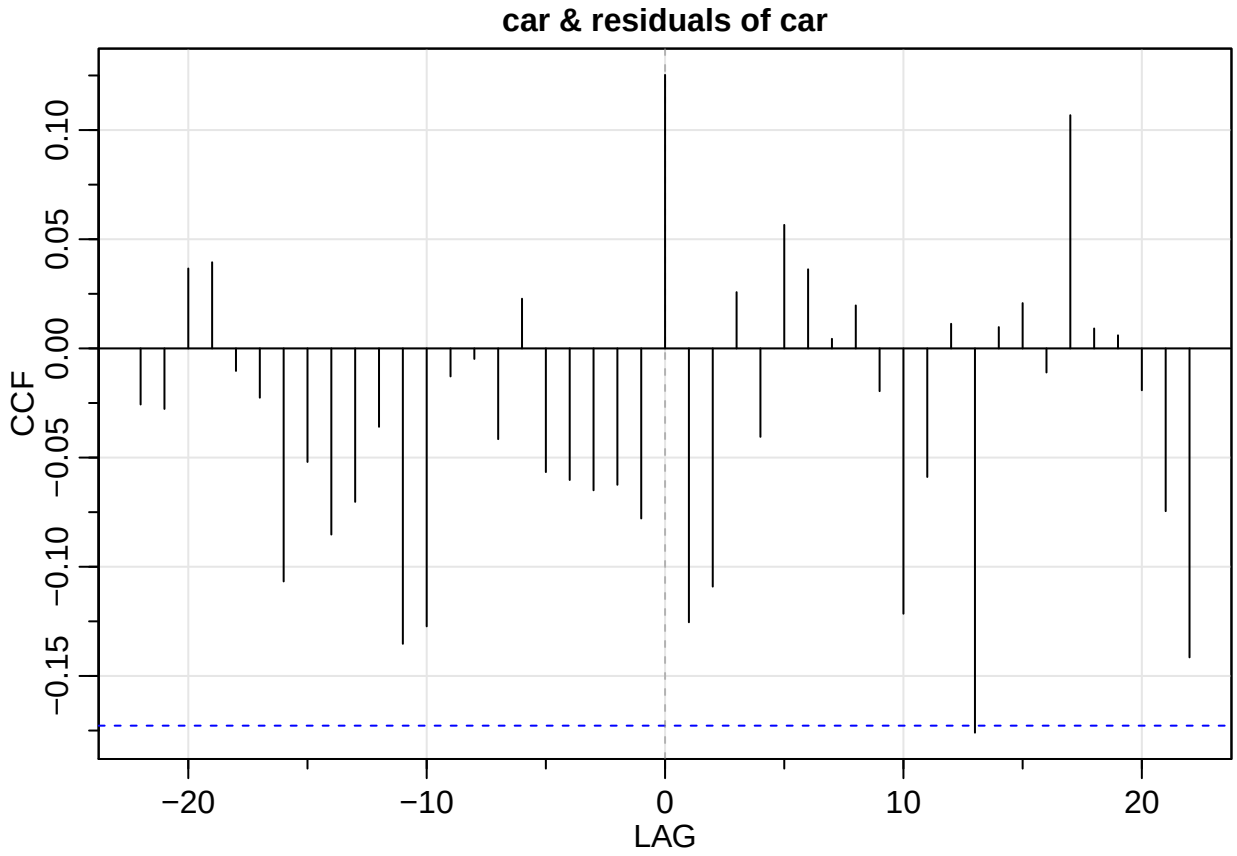

```
ccf2(impute_df$eng_2, dtarget, main = "eng_2 & residuals of target")
```




```
ccf2(impute_df$steel, dtarget, main = "steel & residuals of target")
```




```
ccf2(impute_df$car[67:200], dtarget[67:200], main = "car & residuals of car")
```

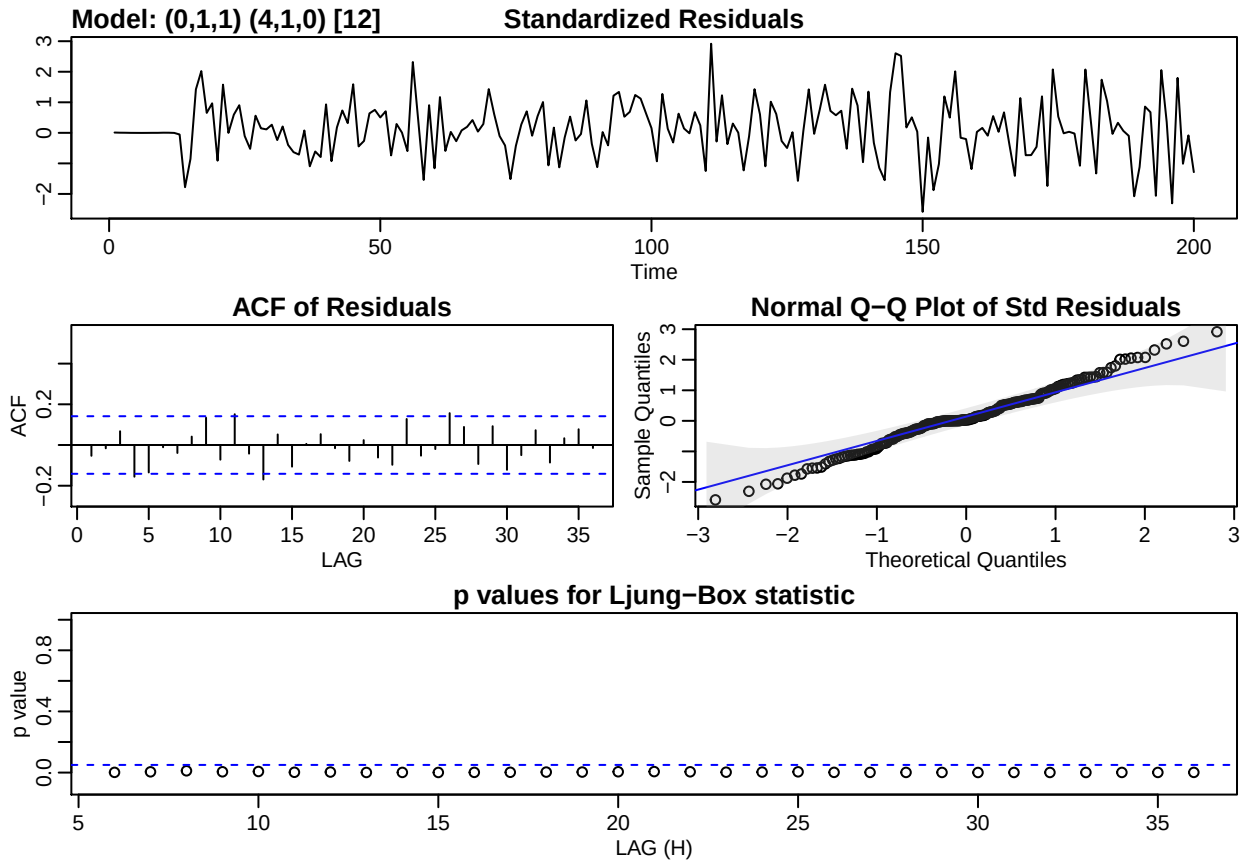


- The ccf with temp , does not show any large cross-correlation. This holds also for `eng_1`, `eng_2` and `steel`.(lower than 0.1)
- The ccf with car shows quite big autocorrelations(bigger than 0.1) at lag 0, +1, and +2. So, it seems that the target series lead the car series and the correlation of target(t) & car(t), target(t) & car(t+1) , and target(t) & car(t+2) are quite big. So I will include these 3 lags in the model for imputing the target series.

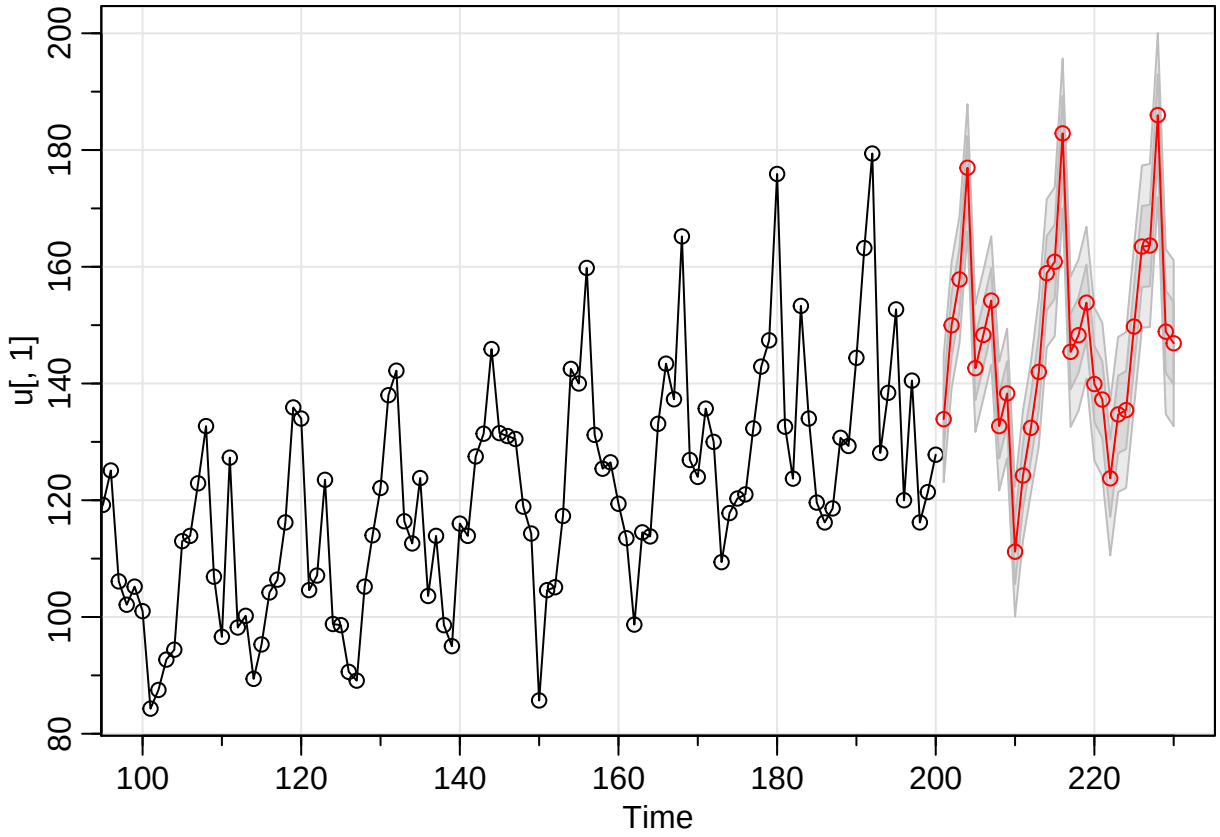
Below, you can see my code for predicting 30 steps ahead for the target value using ARIMAX method and sarima.for function. The regressors(exogenous variables) are lag 0, +1, and +2 of the car series and the parameters for sarima is the same as before.

```
#The values of car before 67 is all NA's so I replaced them with the 67th value of car, it does not aff
impute_df[is.na(impute_df$car), "car"] = 13505
df[is.na(df$car), "car"] = 13505

  u     = ts.intersect(M=ts(impute_df$target) ,  C0 = ts(df$car),
C1=stats::lag(ts(df$car), 1), C2 = stats::lag(ts(df$car), 2))


  newxreg = ts.intersect(C0= ts(df$car), C1=stats::lag(ts(df$car), 1),  C2 = stats::lag(ts(df$car), 2))

  newxreg = newxreg[201:230, 1:3]

sar = sarima(u[,1], 0, 1, 1, 4, 1, 0, 12,  xreg=u[,2:4])
```

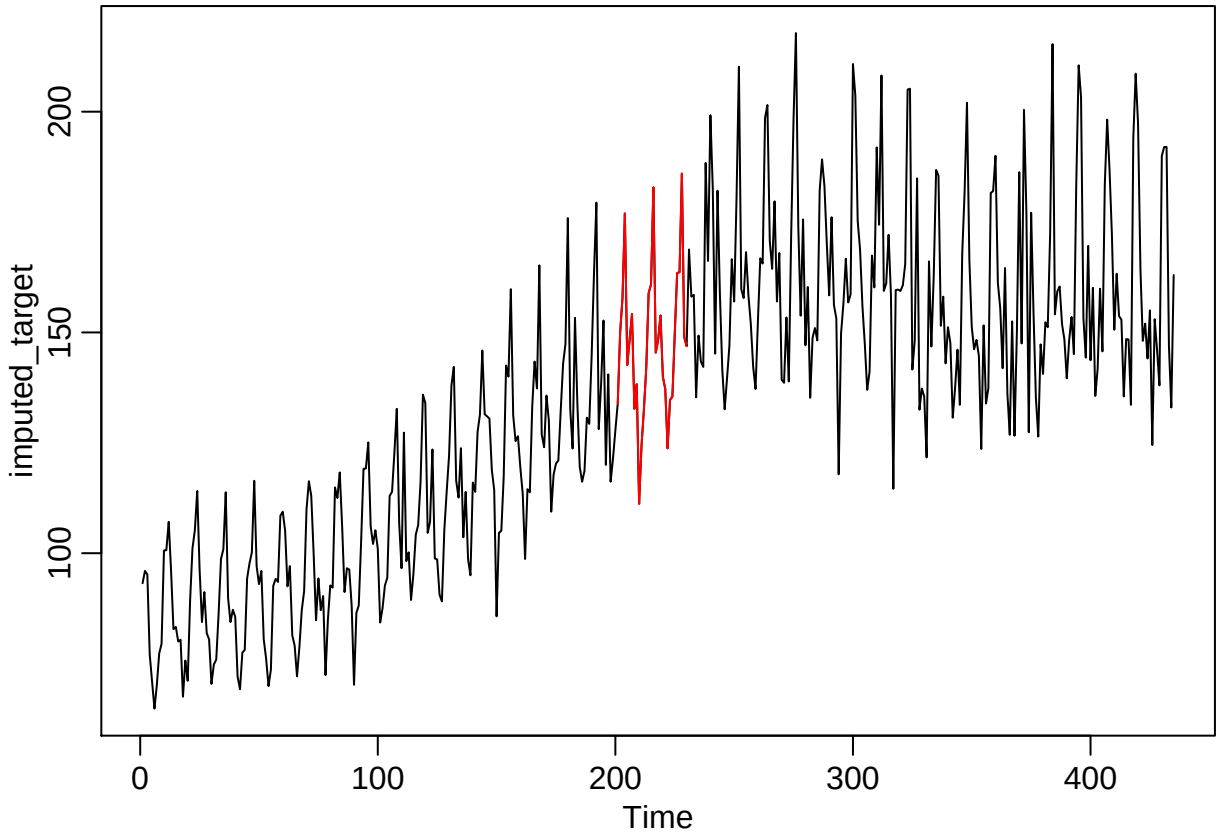
```
## initial  value 2.215156
## iter   2 value 1.957813
## iter   3 value 1.870008
## iter   4 value 1.793144
## iter   5 value 1.764567
## iter   6 value 1.754097
## iter   7 value 1.738034
## iter   8 value 1.729773
## iter   9 value 1.729453
## iter  10 value 1.729426
## iter  11 value 1.729413
## iter  12 value 1.729411
## iter  13 value 1.729411
## iter  14 value 1.729411
## iter  14 value 1.729411
## iter  14 value 1.729411
## final  value 1.729411
## converged
## initial  value 1.731584
## iter   2 value 1.729761
## iter   3 value 1.728625
## iter   4 value 1.727740
## iter   5 value 1.727283
## iter   6 value 1.726875
## iter   7 value 1.726818
## iter   8 value 1.726801
## iter   9 value 1.726800
## iter  10 value 1.726800
## iter  10 value 1.726800
## iter  10 value 1.726800
## final  value 1.726800
## converged
```




```
imputed_values = sarima.for(u[,1], n.ahead = 30,  0, 1, 1, 4, 1, 0, 12,xreg = u[, 2:4], newxreg = newxr
```




```
imputed_target = df$target
imputed_target[is.na(imputed_target)]= imputed_values
plot.ts(imputed_target)
lines(imputed_values, col = "red")
```

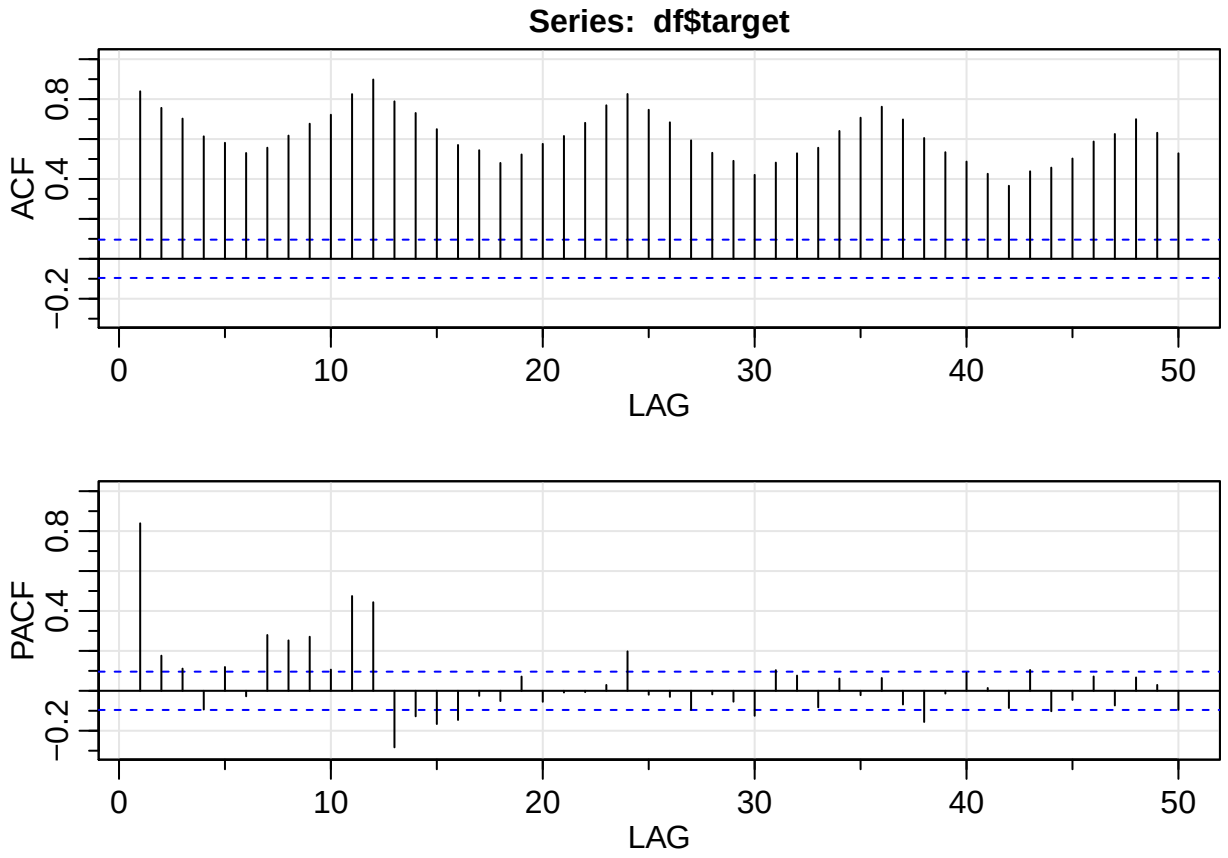



```
df$target = imputed_target
write.csv(file = "hamzeian_Scenario3.csv", imputed_values, row.names = F )
```

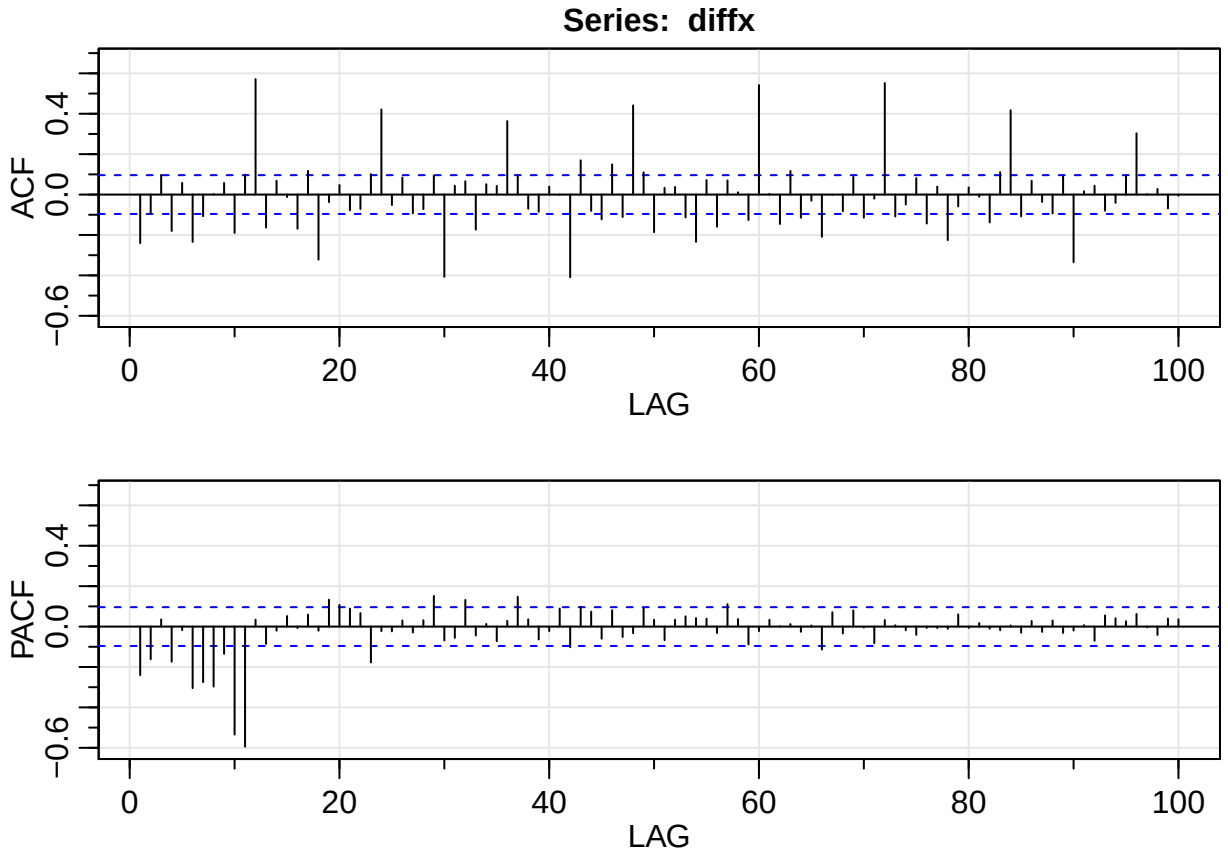
Above, you can see the imputed values(red parts) as a part of the whole series(black parts)

Now, we need to do the forecasting. Once again, we have to find the cross correlation of the residuals of the target series with the exogenous variables. In order to do this, a suitable sarima model should be fitted to the target series and residuals should be obtained. Again, the acf of the target series shows trend and/or seasonality and the acf of the differenced series shows big autocorrelation in lags 12, 24, and etc, so we have seasonality and we need to do the seasonal differencing. By trying different values for parameters of sarima, I determined the parameters to be p=2, q= 1, P = 2, and Q=2.

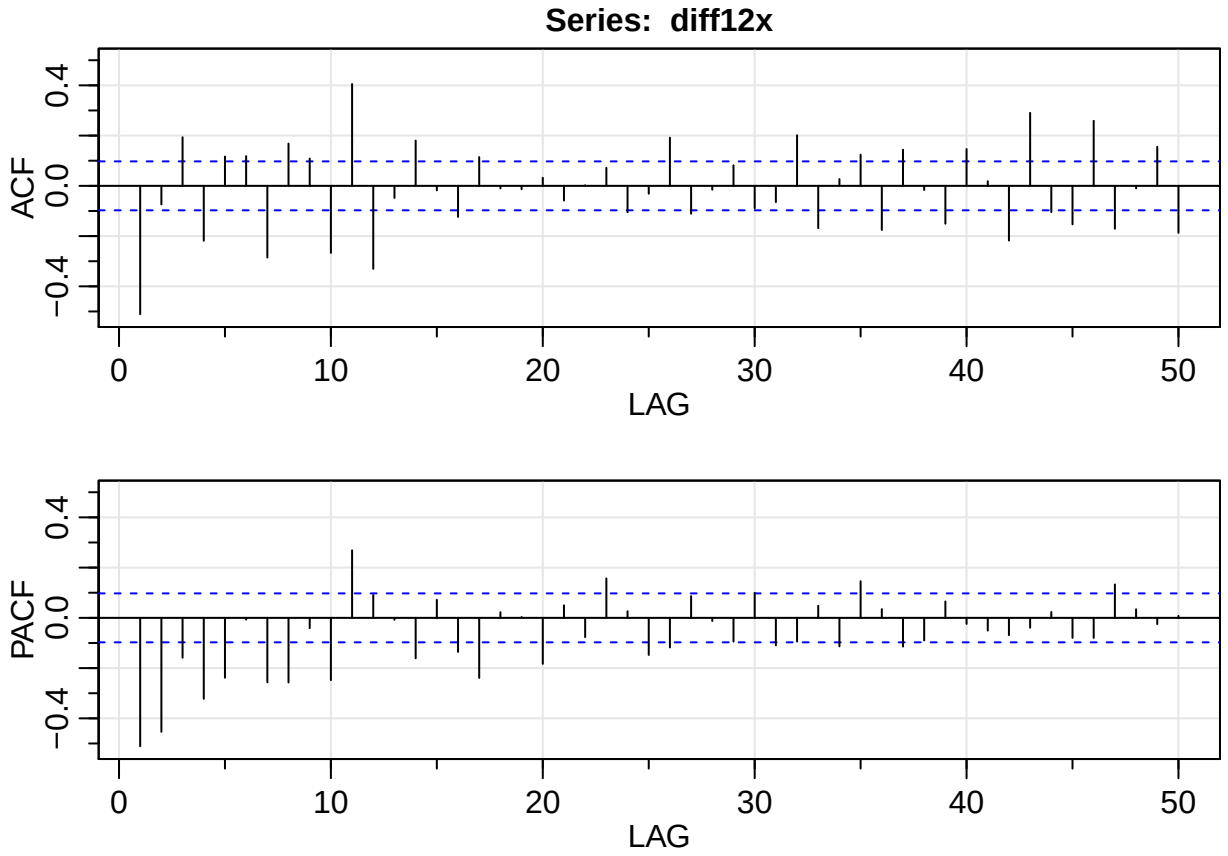
```
acf_0 = acf2(df$target, 50, plot=T)
```




```
diffx = diff(df$target)
diff_acf = acf2(diffx, 100, plot = T)
```




```
diff12x = diff(diffx, 12)
diff12_acf = acf2(diff12x, 50)
```

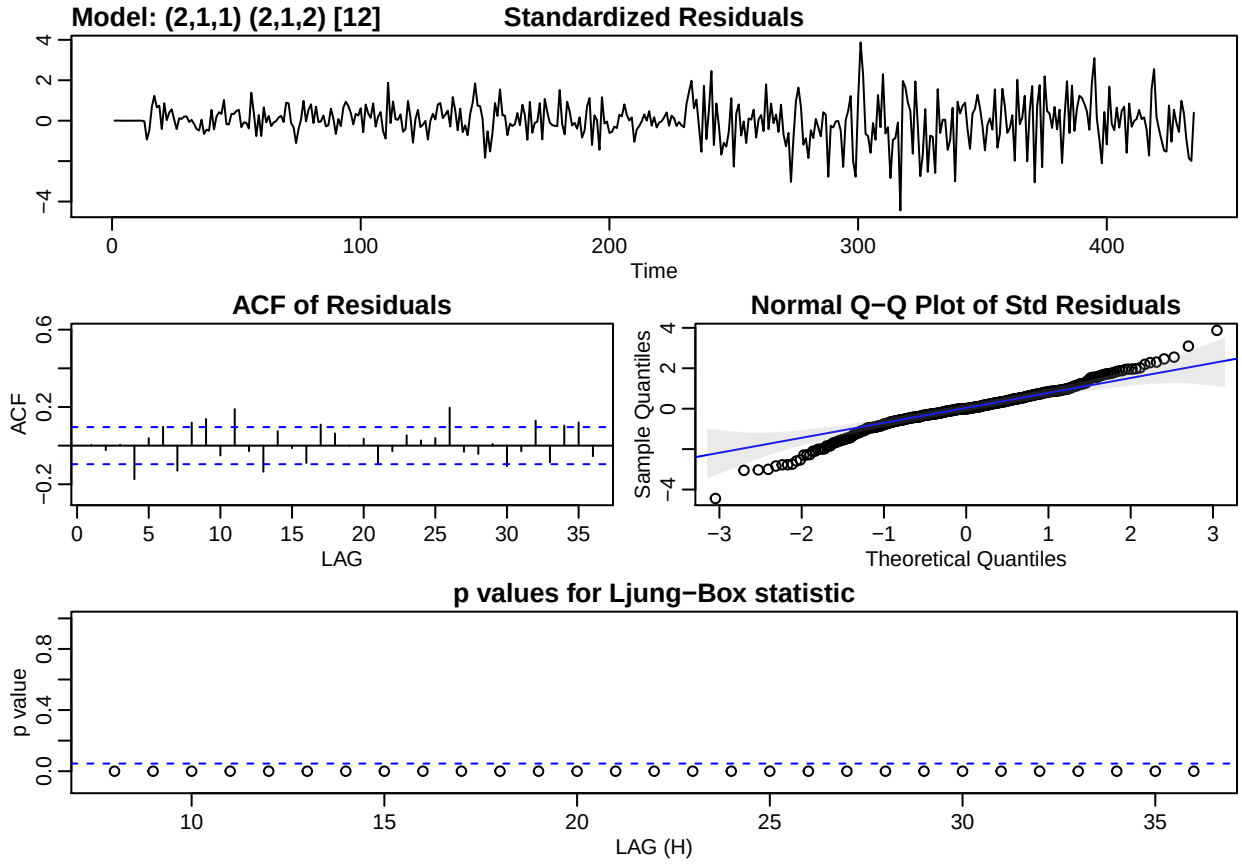



```
fit2 = sarima(df$target, 2, 1, 1, 2, 1, 2, 12 )
```

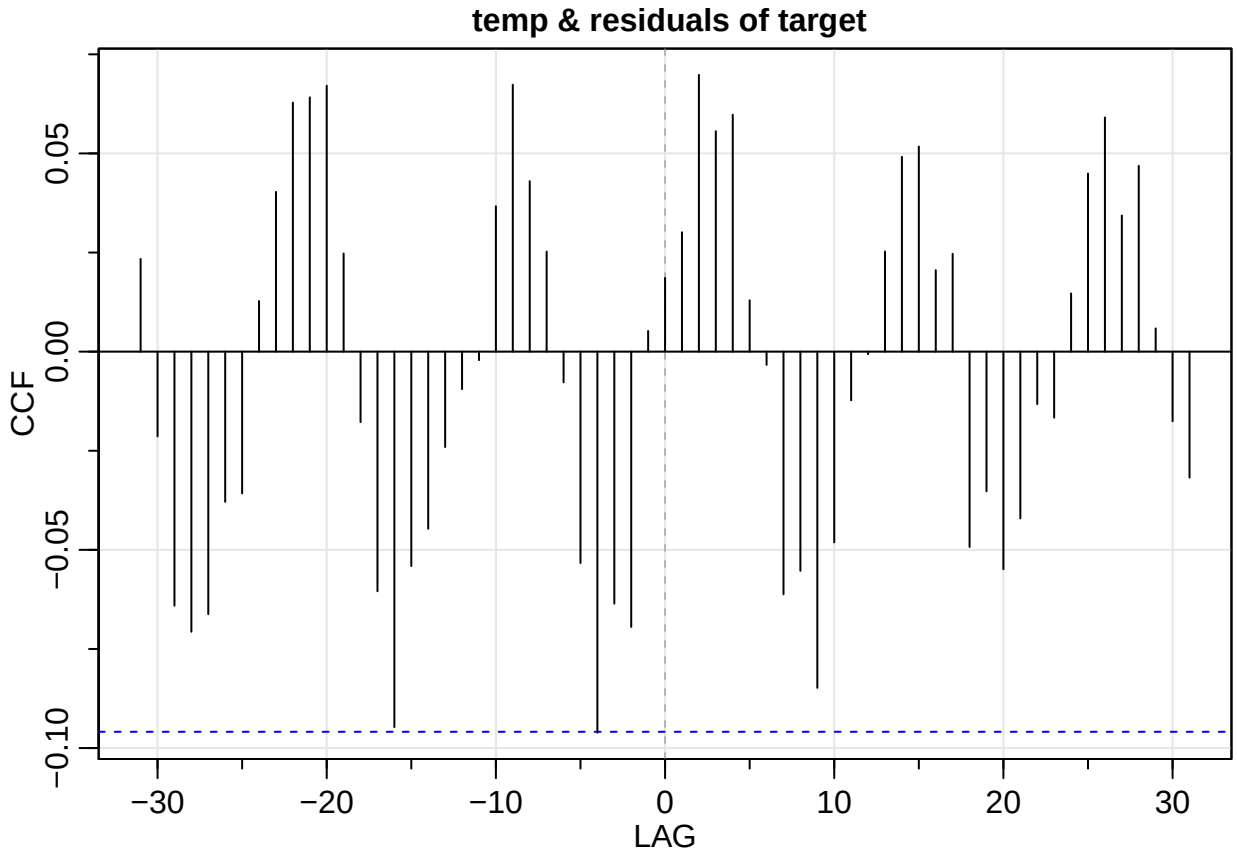
```
## initial  value 2.884608
## iter   2 value 2.427991
## iter   3 value 2.362964
## iter   4 value 2.342339
## iter   5 value 2.322613
## iter   6 value 2.305358
## iter   7 value 2.289319
## iter   8 value 2.284008
## iter   9 value 2.281747
## iter  10 value 2.281208
## iter  11 value 2.281021
## iter  12 value 2.280899
## iter  13 value 2.280672
## iter  14 value 2.280197
## iter  15 value 2.279581
## iter  16 value 2.279098
## iter  17 value 2.278908
## iter  18 value 2.278878
## iter  19 value 2.278829
## iter  20 value 2.278782
## iter  21 value 2.278776
## iter  22 value 2.278774
## iter  23 value 2.278773
## iter  23 value 2.278773
```

```
## iter  23 value 2.278773
## final  value 2.278773
## converged
## initial  value 2.269531
## iter   2 value 2.269340
## iter   3 value 2.269304
## iter   4 value 2.269271
## iter   5 value 2.269249
## iter   6 value 2.269248
## iter   7 value 2.269248
## iter   8 value 2.269248
## iter   9 value 2.269248
## iter  10 value 2.269247
## iter  11 value 2.269244
## iter  12 value 2.269240
## iter  13 value 2.269233
## iter  14 value 2.269214
## iter  15 value 2.269172
## iter  16 value 2.269155
## iter  17 value 2.269130
## iter  18 value 2.269095
## iter  19 value 2.269059
## iter  20 value 2.269045
## iter  21 value 2.269044
## iter  22 value 2.269042
## iter  23 value 2.269035
## iter  24 value 2.269020
## iter  25 value 2.268963
## iter  26 value 2.268924
## iter  27 value 2.268864
## iter  28 value 2.268814
## iter  29 value 2.268708
## iter  30 value 2.268549
## iter  31 value 2.268389
## iter  32 value 2.266389
## iter  33 value 2.265491
## iter  34 value 2.264747
## iter  35 value 2.264692
## iter  36 value 2.264499
## iter  37 value 2.264280
## iter  38 value 2.263911
## iter  39 value 2.262400
## iter  40 value 2.261862
## iter  41 value 2.261339
## iter  42 value 2.259584
## iter  43 value 2.259080
## iter  44 value 2.258778
## iter  45 value 2.258725
## iter  46 value 2.258718
## iter  47 value 2.258718
## iter  48 value 2.258718
## iter  48 value 2.258718
## iter  48 value 2.258718
## final  value 2.258718
```

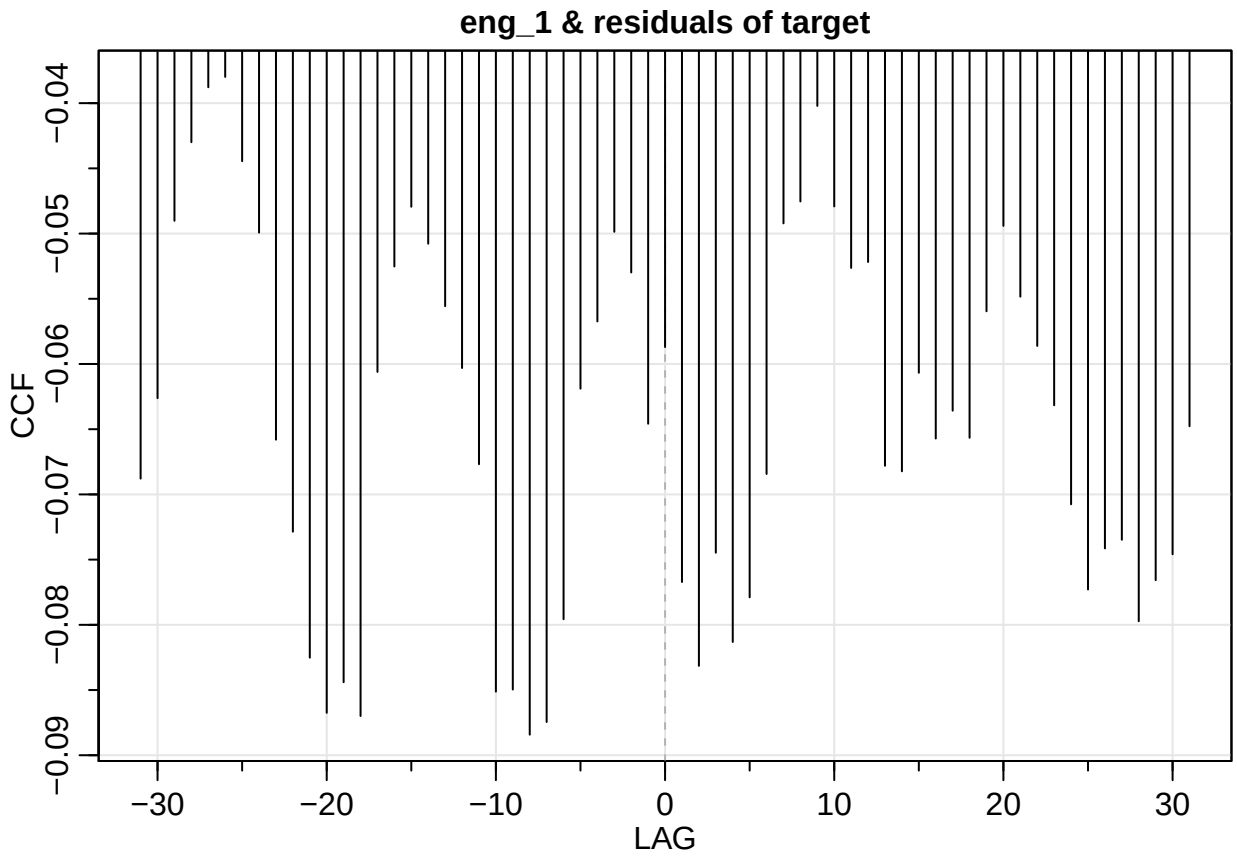
```
## converged
```



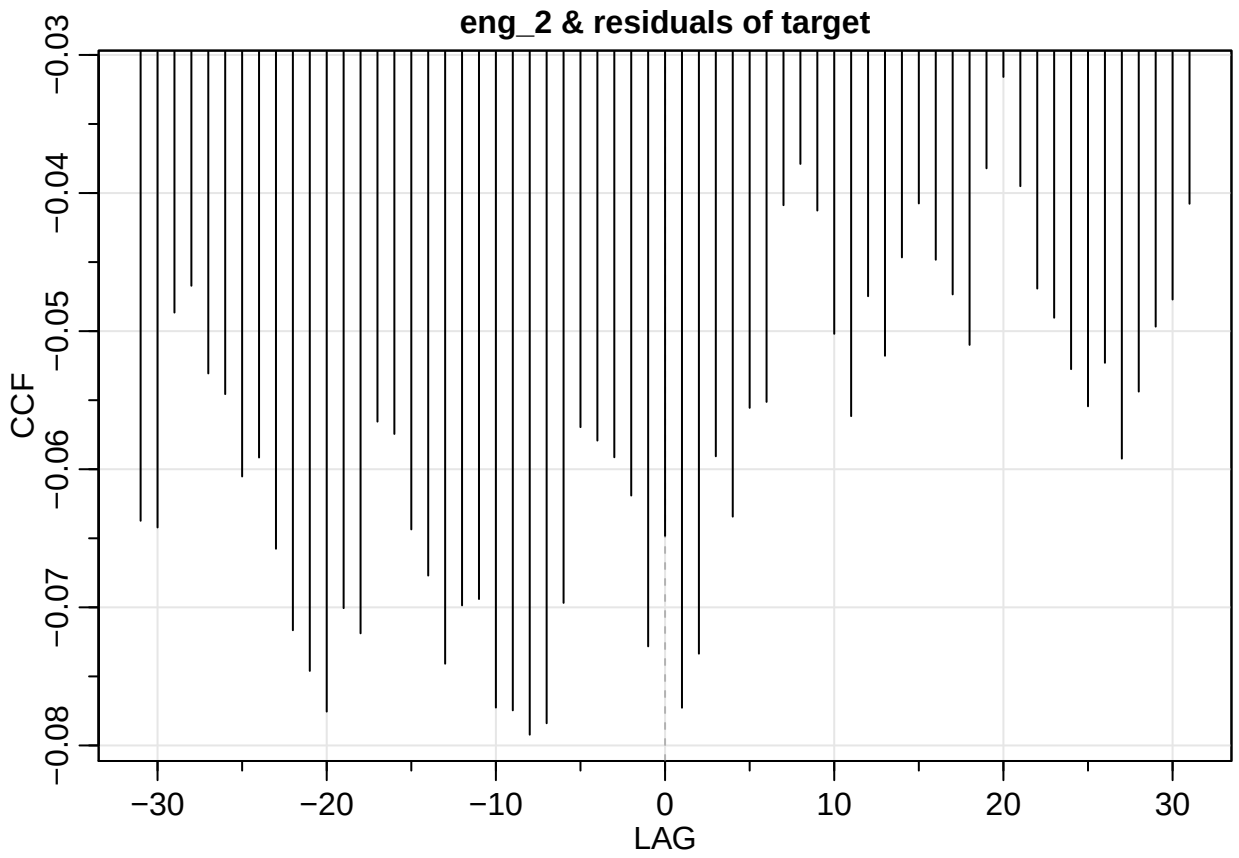

```
dtarget2 = resid(fit2$fit)
ccf2(df$temp, dtarget2, main = " temp & residuals of target")
```



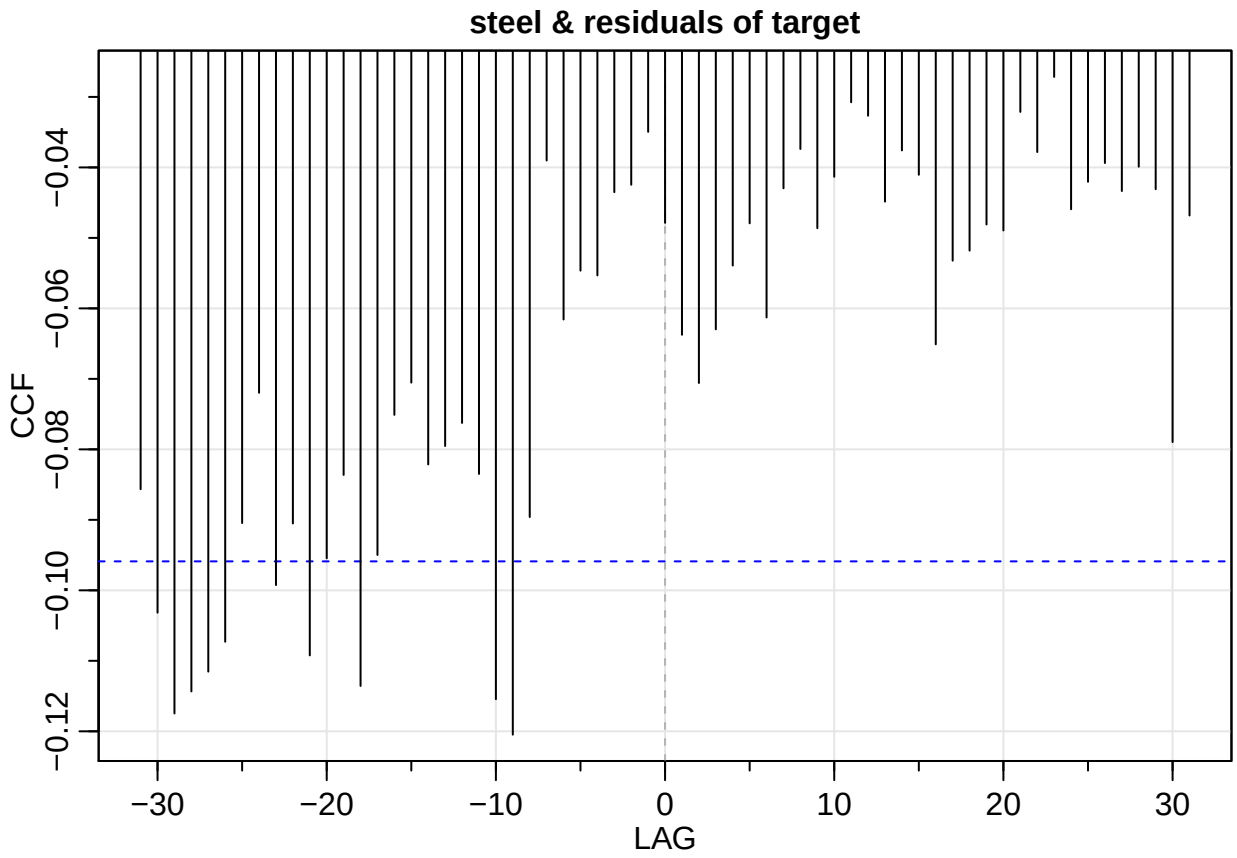

```
ccf2(df$eng_1, dtarget2, main = "eng_1 & residuals of target")
```



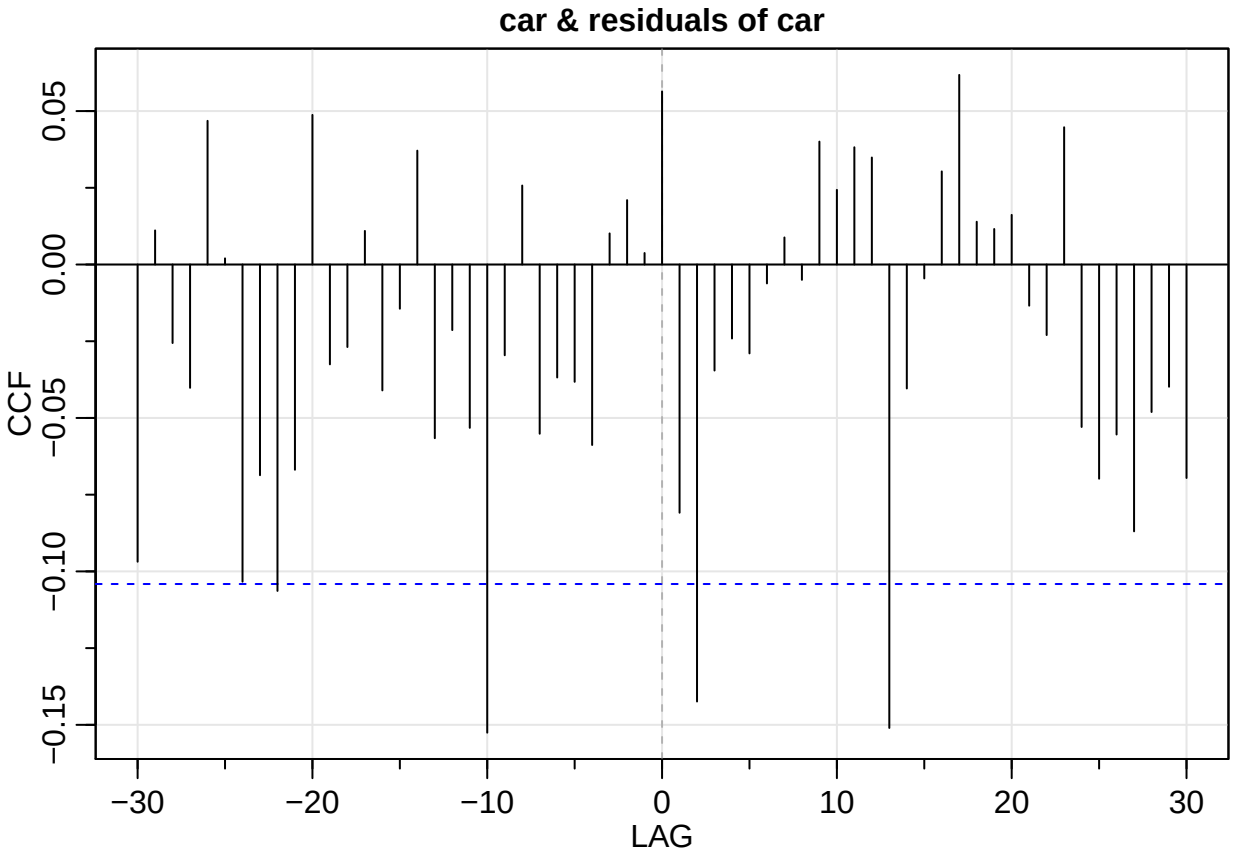

```
ccf2(df$eng_2, dtarget2, main = "eng_2 & residuals of target")
```




```
ccf2(df$steel, dtarget2, main = "steel & residuals of target")
```




```
ccf2(df$car[67:nrow(df)], dtarget2[67:nrow(df)], main = "car & residuals of car")
```

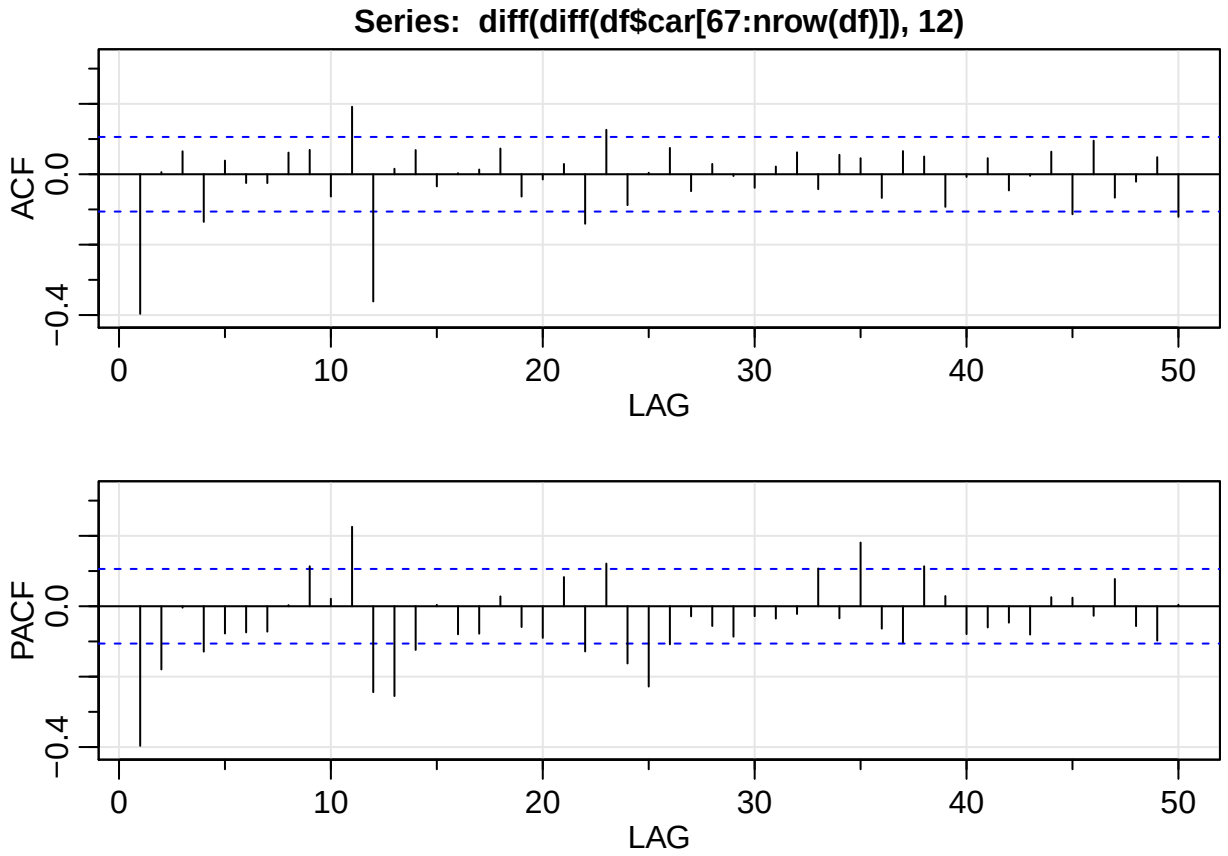


- The ccf with temp shows large correlation at lag -4. (higher than 0.1)
- The ccf with `eng_1` , does not show any large cross-correlation. This holds also for `eng_2` .(lower than 0.1)
- The ccf with steel shows quite big autocorrelation (bigger than 0.1) at lag -9.
- The ccf with car shows quite big autocorrelations(bigger than 0.1) at lag -10 and +2. However, it does not make sense here to use the future values of car to predict the target series, so I only use the lag -10 value of car.

Before using temp(t-4), car(t-10), and steel(t-9) to predict the target, I have to predict car , steel, and temp individually using appropriate sarima models.

Predicting car series 24 steps ahead

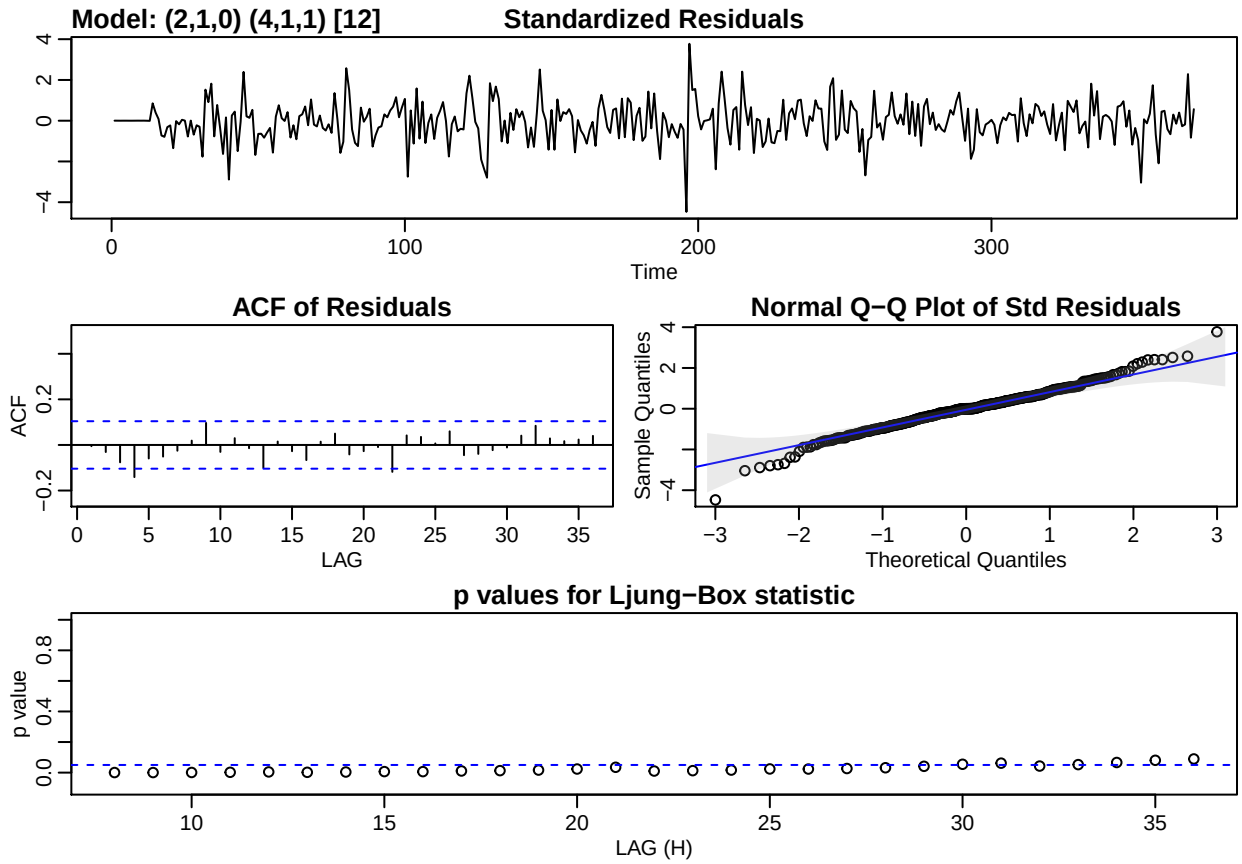
```
acf_car = acf2(diff(diff(df$car[67:nrow(df)]), 12), plot = T, 50)
```



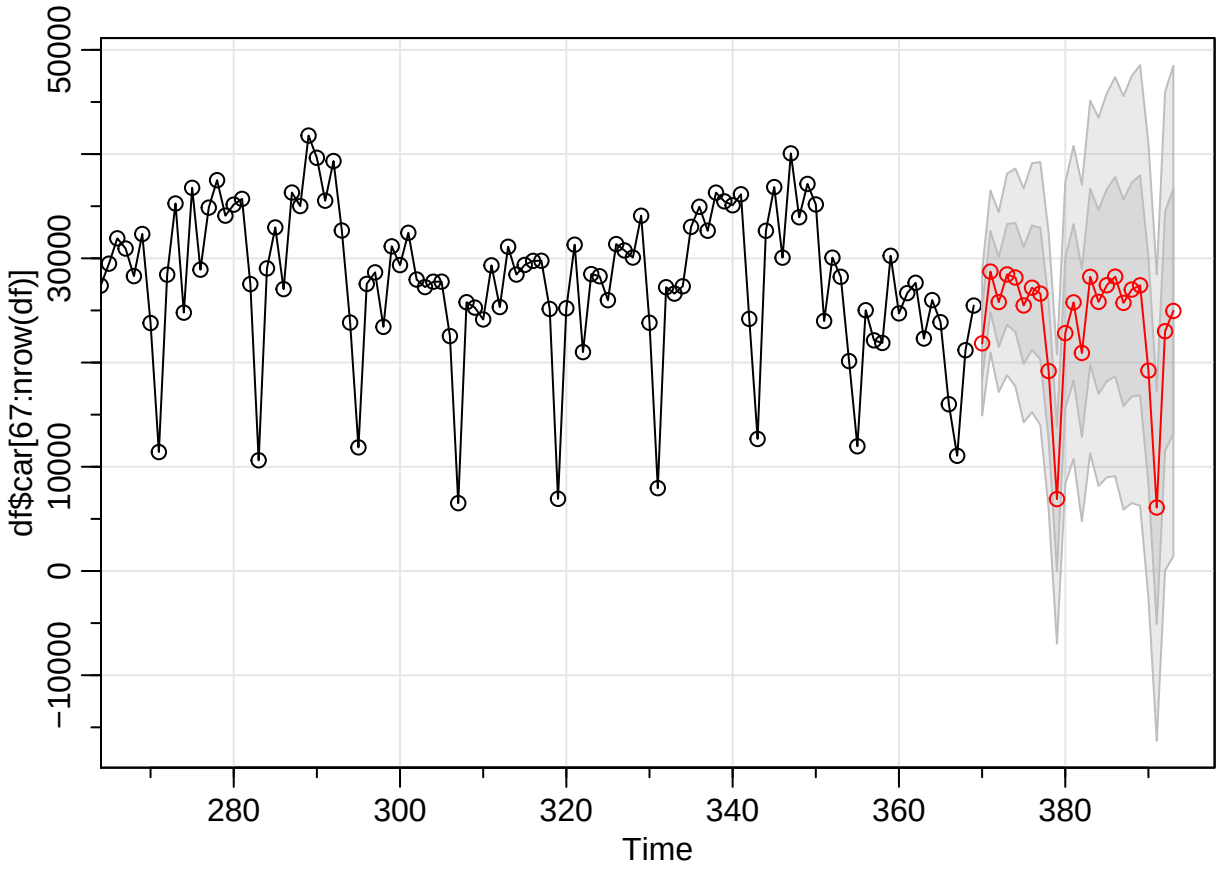

```
fit_car = sarima(df$car[67:nrow(df)], 2,1,0, 4, 1,1, 12)
```

```
## initial  value 8.504401
## iter   2 value 8.248160
## iter   3 value 8.183281
## iter   4 value 8.161375
## iter   5 value 8.157128
## iter   6 value 8.156765
## iter   7 value 8.156672
## iter   8 value 8.156665
## iter   9 value 8.156664
## iter  10 value 8.156662
## iter  11 value 8.156660
## iter  12 value 8.156659
## iter  13 value 8.156659
## iter  13 value 8.156659
## iter  13 value 8.156659
## final  value 8.156659
## converged
## initial  value 8.171062
## iter   2 value 8.169812
## iter   3 value 8.169100
## iter   4 value 8.169040
## iter   5 value 8.169007
## iter   6 value 8.168983
## iter   7 value 8.168962
```

```
## iter   8 value 8.168957
## iter   9 value 8.168957
## iter   9 value 8.168957
## iter   9 value 8.168957
## final  value 8.168957
## converged
```

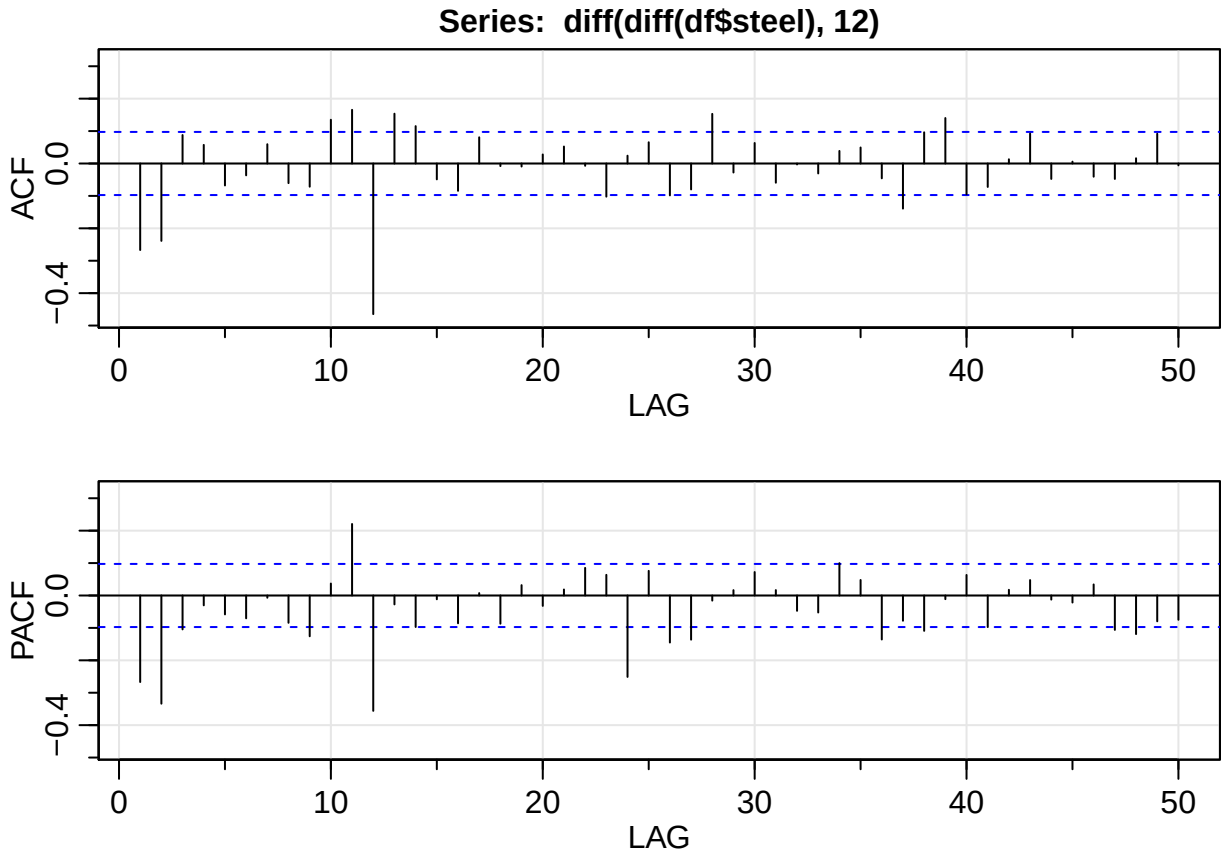



```
pred_car = sarima.for(df$car[67:nrow(df)],24, 2,1,0, 4, 1,1, 12)$pred
```



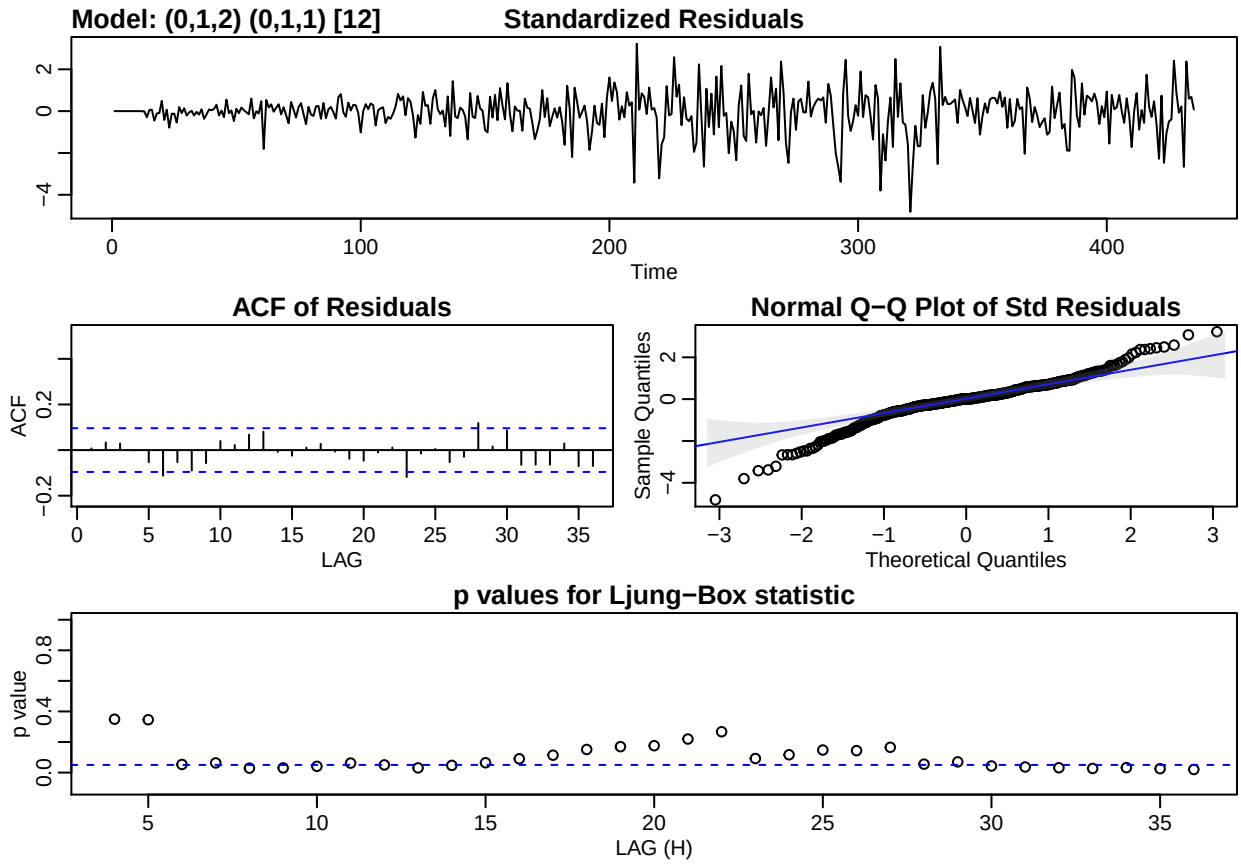
Predicting steel series 24 steps ahead

```
acf_steel = acf2(diff(diff(df$steel), 12), plot = T, 50)
```

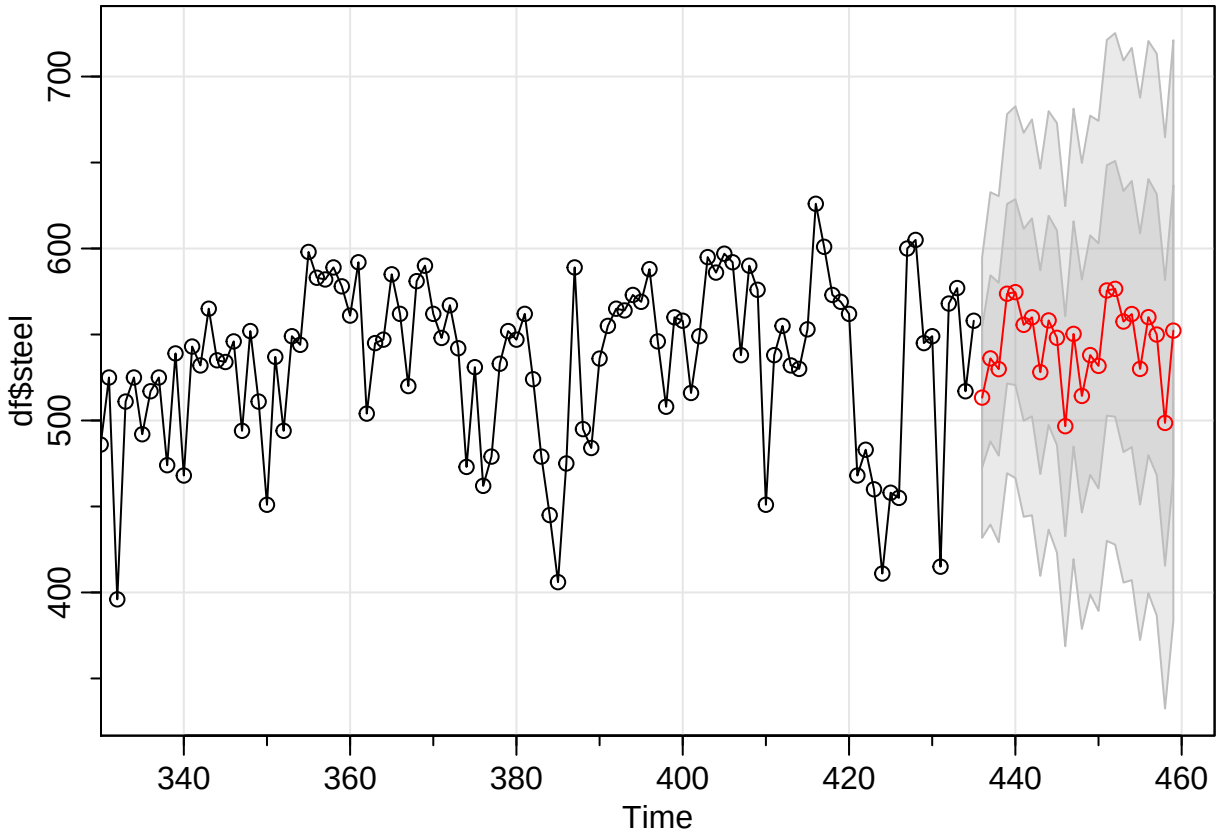



```
fit_steel = sarima(df$steel,  0, 1, 2, 0, 1,1, 12)
```

```
## initial  value 4.069722
## iter   2 value 3.816340
## iter   3 value 3.783017
## iter   4 value 3.743980
## iter   5 value 3.728461
## iter   6 value 3.727957
## iter   7 value 3.727694
## iter   8 value 3.727679
## iter   9 value 3.727679
## iter  10 value 3.727679
## iter  11 value 3.727679
## iter  11 value 3.727679
## iter  11 value 3.727679
## final  value 3.727679
## converged
## initial  value 3.736206
## iter   2 value 3.735069
## iter   3 value 3.734740
## iter   4 value 3.734736
## iter   5 value 3.734736
## iter   5 value 3.734736
## iter   5 value 3.734736
## final  value 3.734736
## converged
```

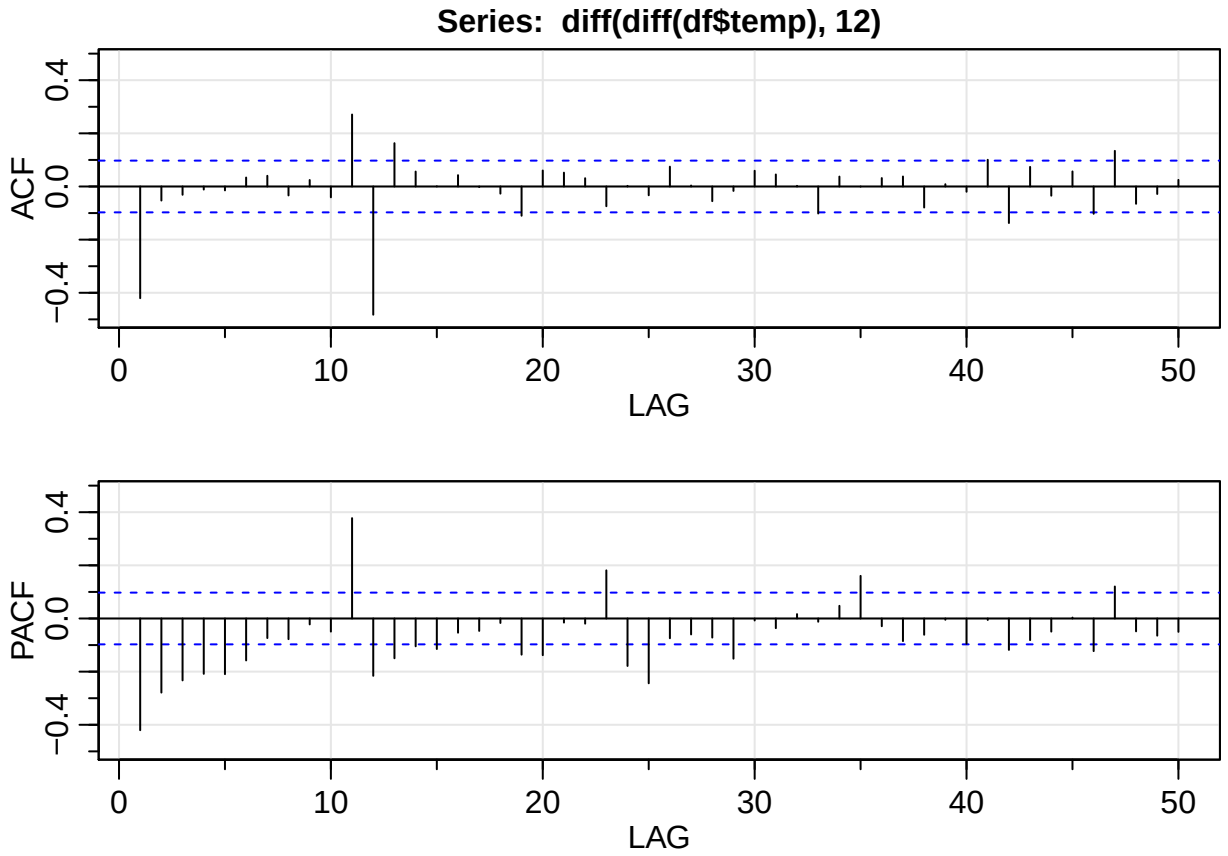



```
pred_steel = sarima.for(df$steel, 24,  0, 1, 2, 0, 1,1, 12)$pred
```



Predicting temp series 24 steps ahead

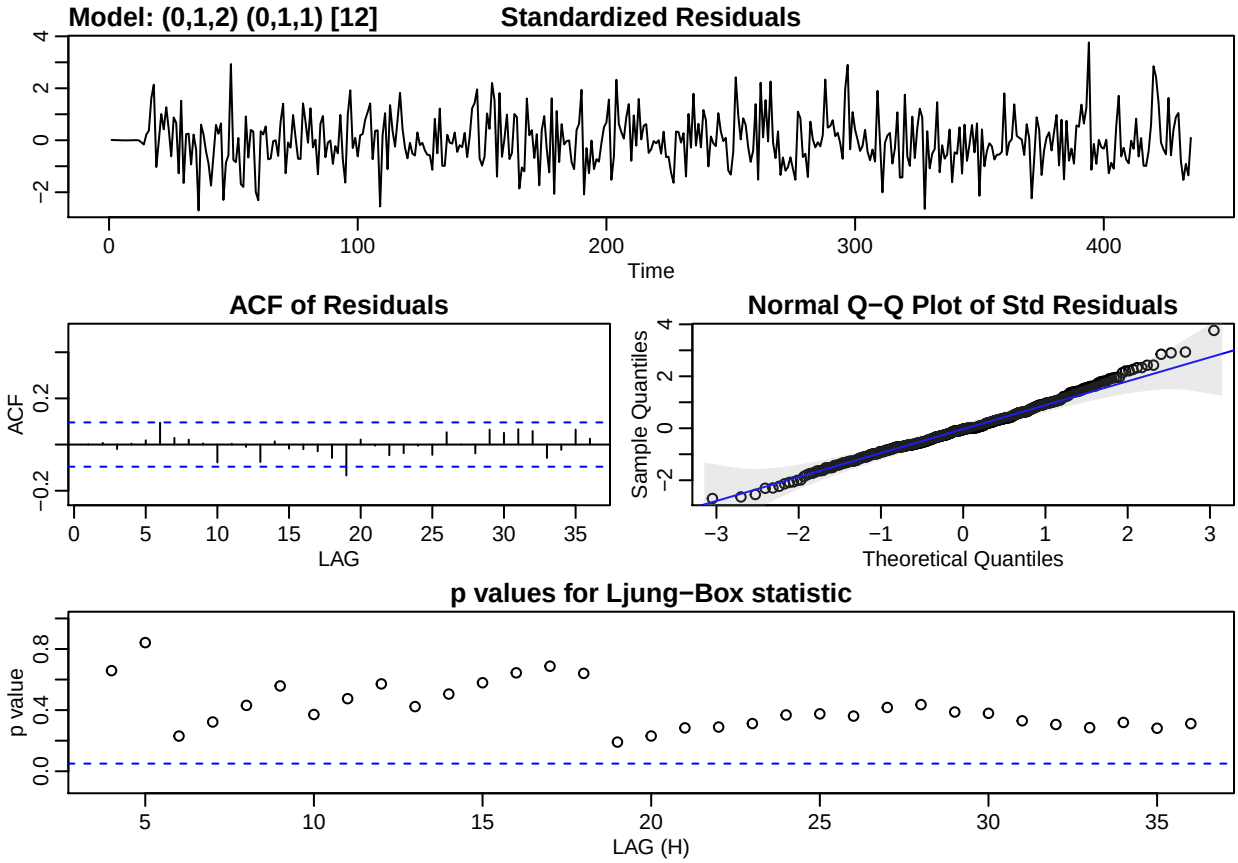
```
acf_temp = acf2(diff(diff(df$temp), 12), plot = T, 50)
```



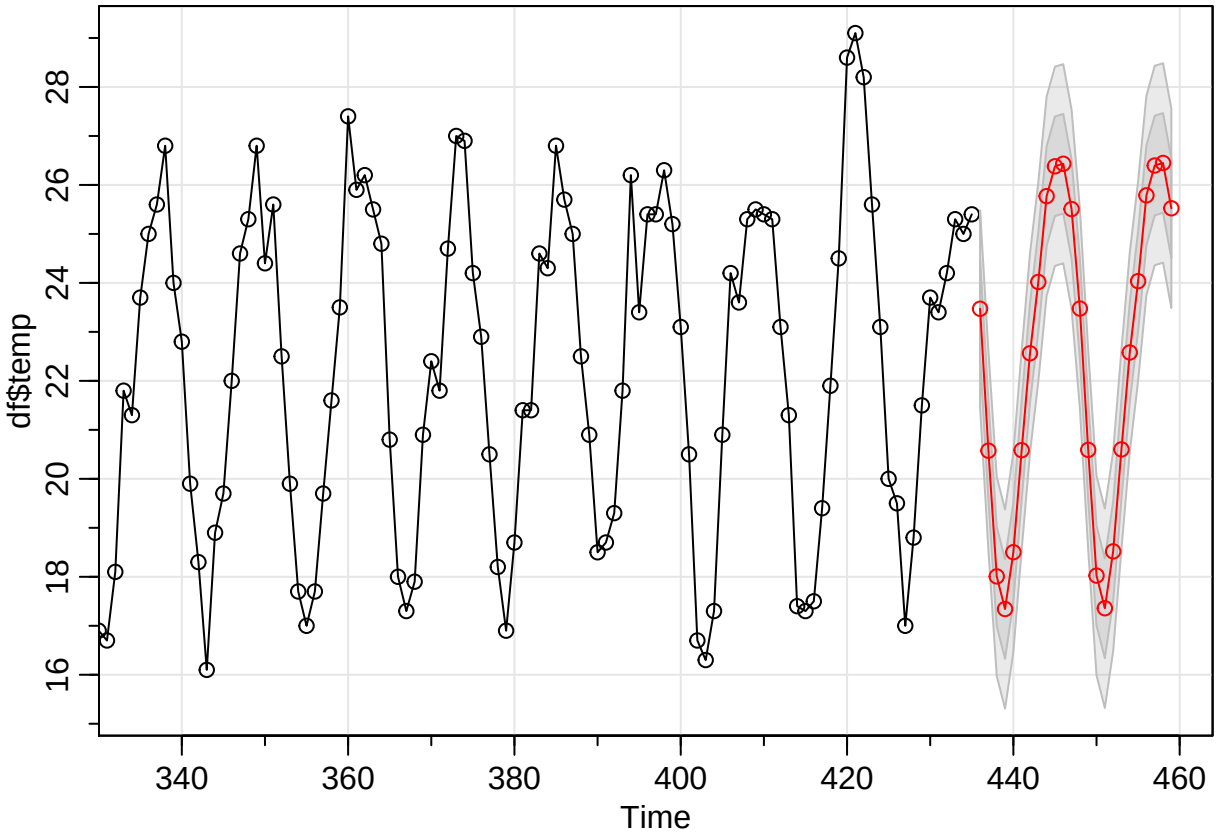

```
fit_temp = sarima(df$temp, 0, 1, 2, 0, 1,1, 12)
```

```
## initial  value 0.601152
## iter   2 value 0.264935
## iter   3 value 0.130484
## iter   4 value 0.113534
## iter   5 value 0.105862
## iter   6 value 0.103072
## iter   7 value 0.101977
## iter   8 value 0.101537
## iter   9 value 0.101523
## iter  10 value 0.101520
## iter  11 value 0.101520
## iter  12 value 0.101520
## iter  13 value 0.101520
## iter  13 value 0.101520
## iter  13 value 0.101520
## final  value 0.101520
## converged
## initial  value 0.084897
## iter   2 value 0.080821
## iter   3 value 0.050162
## iter   4 value 0.049971
## iter   5 value 0.049925
## iter   6 value 0.049826
## iter   7 value 0.049817
```

```
## iter   8 value 0.049797
## iter   9 value 0.049796
## iter  10 value 0.049796
## iter  11 value 0.049796
## iter  12 value 0.049796
## iter  12 value 0.049796
## iter  12 value 0.049796
## final  value 0.049796
## converged
```




```
pred_temp = sarima.for(df$temp,24,  0, 1, 2, 0, 1,1, 12)$pred
```

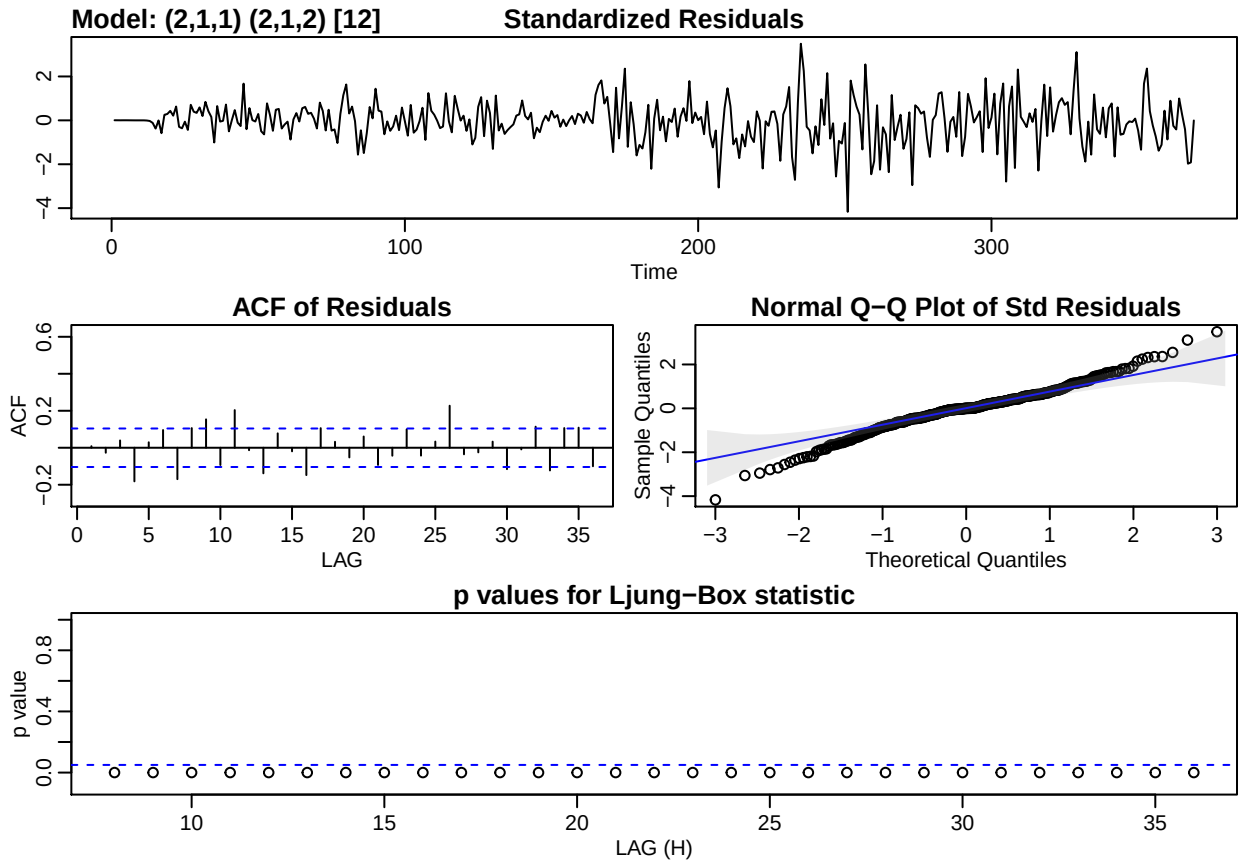


Forecasting target with exogenous variables for 24 steps ahead

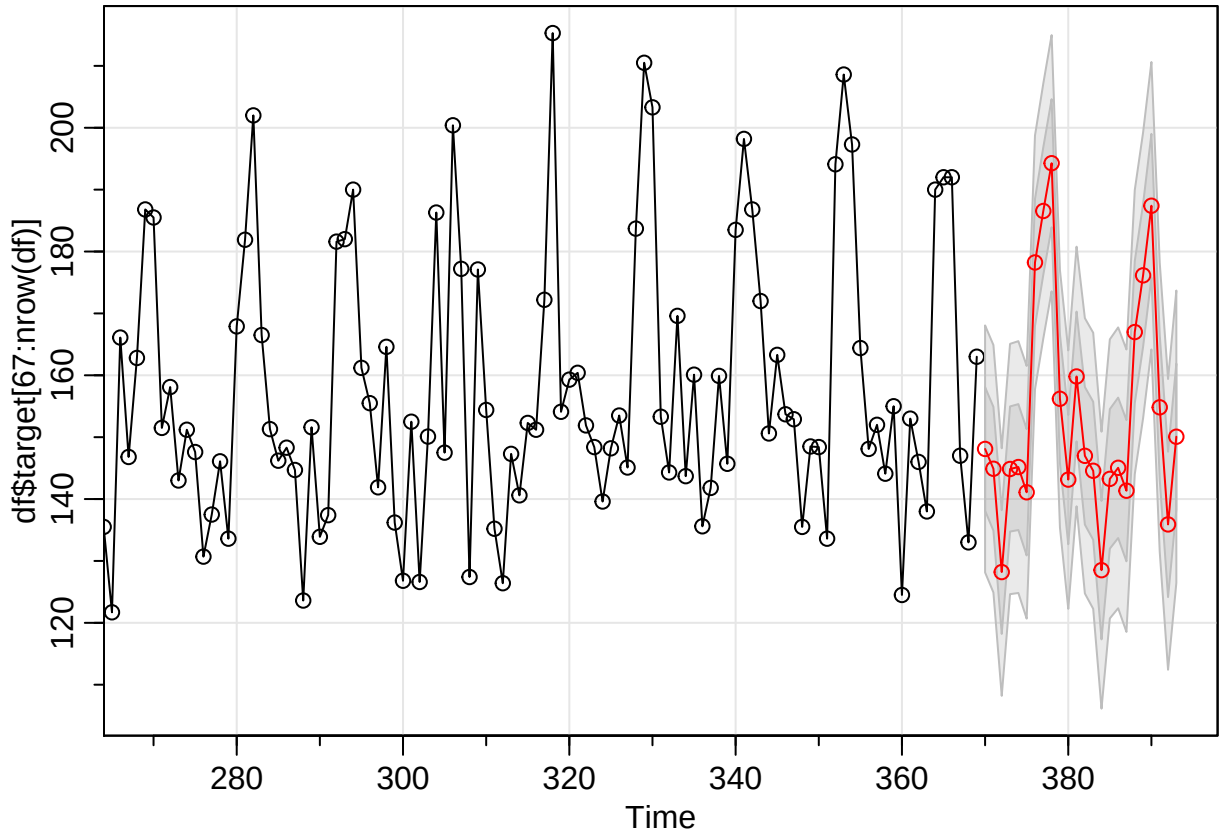
```
xreg= ts.intersect( C1=stats::lag(ts(c(df$car, pred_car)), -10), C2 = stats::lag(ts(c(df$temp, pred_temp

sar = sarima(df$target[67:nrow(df)], 2, 1, 1, 2, 1, 2, 12,  xreg=xreg[67:nrow(df)])
```

```
## initial  value 2.947891
## iter   2 value 2.508844
## iter   3 value 2.434201
## iter   4 value 2.359199
## iter   5 value 2.358100
## iter   6 value 2.340885
## iter   7 value 2.339512
## iter   8 value 2.338381
## iter   9 value 2.338360
## iter  10 value 2.338301
## iter  11 value 2.338289
## iter  12 value 2.338288
## iter  13 value 2.338281
## iter  14 value 2.338269
## iter  15 value 2.338231
## iter  16 value 2.338172
## iter  17 value 2.338076
## iter  18 value 2.337804
## iter  19 value 2.337797
```

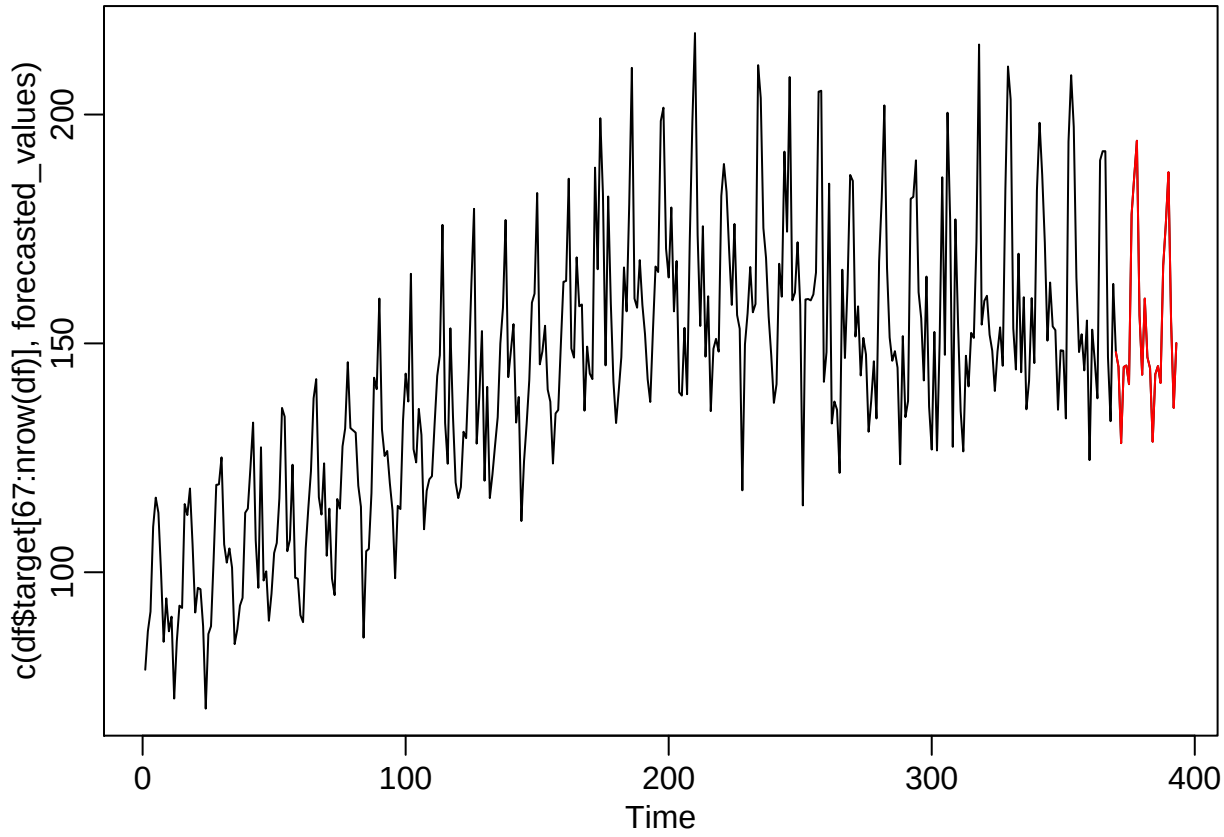
```
## iter  20 value 2.337761
## iter  21 value 2.337741
## iter  22 value 2.337739
## iter  23 value 2.337739
## iter  24 value 2.337739
## iter  25 value 2.337738
## iter  26 value 2.337738
## iter  27 value 2.337735
## iter  28 value 2.337733
## iter  29 value 2.337730
## iter  30 value 2.337730
## iter  31 value 2.337730
## iter  31 value 2.337730
## iter  31 value 2.337730
## final  value 2.337730
## converged
## initial  value 2.323065
## iter   2 value 2.321814
## iter   3 value 2.321409
## iter   4 value 2.321117
## iter   5 value 2.321058
## iter   6 value 2.321052
## iter   7 value 2.321052
## iter   8 value 2.321052
## iter   8 value 2.321052
## iter   8 value 2.321052
## final  value 2.321052
## converged
```




```
forecasted_values = sarima.for(df$target[67:nrow(df)], n.ahead = 24,  2, 1, 1, 2, 1, 2, 12, xreg=xreg[6
```



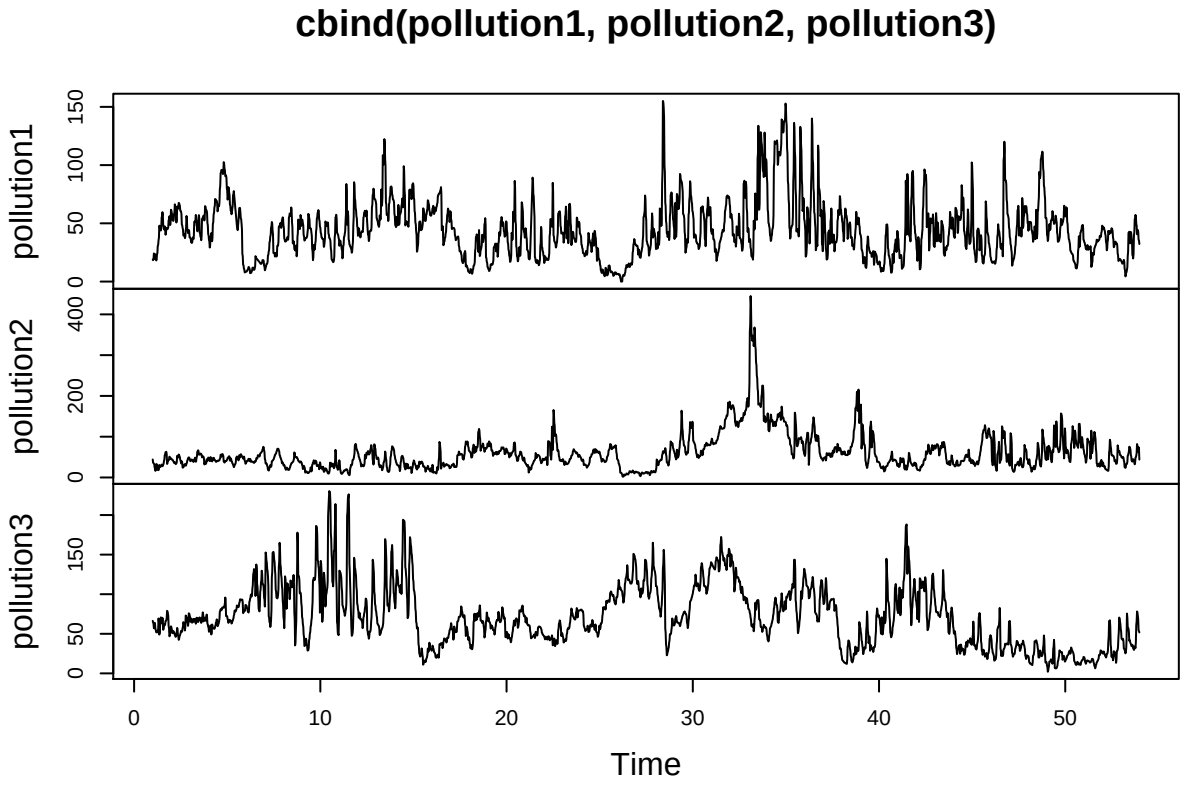

```
plot.ts(c(df$target[67:nrow(df)], forecasted_values) )
lines(forecasted_values, col = "red")
```



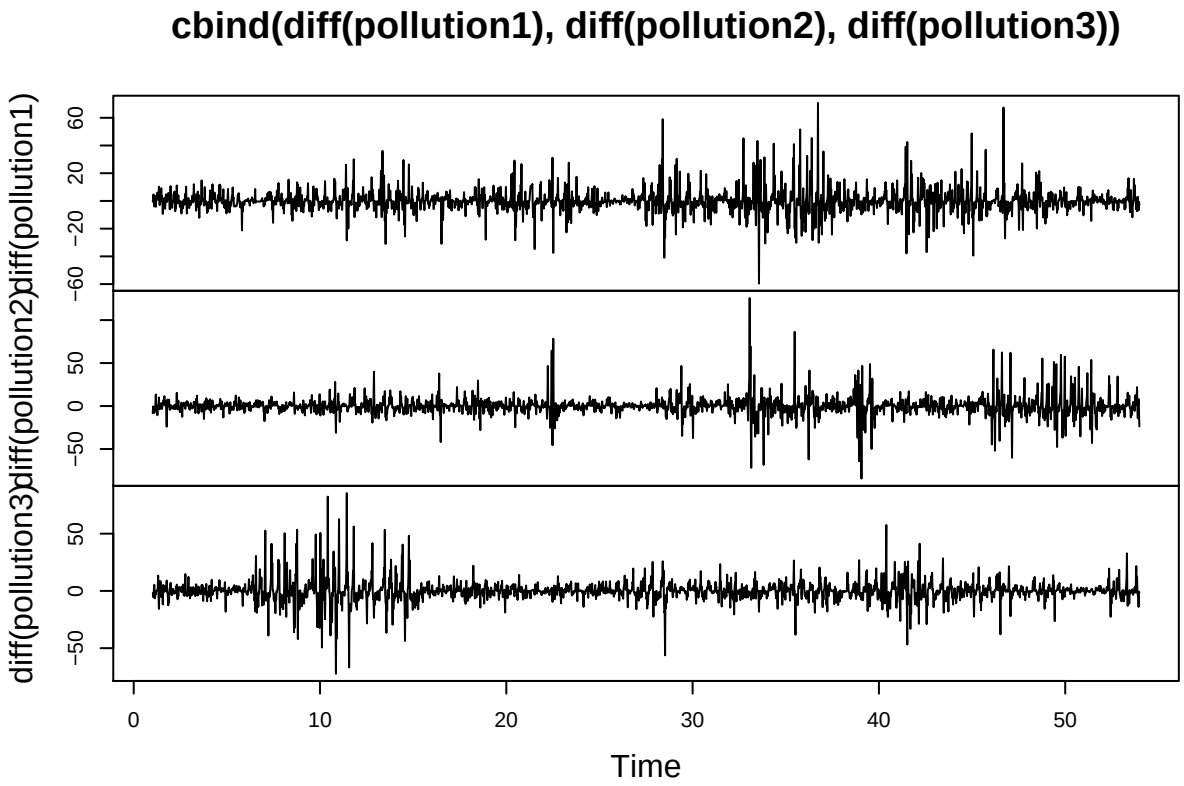

```
write.csv(file = "Hamzeian_Scenario4.csv", forecasted_values, row.names = F)
```

Scenario5

```
pollution1 = ts(read.csv("pollutionCity1.txt")$x, frequency = 48)
pollution2 = read.csv("pollutionCity2.txt")$x
pollution3 = read.csv("pollutionCity3.txt")$x
plot.ts(cbind(pollution1, pollution2, pollution3))
```



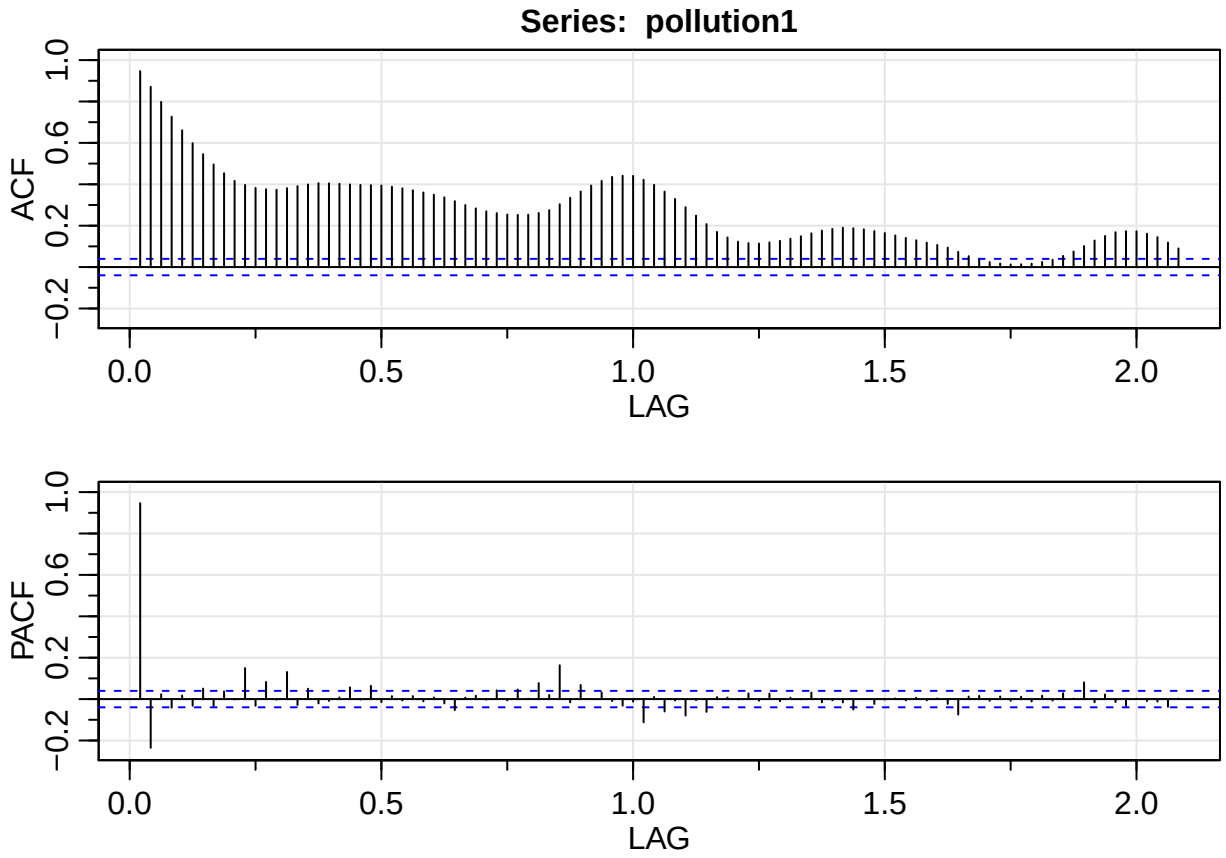

```
plot.ts(cbind(diff(pollution1), diff(pollution2), diff(pollution3)))
```



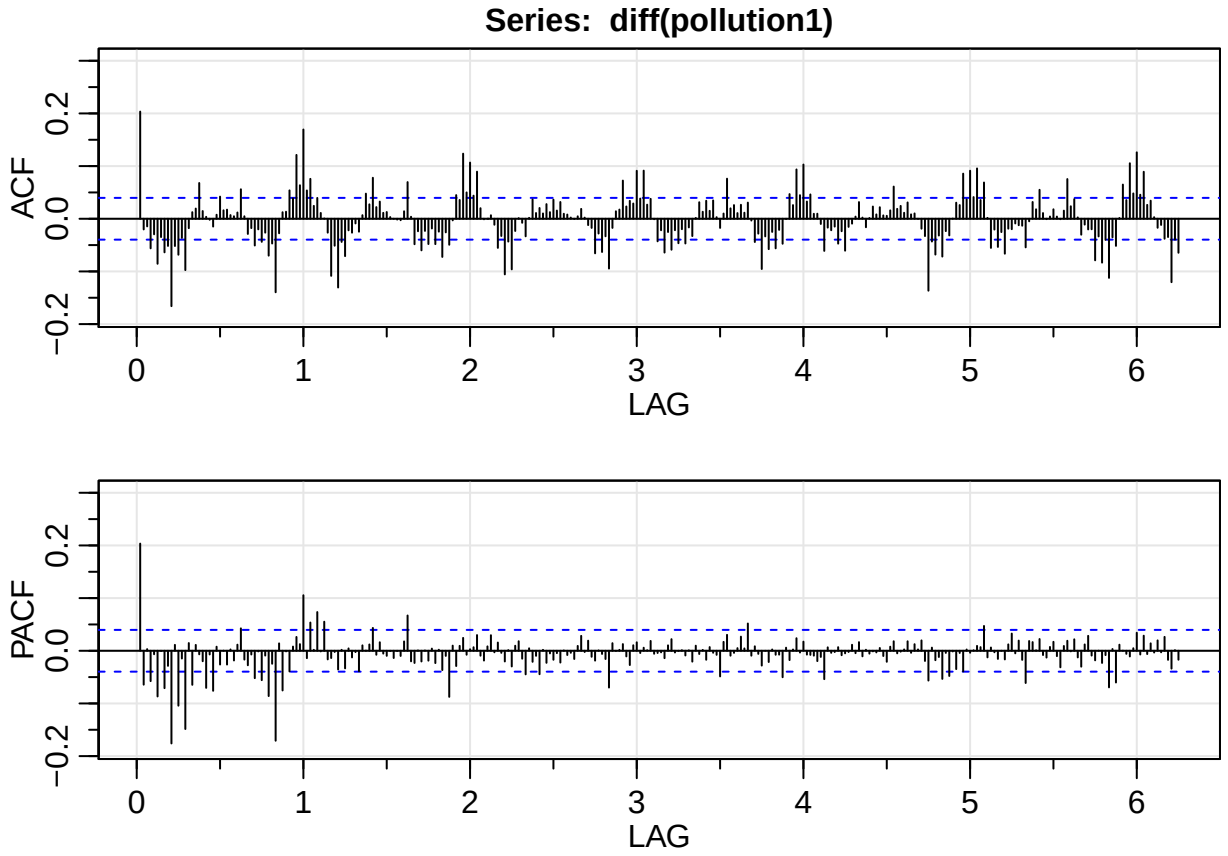
forecast for pollution1

For this scenario I used sarima. The ACF of the series show seasonality and/or trend, so I differenced it. The ACF of the differenced series show seasonality because at lags 48 the autocorrelations are big which makes sense because the data is half- hourly and the cycle is 24*2. Then I took difference of 48. To determine the parameters of the sarima model, I plotted the acf of the diff(diff(pollition1), 48). The seasonality part seems to be MA(ACF cuts off, and PACF trails off) and the non-seasonal component seem to be AR(PACF at seasonality lags cut off and ACF trails off). So, I tried different values and this is the best result that I obtained according to the AIC and significant coefficients.

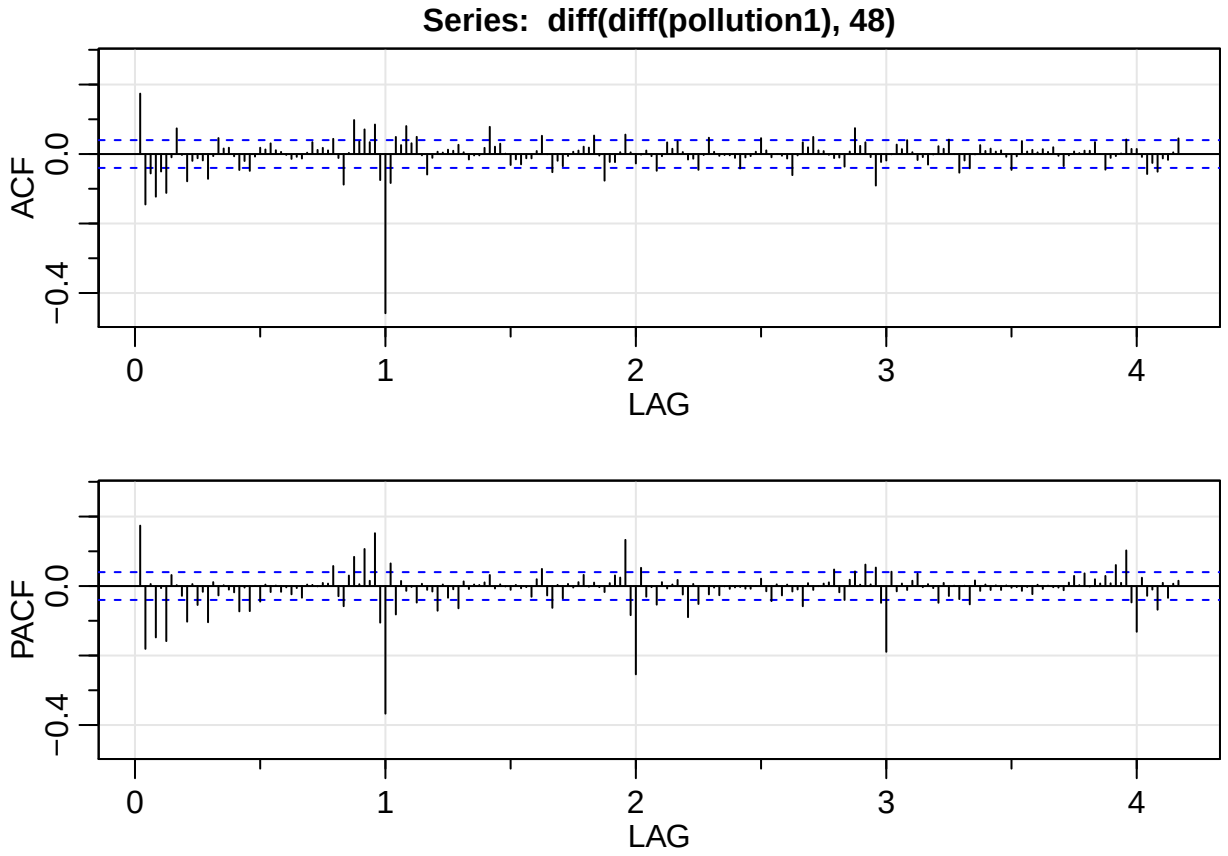
```
acf = acf2(pollution1, 100, plot = T)
```




```
acfx = acf2(diff(pollution1), 300, plot = T)
```



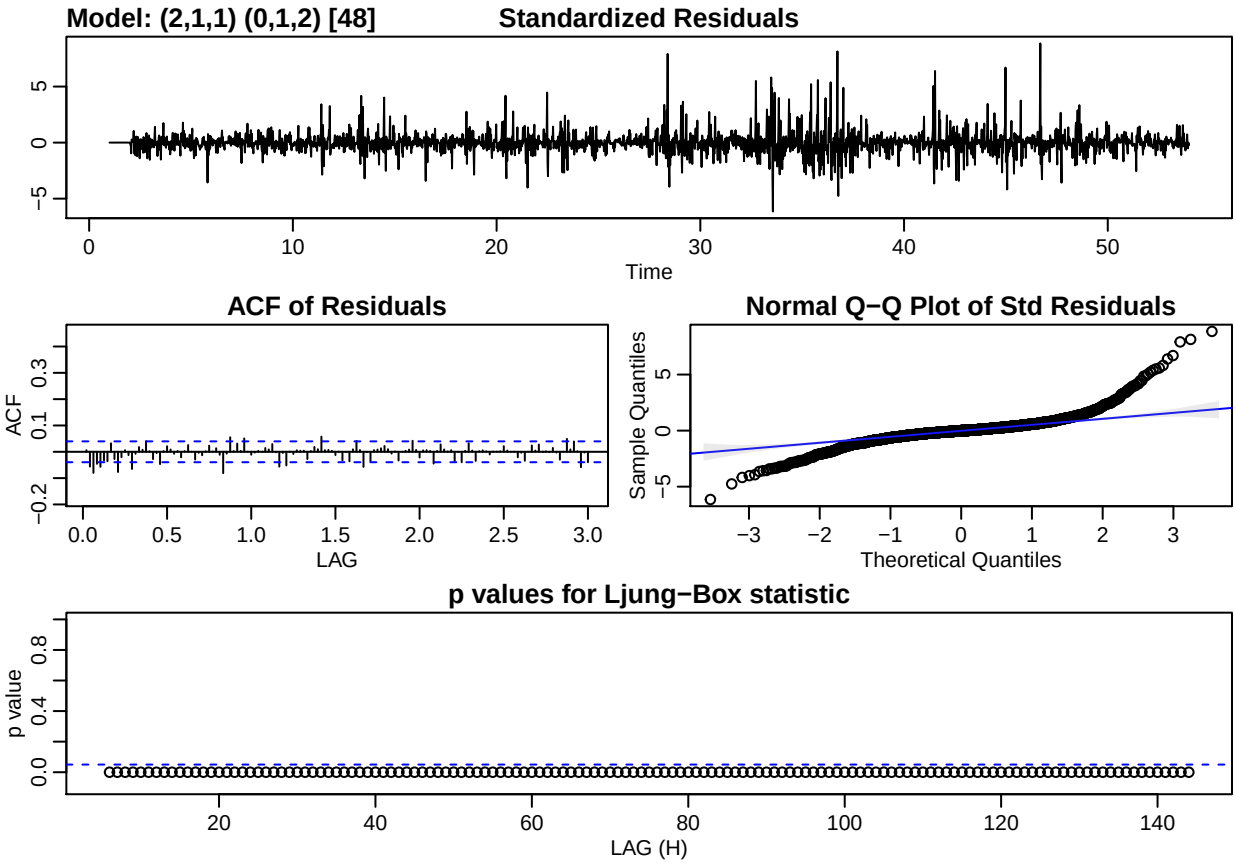

```
acfdiffx = acf2(diff(diff(pollution1), 48),  200, plot = T)
```



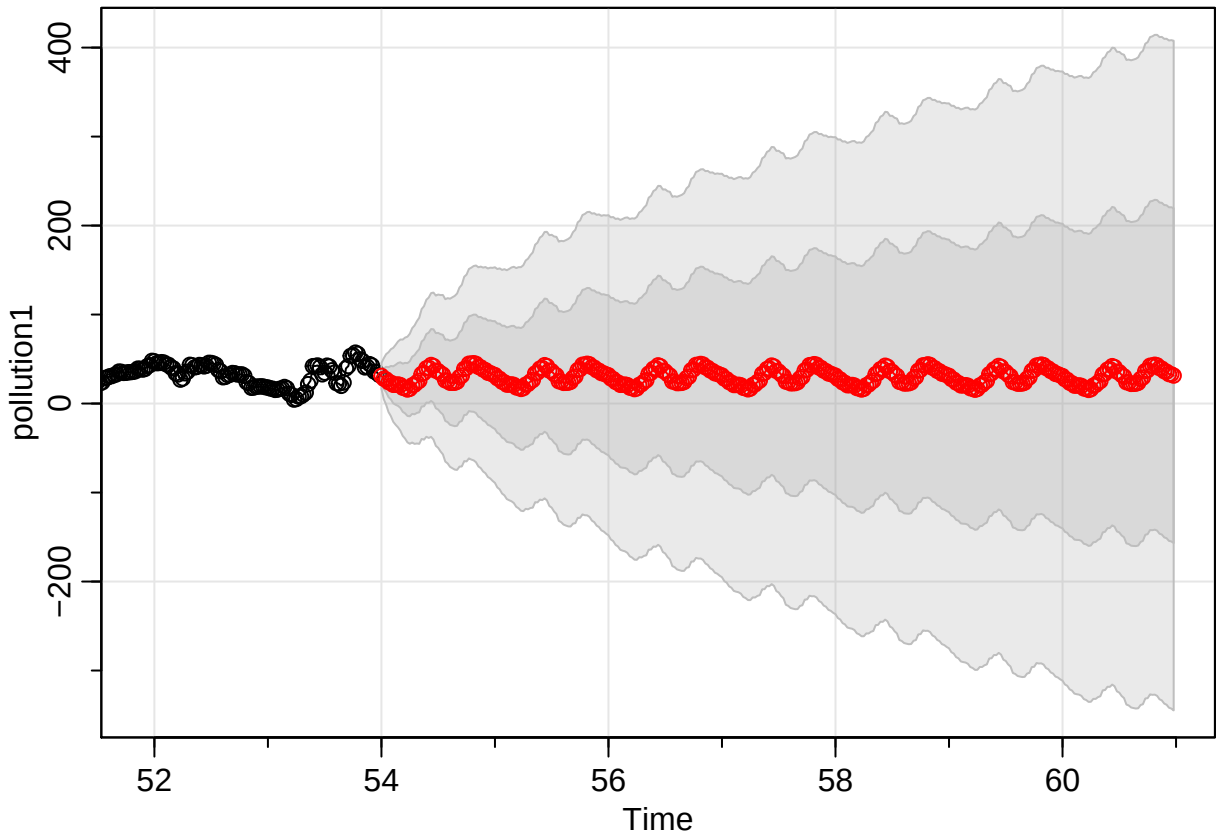

```
sar = sarima(pollution1, 2,1,1, 0,1,2,48)
```

```
## initial  value 2.324573
## iter   2 value 2.129345
## iter   3 value 2.064155
## iter   4 value 2.034019
## iter   5 value 2.022661
## iter   6 value 2.018838
## iter   7 value 2.016317
## iter   8 value 2.016239
## iter   9 value 2.016143
## iter  10 value 2.016135
## iter  11 value 2.016131
## iter  12 value 2.016123
## iter  13 value 2.016121
## iter  14 value 2.016119
## iter  15 value 2.016110
## iter  16 value 2.016104
## iter  17 value 2.016071
## iter  18 value 2.015935
## iter  19 value 2.015789
## iter  20 value 2.015610
## iter  21 value 2.015303
## iter  22 value 2.013614
## iter  23 value 2.011453
## iter  24 value 2.011068
```

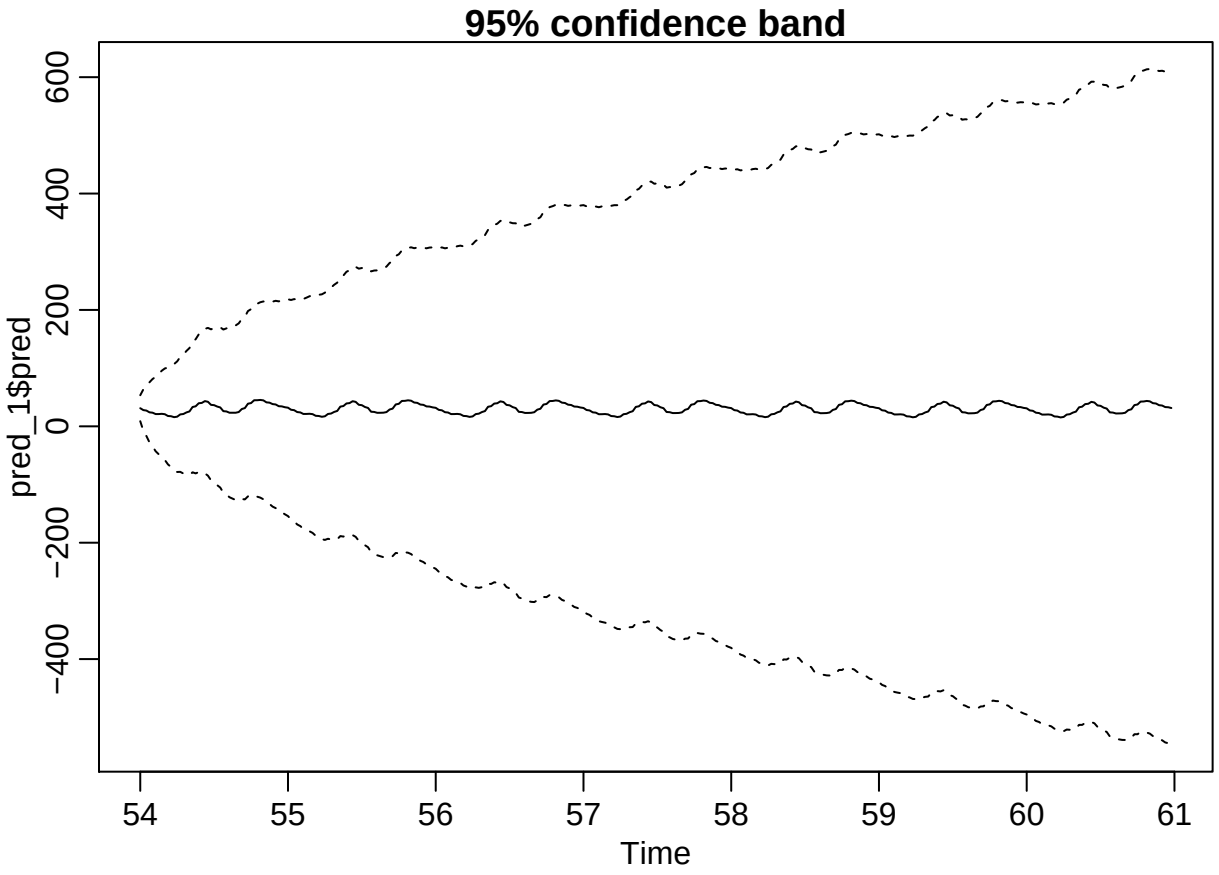
```
## iter  25 value 2.009625
## iter  26 value 2.009557
## iter  27 value 2.009433
## iter  28 value 2.009330
## iter  29 value 2.009296
## iter  30 value 2.009291
## iter  31 value 2.009290
## iter  32 value 2.009290
## iter  32 value 2.009290
## iter  32 value 2.009290
## final  value 2.009290
## converged
## initial  value 2.008086
## iter   2 value 2.004847
## iter   3 value 2.004780
## iter   4 value 2.004455
## iter   5 value 2.004307
## iter   6 value 2.004047
## iter   7 value 2.003940
## iter   8 value 2.003910
## iter   9 value 2.003907
## iter  10 value 2.003905
## iter  10 value 2.003905
## iter  10 value 2.003905
## final  value 2.003905
## converged
```



```
pred_1 = sarima.for(pollution1,n.ahead = 336, 2,1,1, 0,1,2,48)
```



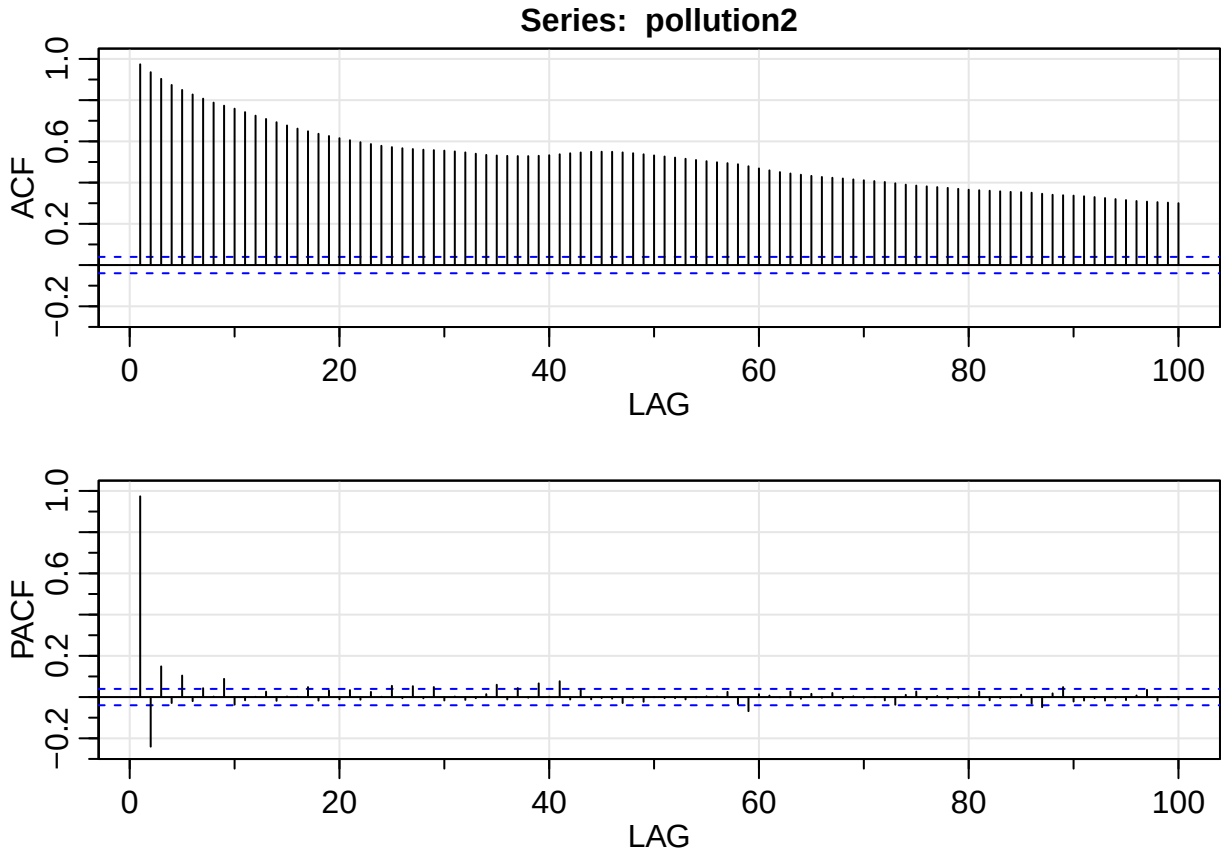

```
lower = pred_1$pred-3.075*pred_1$se
upper = pred_1$pred+3.075*pred_1$se
plot(pred_1$pred, ylim = c(min(lower), max(upper)), main= "95% confidence band")
lines(lower, lty= 2)
lines(upper, lty = 2)
```



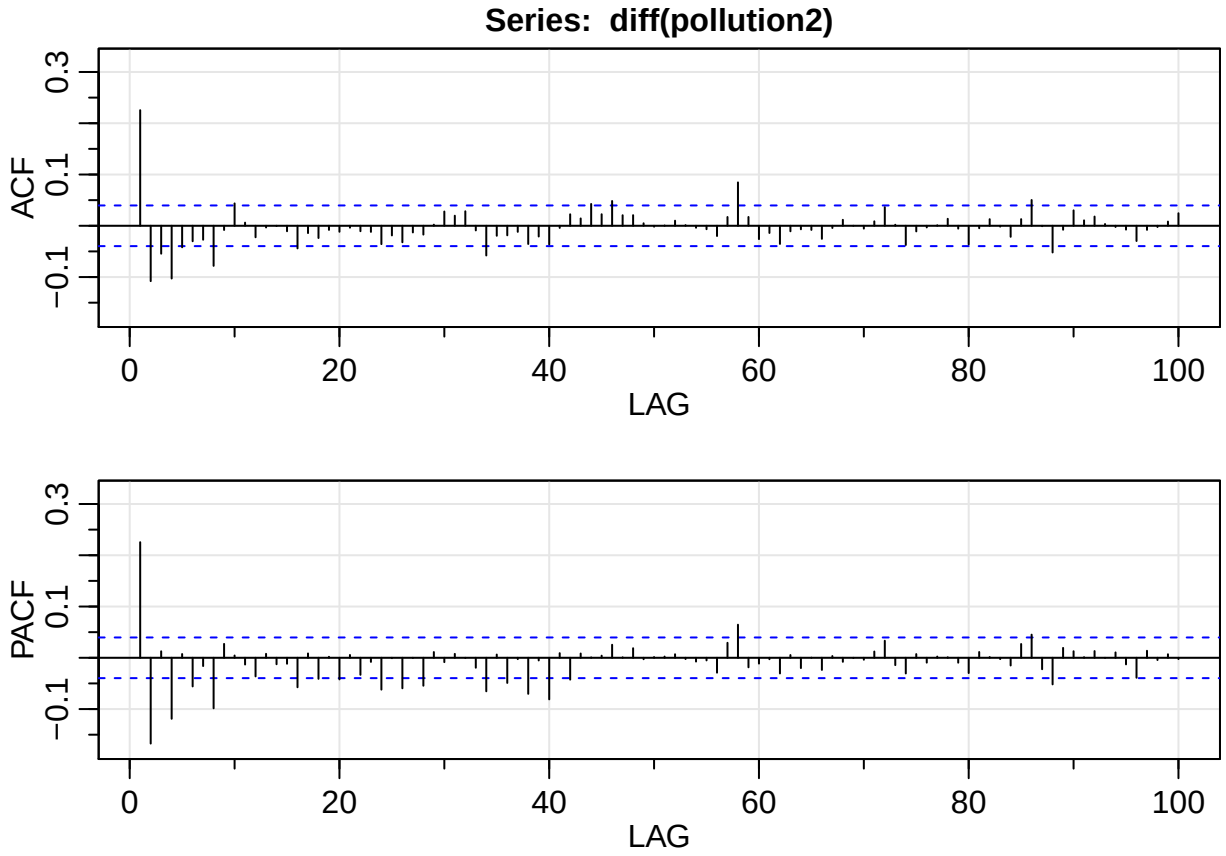
forecast for pollution2

The acf of pollution 2 shows trend, but after differencing it shows no seasonality. Based on the acf and pacf of the differenced series, we can say that it is an arma process. I changed values of p and q to obtain the least AIC with significant coefficients. The Ljung box is not good, but the residuals seem to be stationary.

```
acf = acf2(pollution2, 100, plot = T)
```



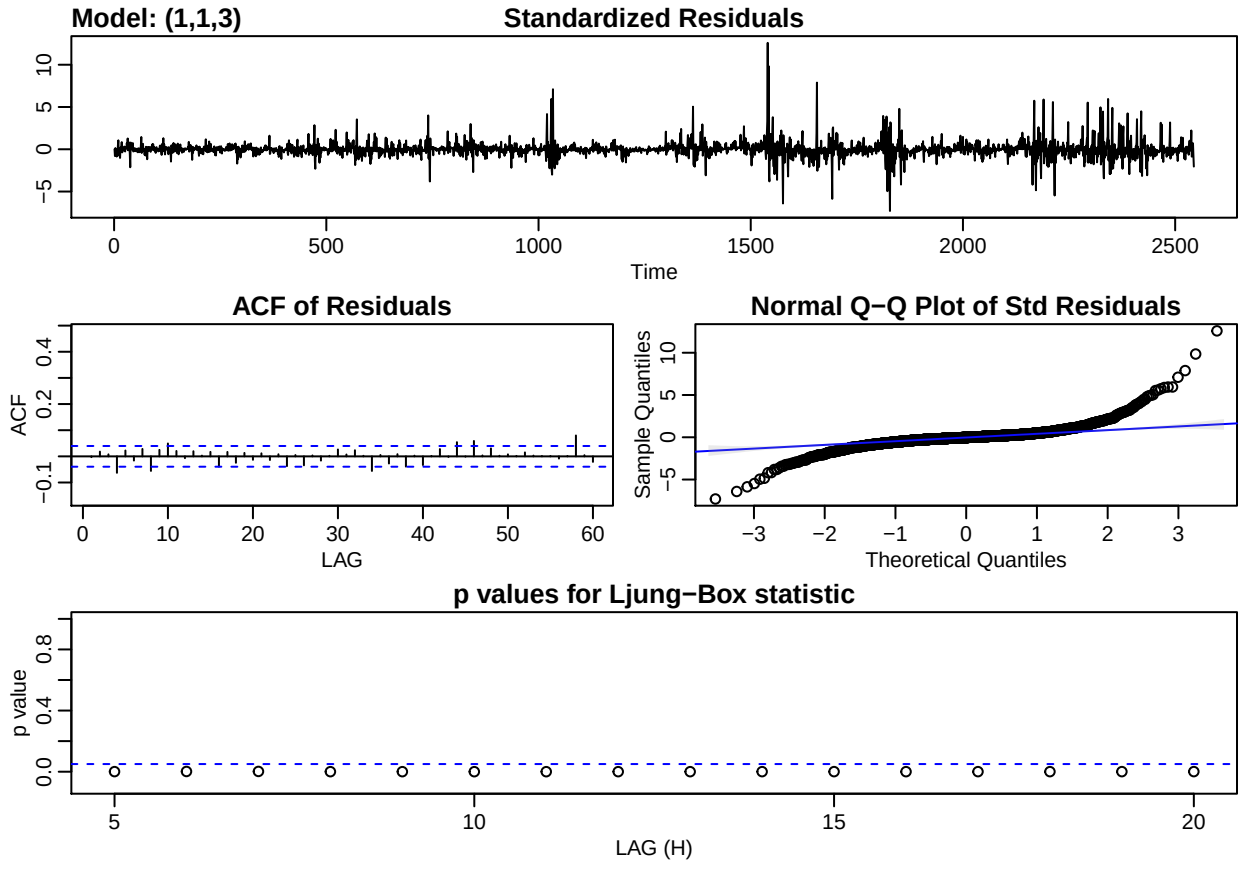

```
acfx = acf2(diff(pollution2), 100, plot = T)
```



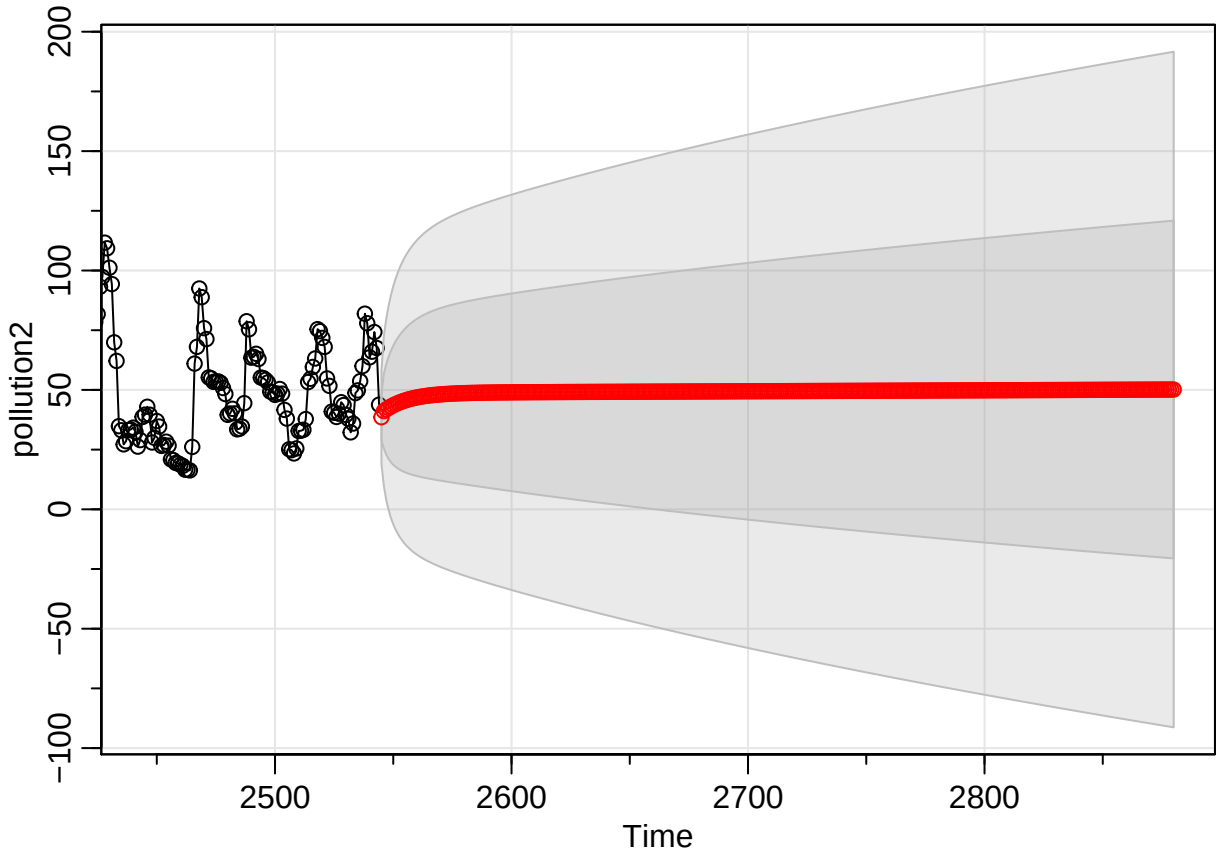

```
arma = sarima(pollution2, 1,1,3)
```

```
## initial  value 2.327532
## iter   2 value 2.303192
## iter   3 value 2.284030
## iter   4 value 2.283783
## iter   5 value 2.283770
## iter   6 value 2.283762
## iter   7 value 2.283739
## iter   8 value 2.283674
## iter   9 value 2.283412
## iter  10 value 2.283264
## iter  11 value 2.283111
## iter  12 value 2.282695
## iter  13 value 2.281391
## iter  14 value 2.281171
## iter  15 value 2.280705
## iter  16 value 2.280628
## iter  17 value 2.278839
## iter  18 value 2.278259
## iter  19 value 2.277661
## iter  20 value 2.276905
## iter  21 value 2.276031
## iter  22 value 2.275592
## iter  23 value 2.275297
## iter  24 value 2.275061
```

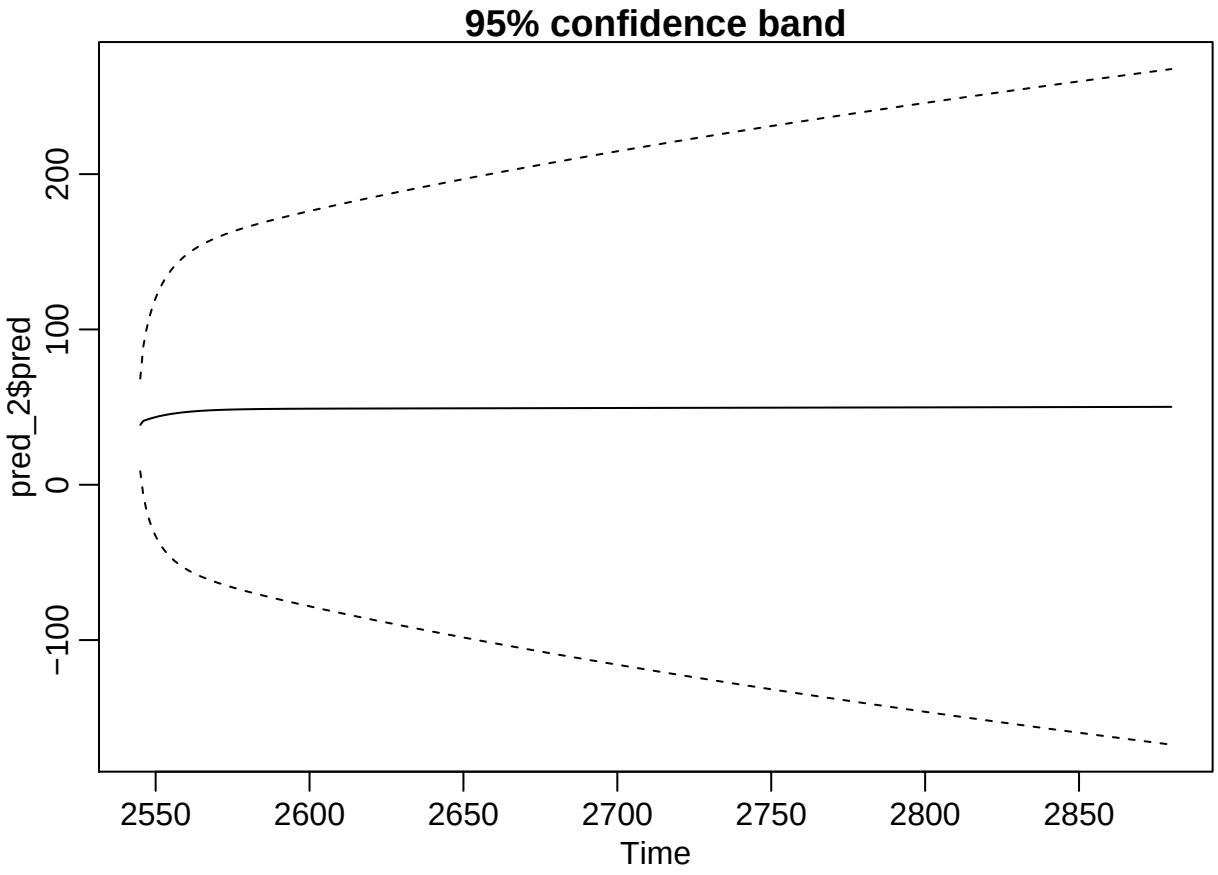
```
## iter  25 value 2.274871
## iter  26 value 2.274770
## iter  27 value 2.274732
## iter  28 value 2.274732
## iter  28 value 2.274732
## iter  28 value 2.274732
## final  value 2.274732
## converged
## initial  value 2.273892
## iter   2 value 2.273862
## iter   3 value 2.273844
## iter   4 value 2.273774
## iter   5 value 2.273742
## iter   6 value 2.273693
## iter   7 value 2.273671
## iter   8 value 2.273658
## iter   9 value 2.273657
## iter  10 value 2.273657
## iter  10 value 2.273657
## iter  10 value 2.273657
## final  value 2.273657
## converged
```




```
pred_2 = sarima.for(pollution2,n.ahead = 336,  1,1,3)
```



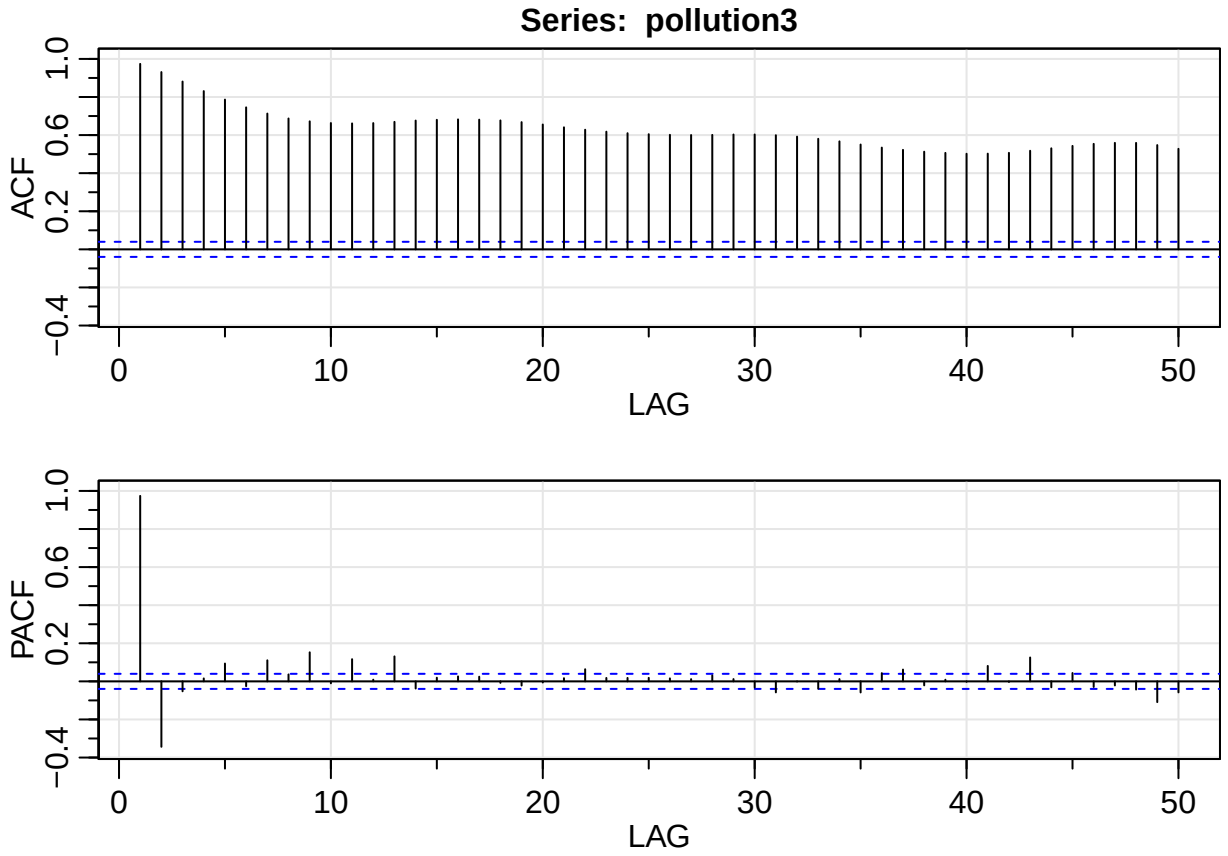

```
lower = pred_2$pred-3.075*pred_2$se
upper = pred_2$pred+3.075*pred_2$se
plot(pred_2$pred, ylim = c(min(lower), max(upper)), main= "95% confidence band")
lines(lower, lty= 2)
lines(upper, lty = 2)
```



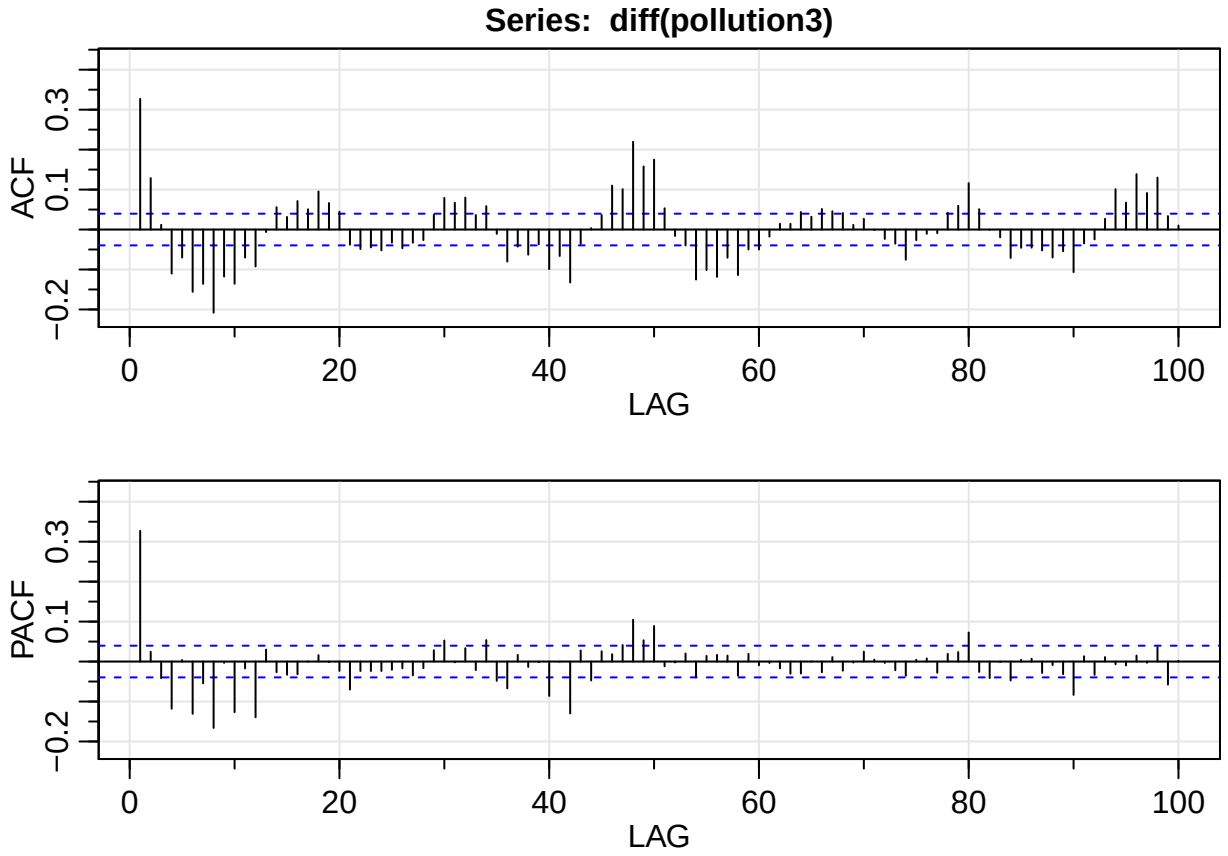
Forecast for pollition3

The ACF of the series shows trend and after differencing the acf shows seasonality of 48. The acf of diff(diff(pollution3), 48) cuts off at seasonality lags so the seasonality component is MA. However, the non-seasonal component is ARMA. I tried different p, q's to find the minimum AIC with significant coefficients.

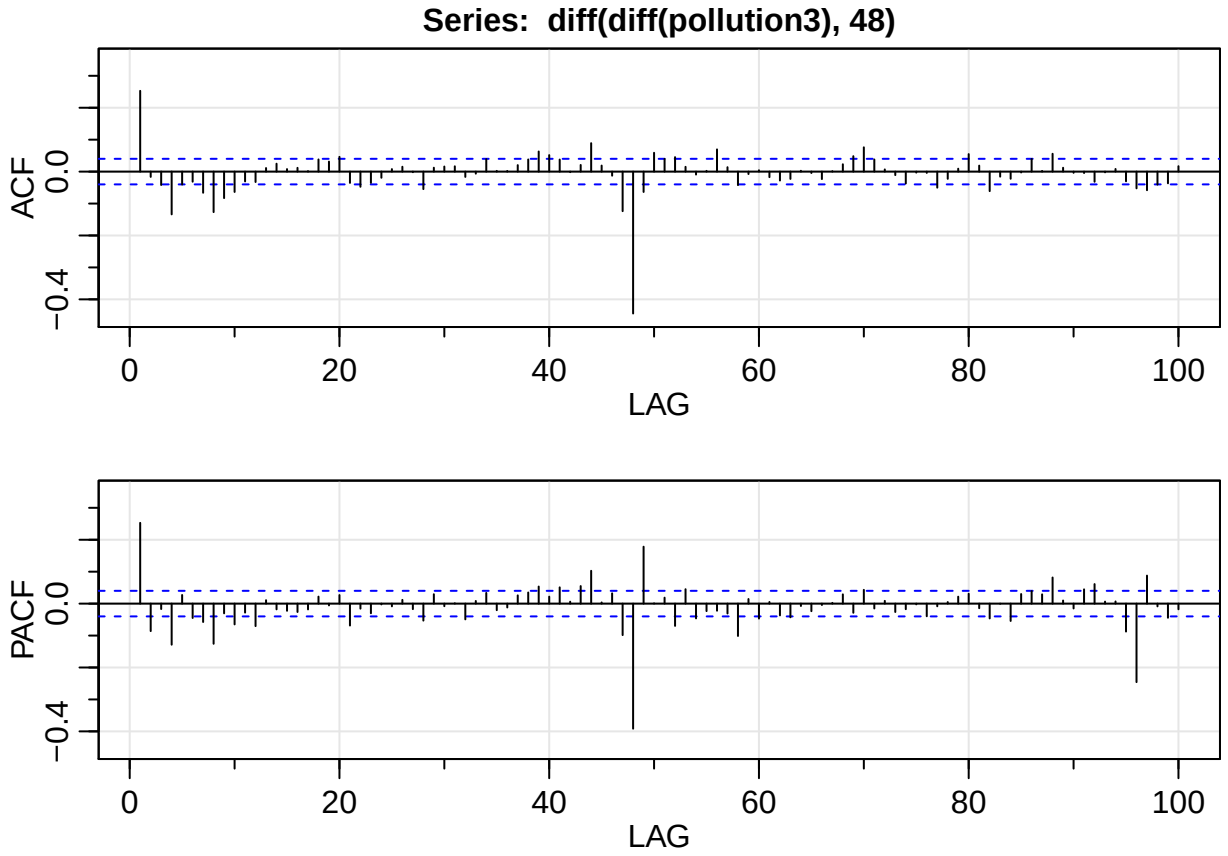
```
acf = acf2(pollution3, 50, plot = T)
```




```
acfx = acf2(diff(pollution3), 100, plot = T)
```



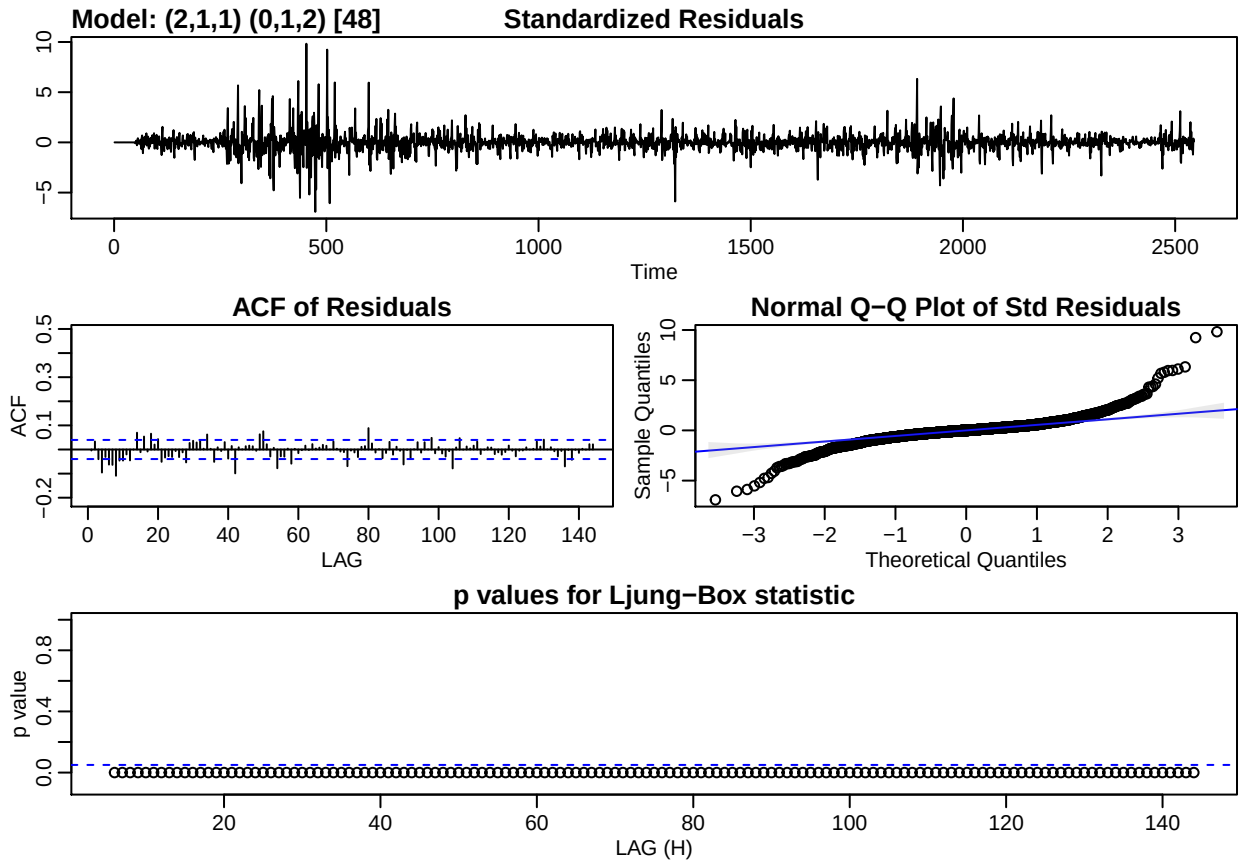

```
acfdiffx = acf2(diff(diff(pollution3), 48), 100, plot = T)
```



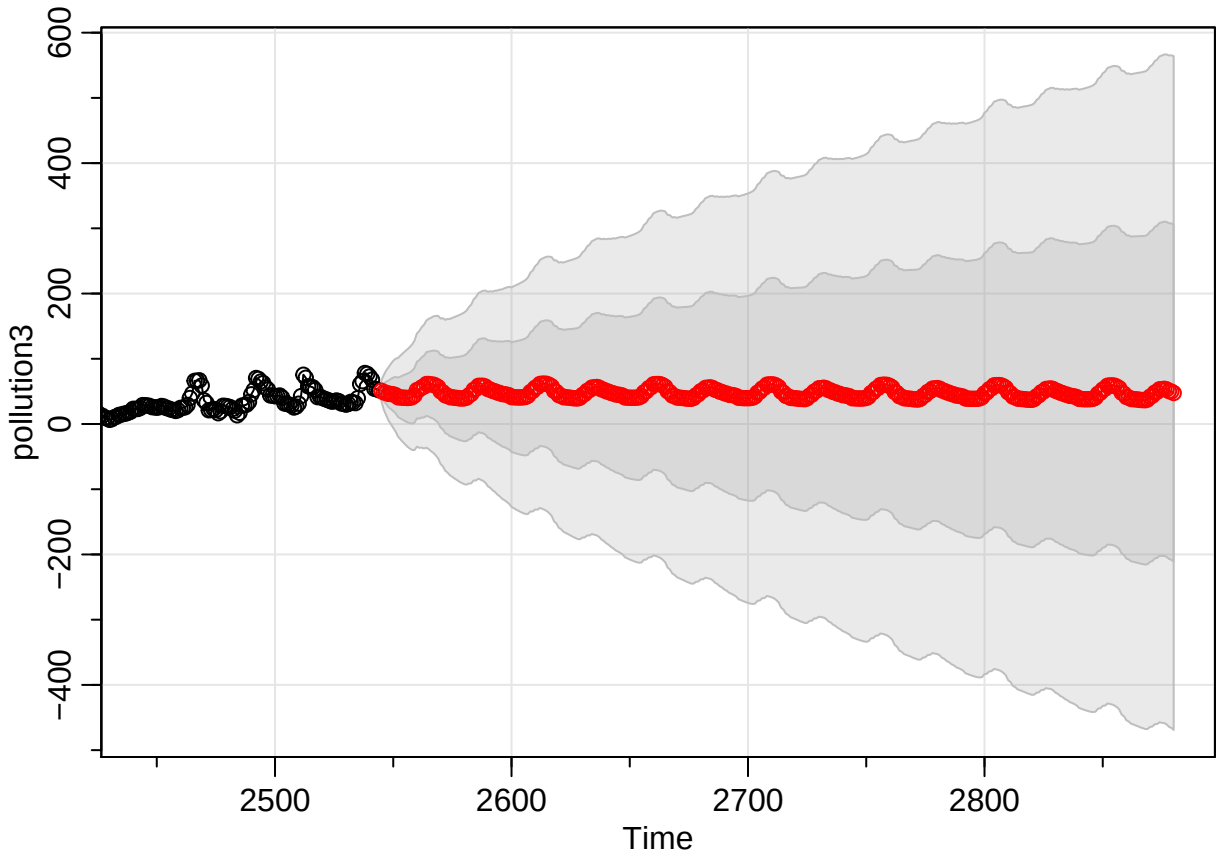

```
sar = sarima(pollution3, 2,1,1, 0,1,2,48)
```

```
## initial  value 2.395639
## iter   2 value 2.215256
## iter   3 value 2.172940
## iter   4 value 2.125796
## iter   5 value 2.119200
## iter   6 value 2.112532
## iter   7 value 2.112080
## iter   8 value 2.111933
## iter   9 value 2.111928
## iter  10 value 2.111926
## iter  11 value 2.111925
## iter  12 value 2.111925
## iter  13 value 2.111925
## iter  14 value 2.111925
## iter  15 value 2.111924
## iter  16 value 2.111923
## iter  17 value 2.111919
## iter  18 value 2.111905
## iter  19 value 2.111796
## iter  20 value 2.111553
## iter  21 value 2.111488
## iter  22 value 2.111470
## iter  23 value 2.111446
## iter  24 value 2.111433
```

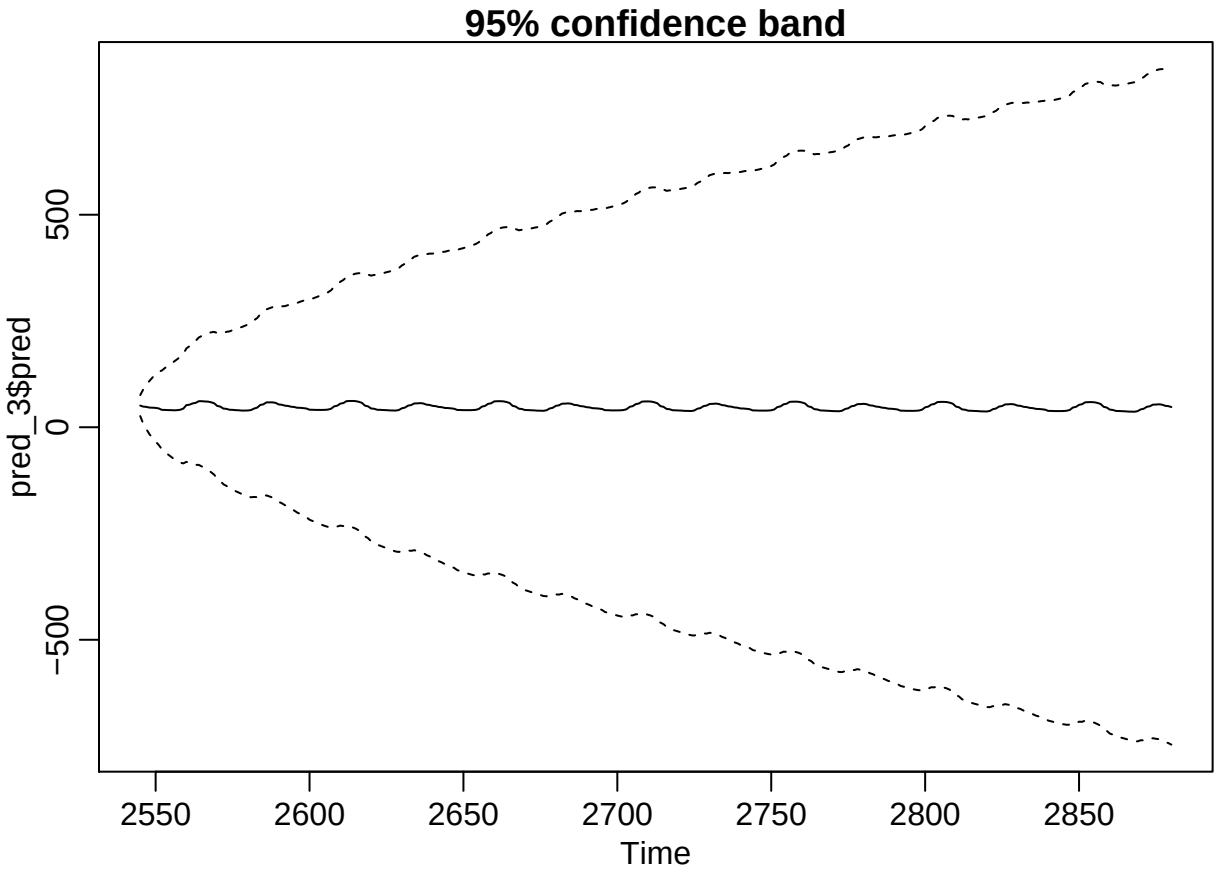
```
## iter  25 value 2.110249
## iter  26 value 2.110030
## iter  27 value 2.109867
## iter  28 value 2.109569
## iter  29 value 2.109409
## iter  30 value 2.107013
## iter  31 value 2.106596
## iter  32 value 2.105340
## iter  33 value 2.104940
## iter  34 value 2.104292
## iter  35 value 2.103799
## iter  36 value 2.103490
## iter  37 value 2.103450
## iter  38 value 2.103448
## iter  39 value 2.103447
## iter  39 value 2.103447
## iter  39 value 2.103447
## final  value 2.103447
## converged
## initial  value 2.103263
## iter   2 value 2.101574
## iter   3 value 2.101476
## iter   4 value 2.101467
## iter   5 value 2.101457
## iter   6 value 2.101450
## iter   7 value 2.101448
## iter   8 value 2.101438
## iter   9 value 2.101433
## iter  10 value 2.101431
## iter  11 value 2.101431
## iter  11 value 2.101431
## iter  11 value 2.101431
## final  value 2.101431
## converged
```




```
pred_3 = sarima.for(pollution3,n.ahead = 336, 2,1,1, 0,1,2,48)
```




```
lower = pred_3$pred-3.075*pred_3$se
upper = pred_3$pred+3.075*pred_3$se
plot(pred_3$pred, ylim = c(min(lower), max(upper)), main= "95% confidence band")
lines(lower, lty= 2)
lines(upper, lty = 2)
```




```
write.csv(file = "hamzeian_Scenario5.csv", data.frame(city1 = pred_1$pred, city2 = pred_2$pred, city3 =
```