MSCA 31006 Time Series Final Project - Divvy

Hyejeong Lee, Kunal Shukla, WanQi Tay, Yingkun Zhu

August 22, 2018

A. Data Preparation

Original Tutorial: https://cran.r-project.org/web/packages/bikedata/vignettes/bikedata.html#3_downloading_data

Load required packages (install first if necessary)

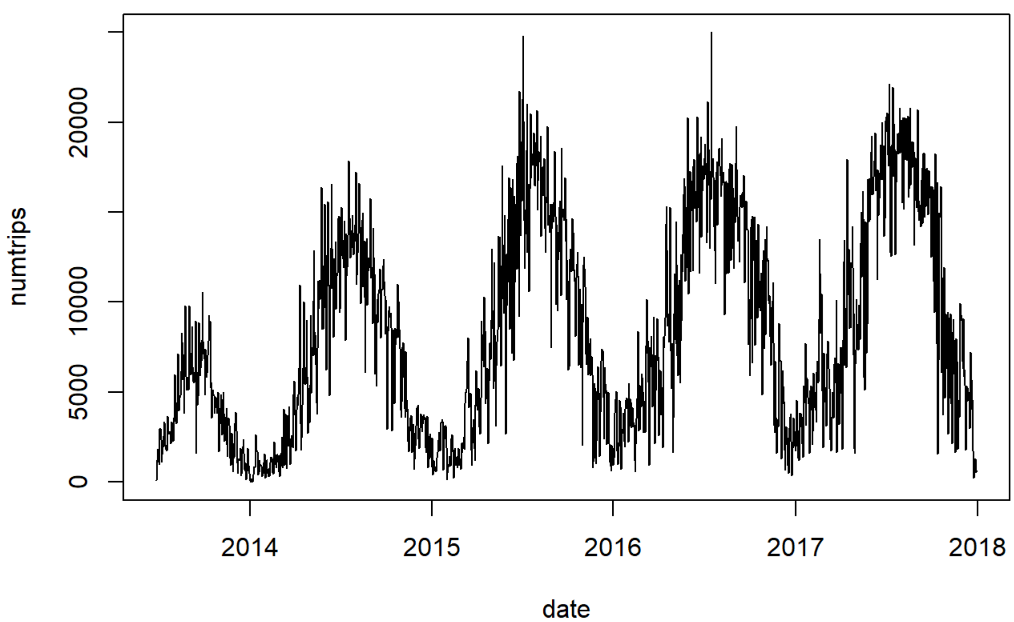
```
library(bikedata)
library(RSQLite)
library(tseries)
library(xts)
library(forecast)
library(ggplot2)
library(tibble)
library(expsmooth)
library(vars)
library(TSA)
library(dplyr)
library(foreach)
library(imputeTS)
```

```
## Load this image to skip running all script below.
load("Divvy_Project_Summer2018.RData")
```

Download and import Divvy Trips data

Examine daily number of trips

```
plot(time.series, type = "l")
```



Divvy Trips dataset has missing values - 2014/1/7 and 2014/1/8. We substitue the missing values by using the average value of 2014/01/06 and 2014/01/09

```
time.series2 = add_row(time.series, date = c(as.Date("2014-01-07"), as.Date("2014-01-08")), numtrips = c((time.series[[194
,2]]+time.series[[195,2]])/2, (time.series[[194,2]]+time.series[[195,2]])/2), .after = 194)
```

Remove leap year day for simplicity

```
which(time.series2$date == "2016/02/29")
```

```
## [1] 978
```

```
time.series2 = time.series2[-c(978),]
```

Combine and import Divvy Stations data

```
setwd("D:/1 UOC/1 Summer 2018/MSCA 31006 Time Series Analysis and Forecasting/1 Project/")
filenames = dir("D:/1 UOC/1 Summer 2018/MSCA 31006 Time Series Analysis and Forecasting/1 Project/")
stations.data = lapply(filenames[grep("Divvy_Stations_2", filenames)], read.csv)
```

```
stations.data.combined = stations.data[[1]]

for(i in 2:7) {
  stations.data.combined = merge(stations.data.combined, stations.data[[i]], all.y = TRUE)
}
```

```
# Reformat online_date column to date %m/%d/%Y format
stations.data.combined$online_date = as.Date(stations.data.combined$online_date, format = "%m/%d/%Y")
```

Create data frame to record total number of active stations on a given date

```
x = table(stations.data.combined$online_date)
stations.info = data.frame(Date = x, Cumulative = cumsum(as.vector(x)))
stations.info$Date.Var1 = as.Date(stations.info$Date.Var1, format = '%Y-%m-%d')
head(stations.info)
```

```
##    Date.Var1 Date.Freq Cumulative
## 1 2013-06-10         5          5
## 2 2013-06-19         1          6
## 3 2013-06-21         3          9
## 4 2013-06-22        23         32
## 5 2013-06-24         1         33
## 6 2013-06-25        33         66
```

Combine stations and trips data into data frame - divvy

```
divvy = as.data.frame(time.series2)

for (i in 1:nrow(time.series2)){
  for(j in 1:(nrow(stations.info)-1))  {
        if(divvy[i,1] ==  stations.info$Date.Var1[j]) {
            divvy[i,3] = stations.info$Cumulative[j]
        } else if (divvy[i,1] > stations.info$Date.Var1[j] && divvy[i,1] < stations.info$Date.Var1[j+1]) {
              divvy[i,3] = stations.info$Cumulative[j]
          } else if (divvy[i,1] >  stations.info$Date.Var1[j]) {
                divvy[i,3] = stations.info$Cumulative[nrow(stations.info)]
        }
  }
}
head(divvy)
```

```
##         date numtrips V3
## 1 2013-06-27       95 84
## 2 2013-06-28      897 85
## 3 2013-06-29     1201 85
## 4 2013-06-30     1812 86
## 5 2013-07-01     1559 86
## 6 2013-07-02     1108 86
```

Compute the average number of trips for each day and convert the dataframe into time series

Average Number of Trips =

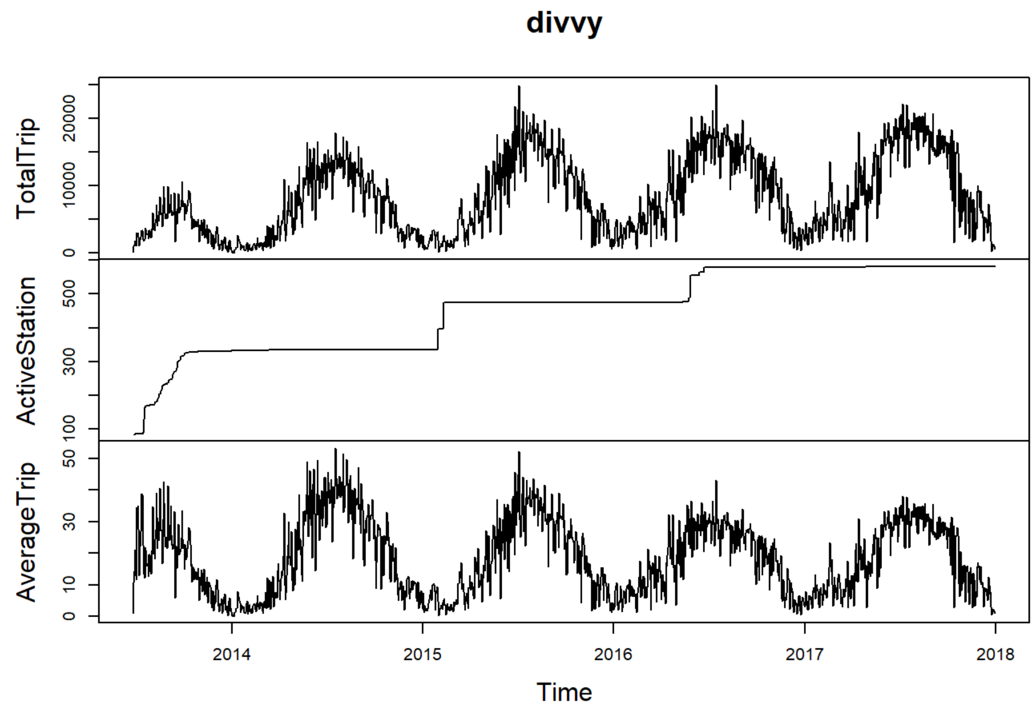
$$\frac{Total\ Number\ of\ Trips_{i=date}}{Number\ of\ Active\ Stations_{i=date}}$$

```
divvy[,4] = divvy[,2]/divvy[,3]
colnames(divvy) = c("Date", "TotalTrip", "ActiveStation", "AverageTrip")

# Convert data frame into time series
divvy = ts(divvy[,2:4], start = c(2013,178), frequency = 365)
head(divvy)
```

```
## Time Series:
## Start = c(2013, 178)
## End = c(2013, 183)
## Frequency = 365
##          TotalTrip ActiveStation AverageTrip
## 2013.485        95            84    1.130952
## 2013.488       897            85   10.552941
## 2013.490      1201            85   14.129412
## 2013.493      1812            86   21.069767
## 2013.496      1559            86   18.127907
## 2013.499      1108            86   12.883721
```

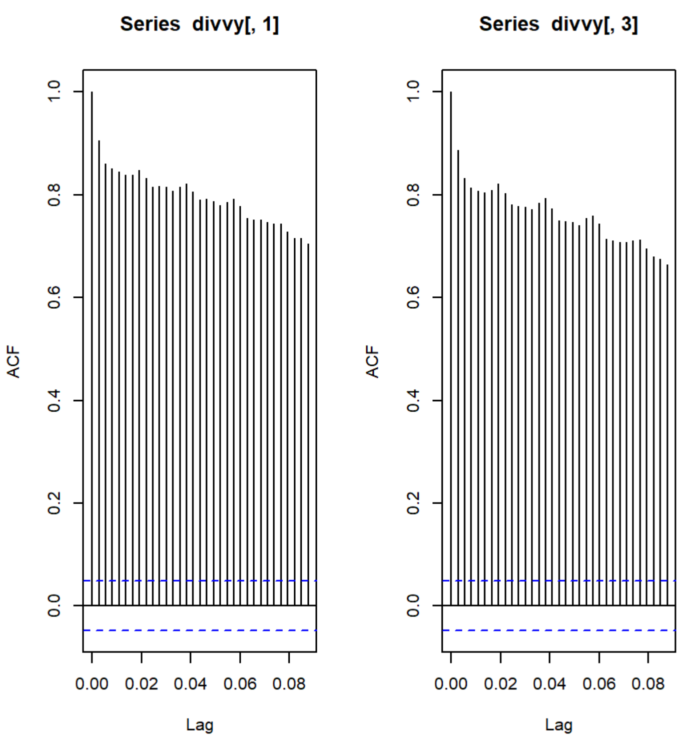
B. Data Analysis

```
plot(divvy)
```



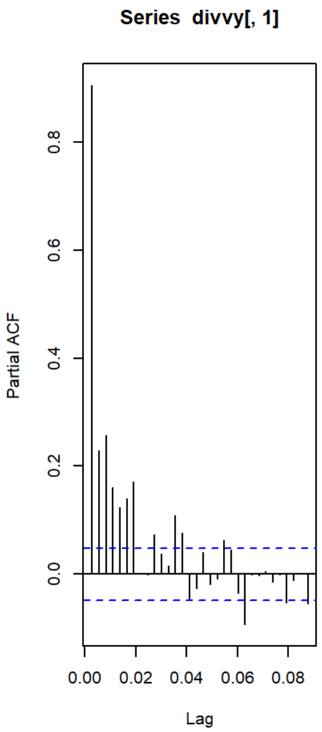
The Divvy total trip data is showing an upward trend with multiplicative seasonality. However, the average trip data is showing a downward trend with multiplicative seasonality. This indicates that the bike rental demand per station is actually decreasing as more stations are being added.

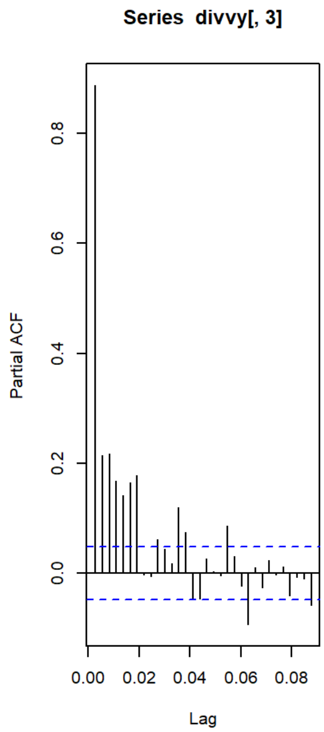
```
par(mfrow = c(1,3))
stats::acf(divvy[,1]) # Total Trip
stats::acf(divvy[,3]) # Average Trip
```



Both acf plots show an oscillation which indicates that both data is seasonal and non-stationary.

```
par(mfrow = c(1,3))
stats::pacf(divvy[,1]) # Total Trip
stats::pacf(divvy[,3]) # Average Trip
```





Both PACF plots have too many significant spikes, it is hard to determine which lag value or AR model to use.

Split data into Train and Test Sets

```
train.set = ts(divvy[1:1500,], frequency = 365, start = c(2013, 178), end = c(2017, 217))
test.set = ts(divvy[1501:1648,], frequency = 365, start = c(2017, 218))
```

C. Data Normalization

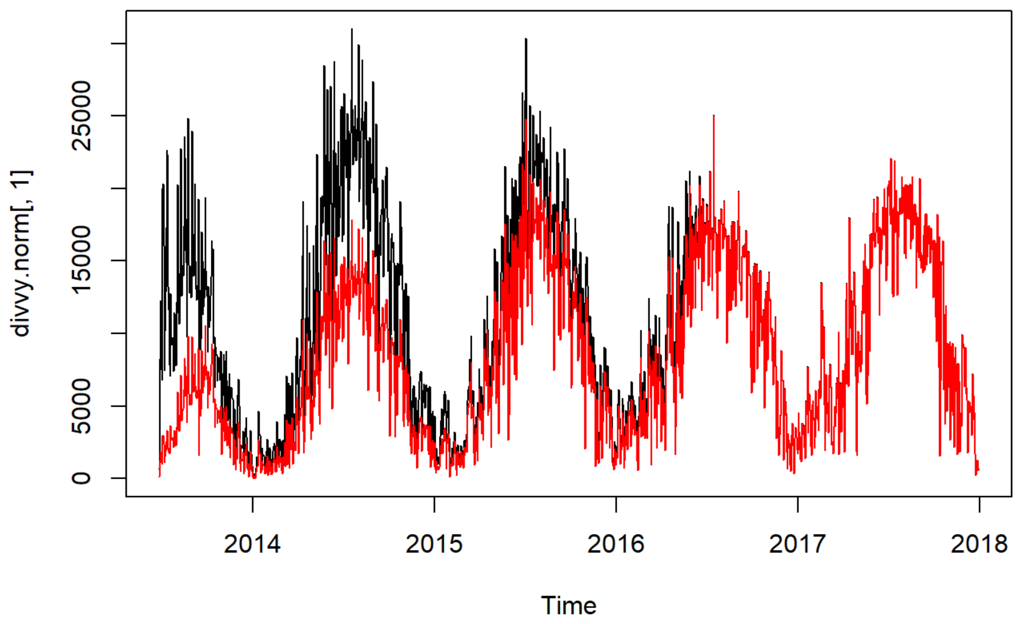
Normalized Total Number of Trips =

$$\frac{Total\ Number\ of\ Trips_{i\,=\,date} * Total\ Number\ of\ Active\ Stations}{Number\ of\ Active\ Stations_{i\,=\,date}}$$

```
divvy.norm = divvy

m = max(divvy.norm[,2])

for (i in 1:dim(divvy.norm)[1]) {
  divvy.norm[i, 1] = divvy.norm[i, 1] * (m/divvy.norm[i,2])
}
```

Comparing original data and normalized data

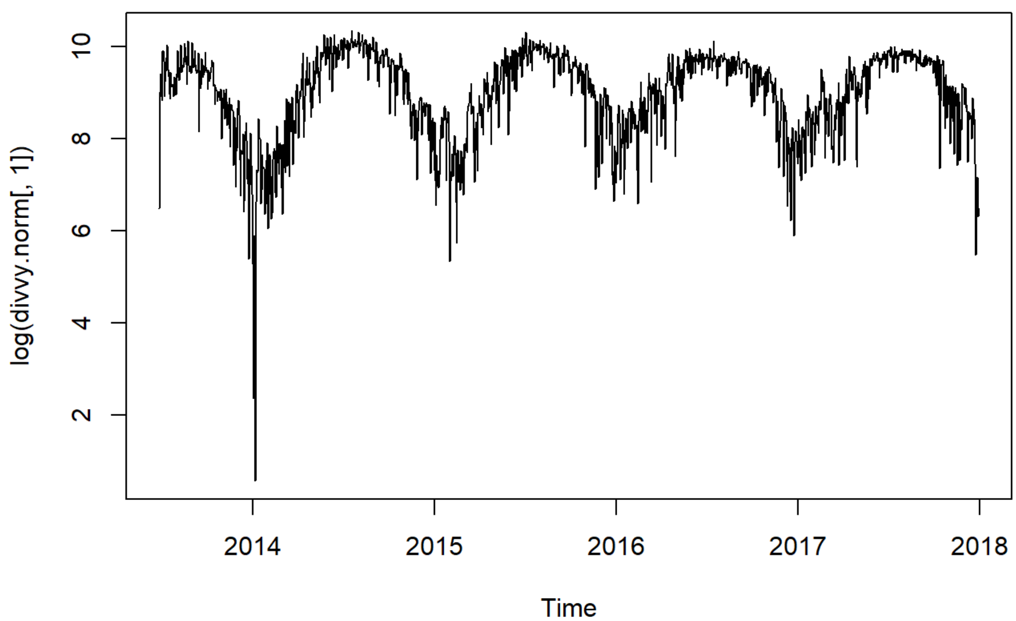
```
plot(divvy.norm[,1])
lines(divvy[,1], col = 'red')
```



Instead of using the average number of trips per day, we decided to use the above normalization method so that the data is more interpretable.

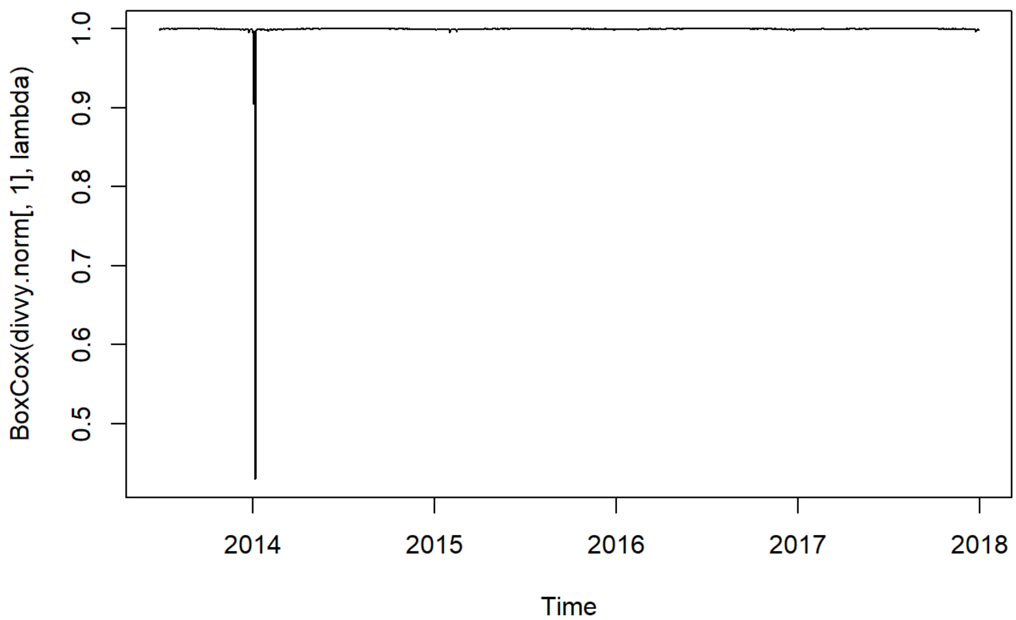
Apply natural log transformation to check if that helps stabilize the normalized data

```
plot(log(divvy.norm[,1])) # low outlier corresponds to Jan 6, 2014 - coldest Jan 6 in Chicago history dating back to 1870
```



Apply Box-Cox transformation to check if that helps stabilize the normalized data

```
lambda = BoxCox.lambda(divvy.norm[,1]) # auto-generated lambda does not help to stabilize variance
plot(BoxCox(divvy.norm[,1],lambda))
```



Natural log transformation seems to stabilize the data's variance more than Box-Cox transformation. Hence, we are going to use natural log transformation in our sArima model.

Split data into Train and Test Sets

```
train.norm = ts(divvy.norm[1:1500,], frequency = 365, start = c(2013, 178), end = c(2017, 217))
test.norm = ts(divvy.norm[1501:1648,], frequency = 365, start = c(2017, 218))
```

D. Modeling

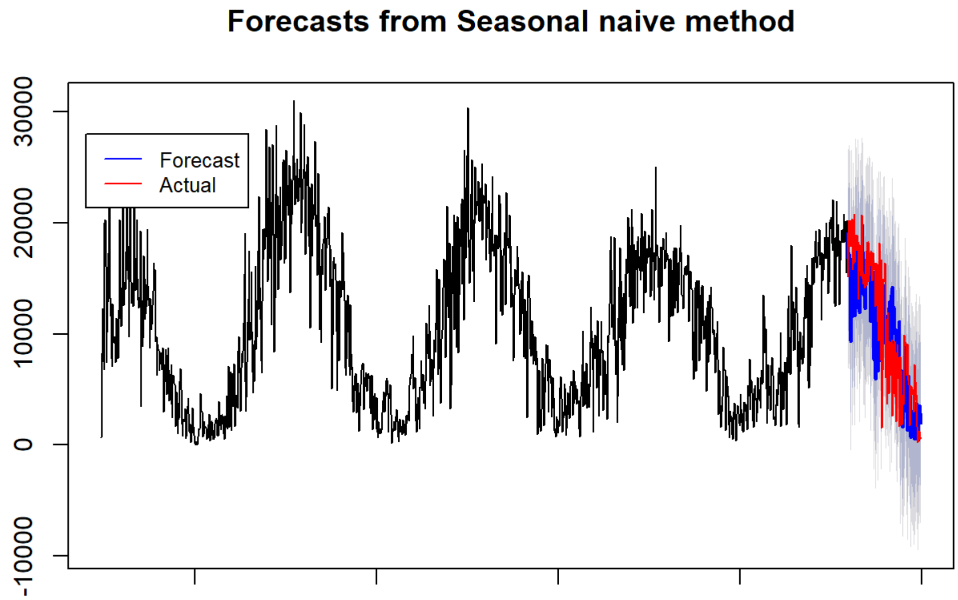
Next, we are going to try out different time series analysis approaches and see which model fits our data best.

1. sNaive [with Normalized Divvy data]

Forecast total number of trips per day using sNaive method.

```
forecast.snaive = snaive(train.norm[,1], h = 148)
```

```
par(xpd = TRUE)
plot(forecast.snaive)
lines(test.norm[,1], type  = "l", col = "red")
legend(2013.4, 28000, inset = c(-0.2,5), legend = c("Forecast", "Actual"), col = c("blue", "red"), lty = 1, cex = 0.8)
```

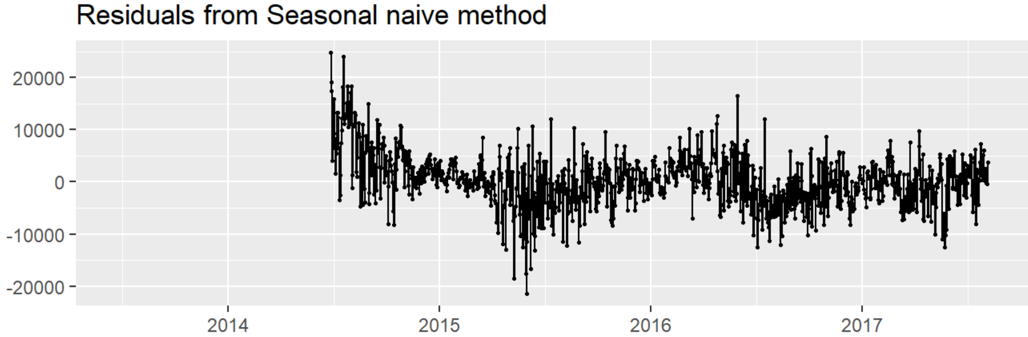


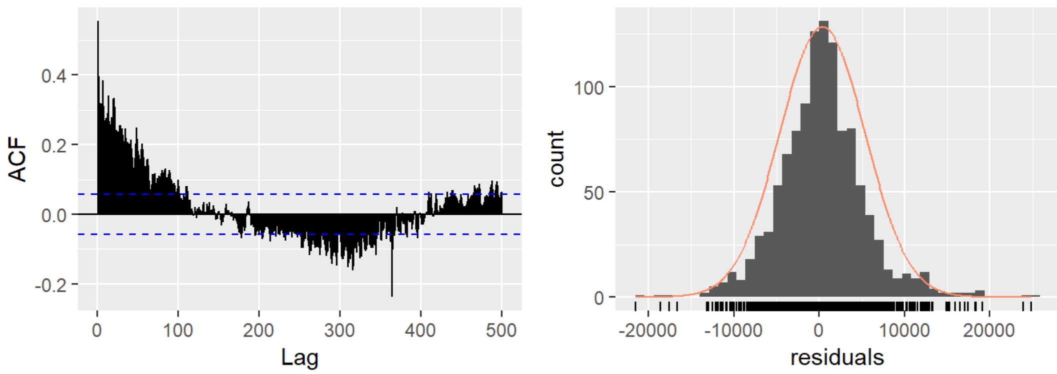
Compute the accuracy score and check the sNaive model's residuals

```
(acc.snaive = accuracy(forecast.snaive, test.norm[,1]))
```

```
##                    ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 340.7354 5030.764 3682.816 -11.40200 48.33773 1.0000000
## Test set     633.1860 3960.865 3051.240 -21.61829 53.86553 0.8285073
##                   ACF1 Theil's U
## Training set 0.5546004        NA
## Test set     0.3811809  1.427455
```

```
checkresiduals(forecast.snaive)
```






```
##
##  Ljung-Box test
##
## data:  Residuals from Seasonal naive method
## Q* = 5946.6, df = 300, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 300
```

The residuals are normally distributed but there is still clearly a seasonal pattern remaining in the ACF plot. It looks like the seasonal naive method cannot handle time series with multiple seasonality.

2. sArima [with Normalized Divvy data]

Build the sArima model with natural log transformation

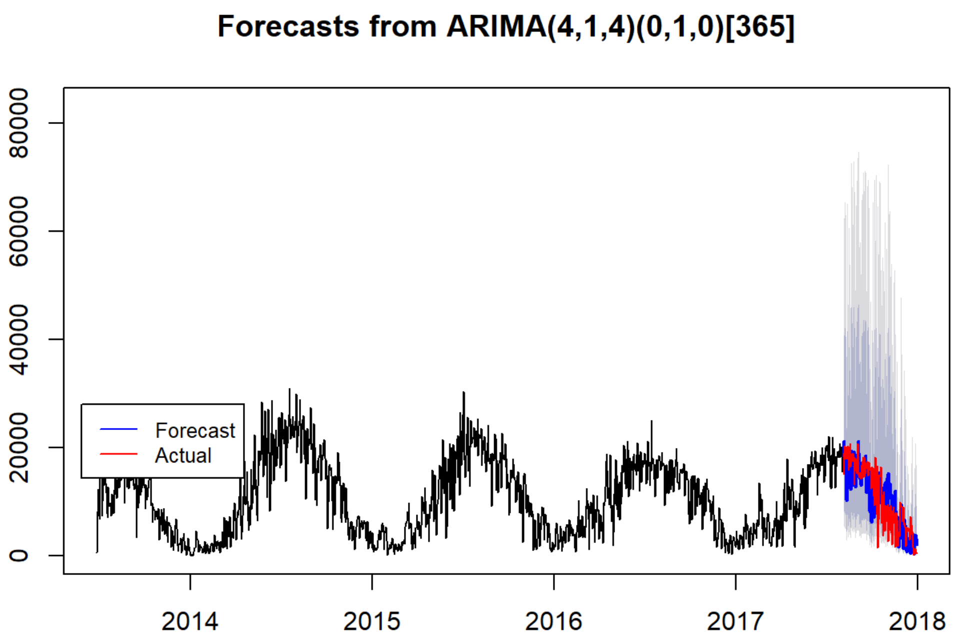
```
(sArima.model = auto.arima(train.norm[,1], lambda = 0))
```

```
## Series: train.norm[, 1]
## ARIMA(4,1,4)(0,1,0)[365]
## Box Cox transformation: lambda= 0
##
## Coefficients:
##           ar1      ar2      ar3     ar4     ma1     ma2      ma3      ma4
##       -0.7878  -0.7723  -0.1686  0.3504  0.3139  0.0890  -0.4779  -0.7214
## s.e.   0.1261   0.0481   0.0996  0.0691  0.1208  0.0653   0.0617   0.1048
##
## sigma^2 estimated as 0.3073:  log likelihood=-940.63
## AIC=1899.26   AICc=1899.42   BIC=1944.56
```

Forecast total number of trips per day using sArima model

```
forecast.sArima = forecast(sArima.model, h = 148)
```

```
plot(forecast.sArima)
lines(test.norm[,1], type  = "l", col = "red")
legend(2013.4, 28000, inset = c(-0.2,5), legend = c("Forecast", "Actual"), col = c("blue", "red"), lty = 1, cex = 0.8)
```

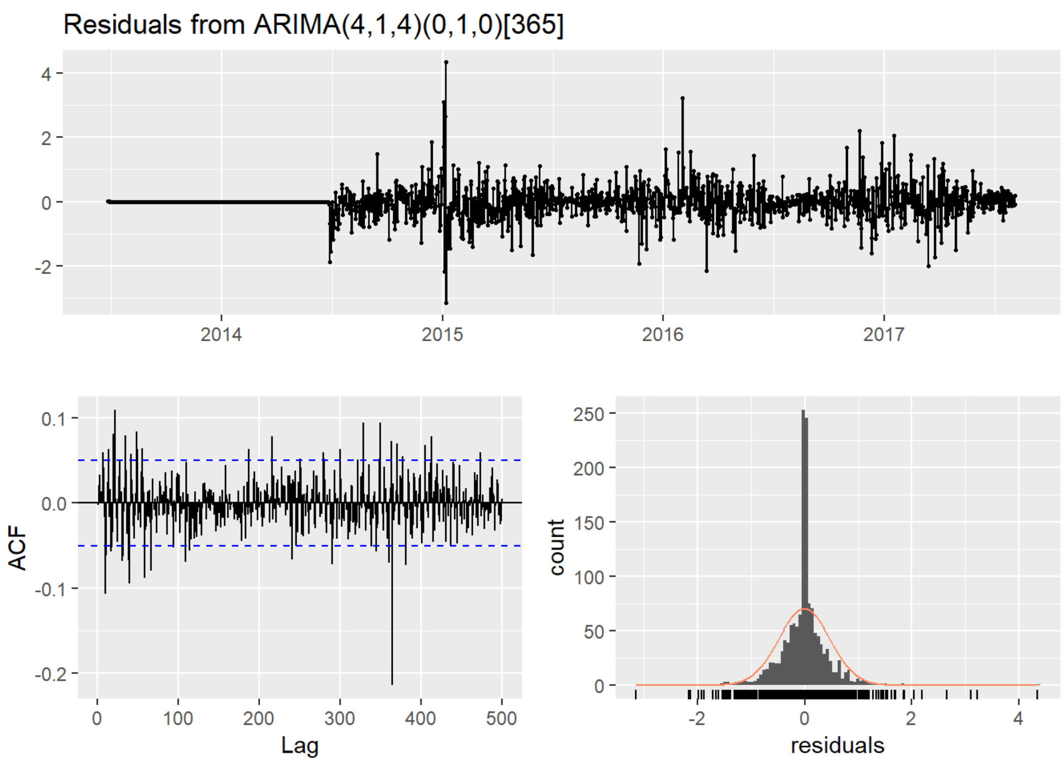


Compute the accuracy score and check the sArima model's residuals

```
(accuracy(forecast.sArima$mean, test.norm[,1]))
```

```
##                  ME     RMSE      MAE       MPE     MAPE      ACF1
## Test set -130.5531 3995.489 2958.907 -30.53164 56.67661 0.3639692
##          Theil's U
## Test set  1.543682
```

```
checkresiduals(forecast.sArima)
```



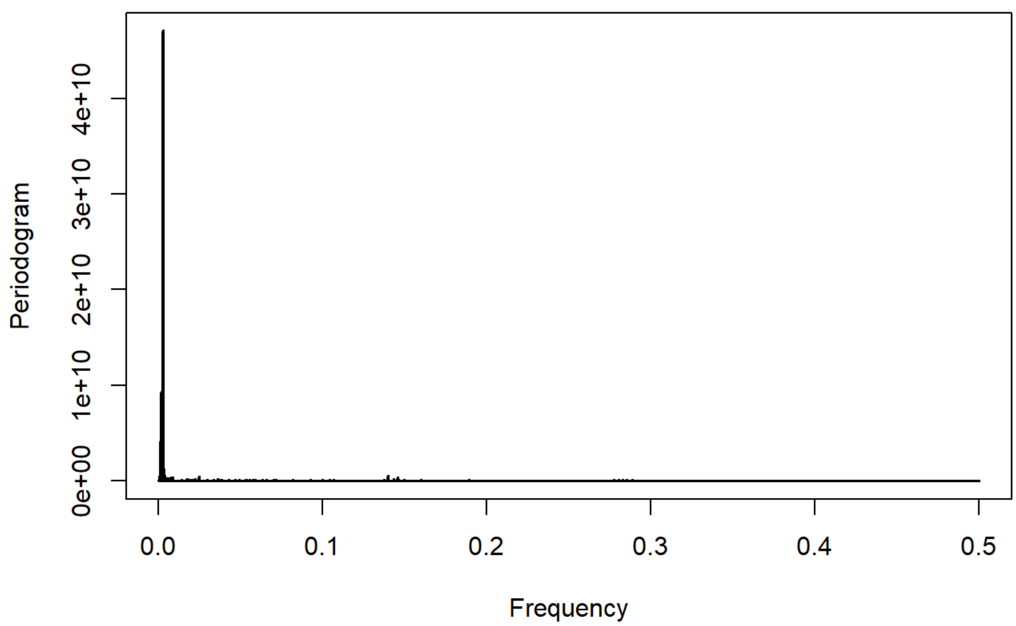
```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(4,1,4)(0,1,0)[365]
## Q* = 455.23, df = 292, p-value = 2.94e-09
##
## Model df: 8.   Total lags used: 300
```

The residuals are slightly right skewed but there is no obvious seasonal pattern showing on the ACF plot. The sArima model is doing well in capturing the multi-seanality patterns.

3. Dynamic Harmonic Regression [with Normalized Divvy data]

Build the Dynamic Harmonic Regression (DHR) model

```
DHR.p = periodogram(divvy.norm[,1])
```




```
max.spec = max(DHR.p$spec)
f = DHR.p$freq[DHR.p$spec == max.spec]
DHR.period = 1/f

DHR.model = list(aicc = Inf)

for(i in 1:25) {
  DHR.fit = auto.arima(train.norm[,1], xreg = fourier(train.norm[,1], i), seasonal = FALSE)
  if(DHR.fit$aicc < DHR.model$aicc) DHR.model = DHR.fit
}
```

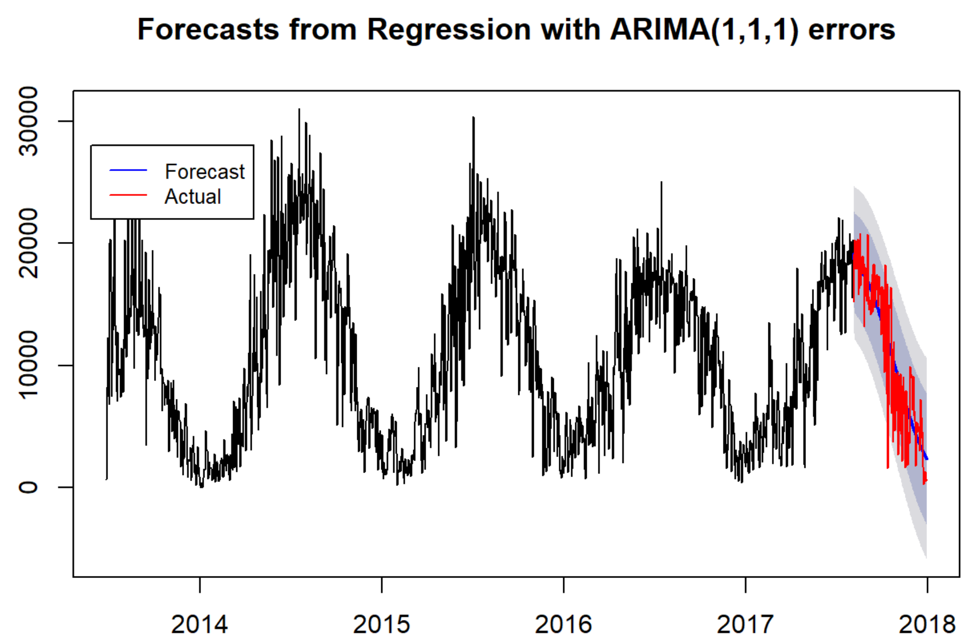
```
summary(DHR.model)
```

```
## Series: train.norm[, 1]
## Regression with ARIMA(1,1,1) errors
##
## Coefficients:
##          ar1      ma1     S1-365     C1-365
##       0.4183  -0.9524  4494.5115  7206.9619
## s.e.  0.0265   0.0089   519.0887   518.8199
##
## sigma^2 estimated as 7786130:  log likelihood=-14018.72
## AIC=28047.44   AICc=28047.48   BIC=28074
##
## Training set error measures:
##                   ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 74.1269 2785.709 2011.926 -29.8146 48.66382 0.5463011
##                    ACF1
## Training set 0.01562389
```

Forecast total number of trips per day using Dynamic Harmonic Regression (DHR) model

```
forecast.DHR = forecast(DHR.model, xreg = fourier(train.norm[,1], 1, 148))
```

```
plot(forecast.DHR)
lines(test.norm[,1], type  = "l", col = "red")
legend(2013.4, 28000, inset = c(-0.2,5), legend = c("Forecast", "Actual"), col = c("blue", "red"), lty = 1, cex = 0.8)
```

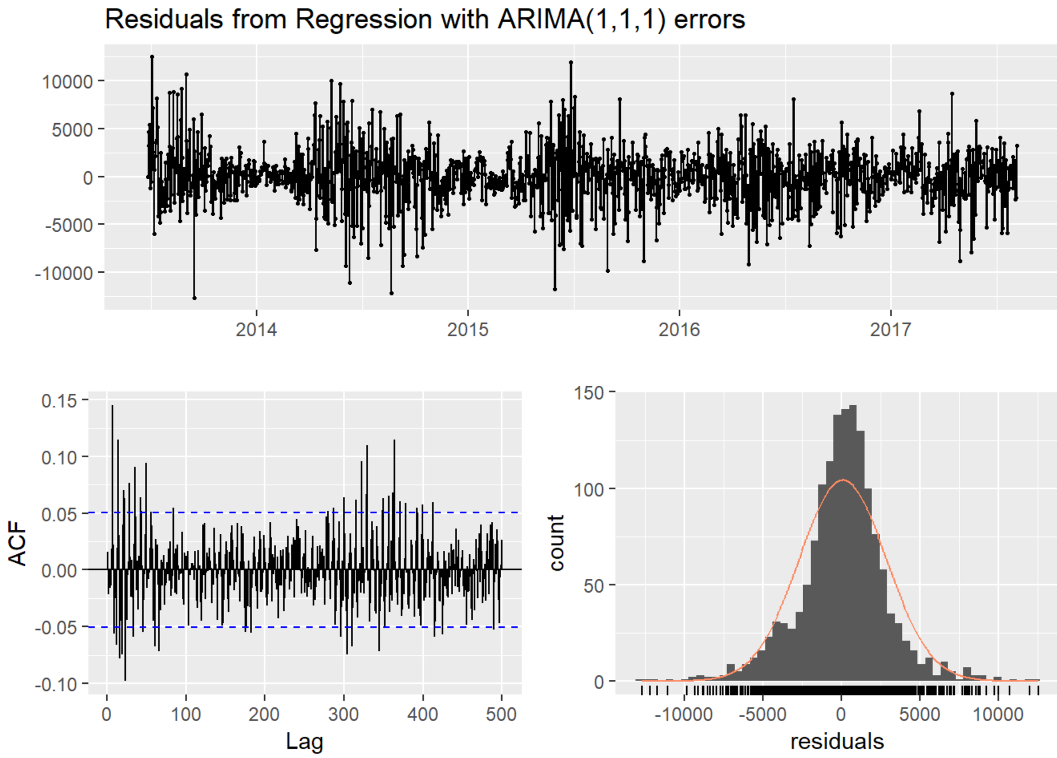


Compute the accuracy score and check the sNaive model's residuals

```
accuracy(forecast.DHR$mean, test.norm[,1])
```

```
##                 ME     RMSE      MAE       MPE     MAPE      ACF1
## Test set -34.53916 2691.286 2147.425 -35.51012 54.00101 0.3772269
##          Theil's U
## Test set  1.789388
```

```
checkresiduals(forecast.DHR)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,1,1) errors
## Q* = 458.34, df = 296, p-value = 4.12e-09
##
## Model df: 4.   Total lags used: 300
```

The Dynamic Harmonic Regression performs better than sArima model. Its residuals are more normally distributed and there is no obvious seasonal patterns showing on the ACF plot. There are still a few lags that are beyond the significant boundary which means there are something left explained by the model.

Next, we are going to include the Chicago weather data in our VAR model and Regression with Arima Errors model in order to see how those variables affect the time series forecast.

4. VAR [with Normalized Divvy data + Chicago Weather Data]

Import Chicago Weather Data

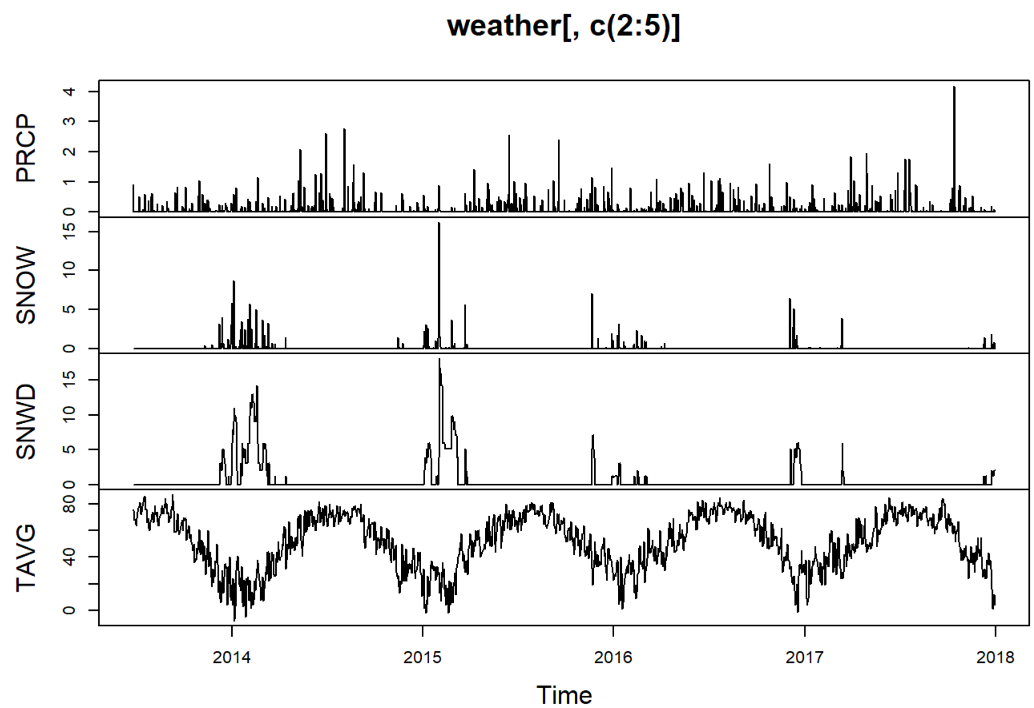
```
weather = read.csv("chicago_weather.csv")
```

```
# Reformat Variables
weather$DATE = as.Date(weather$DATE, format = "%m/%d/%Y")
weather$AWND = as.numeric(weather$AWND)
weather$PRCP = as.numeric(weather$PRCP)
weather$SNOW = as.numeric(weather$SNOW)
weather$SNWD = as.numeric(weather$SNWD)
weather$TAVG = as.numeric(weather$TAVG)
weather$TMAX = as.numeric(weather$TMAX)
weather$TMIN = as.numeric(weather$TMIN)
weather$WDF2 = as.numeric(weather$WDF2)
weather$WDF5 = as.numeric(weather$WDF5)
weather$WSF2 = as.numeric(weather$WSF2)
weather$WSF5 = as.numeric(weather$WSF5)
```

```
weather = ts(weather[,2:12], start = c(2013,178), frequency = 365)
head(weather)
```

```
## Time Series:
## Start = c(2013, 178)
## End = c(2013, 183)
## Frequency = 365
##            AWND PRCP SNOW SNWD TAVG TMAX TMIN WDF2 WDF5 WSF2 WSF5
## 2013.485  7.16 0.92    0    0   76   90   64  310  300 32.0 48.1
## 2013.488 10.96 0.00    0    0   75   83   67  300  290 25.9 33.1
## 2013.490 12.30 0.00    0    0   68   72   64   20   50 23.0 32.0
## 2013.493 13.87 0.00    0    0   68   77   62   20   20 29.1 36.9
## 2013.496 12.30 0.00    0    0   68   75   59   40   50 23.0 35.1
## 2013.499 11.18 0.00    0    0   65   68   60   40   40 17.0 25.1
```

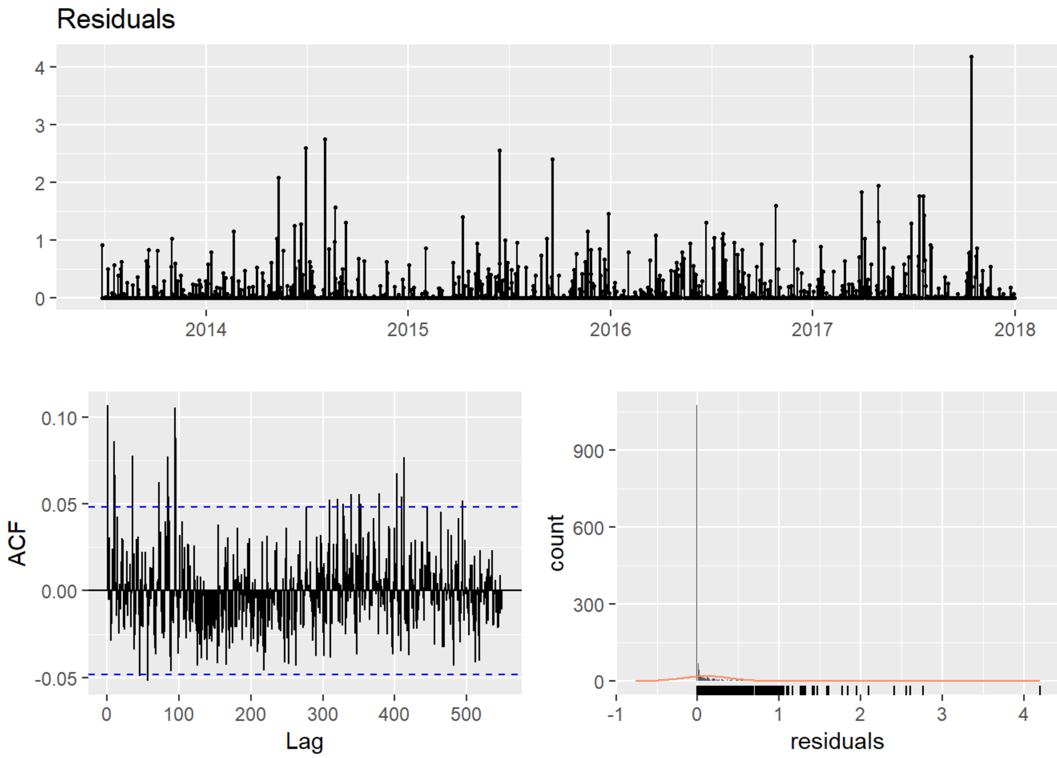
```
plot(weather[,c(2:5)])
```



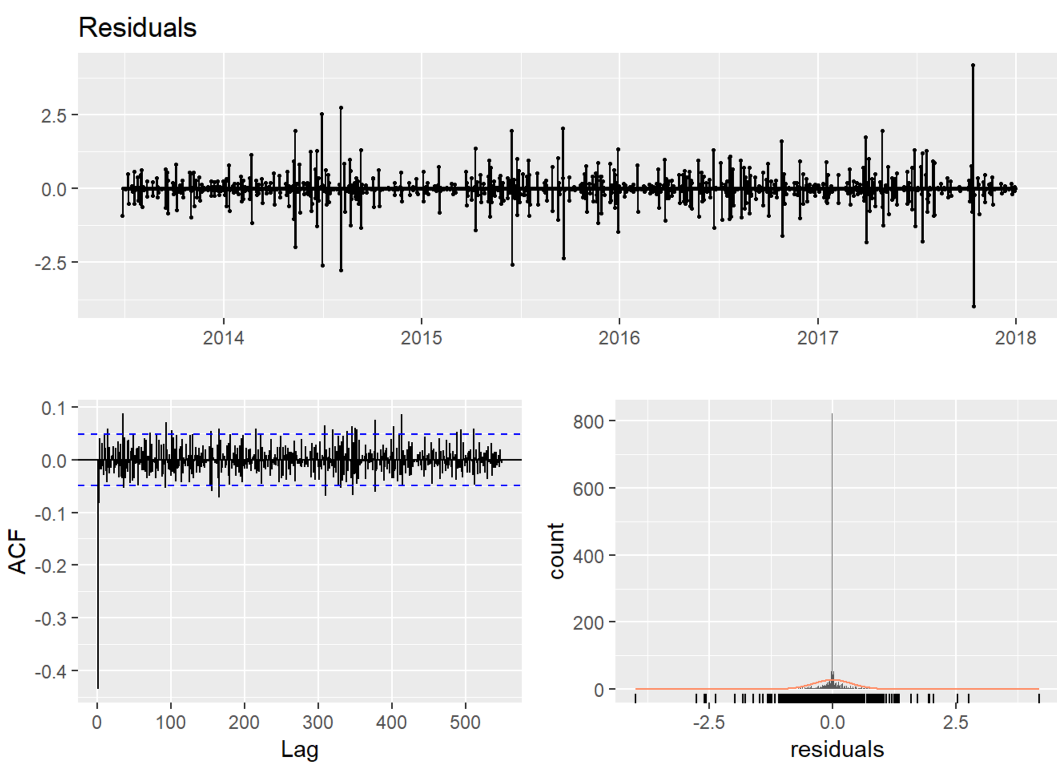
PRCP - Precipitation (mm or inches as per user preference, inches to hundredths on Daily Form pdf file)
SNOW - Snowfall (mm or inches as per user preference, inches to tenths on Daily Form pdf file)
SNWD - Snow depth (mm or inches as per user preference, inches on Daily Form pdf file)
TAVG = Average temperature (Fahrenheit or Celsius as per user preference, Fahrenheit to tenths on Daily Form pdf file

Weather Data Differencing and Analysis

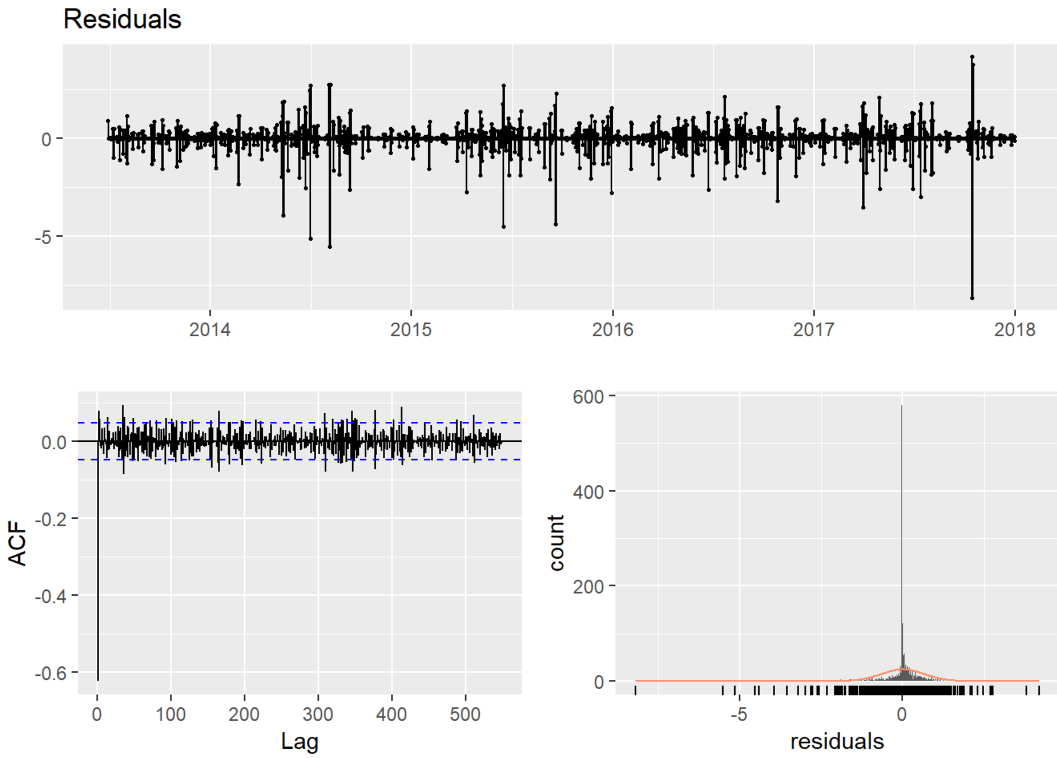
```
checkresiduals(weather[,2])
```



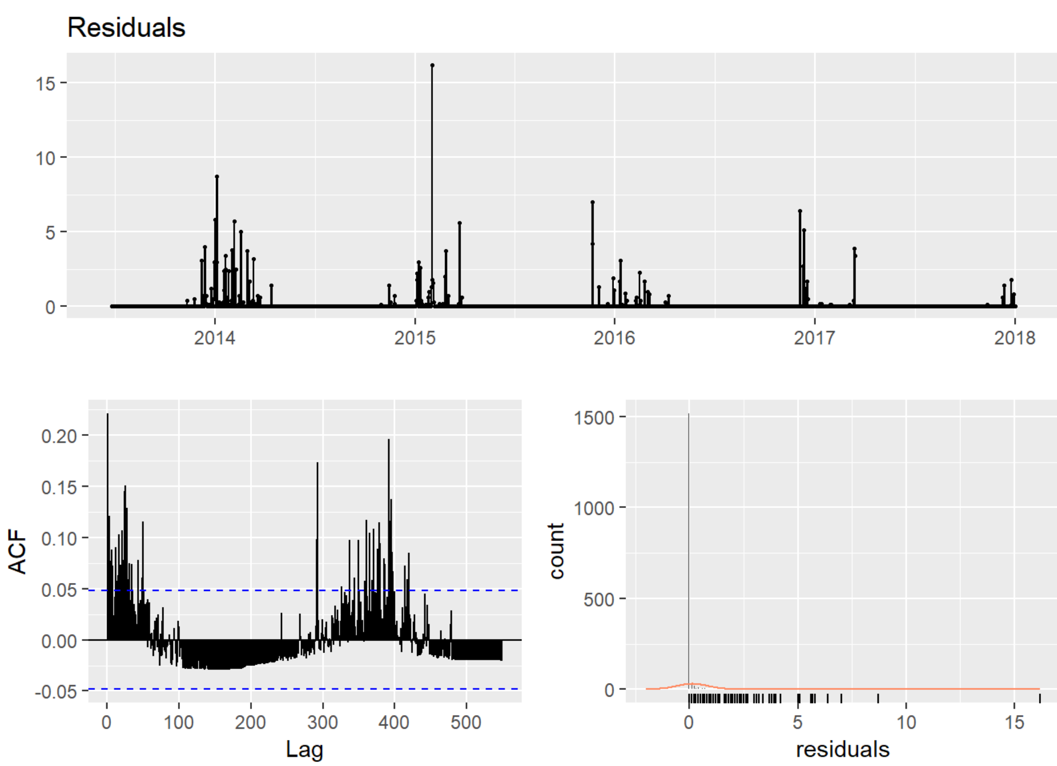

```
checkresiduals(diff(weather[,2]))
```



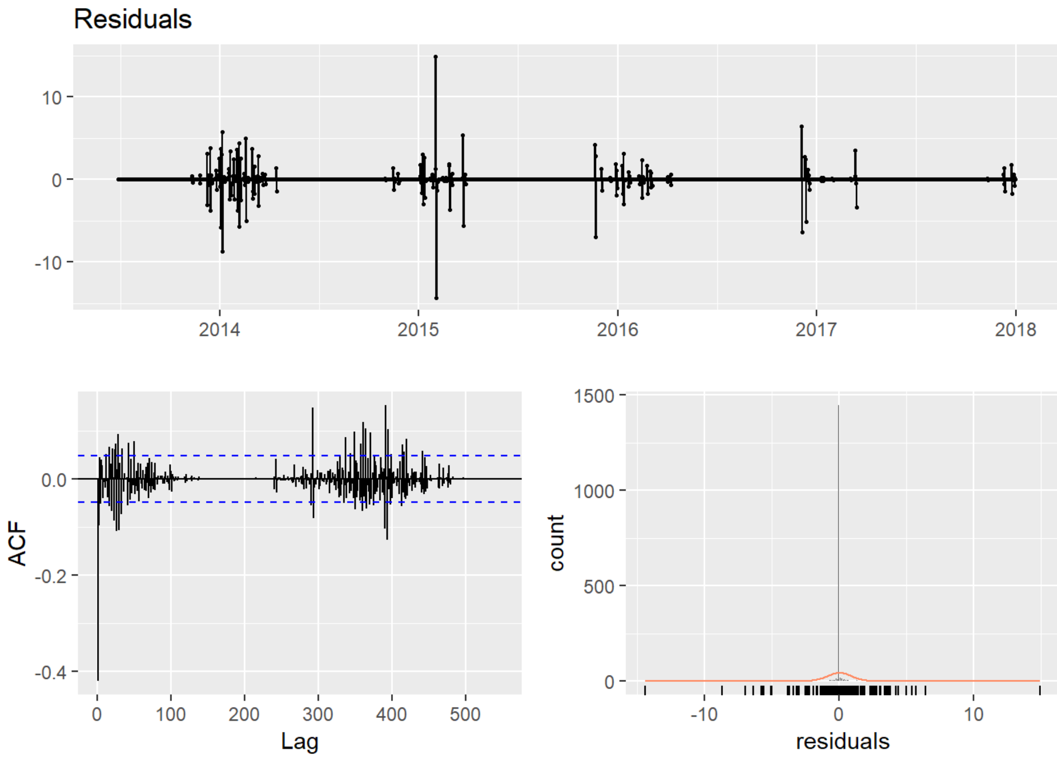

```
checkresiduals(diff(diff(weather[,2])))
```



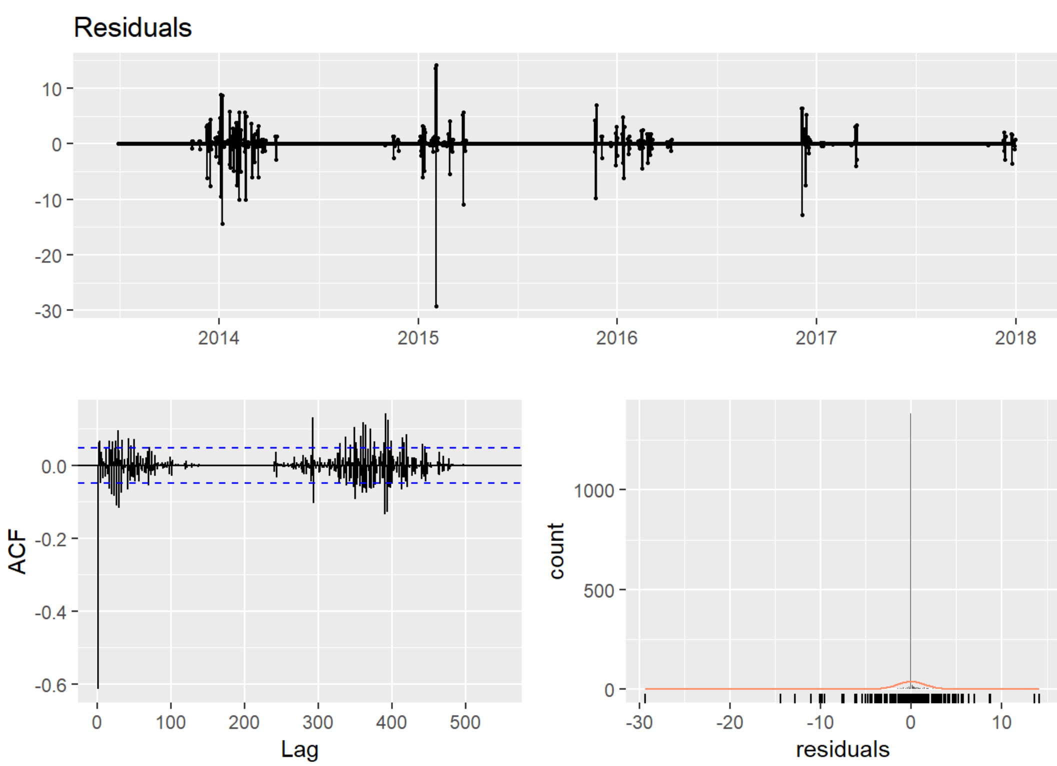

```
checkresiduals(weather[,3])
```



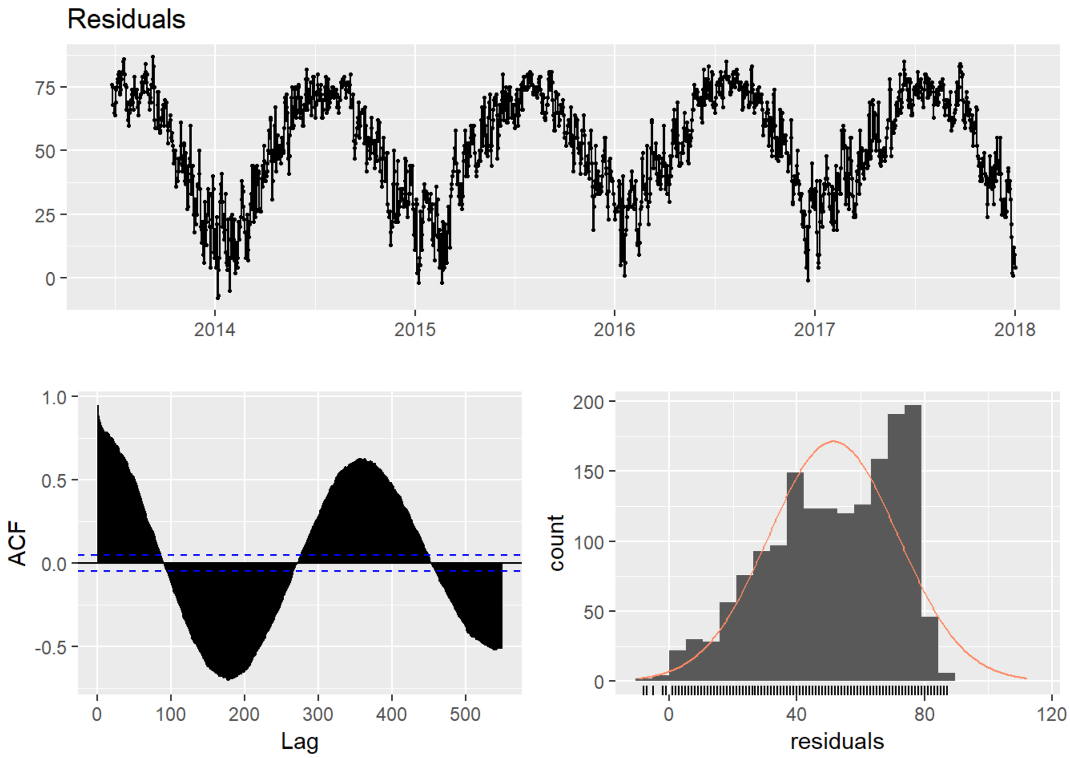

```
checkresiduals(diff(weather[,3]))
```



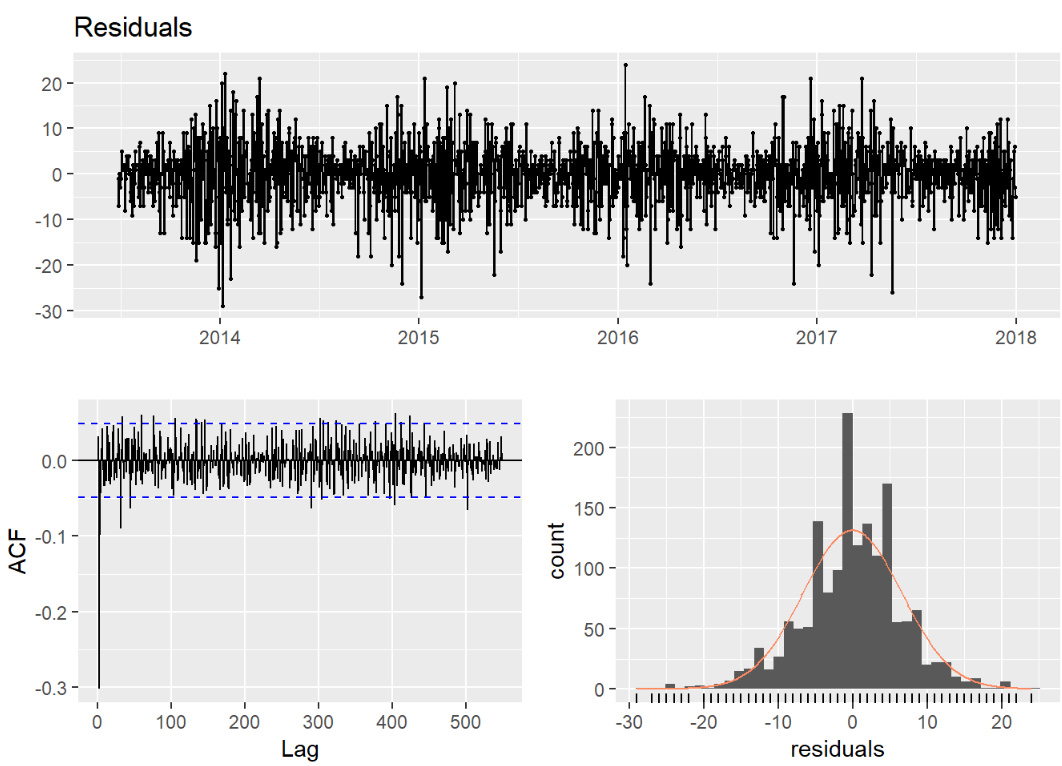

```
checkresiduals(diff(diff(weather[,3])))
```



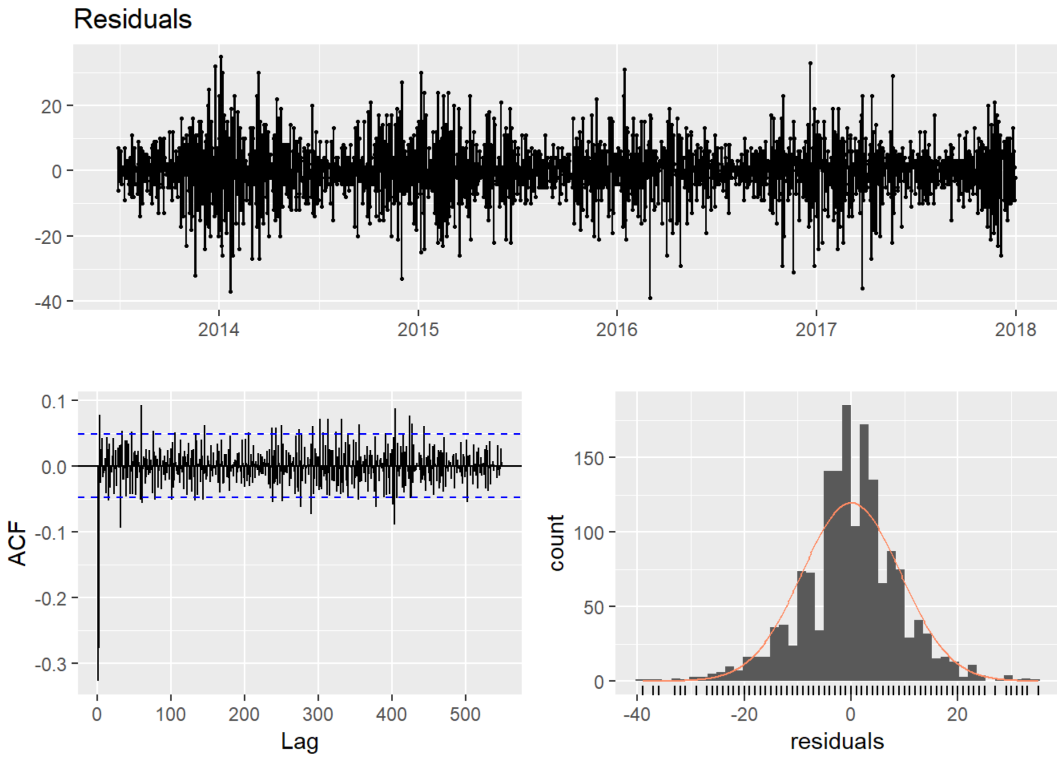

```
checkresiduals(weather[,5])
```



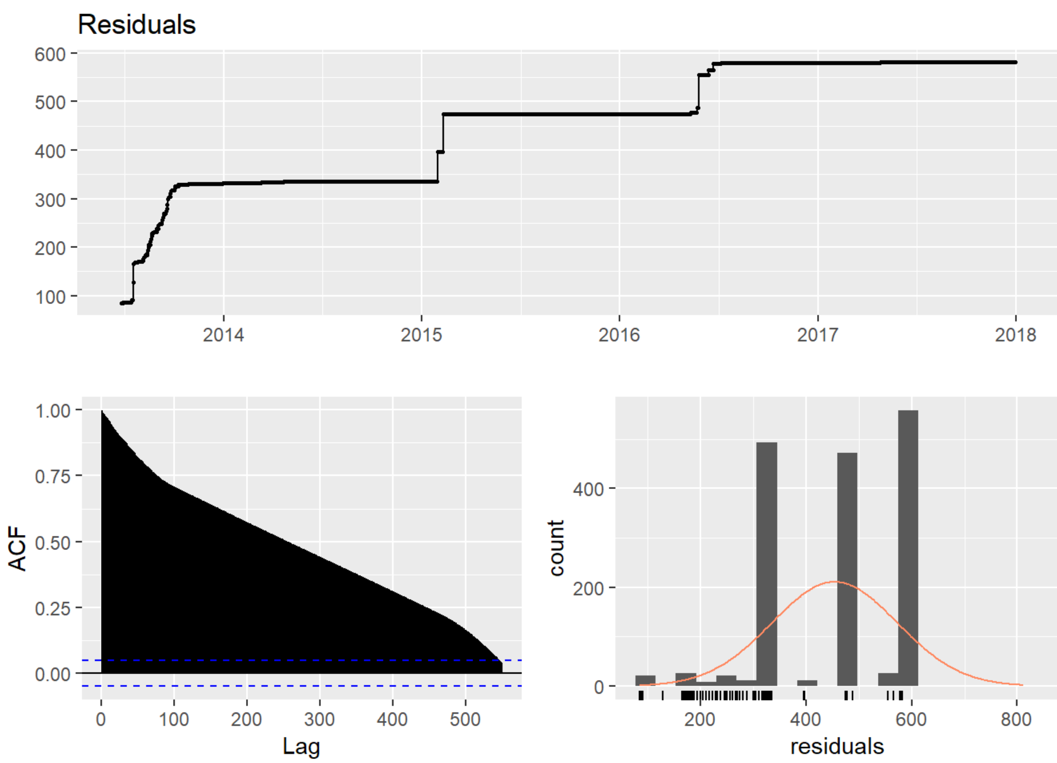

```
checkresiduals(diff(weather[,5]))
```



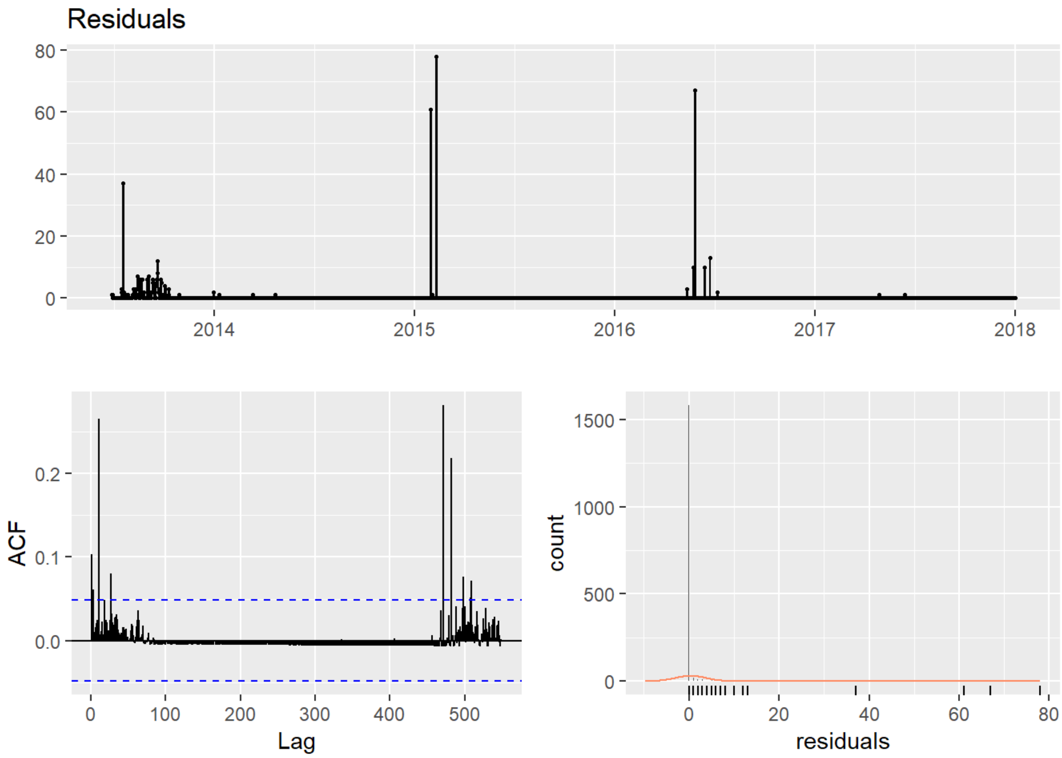

```
checkresiduals(diff(diff(weather[,5])))
```



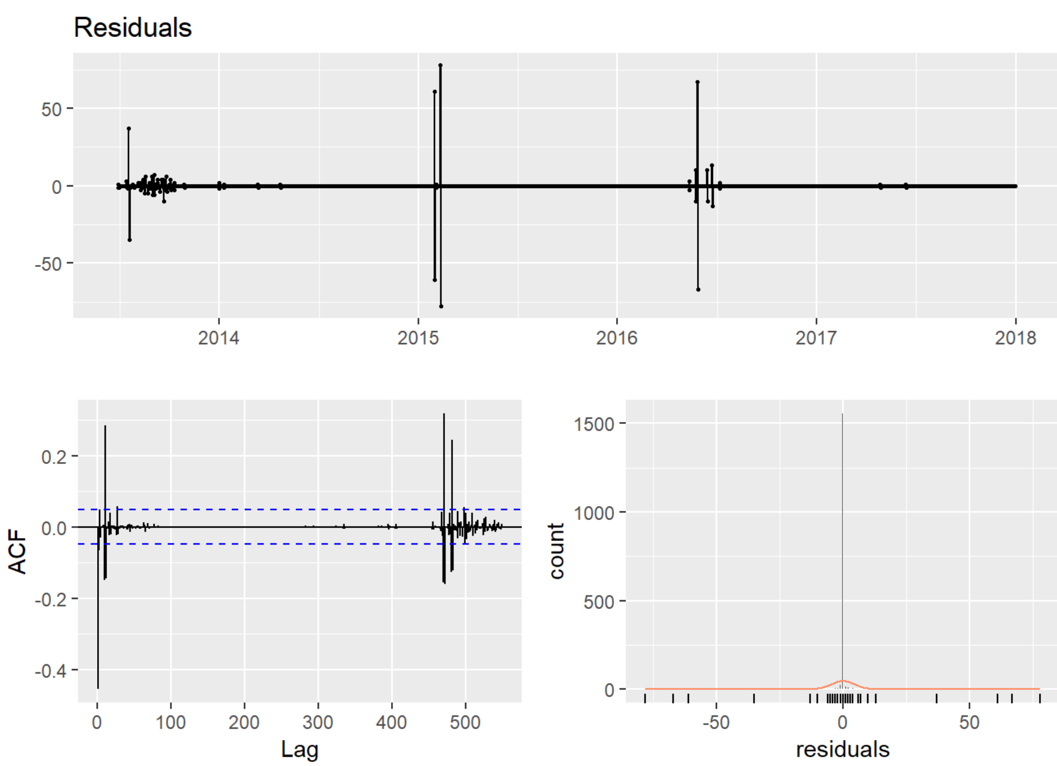

```
checkresiduals(divvy[,2])
```




```
checkresiduals(diff(divvy[,2]))
```




```
checkresiduals(diff(diff(divvy[,2])))
```



First order diffrencing makes those variables look more stationary but the second order differencing does not improve the results any further.

Split the data into Train and Test Set

```
weather.train = ts(weather[1:1500,], frequency = 365, start = c(2013, 178), end = c(2017, 217))
weather.test = ts(weather[1501:1648,], frequency = 365, start = c(2017, 218))
```

Build the VAR model

Precipitation, snowfall, snow depth, and average temparature are used in the VAR model.

```
VAR.data.merged = cbind(train.norm[,1], weather.train[,2:5])
colnames(VAR.data.merged) = c("TotalTrip", "PRCP", "SNOW", "SNWD", "TAVG")

VARselect(VAR.data.merged, lag.max=10, type="both", season=365)$selection
```

```
## AIC(n)  HQ(n)  SC(n) FPE(n)
##      7      7      2      7
```

```
VARselect(VAR.data.merged, lag.max=100, type="both", season=365)$selection
```

```
## AIC(n)  HQ(n)  SC(n) FPE(n)
##     15      2      2      7
```

```
var1.model = VAR(VAR.data.merged, p=1, type="both", season=365)
serial.test(var1.model, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var1.model
## Chi-squared = 750.35, df = 225, p-value < 2.2e-16
```

```
var2.model = VAR(VAR.data.merged, p=2, type="both", season=365)
serial.test(var2.model, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var2.model
## Chi-squared = 539.75, df = 200, p-value < 2.2e-16
```

```
var3.model = VAR(VAR.data.merged, p=3, type="both", season=365)
serial.test(var3.model, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var3.model
## Chi-squared = 501.02, df = 175, p-value < 2.2e-16
```

```
var4.model = VAR(VAR.data.merged, p=4, type="both", season=365)
serial.test(var4.model, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var4.model
## Chi-squared = 452.87, df = 150, p-value < 2.2e-16
```

```
var5.model = VAR(VAR.data.merged, p=5, type="both", season=365 )
serial.test(var5.model, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var5.model
## Chi-squared = 376.27, df = 125, p-value < 2.2e-16
```

```r
var6.model = VAR(VAR.data.merged, p=6, type="both", season=365)
serial.test(var6.model, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var6.model
## Chi-squared = 230.7, df = 100, p-value = 2.478e-12
```

```r
var7.model = VAR(VAR.data.merged, p=7, type="both", season=365)
serial.test(var7.model, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var7.model
## Chi-squared = 131.29, df = 75, p-value = 6.332e-05
```

```r
var15.model = VAR(VAR.data.merged, p=15, type="both", season=365)
serial.test(var10.model, lags.pt=10, type="PT.asymptotic")
```

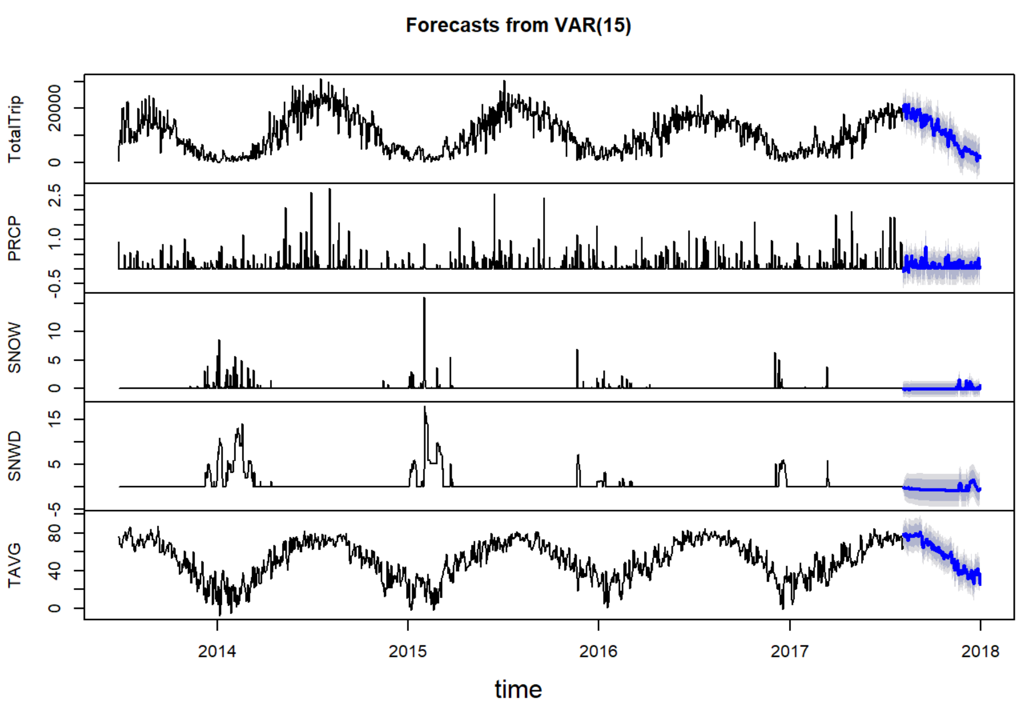
```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var10.model
## Chi-squared = 51.379, df = 0, p-value < 2.2e-16
```

We choose VAR(7) model as our best VAR model because it has the lowest AIC value.

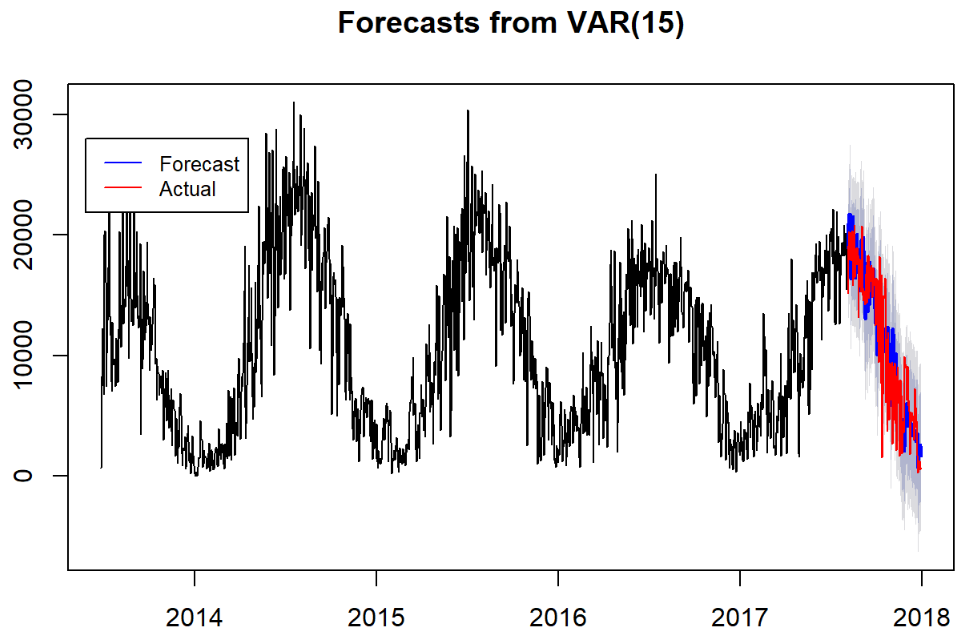
Forecast total number of trips per day using VAR model

```r
forecast.var15 = forecast(var15.model, h = 148)
```

```r
plot(forecast.var15)
```



```
plot(forecast.var15$forecast$TotalTrip)
lines(test.norm[,1], type  = "l", col = "red")
legend(2013.4, 28000, inset = c(-0.2,5), legend = c("Forecast", "Actual"), col = c("blue", "red"), lty = 1, cex = 0.8)
```



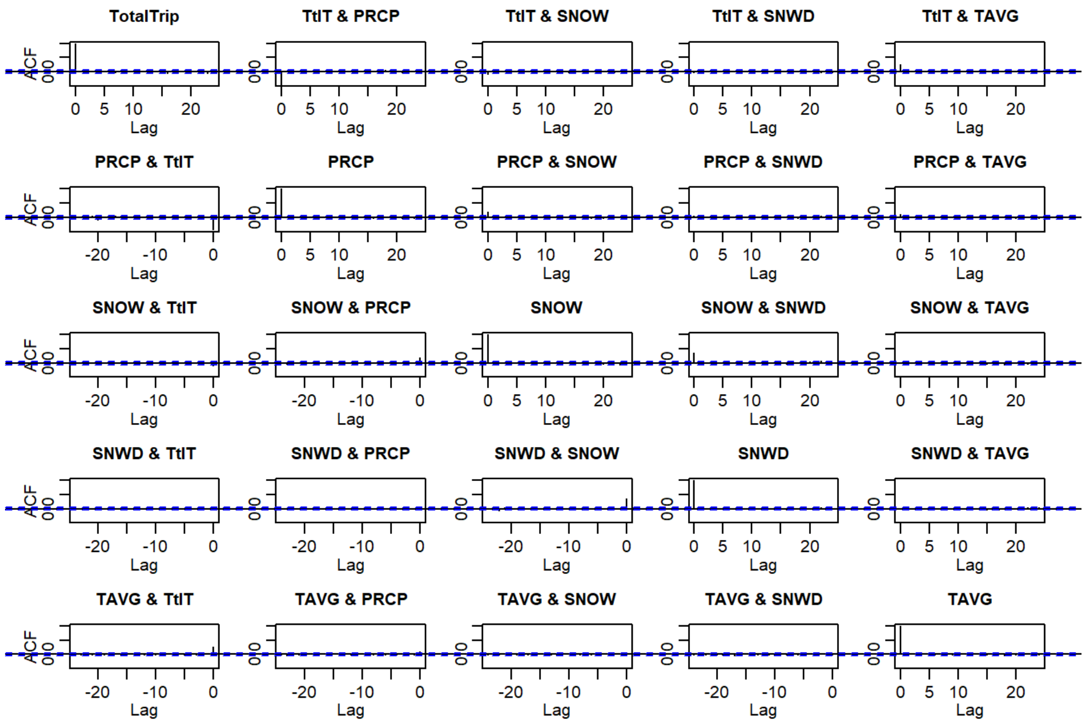
Compute the accuracy score and check the VAR model’s residuals

```
accuracy(forecast.var15$forecast$TotalTrip$mean, test.norm[,1])
```

```
##                 ME     RMSE      MAE       MPE     MAPE      ACF1
## Test set -237.1451 3167.418 2460.044 -25.76111 46.42776 0.3803305
##          Theil's U
## Test set  1.250084
```

```
par(oma=c(0,0,2,0))
stats::acf(residuals(var15.model), xpd = par("xaxs"))
```

```
## Warning in par(mfrow = rep(nr, 2L), mar = mar, oma = oma, mgp = mgp, ask =
## ask, : NAs introduced by coercion
```



Both individual ACFs and cross-correlation ACFs resemble white noise.

4. Regression With Arima Errors model

Build the Regression With Arima Errors model

```
# With stepwise = FALSE, Approx = FALSE

xreg.train = cbind(PRCP = weather.train[,2], SNOW = weather.train[,3], TEMP = weather.train[,5], STAT = train.set[,2])
reg.model1 = auto.arima(train.set[,1], lambda = "auto", xreg = xreg.train, stepwise = FALSE, approx = FALSE)
```

```
summary(reg.model1)
```

```
## Series: train.set[, 1]
## Regression with ARIMA(2,1,3) errors
## Box Cox transformation: lambda= 0.5956636
##
## Coefficients:
##          ar1      ar2      ma1     ma2      ma3       PRCP     SNOW    TEMP
##       1.2295  -0.9648  -1.9408  1.7887  -0.6874  -96.0799  -4.4294  3.3701
## s.e.  0.0126   0.0130   0.0278  0.0439   0.0273    4.2066   1.6169  0.1667
##         STAT
##       0.2663
## s.e.  0.1828
##
## sigma^2 estimated as 2163:  log likelihood=-7878.79
## AIC=15777.59   AICc=15777.73   BIC=15830.71
##
## Training set error measures:
##                     ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 102.373 1741.587 1247.143 -8.90314 27.15933 0.4023382
##                     ACF1
## Training set 0.07903325
```

```
# Without stepwise = FALSE, Approx = FALSE
reg.model2 = auto.arima(train.set[,1], lambda = "auto", xreg = xreg.train)
```

```
summary(reg.model2)
```

```
## Series: train.set[, 1]
## Regression with ARIMA(4,1,5) errors
## Box Cox transformation: lambda= 0.5956636
##
## Coefficients:
```

```
## Warning in sqrt(diag(x$var.coef)): NaNs produced
```

```
##          ar1      ar2     ar3      ar4      ma1     ma2      ma3     ma4
##       0.5814  -0.5241  -0.208  -0.3389  -1.1790  0.6443  -0.0155  0.1483
## s.e.  0.1463      NaN     NaN      NaN   0.1389     NaN      NaN     NaN
##           ma5      PRCP     SNOW    TEMP    STAT
##       -0.3175  -92.0375  -3.7265  3.4172  0.3270
## s.e.      NaN    4.1556   1.5608  0.1742  0.1722
##
## sigma^2 estimated as 2115:  log likelihood=-7860.07
## AIC=15748.14   AICc=15748.43   BIC=15822.52
##
## Training set error measures:
##                    ME    RMSE      MAE       MPE     MAPE      MASE
## Training set 106.4058 1724.43 1225.232 -8.957166 26.73741 0.3952695
##                     ACF1
## Training set -0.03221982
```

```
xreg.train.diff = cbind(PRCP = diff(weather.train[,2]), SNOW = diff(weather.train[,3]), TEMP = diff(weather.train[,5]), ST
AT = diff(train.set[,2]))
reg.model3 = auto.arima(diff(train.set[,1]), lambda = "auto", xreg = xreg.train.diff)
```

```
summary(reg.model3)
```

```
## Series: diff(train.set[, 1])
## Regression with ARIMA(1,0,1) errors
## Box Cox transformation: lambda= 1.436525
##
## Coefficients:
##          ar1      ma1  intercept        PRCP      SNOW       TEMP
##       0.2424  -0.7656  -795.1029  -95666.399  1760.871  2307.8958
## s.e.  0.0383   0.0247   432.2896    4657.439  1812.219   199.7691
##           STAT
##       131.2010
## s.e.  272.7037
##
## sigma^2 estimated as 2.801e+09:  log likelihood=-18427.75
## AIC=36871.5   AICc=36871.6   BIC=36914
##
## Training set error measures:
##                    ME     RMSE      MAE MPE MAPE      MASE       ACF1
## Training set 56.02977 1878.466 1430.363 NaN  Inf 0.5548142 0.09500399
```

We try to use differenced variables (more stationary) in the model but they do not make the model better.

The reg.model2 has lower RMSE and MAPE, hence, it is a better model.

```
reg.model = Arima(train.set[,1], order = c(4,1,5), lambda = "auto", xreg = xreg.train)
```

```
summary(reg.model)
```

```
## Series: train.set[, 1]
## Regression with ARIMA(4,1,5) errors
## Box Cox transformation: lambda= 0.5956636
##
## Coefficients:
```

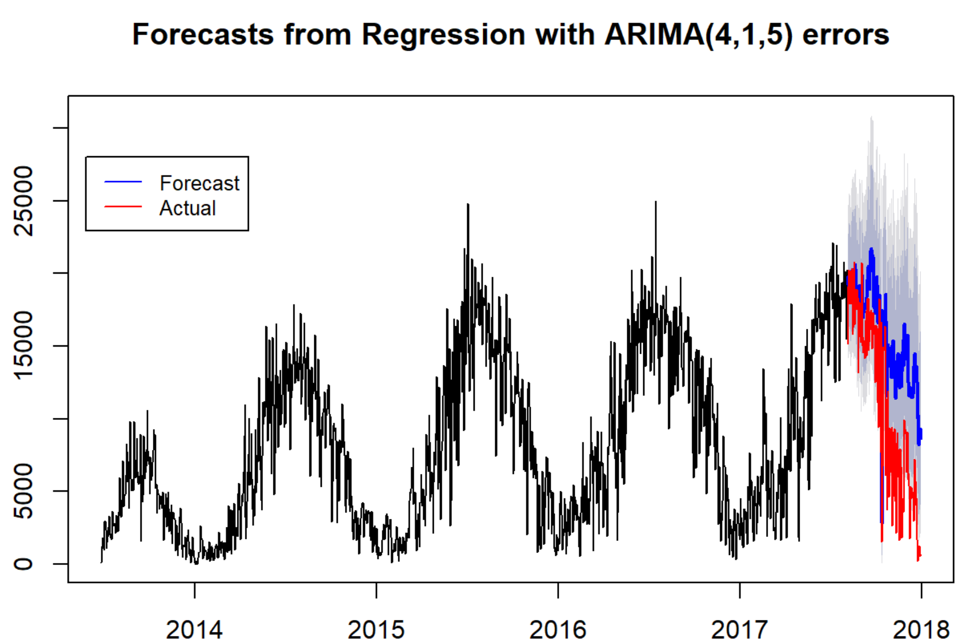
```
## Warning in sqrt(diag(x$var.coef)): NaNs produced
```

```
##          ar1      ar2     ar3      ar4      ma1     ma2      ma3     ma4
##       0.5814  -0.5241  -0.208  -0.3389  -1.1790  0.6443  -0.0155  0.1483
## s.e.  0.1463      NaN     NaN      NaN   0.1389     NaN      NaN     NaN
##          ma5      PRCP     SNOW    TEMP    STAT
##      -0.3175  -92.0375  -3.7265  3.4172  0.3270
## s.e.     NaN    4.1556   1.5608  0.1742  0.1722
##
## sigma^2 estimated as 2115:  log likelihood=-7860.07
## AIC=15748.14   AICc=15748.43   BIC=15822.52
##
## Training set error measures:
##                    ME    RMSE      MAE       MPE     MAPE      MASE
## Training set 106.4058 1724.43 1225.232 -8.957166 26.73741 0.3952695
##                     ACF1
## Training set -0.03221982
```

Forecast total number of trips per day using Regression with Arima Errors with number of active stations, temperature, precipitation, and snow as predictors

```
xreg.test = cbind(PRCP = weather.test[,2], SNOW = weather.test[,3], TEMP = weather.test[,5], STAT = test.set[,2])
forecast.reg = forecast(reg.model, xreg = xreg.test, h = 148)
```

```
par(xpd = TRUE)
plot(forecast.reg)
lines(test.set[,1], type  = "l", col = "red")
legend(2013.4, 28000, inset = c(-0.2,5), legend = c("Forecast", "Actual"), col = c("blue", "red"), lty = 1, cex = 0.8)
```

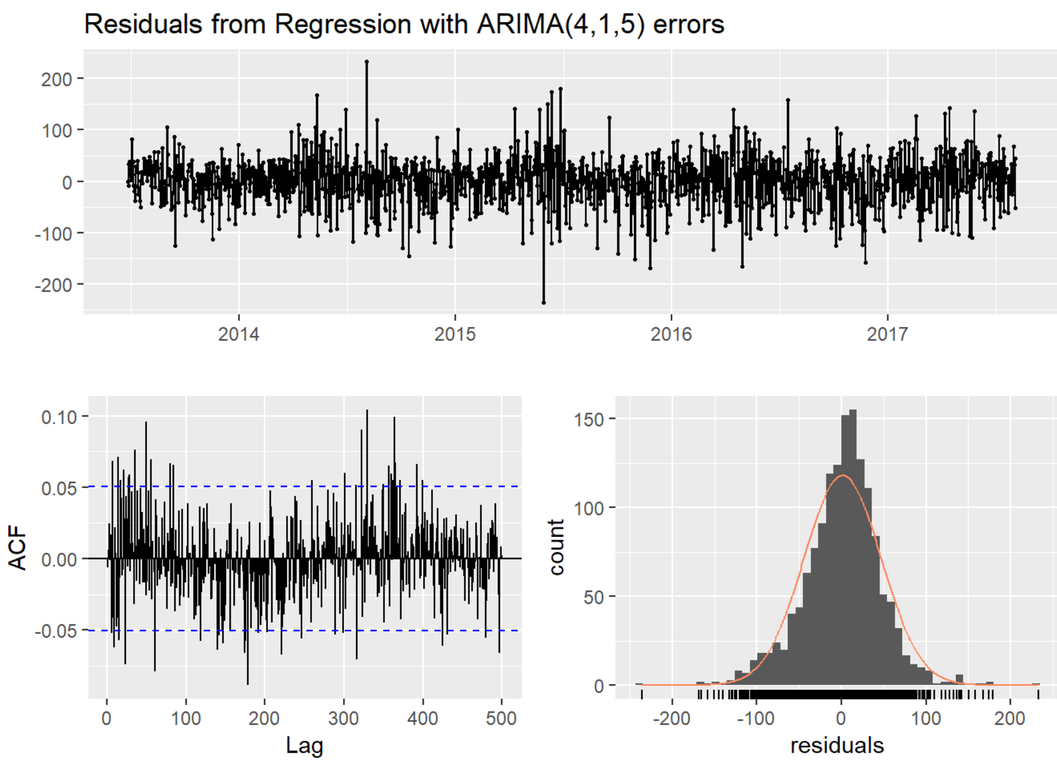


Compute the accuracy score

```
acc.reg = accuracy(forecast.reg, test.set[,1])
acc.reg
```

```
##                       ME     RMSE      MAE         MPE      MAPE      MASE
## Training set   106.4058 1724.43 1225.232   -8.957166  26.73741 0.3952695
## Test set     -4926.4528 5905.62 5012.679 -167.781833 168.20793 1.6171297
##                     ACF1 Theil's U
## Training set -0.03221982        NA
## Test set      0.69869419  6.801634
```

```
checkresiduals(forecast.reg)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(4,1,5) errors
## Q* = 457.24, df = 287, p-value = 6.224e-10
##
## Model df: 13.   Total lags used: 300
```

The residuals are normally distributed and there is no obvious seasonal patterns showing on the ACF plot.

E. Cross Validation (excludes sARIMA model due to high computational complexity)

1. sNaive

```
n = length(divvy.norm[,1]) # number of data points
p = 365 # period
H = 366 # forecast horizon

st = tsp(divvy)[1] # gives the start time in time units

error.expanding.sNaive = matrix(NA, floor(n/H), H)
error.sliding.sNaive = matrix(NA, floor(n/H), H)

for (i in 1:floor(n/H)){

  train.expanding = window(divvy.norm, end = st + i) # expanding window
  train.sliding = window(divvy.norm, start = st + i - 1, end = st + i) # sliding window

  test = window(divvy.norm, start = st + i + 1/p, end = st + i + 1 + 1/p)

  fcast.expanding.sNaive = forecast(train.expanding[,1], h = H)
  fcast.sliding.sNaive = forecast( train.sliding[,1], h = H)

  error.expanding.sNaive[i, 1:length(test[,1])] = (abs(fcast.expanding.sNaive[['mean']] - test[,1])/test[,1])/length(test
[,1])*100
  error.sliding.sNaive[i, 1:length(test[,1])] = (abs(fcast.sliding.sNaive[['mean']] - test[,1])/test[,1])/length(test[,1])
*100

}
```

```
## Warning in window.default(x, ...): 'end' value not changed
```

2. Dynamic Harmonic Regression

```
n = length(divvy.norm[,1]) # number of data points
p = 365 # period
H = 366 # forecast horizon

st = tsp(divvy)[1] # gives the start time in time units

error.expanding.DHR = matrix(NA, floor(n/H), H)
error.sliding.DHR = matrix(NA, floor(n/H), H)

for (i in 1:floor(n/H)){

  train.expanding = window(divvy.norm, end = st + i) # expanding window
  train.sliding = window(divvy.norm, start = st + i - 1, end = st + i) # sliding window

  test = window(divvy.norm, start = st + i + 1/p, end = st + i + 1 + 1/p)

  fit.expanding.DHR = Arima(train.expanding[,1], xreg = fourier(train.expanding[,1], 1), order = c(1,1,1))
  fit.sliding.DHR = Arima(train.sliding[,1], xreg = fourier(train.sliding[,1], 1), order = c(1,1,1))

  fcast.expanding.DHR = forecast(fit.expanding.DHR, xreg = fourier(train.expanding[,1], 1), h = H)
  fcast.sliding.DHR = forecast(fit.sliding.DHR, xreg = fourier(train.sliding[,1], 1), h = H)

  error.expanding.DHR[i, 1:length(test[,1])] = (abs(fcast.expanding.DHR[['mean']] - test[,1])/test[,1])/length(test[,1])*1
00
  error.sliding.DHR[i, 1:length(test[,1])] = (abs(fcast.sliding.DHR[['mean']] - test[,1])/test[,1])/length(test[,1])*100

}
```

```
## Warning in window.default(x, ...): 'end' value not changed
```

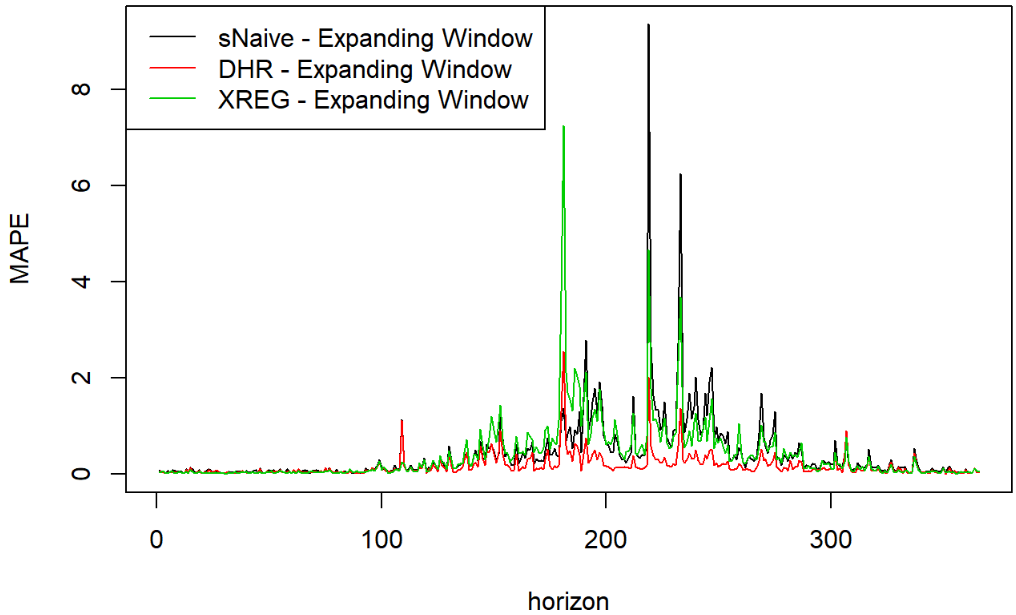
3. Regression with Arima errors

```
n = length(divvy[,1]) # number of data points
p = 365 # period
H = 366 # forecast horizon

st = tsp(divvy)[1] # gives the start time in time units

error.expanding.xreg = matrix(NA, floor(n/H), H)
error.sliding.xreg = matrix(NA, floor(n/H), H)

for (i in 1:floor(n/H)){

  train.expanding = window(divvy, end = st + i) # expanding window
  train.sliding = window(divvy, start = st + i - 1, end = st + i) # sliding window

  weather.train.expanding = window(weather, end = st + i)
  weather.train.sliding = window(weather, start = st + i - 1, end = st + i)

  test = window(divvy, start = st + i + 1/p, end = st + i + 1 + 1/p)

  weather.test = window(weather, start = st + i + 1/p, end = st + i + 1 + 1/p)

  xreg.expanding = cbind(PRCP = weather.train.expanding[,2], SNOW = weather.train.expanding[,3], TEMP = weather.train.expa
nding[,5], STAT = train.expanding[,2])
  xreg.sliding = cbind(PRCP = weather.train.sliding[,2], SNOW = weather.train.sliding[,3], TEMP = weather.train.sliding[,5
], STAT = train.sliding[,2])
  xreg.test = cbind(PRCP = weather.test[,2], SNOW = weather.test[,3], TEMP = weather.test[,5], STAT = test[,2])

  fit.expanding.xreg = Arima(train.expanding[,1], order = c(4,1,5), lambda = "auto", xreg = xreg.expanding)
  fit.sliding.xreg = Arima(train.sliding[,1], order = c(4,1,5), lambda = "auto", xreg = xreg.sliding)

  fcast.expanding.xreg = forecast(fit.expanding.xreg, xreg = xreg.test, h = H)
  fcast.sliding.xreg = forecast(fit.sliding.xreg, xreg = xreg.test, h = H)

  error.expanding.xreg[i, 1:length(test[,1])] = (abs(fcast.expanding.xreg[['mean']] - test[,1])/test[,1])/length(test[,1])
*100
  error.sliding.xreg[i, 1:length(test[,1])] = (abs(fcast.sliding.xreg[['mean']] - test[,1])/test[,1])/length(test[,1])*100

}
```

```
## Warning in window.default(x, ...): 'end' value not changed

## Warning in window.default(x, ...): 'end' value not changed
```

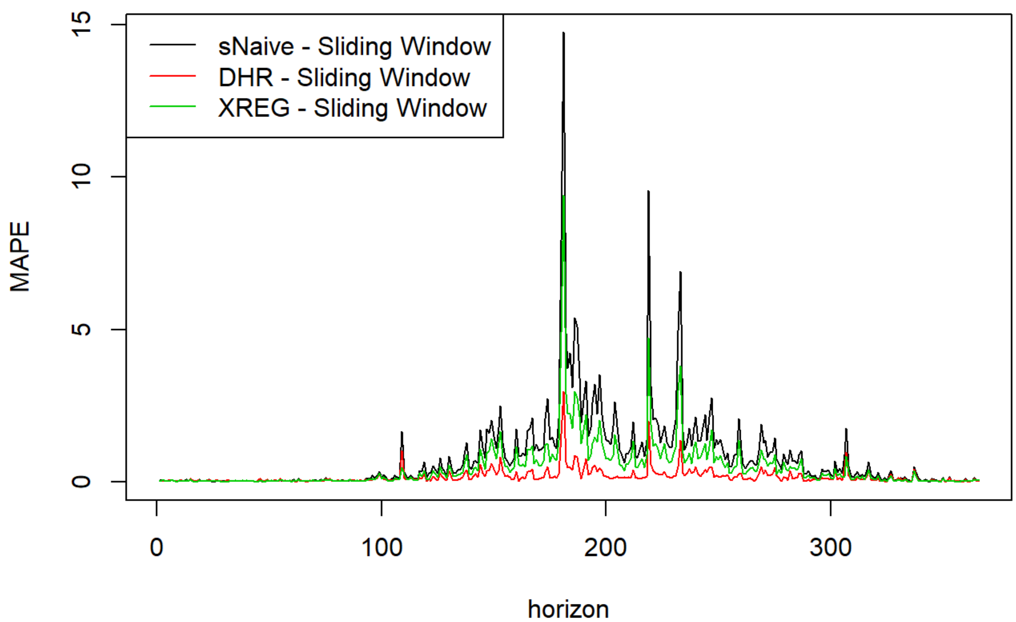
We are using MAPE to evaluate forecast accuracy because some of the models are using normalized data and some of them are using regular data. MAPE is independent of scale and hence it is the best accurary measure for our analysis.

```
plot(1:366, colMeans(error.expanding.sNaive, na.rm = TRUE), type = "l", col = 1, xlab = "horizon", ylab = "MAPE", ylim = c
(0, max(colMeans(error.expanding.sNaive, na.rm = TRUE))))
lines(1:366, colMeans(error.expanding.DHR, na.rm = TRUE), type = "l", col = 2)
lines(1:366, colMeans(error.expanding.xreg, na.rm = TRUE), type = "l", col = 3)
legend("topleft",legend=c("sNaive - Expanding Window","DHR - Expanding Window", "XREG - Expanding Window"), col = 1:3, lty
=1)
```



Under expanding window method, DHR model performs the best as it has the lowest MAPE throughout the forecast horizon.

```
plot(1:366, colMeans(error.sliding.sNaive, na.rm = TRUE), type = "l", col = 1, xlab = "horizon", ylab = "MAPE", ylim = c(0
, max(colMeans(error.sliding.sNaive, na.rm = TRUE))))
lines(1:366, colMeans(error.sliding.DHR, na.rm = TRUE), type = "l", col = 2)
lines(1:366, colMeans(error.sliding.xreg, na.rm = TRUE), type = "l", col = 3)
legend("topleft",legend=c("sNaive - Sliding Window","DHR - Sliding Window", "XREG - Sliding Window"), col = 1:3, lty=1)
```



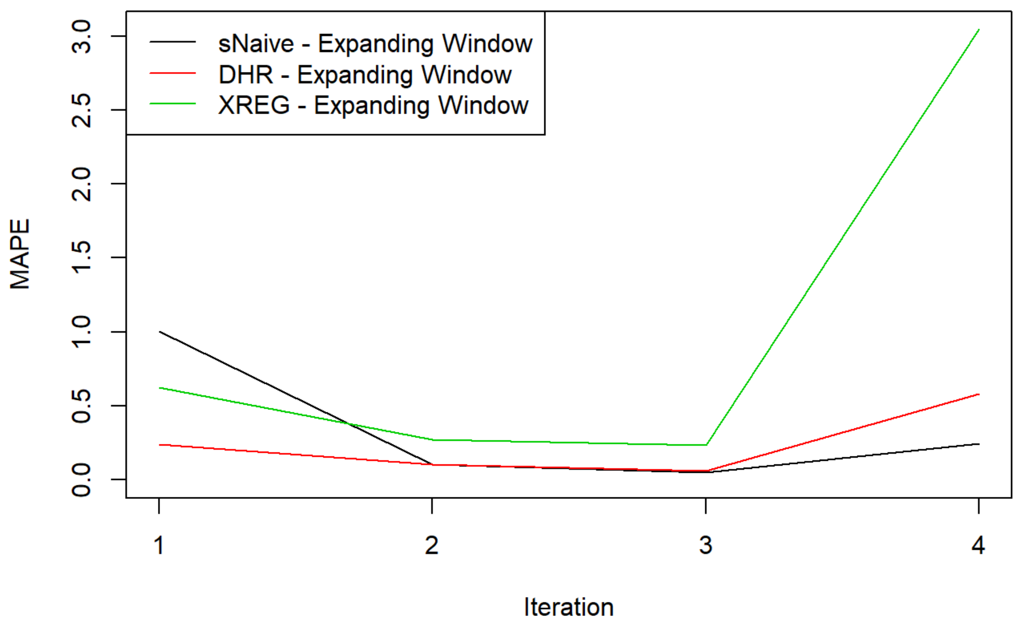
DHR model has the lowest MAPE throughout the forecast horizon and hence it is also the best performing model under sliding window approach.

Overall, models trained by expanding window are better than models trained by sliding window approach.

```
sNaive.mape.expanding = cbind(mean(error.expanding.sNaive[1,]), mean(error.expanding.sNaive[2,]), mean(error.expanding.sNaive[3,]^2), mean(error.expanding.sNaive[4,]^2, na.rm = TRUE))
DHR.mape.expanding = cbind(mean(error.expanding.DHR[1,]), mean(error.expanding.DHR[2,]), mean(error.expanding.DHR[3,]^2),
 mean(error.expanding.DHR[4,]^2, na.rm = TRUE))
xreg.mape.expanding = cbind(mean(error.expanding.xreg[1,]), mean(error.expanding.xreg[2,]), mean(error.expanding.xreg[3,]^2), mean(error.expanding.xreg[4,]^2, na.rm = TRUE))

MAPE.expanding = rbind(sNaive.mape.expanding, DHR.mape.expanding, xreg.mape.expanding)
colnames(MAPE.expanding) = c("366:366", "731:366", "1096:366", "1461:366")
rownames(MAPE.expanding) = c("sNaive MAPE", "DHR MAPE", "XREG MAPE")
MAPE.expanding
```

```
##              366:366   731:366   1096:366  1461:366
## sNaive MAPE 1.0059804 0.1062657 0.04907771 0.2425883
## DHR MAPE    0.2401221 0.1059585 0.05813865 0.5805970
## XREG MAPE   0.6251870 0.2701163 0.23543300 3.0445922
```

```
plot(1:4, MAPE.expanding[1,], type = "l", col = 1, xlab = "Iteration", ylim=c(0,max(MAPE.expanding)), ylab = "MAPE",xaxt='n')
lines(1:4, MAPE.expanding[2,], type = "l", col = 2)
lines(1:4, MAPE.expanding[3,], type = "l", col = 3)
legend("topleft",legend=c("sNaive - Expanding Window","DHR - Expanding Window","XREG - Expanding Window"), col = 1:3, lty=1)
axis(side=1, at=c(1:4))
```

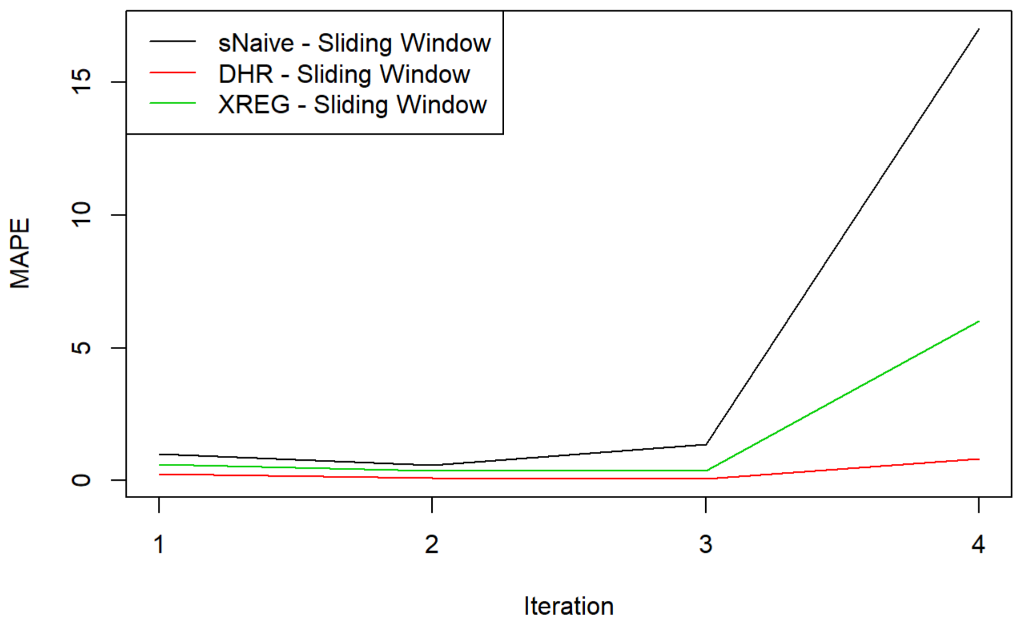


DHR model has the highest stability and accurary throughout the sampling iterations. sNaive, DHR, and XREG models have the lowest MAPE when data is split using 3:1 ratio. VAR model has lowest MAPE at 2:1 split.

```
sNaive.mape.sliding = cbind(mean(error.sliding.sNaive[1,]), mean(error.sliding.sNaive[2,]), mean(error.sliding.sNaive[3,]^
2), mean(error.sliding.sNaive[4,]^2, na.rm = TRUE))
DHR.mape.sliding = cbind(mean(error.sliding.DHR[1,]), mean(error.sliding.DHR[2,]), mean(error.sliding.DHR[3,]^2), mean(err
or.sliding.DHR[4,]^2, na.rm = TRUE))
xreg.mape.sliding = cbind(mean(error.sliding.xreg[1,]), mean(error.sliding.xreg[2,]), mean(error.sliding.xreg[3,]^2), mean
(error.sliding.xreg[4,]^2, na.rm = TRUE))

MAPE.sliding = rbind(sNaive.mape.sliding, DHR.mape.sliding, xreg.mape.sliding)
colnames(MAPE.sliding) = c("366:366", "366:366", "366:366", "366:366")
rownames(MAPE.sliding) = c("sNaive MAPE", "DHR MAPE", "XREG MAPE")
MAPE.sliding
```

```
##              366:366   366:366    366:366    366:366
## sNaive MAPE 1.0059804 0.5755332 1.35352922 17.0228219
## DHR MAPE    0.2401221 0.1102577 0.06115787  0.8273782
## XREG MAPE   0.6251870 0.3787146 0.35613641  6.0083155
```

```
plot(1:4, MAPE.sliding[1,], type = "l", col = 1, xlab = "Iteration", ylab = "MAPE", ylim = c(min(MAPE.sliding), max(MAPE.s
liding)), xaxt='n')
lines(1:4, MAPE.sliding[2,], type = "l", col = 2)
lines(1:4, MAPE.sliding[3,], type = "l", col = 3)
legend("topleft",legend=c("sNaive - Sliding Window","DHR - Sliding Window", "XREG - Sliding Window"), col = 1:4, lty=1)
axis(side=1, at=c(1:4))
```



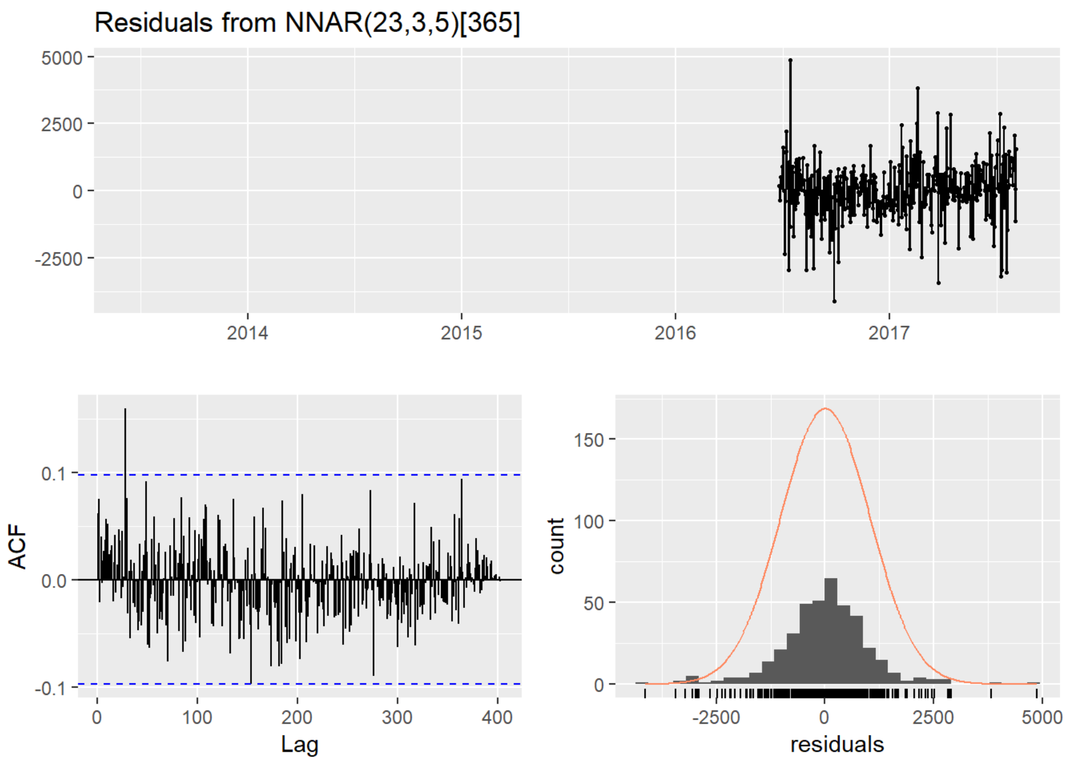
Again, DHR model has the highest stability and accurary throughout the sampling iterations. sNaive and VAR models have the lowest MAPE under second sliding window while DHR and XREG models perform the best under third sliding window.

F. Future Work - Neural Networks

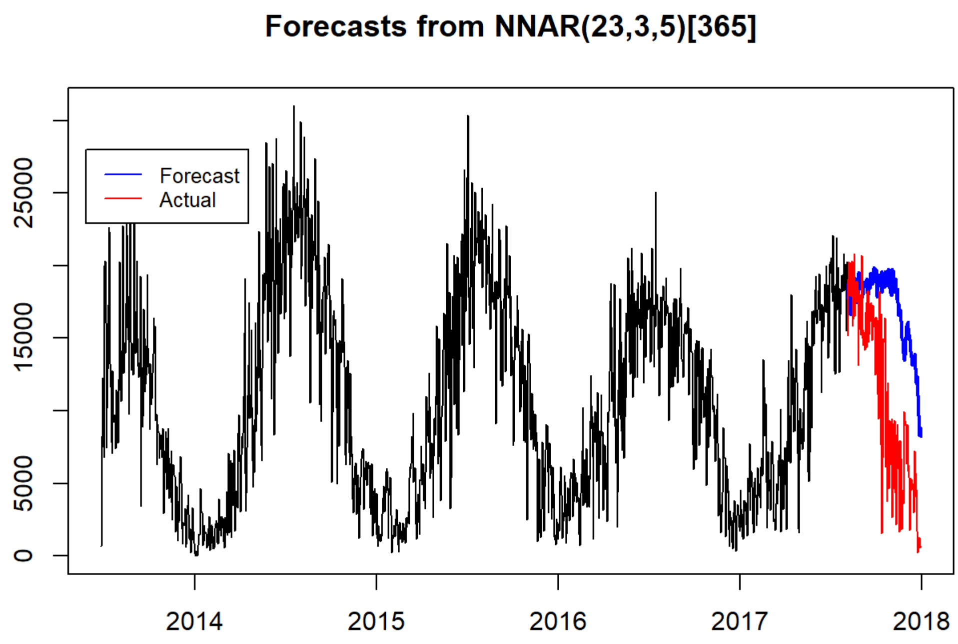
```
#nnetar() in forecast library
nn2 = nnetar(train.norm[,1], P = 3, size = 5, repeats = 10)
nn2
```

```
## Series: train.norm[, 1]
## Model:  NNAR(23,3,5)[365]
## Call:   nnetar(y = train.norm[, 1], P = 3, size = 5, repeats = 10)
##
## Average of 10 networks, each of which is
## a 26-5-1 network with 141 weights
## options were - linear output units
##
## sigma^2 estimated as 1058940
```

```
nn2.forecast = forecast(nn2, h = 148)
checkresiduals(nn2.forecast)
```




```
plot(nn2.forecast)
lines(test.norm[,1], type  = "l", col = "red")
legend(2013.4, 28000, inset = c(-0.2,5), legend = c("Forecast", "Actual"), col = c("blue", "red"), lty = 1, cex = 0.8)
```




```
acc.nn2 = accuracy(nn2.forecast$mean, test.norm[,1])
acc.nn2
```

```
##                 ME     RMSE      MAE       MPE     MAPE      ACF1
## Test set -6172.416 7807.259 6471.636 -205.1846 206.7083 0.8095958
##          Theil's U
## Test set  7.945414
```