STAT 443 Group 21 Project Code

Fall 2025

Packages Used

```
#install.packages('zoo')
#install.packages('forecast')
#install.packages('MuMIn')
#install.packages("gridExtra")
#install.packages("cowplot")
#install.packages('ggplot')
#install.packages('ggplot2')
#install.packages('gridExtra')
#install.packages('ggfortify')
#install.packages('qqplotr')
#install.packages("ggpubr")
library(gridExtra)
library(MuMIn)
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(cowplot)
library(ggplot2)
library(gridExtra)
library(ggfortify)
```

```
## Registered S3 methods overwritten by 'ggfortify':
##   method                 from
##   autoplot.Arima         forecast
##   autoplot.acf           forecast
##   autoplot.ar            forecast
##   autoplot.bats          forecast
##   autoplot.decomposed.ts forecast
##   autoplot.ets           forecast
##   autoplot.forecast      forecast
##   autoplot.stl           forecast
##   autoplot.ts            forecast
##   fitted.ar              forecast
##   fortify.ts             forecast
##   residuals.ar           forecast
```

```
library(qqplotr)
```

```
##
## Attaching package: 'qqplotr'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     stat_qq_line, StatQqLine
```

```
library(ggpubr)
```

```
## Registered S3 methods overwritten by 'broom':
##   method        from
##   nobs.fitdistr MuMIn
##   nobs.multinom MuMIn
```

```
##
## Attaching package: 'ggpubr'
```

```
## The following object is masked from 'package:cowplot':
##
##     get_legend
```

```
## The following object is masked from 'package:forecast':
##
##     gghistogram
```

Data

```
set.seed(21) # our lucky number, group 21
data <- read.csv('Data_Group21.csv', header=TRUE)
library(zoo)
```

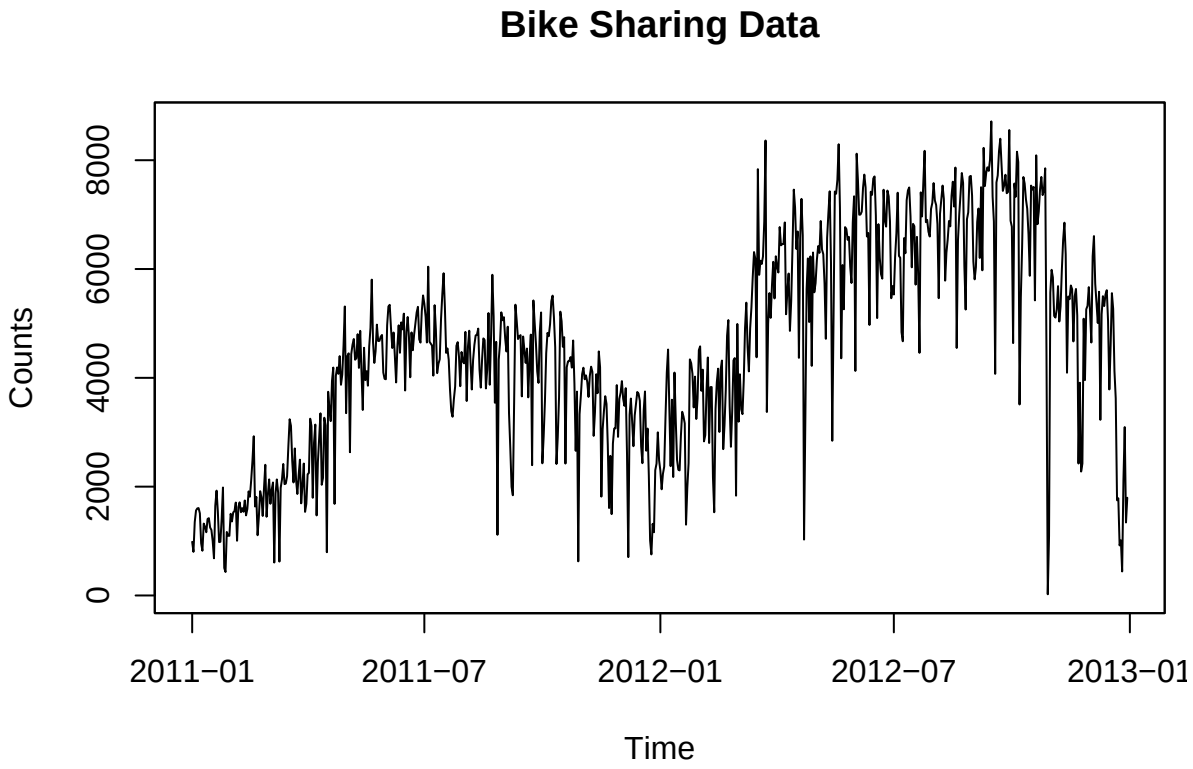
```
##
## Attaching package: 'zoo'
```

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

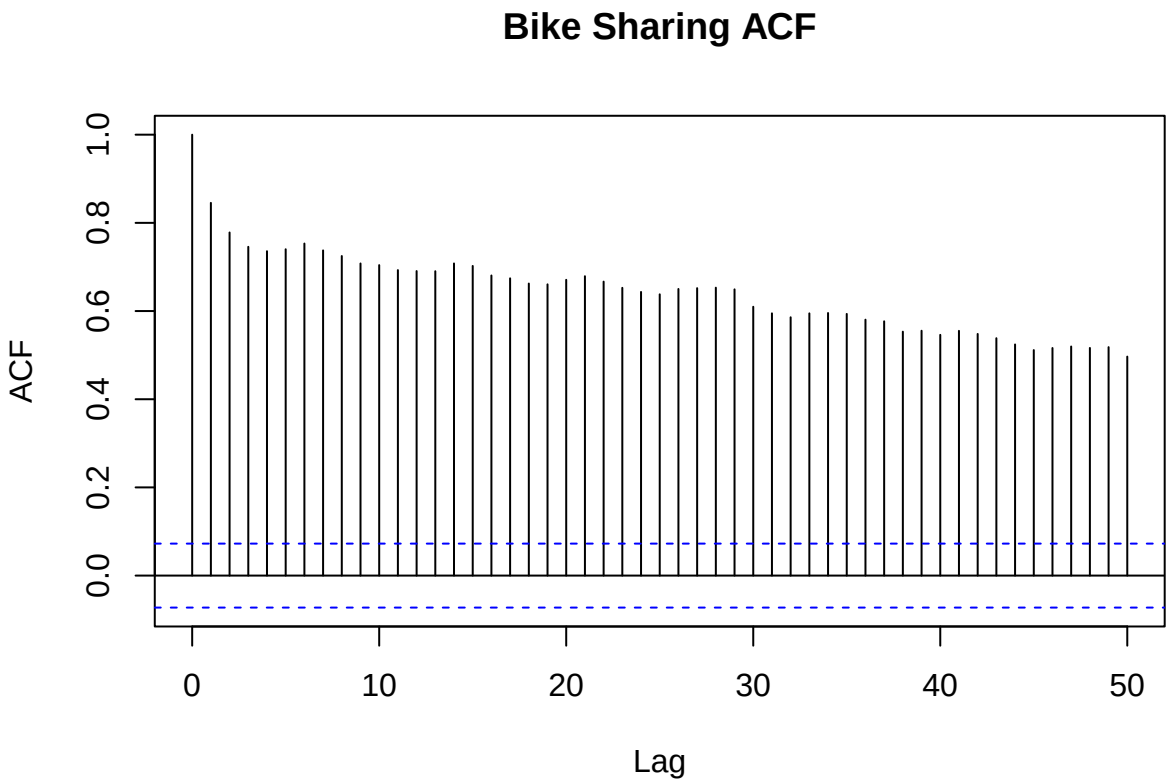
```
dates <- seq(as.Date('2011-01-01'), as.Date('2012-12-30'), by='day')
num.obs <- length(dates) # total 730 observations

# dataset
bike.data <- data.frame(dates = dates,
                        bikes = data$BikeSharing[1:num.obs],
                        t = 1:num.obs)
# time series for data
Bike <- zoo(data$BikeSharing[1:num.obs], order.by=dates)

# prediction indices (for prediction untill 2013-01-31)
#pred.dates.full <- seq(as.Date('2011-01-01'), as.Date('2013-01-31'), by='day')
#plot.range <- as.Date(c('2011-01-01', '2013-02-01'))
pred.dates.full <- seq(as.Date('2011-01-01'), as.Date('2013-12-31'), by='day')
plot.range <- as.Date(c('2011-01-01', '2013-12-31'))


# number of days to be predicted (2012-12-31 to 2013-01-31)
pred.days <- length(pred.dates.full) - length(dates)

# training indices + prediction indices
```

```r
t.full <- 1:length(pred.dates.full)

# dummy y for obtaining matrix
y.dummy <- rep(1, length(t.full))

# indices for prediction only
t.pred <- length(bike.data$t) + 1:pred.days

# dates for prediction only
dates.pred = pred.dates.full[t.pred]
```

```r
# plots for raw data
plot(Bike, type='l', main='Bike Sharing Data', xlab='Time', ylab='Counts')
```




```r
acf(Bike, main='Bike Sharing ACF', lag.max = 50)
```



Utility functions for plotting residuals:

```
### advanced ggplot
super.residual.plot <- function(name, res, fitted, res.band, acf.lags=50) {
    # qq plot
    #res <- model$residuals
    res.qqplot <- ggpubr::ggqqplot(res, xlab='N(0,1)', ylab='Residuals',
                                   conf.int=TRUE, conf.level=0.95) +
                                   ggtitle(paste0('Residual QQ-plot: ', name))

    # acf
    res.acf <- ggAcf(res, lag.max = acf.lags) +
               ggtitle(paste0('Residual ACF: ', name))+
               theme_minimal()

    # residuals vs fitted
    res.fit.plot <- ggplot(data.frame(fitted=fitted, residuals=res),
                           aes(x=fitted, y=res)) +
                           geom_point(shape=1) +
                           geom_hline(yintercept=0,linetype='dashed',color='red')+
                           ggtitle(paste0('Residuals vs Fitted: ', name)) +
                           xlab('Fitted') + ylab('Residuals') + theme_minimal()
    # residuals vs time
    # residual 95% CI
    #H <- matrix.train %*% X.T.X.inv %*% t(matrix.train) # projection matrix
    #res.band <- 1.96 * model.sd * (1 - diag(H))
    res.time.plot <- ggplot(data.frame(time=1:length(res), residuals=res,
                                       res.band=res.band),
                            aes(x=time, y=residuals)) +
                     geom_ribbon(aes(ymin=res-res.band, ymax=res+res.band),
```

```
                                fill='blue',alpha=0.4) +
                    geom_line(color='black') +
                    geom_hline(yintercept=0,linetype='dashed',color='red') +
                    ggtitle(paste0('Residuals vs Time: ', name)) +
                    xlab('Time') + ylab('Residuals') + theme_minimal()
    grid.arrange(res.qqplot, res.acf, res.fit.plot, res.time.plot, nrow=2)
}
```

Holt-Winters

```
#plot(FIT$model)
bikes <- ts(Bike, start = 2011, frequency = 365)

hw <- HoltWinters(bikes, seasonal = 'mult')
head(hw$fitted[,'xhat'])
hw
#head(hw)
pred.hw <- predict(hw, n.ahead = pred.days, prediction.interval = TRUE, level = 0.95)
#head(hw$fitted)
frequency(bikes)
head(pred.hw)
#head(pred.hw[, 'fit'])
plot(hw)
plot(hw, pred.hw)
plot(pred.hw)
```

```
bikes <- ts(bike.data$bikes, frequency = 365, start = 2011)


hw.mse <- function(residuals) {
  return(mean(residuals^2))
}

hw.apse <- function(data, train.size, seasonal) {
  # train size is at least one cycle
  # if(train.size < (2 * frequency(data))) {
  #   return(NA)
  # }
  # recommend using last 1/5 data for validation
  train.idx <- 1:train.size
  data.train <- ts(data[train.idx], frequency=frequency(data), start=start(data))
  data.val <- data[-train.idx]
  model <- HoltWinters(data.train, seasonal = seasonal)
  pred.model <- predict(model, n.ahead = length(data.val))
  pred.values <- pred.model[, 'fit']
  return(hw.mse(data.val - pred.values))
}

plot.hw.residuals <- function(fitted, residuals) {
  # normality
  # some code are from tutorial code ResidualDiagnostics-Rcodes.R
  par(mfrow = c(2, 2))
```

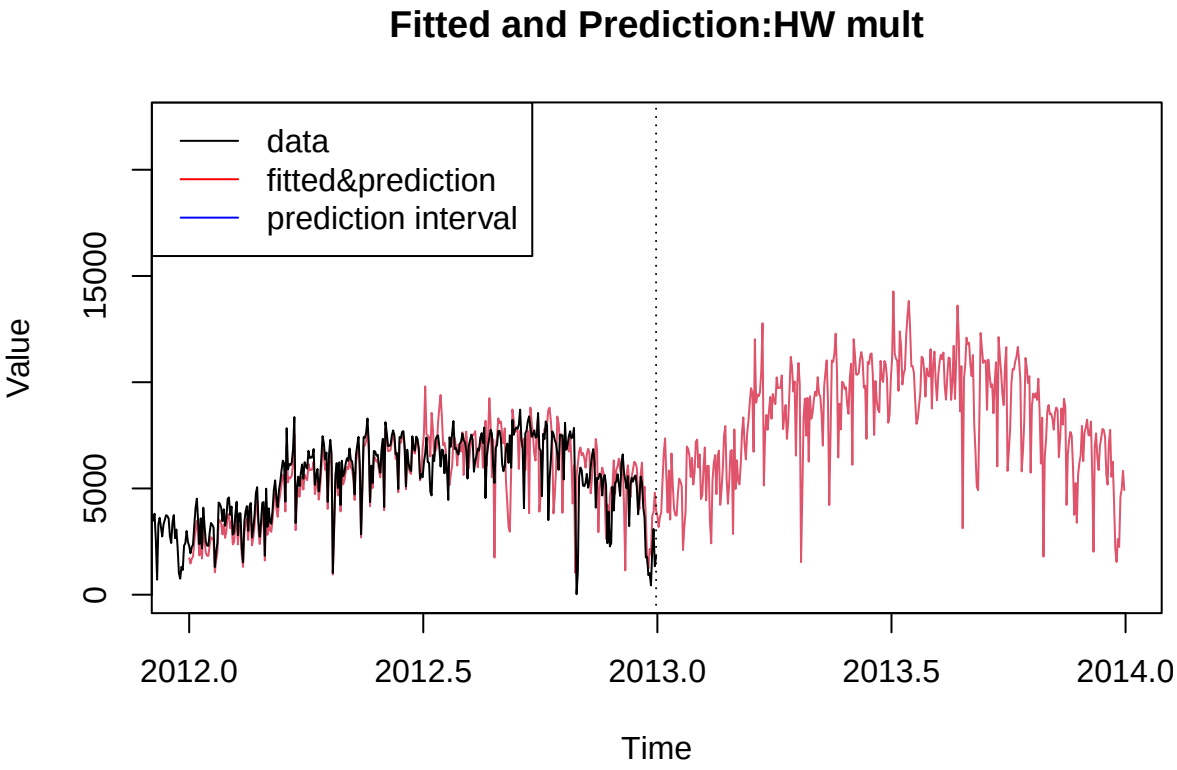
```
  # qq plot
  car::qqPlot(
    residuals,
    pch = 16,
    col = adjustcolor("black", 0.7),
    xlab = "Normal (0,1)",
    ylab = "Residuals",
    main = paste0("Q-Q Plot")
  )
  # acf
  acf(residuals, main = 'Residual ACF', lag.max = 60)

  # residuals vs fitted
  plot(as.vector(fitted), as.vector(residuals), main = 'Residuals vs Fitted',
       xlab = 'Fitted', ylab = 'Residuals')
  abline(h = 0, col = 'red')

  # residuals vs time
  plot(residuals, type = 'l', main = 'Residuals vs Time',
       xlab = 'Residuals', ylab = 'Time')
  abline(h = 0, col = 'red')
  par(mfrow = c(1, 1))
}
```

Analytical tools:

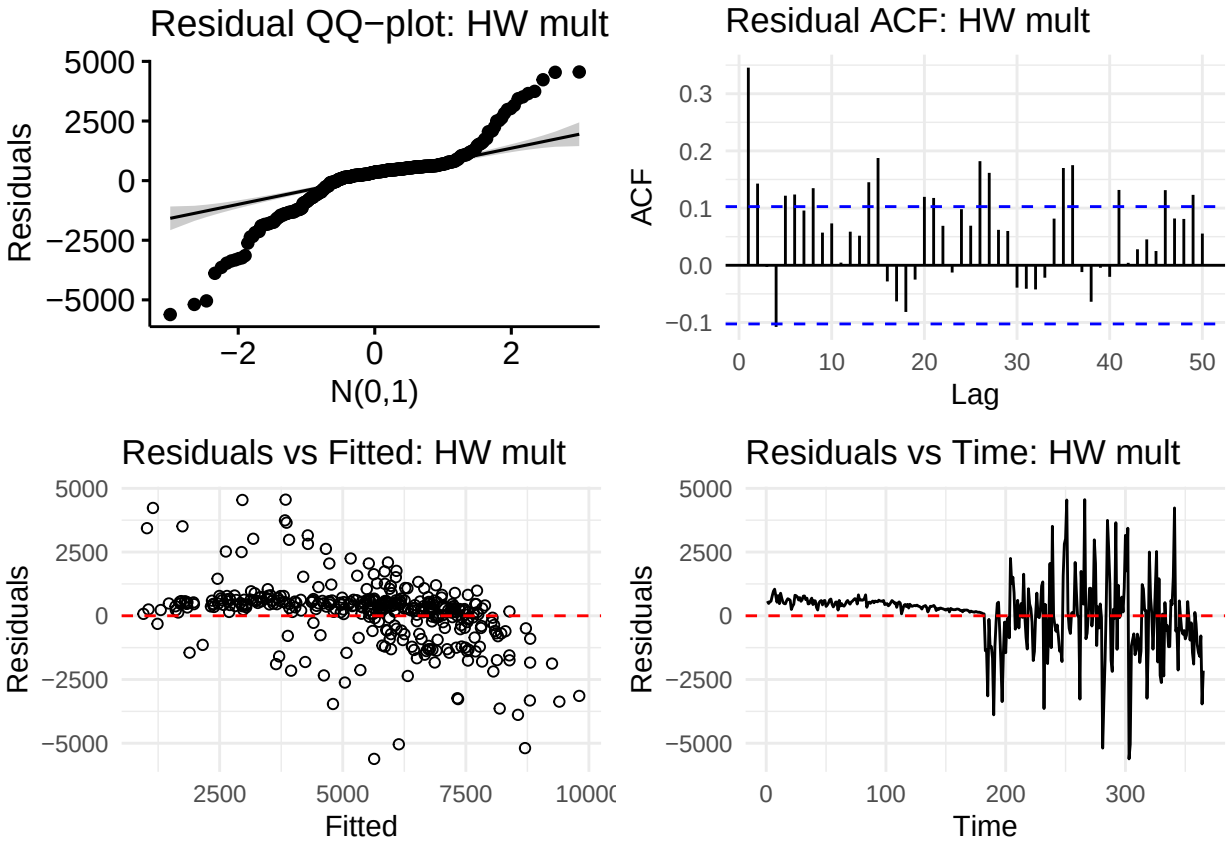
```
hw.analysis <- function(name, data, seasonal,
                        pred.days,
                        params = FALSE,
                        print.summary = FALSE,
                        plot = TRUE, plot.residuals = TRUE) {
  fit.model <- NA
  if(!isFALSE(params)) {
    fit.model <- HoltWinters(data, alpha = params['alpha'],
                             beta = params['beta'],
                             gamma = params['gamma'],
                             seasonal = seasonal)
  } else {
    fit.model <- HoltWinters(data, seasonal = seasonal)
  }

  pred.model <- predict(fit.model, n.ahead = pred.days,
                        prediction.interval = FALSE, level = 0.95)
  fitted <- fit.model$fitted[, 'xhat']
  # the original data that are being fitted
  data.fitted <- data[-(1:frequency(data))]
  residuals <- data.fitted - fitted
  mse <- hw.mse(residuals)
  # apse data has less than two periods. not valid
  #apse <- hw.apse(data, as.integer(length(data) * (4/5)), seasonal)
  #pred.se <- (pred.model[10, 'upr'] - pred.model[10, 'lwr']) / 2
  normality <- shapiro.test(residuals)$p.value
  randomness <- randtests::runs.test(residuals)$p.value
```

```
  if(print.summary) {
    print(fit.model)
  }

  if(plot) {
    plot(fit.model, pred.model, main = paste0('Fitted and Prediction:', name),
         xlab = 'Time', ylab = 'Value',
         # ylim = c(min(c(data, pred.model[, 'lwr'])),
         #          max(c(data, pred.model[, 'upr'])) + 8000))
         ylim = c(min(c(data, pred.model[, 'fit'])),
                  max(c(data, pred.model[, 'fit'])) + 8000))
    legend('topleft', legend = c('data',
                                 'fitted&prediction',
                                 'prediction interval'),
           col = c('black', 'red', 'blue'),
           lty = 1)
  }

  if(plot.residuals) {
    #plot.hw.residuals(fitted, as.vector(residuals)) # update technology
    super.residual.plot(name, as.vector(residuals), fitted, 0)
  }

  return(data.frame(#pred.se = pred.se,
                    mse = mse,
                    #apse = apse,
                    normality = normality,
                    randomness = randomness))
}
```

```
bikes <- ts(bikes, frequency = 365, start = 2011)
init.guess <- c(alpha = 0.1, beta = 0.7, gamma = 0.6)
hw.analysis('HW mult', bikes, 'mult', 365)
```

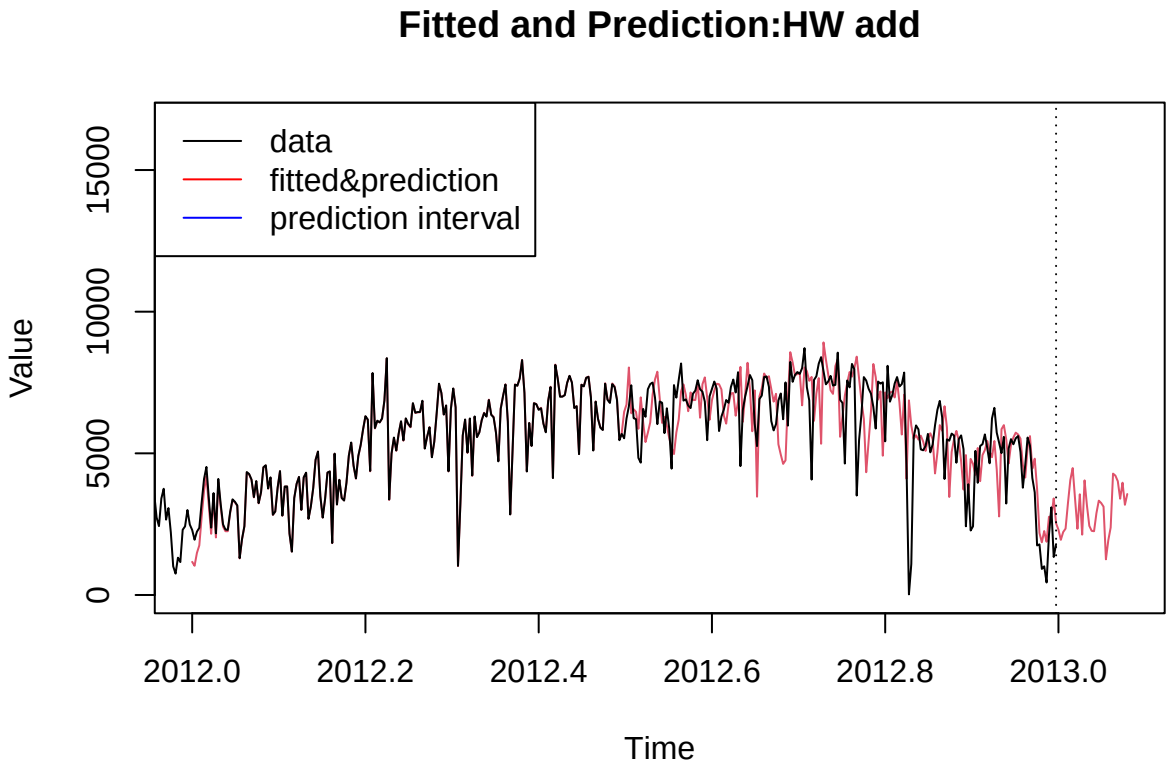



```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```

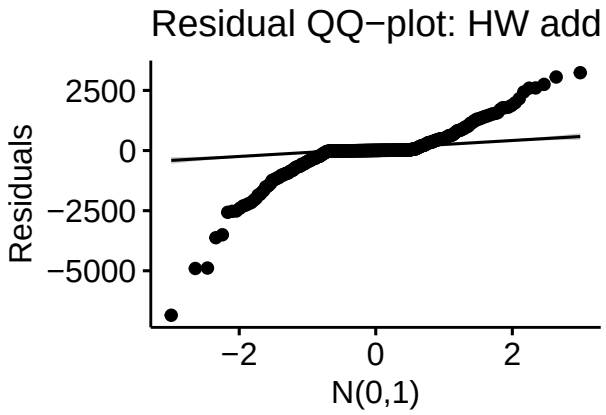


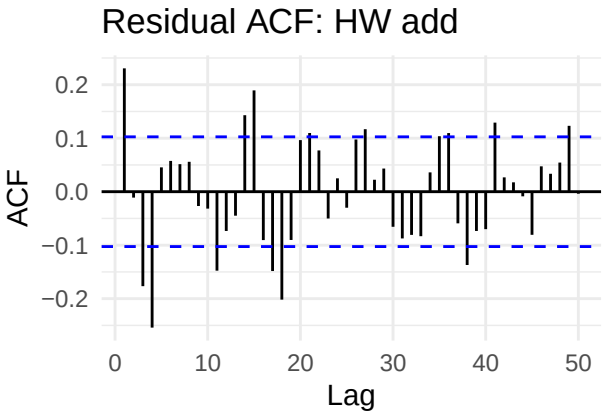
```
##       mse    normality   randomness
## 1 1569180 2.277176e-17 2.907697e-26
```

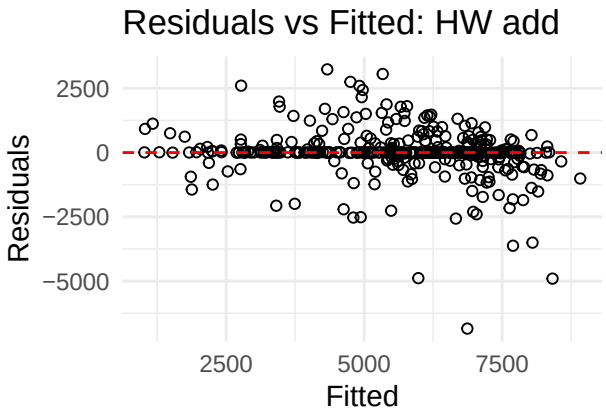
```
hw.analysis('HW add', bikes, 'add', 30)
```

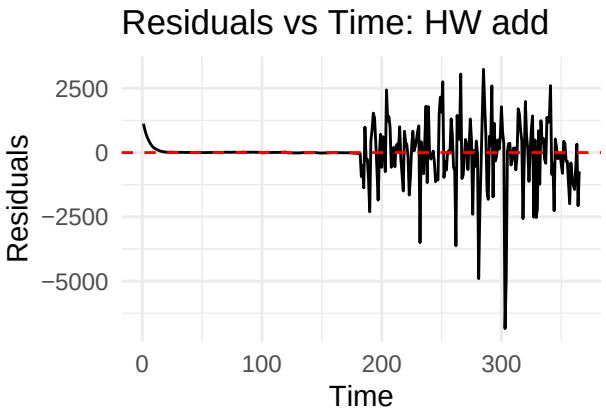



```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



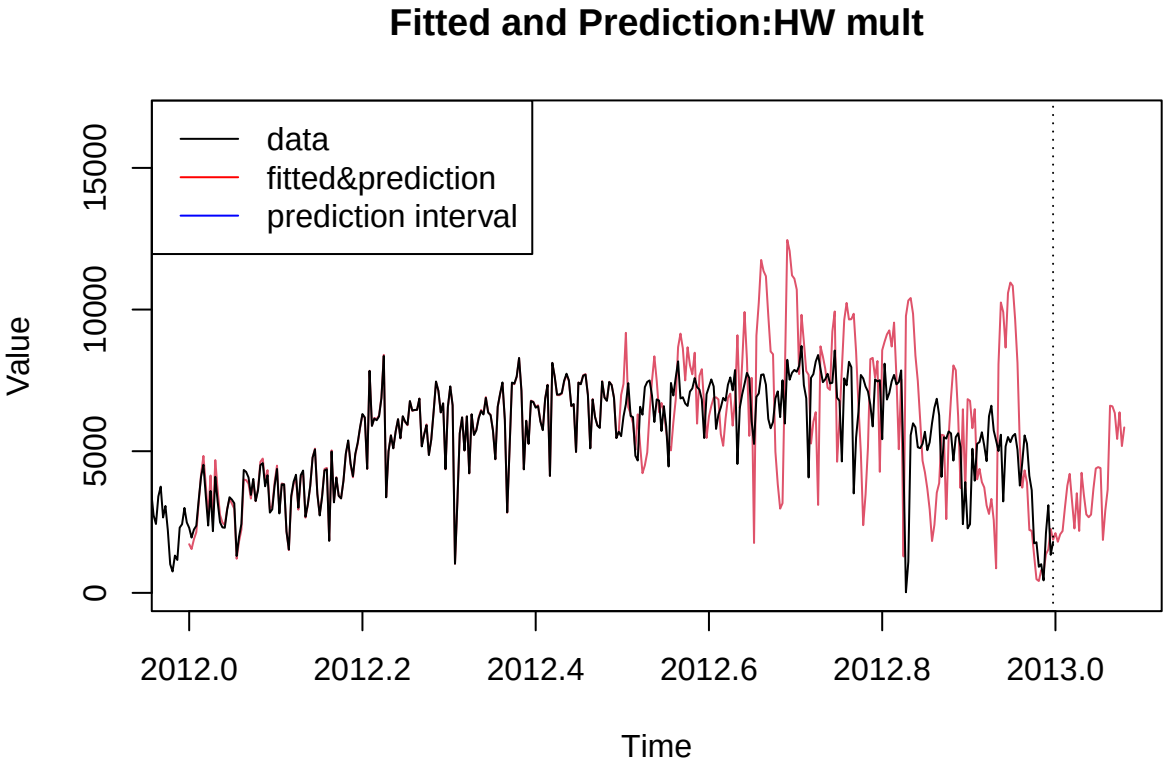




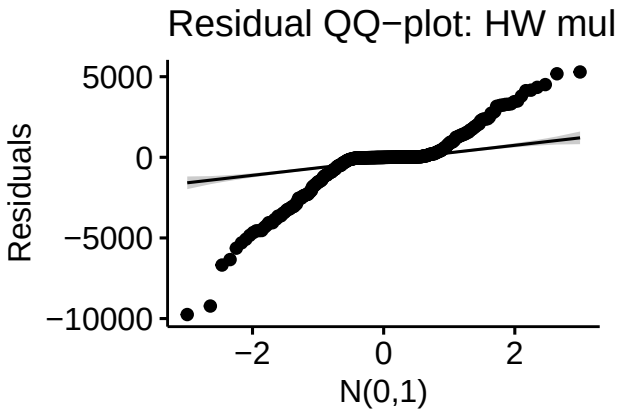


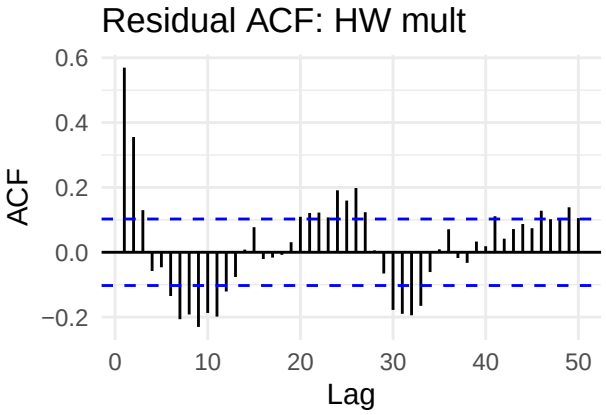

```
##      mse    normality   randomness
## 1 954591 7.192613e-22 1.630761e-22
```

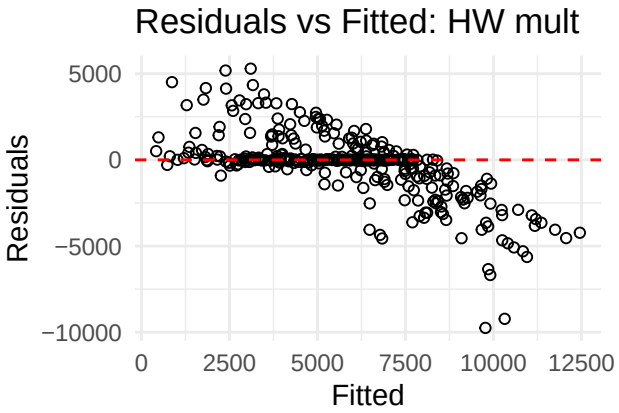
```
hw.analysis('HW mult', bikes, 'mult', 30, init.guess)
```

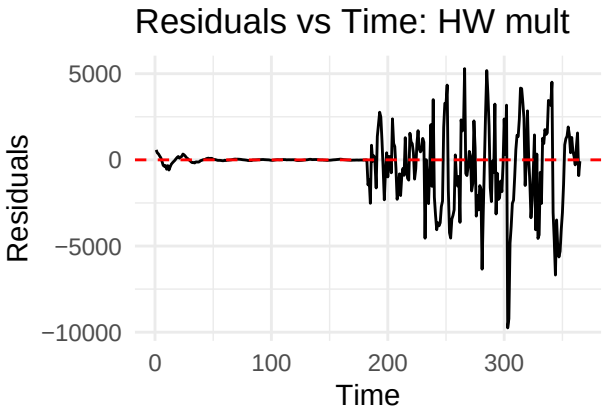



```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



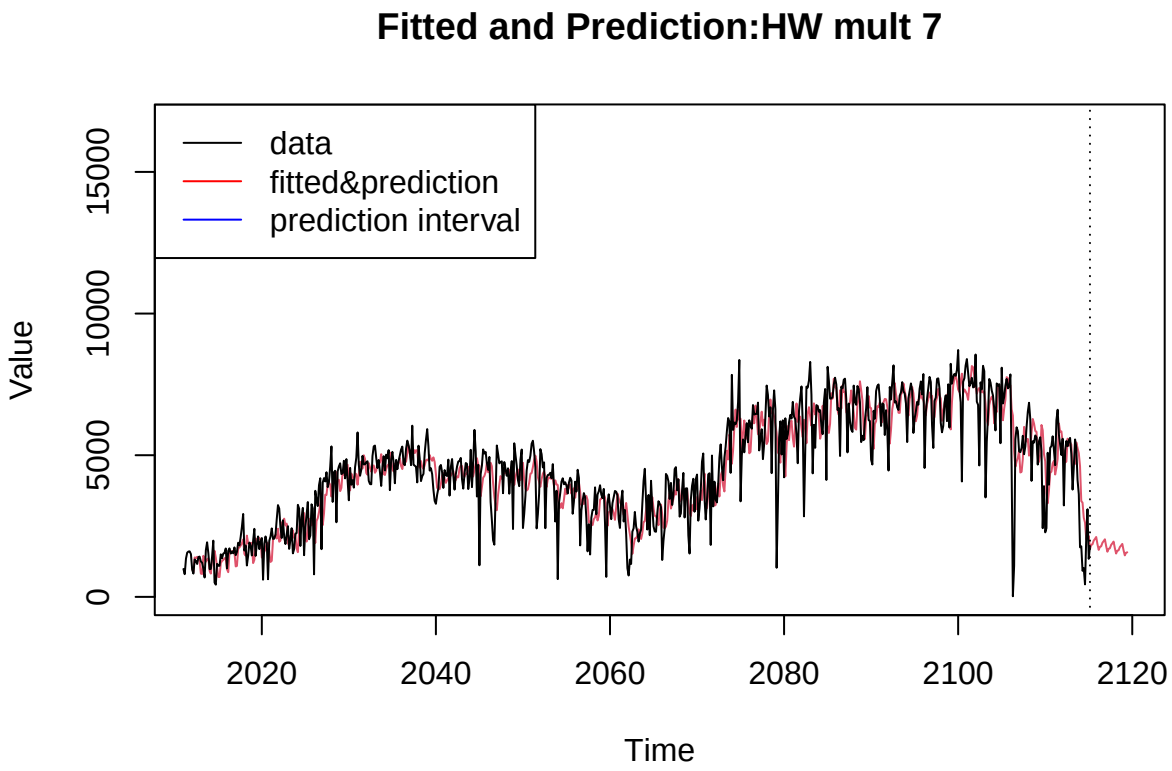




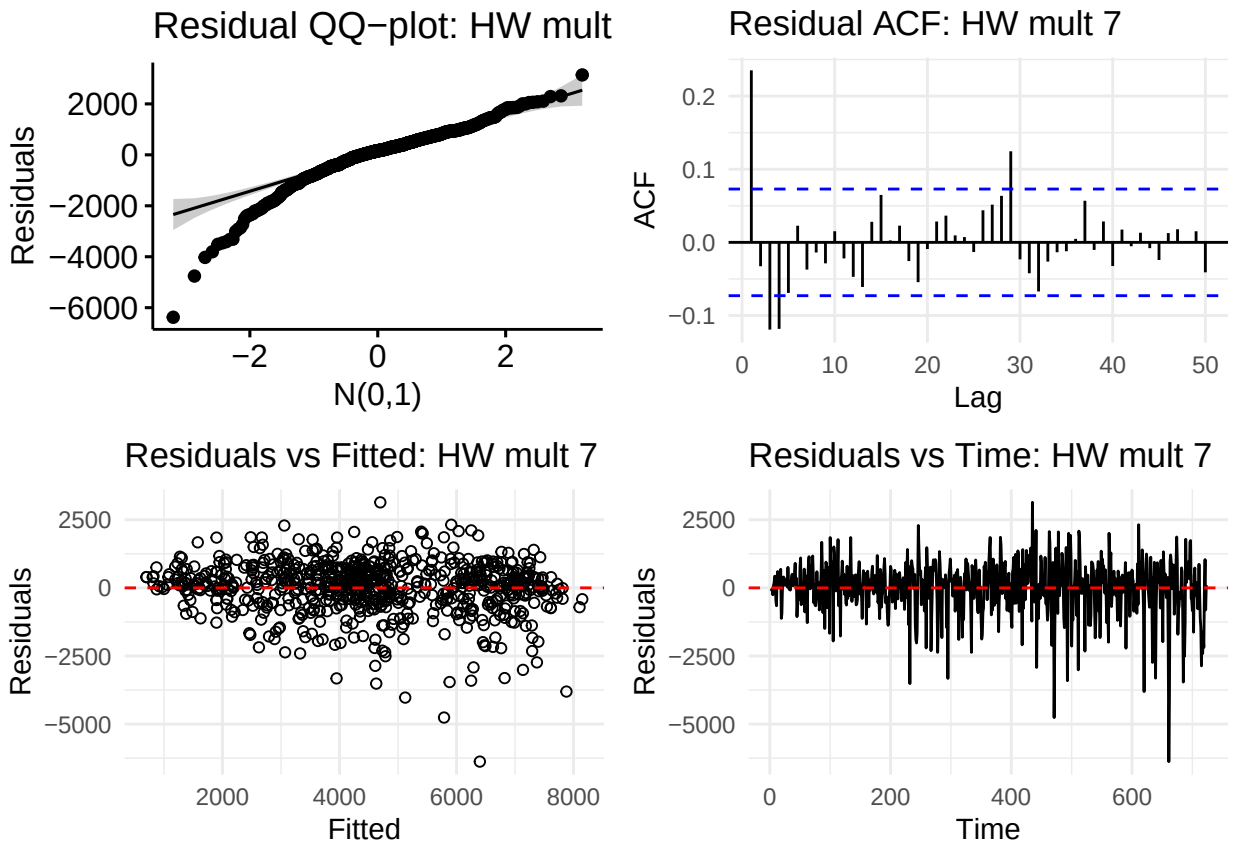


```
##       mse   normality  randomness
## 1 3460220 5.11957e-18 2.24077e-31
```

```
bikes <- ts(bikes, frequency = 7, start = 2011)
hw.analysis('HW mult 7', bikes, 'mult', 30)
```




```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```




```
##        mse    normality  randomness
## 1 956499.3 3.375413e-18 5.40851e-08
```

```
# apse of ts with period 7
hw.apse(bikes, as.integer(length(bikes) * (4/5)), 'mult')
```

```
## [1] 7819302
```

Differencing

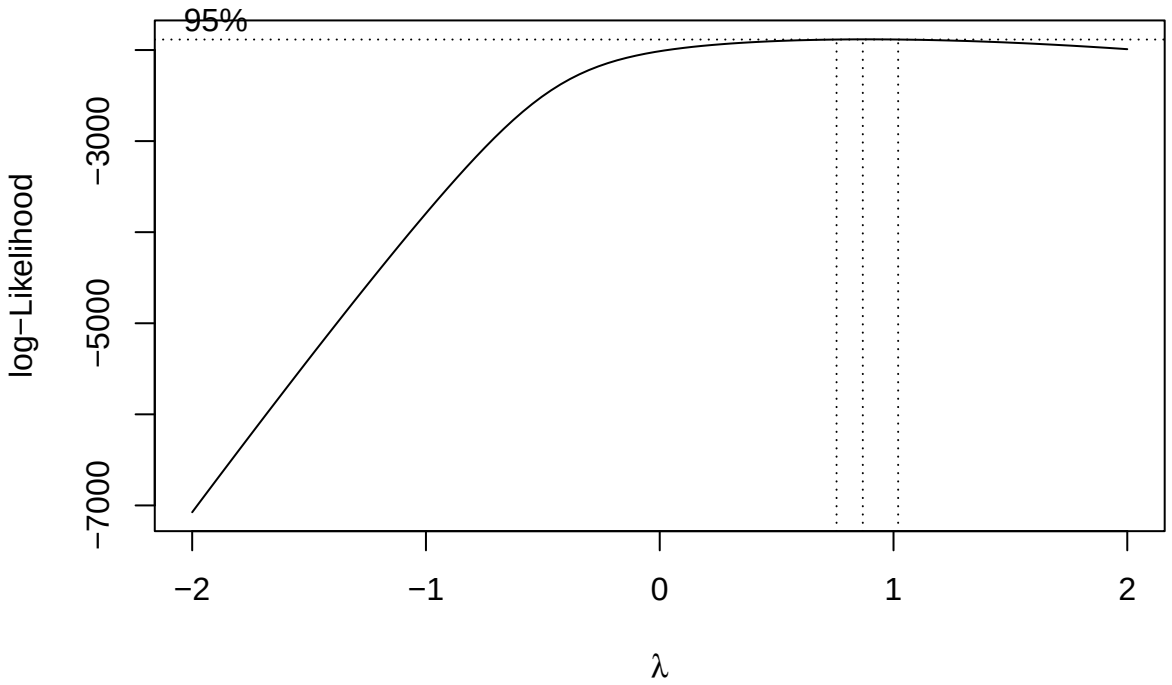
Stationarity analsis/test on data

```
library(astsa)
```

```
##
## Attaching package: 'astsa'
```

```
## The following object is masked from 'package:forecast':
##
##     gas
```

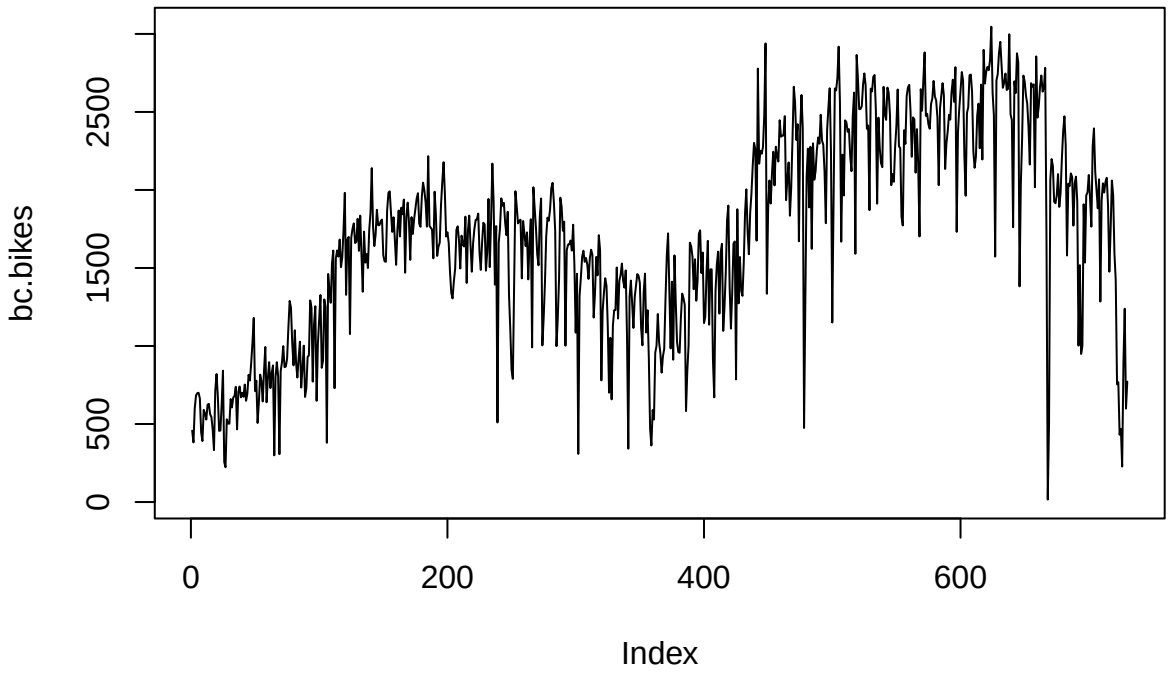
```
# box-cox trans
bc.bikes = MASS::boxcox(bike.data$bikes ~ 1)
```



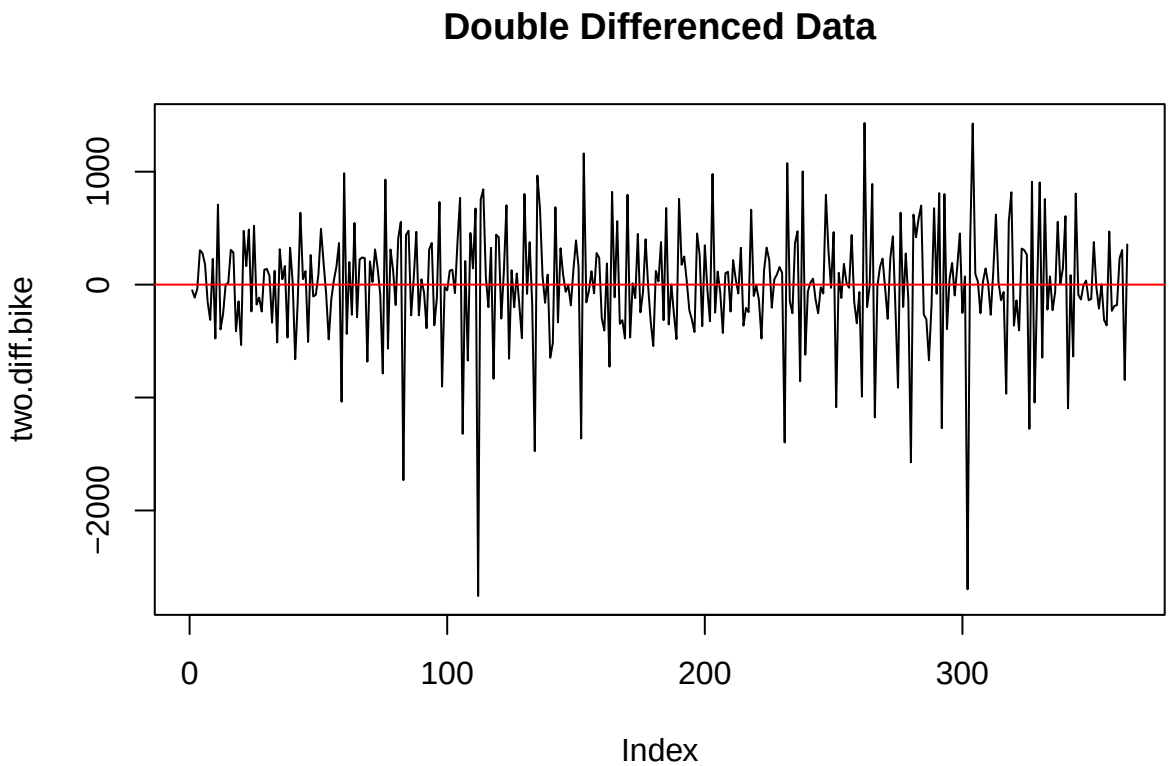

```r
summary(bc.bikes)
```

```
##   Length Class  Mode
## x 100    -none- numeric
## y 100    -none- numeric
```

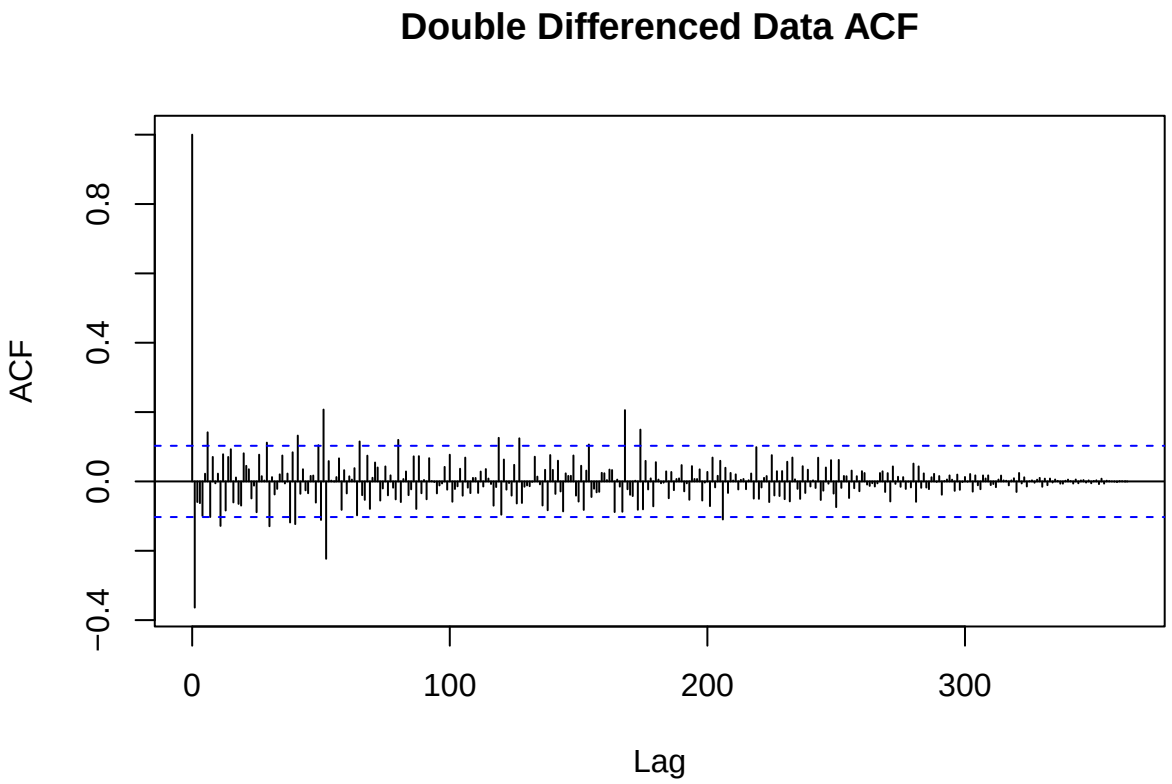
```r
lambda <- bc.bikes$x[which.max(bc.bikes$y)]
# need not boxcox
bc.bikes <- (bike.data$bikes^lambda - 1) / lambda

# plot(diff(diff(Bike, lag=365)))
# acf(diff(Bike, lag=365))
# pacf(diff(Bike, lag=365))

# ARMA(2,1) 365

# seasonal difference + trend difference
plot(bc.bikes, type = 'l')
```



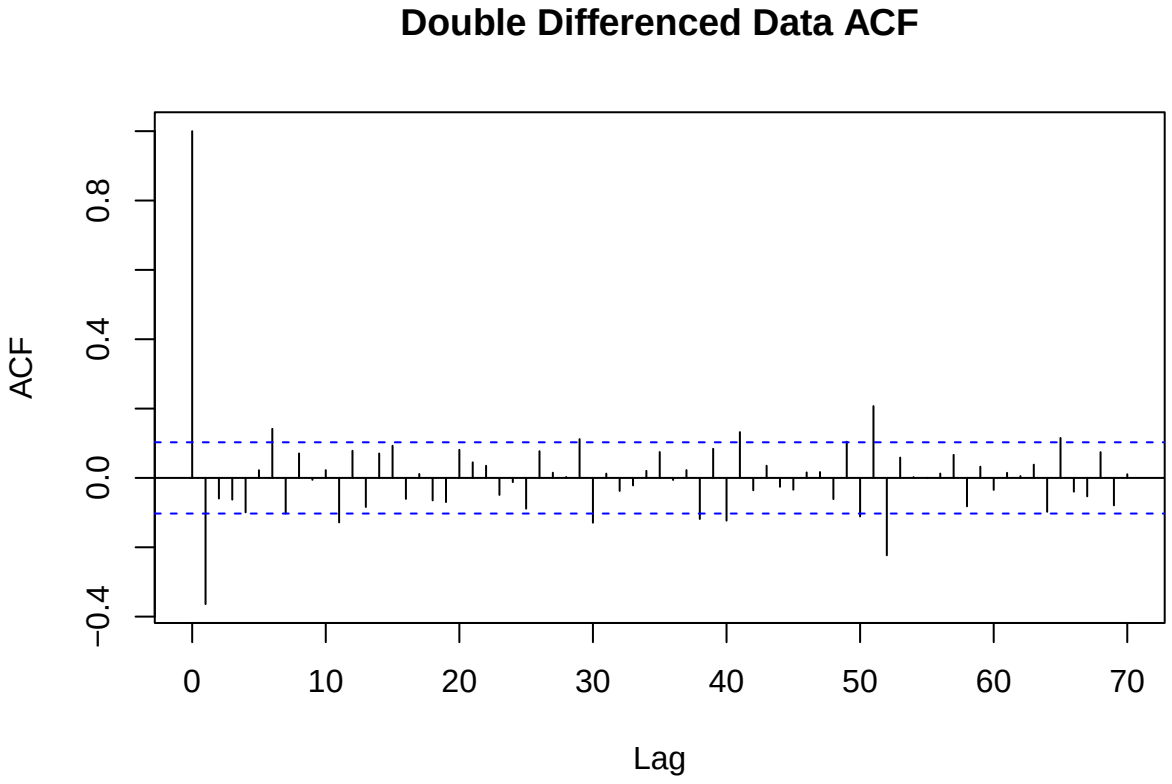

```
two.diff.bike <- diff(diff(bc.bikes, lag = 365))
plot(two.diff.bike, type = 'l', main = 'Double Differenced Data')
abline(h = 0, col='red')
```



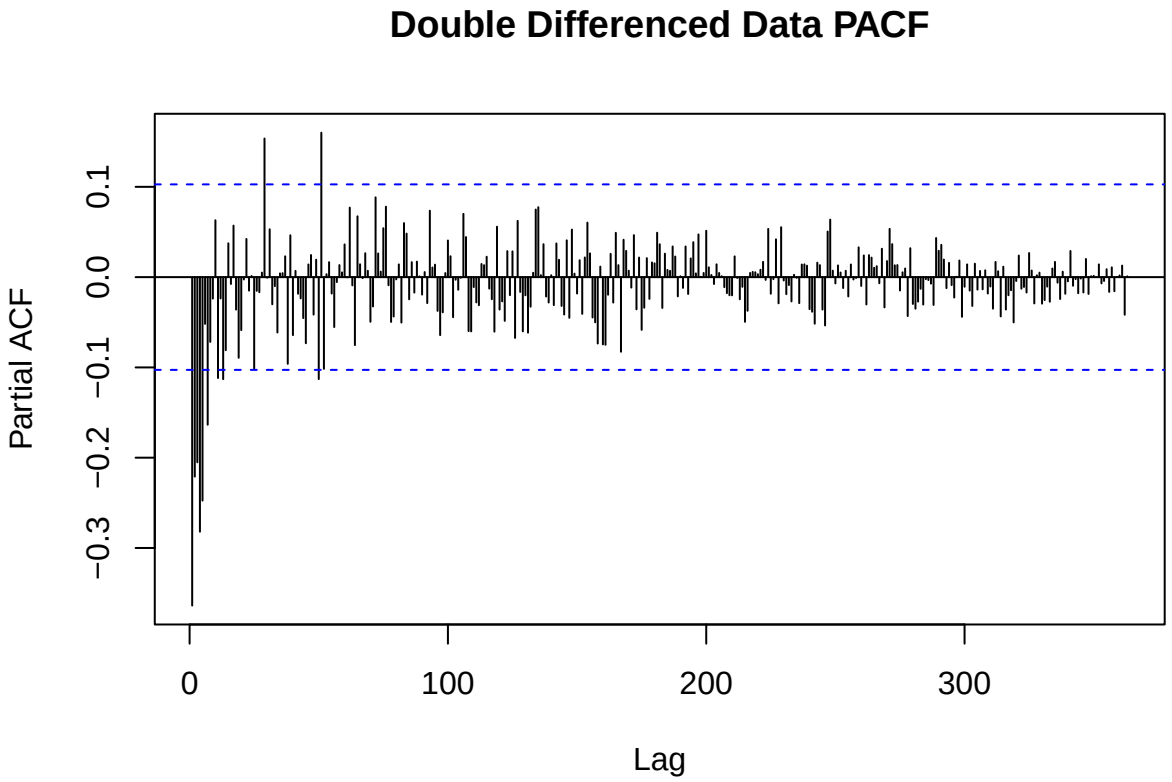

```
acf(two.diff.bike, lag.max = 365, main = 'Double Differenced Data ACF')
```



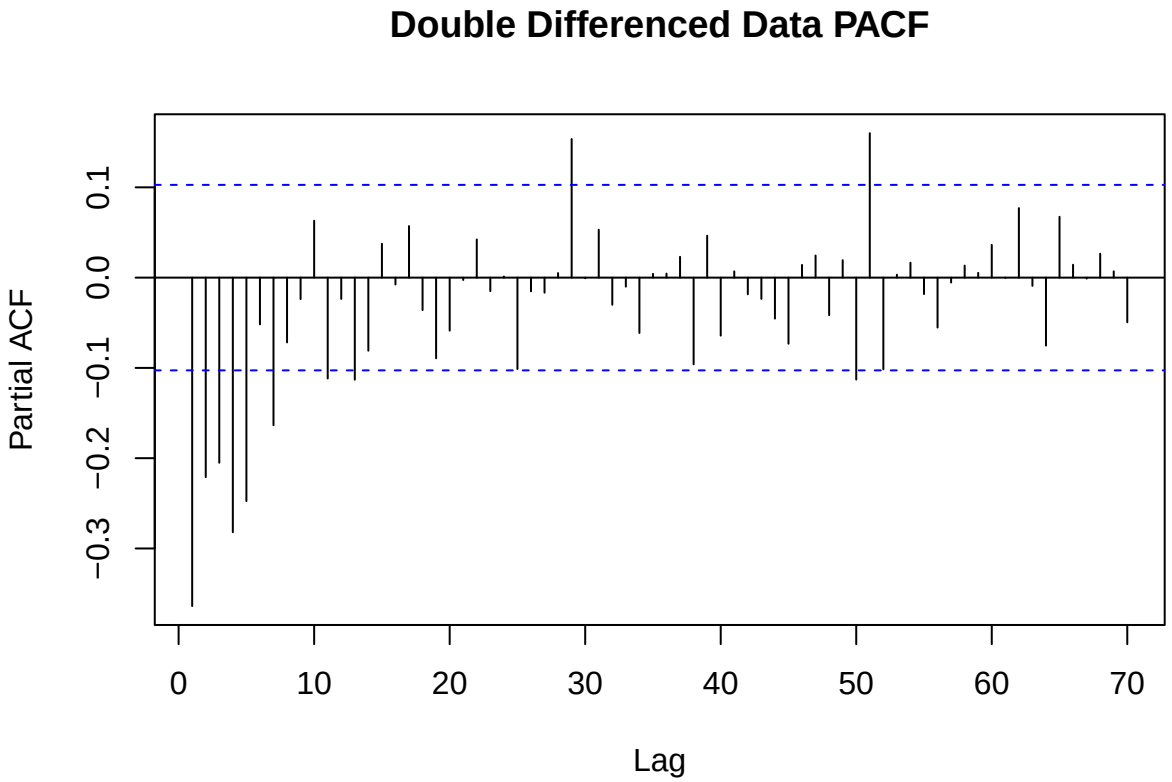

```
acf(two.diff.bike, lag.max = 70, main = 'Double Differenced Data ACF')
```




```
pacf(two.diff.bike, lag.max = 365, main = 'Double Differenced Data PACF')
```

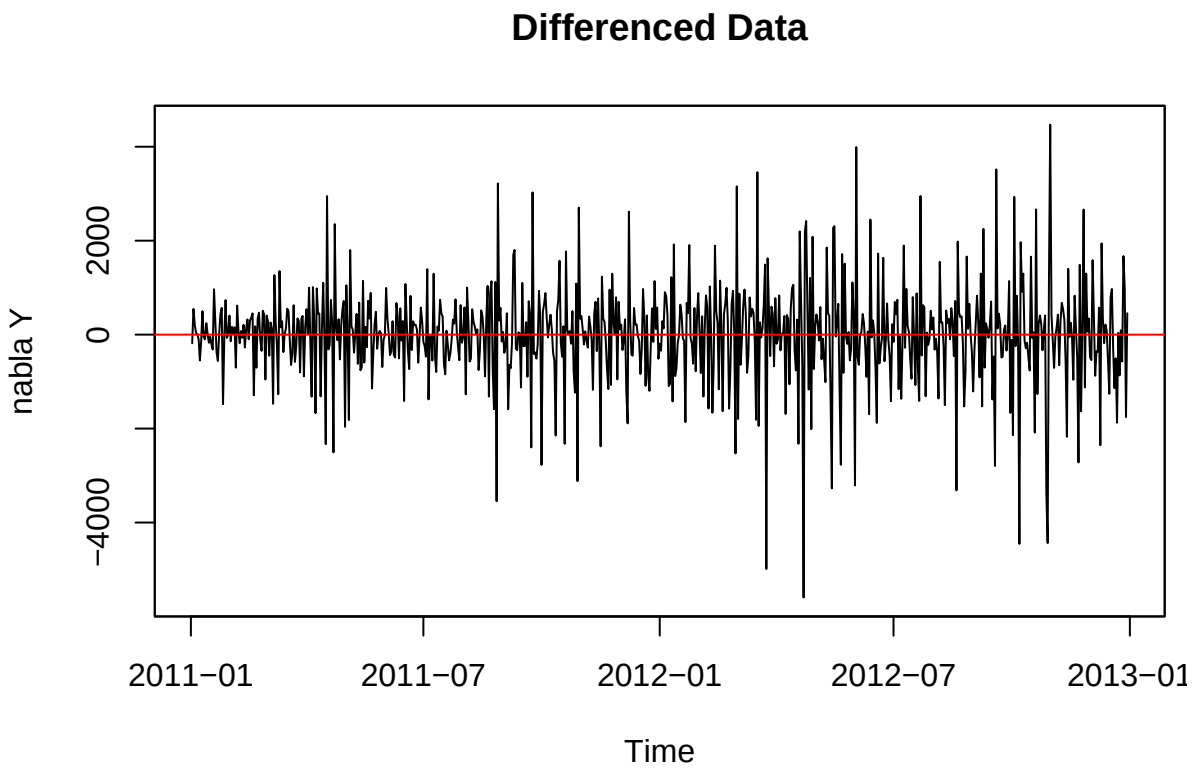



```
pacf(two.diff.bike, lag.max = 70, main = 'Double Differenced Data PACF')
```

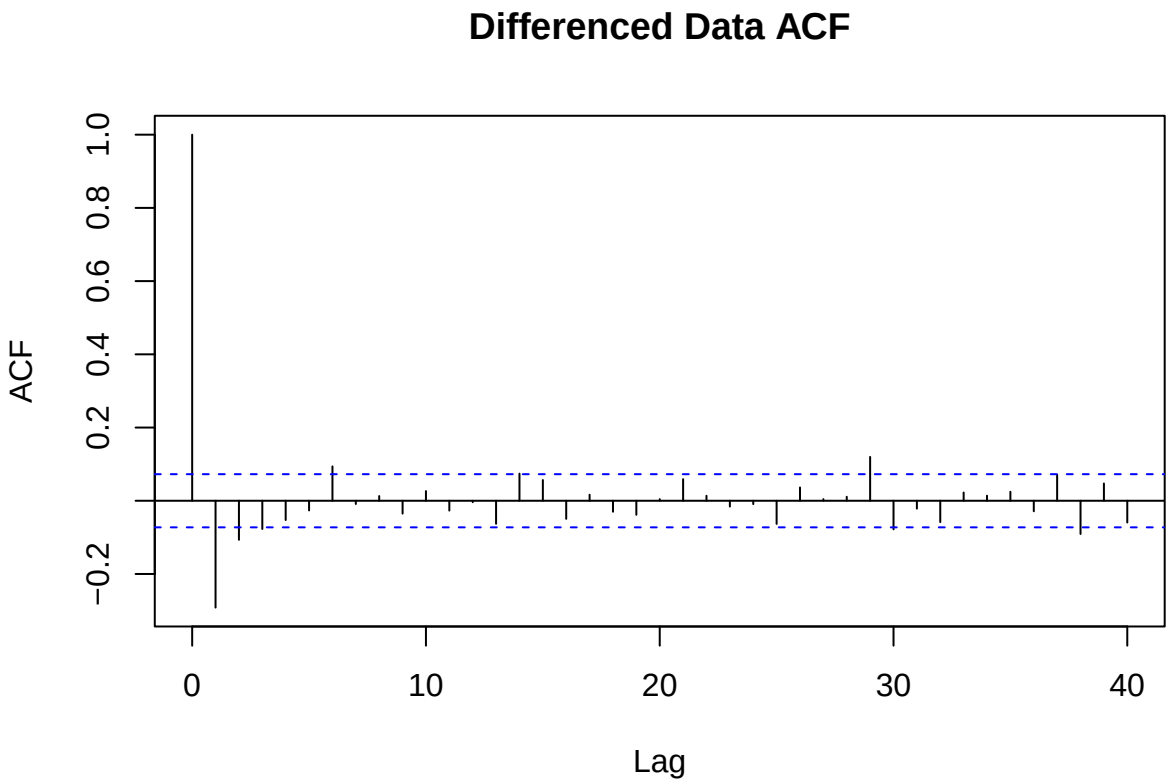



```
# one-time difference
diff.bike <- diff(Bike)
plot(diff.bike, main='Differenced Data', xlab='Time', ylab='nabla Y')
```

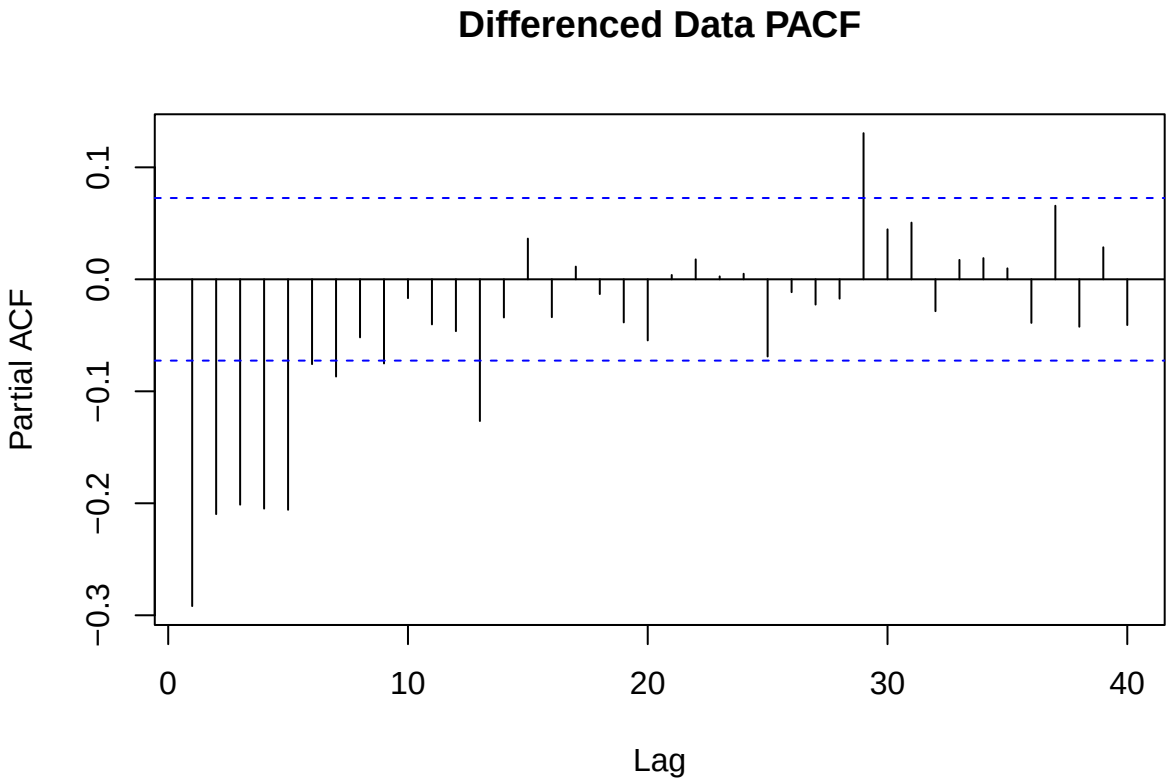
```r
abline(h=0, col='red')
```



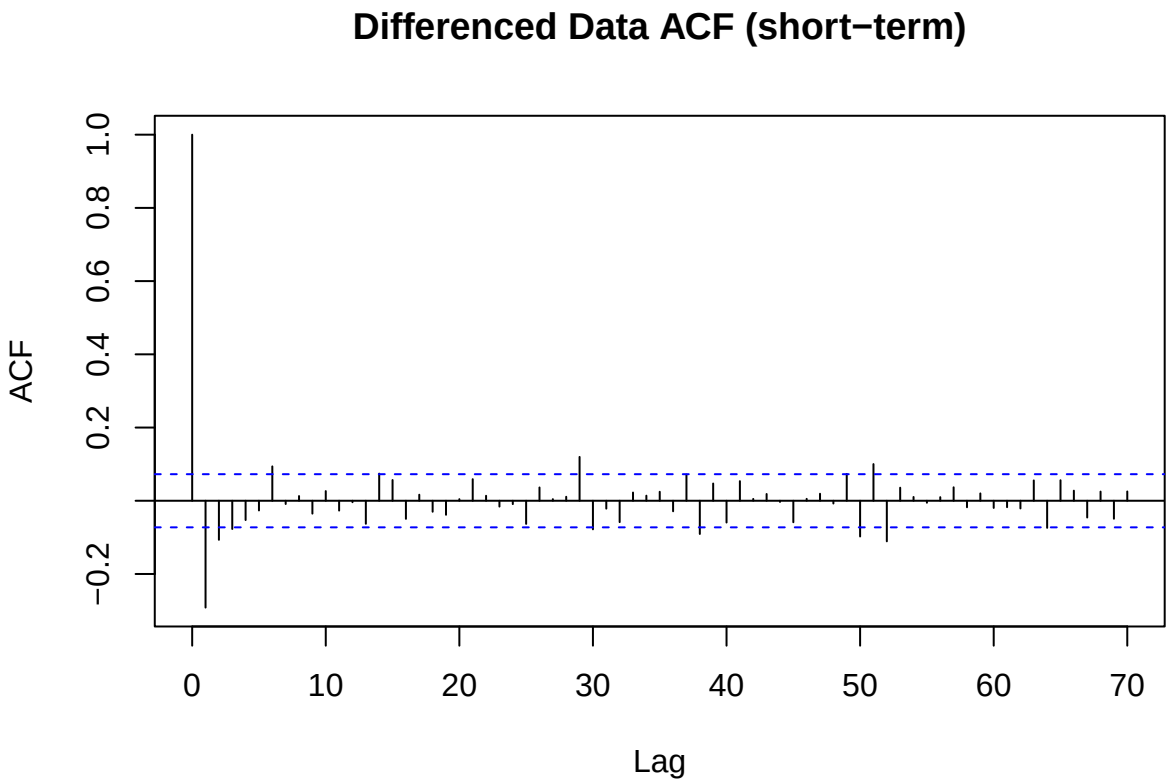

```r
#par(mfrow = c(1, 2))

# short-term acf/pacf
acf(diff.bike, main = 'Differenced Data ACF', lag.max = 40)
```



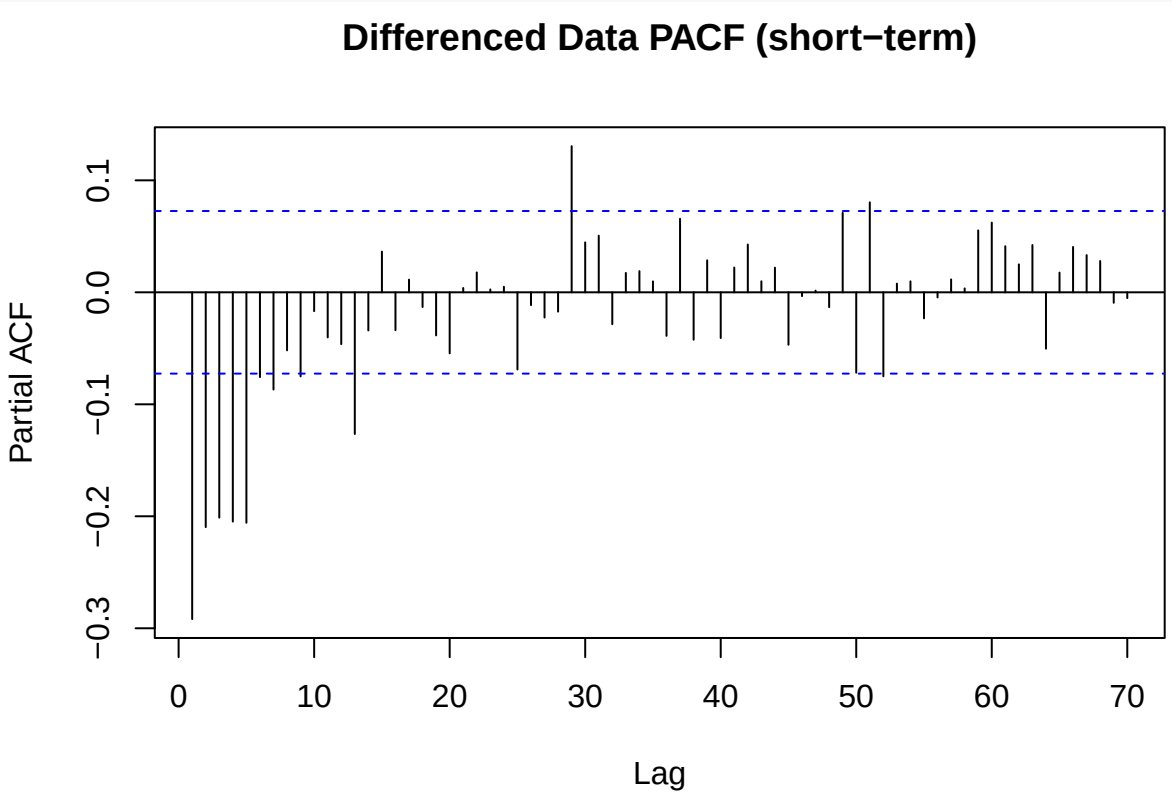

```
acf(diff.bike, type = 'partial', main = 'Differenced Data PACF', lag.max = 40)
```



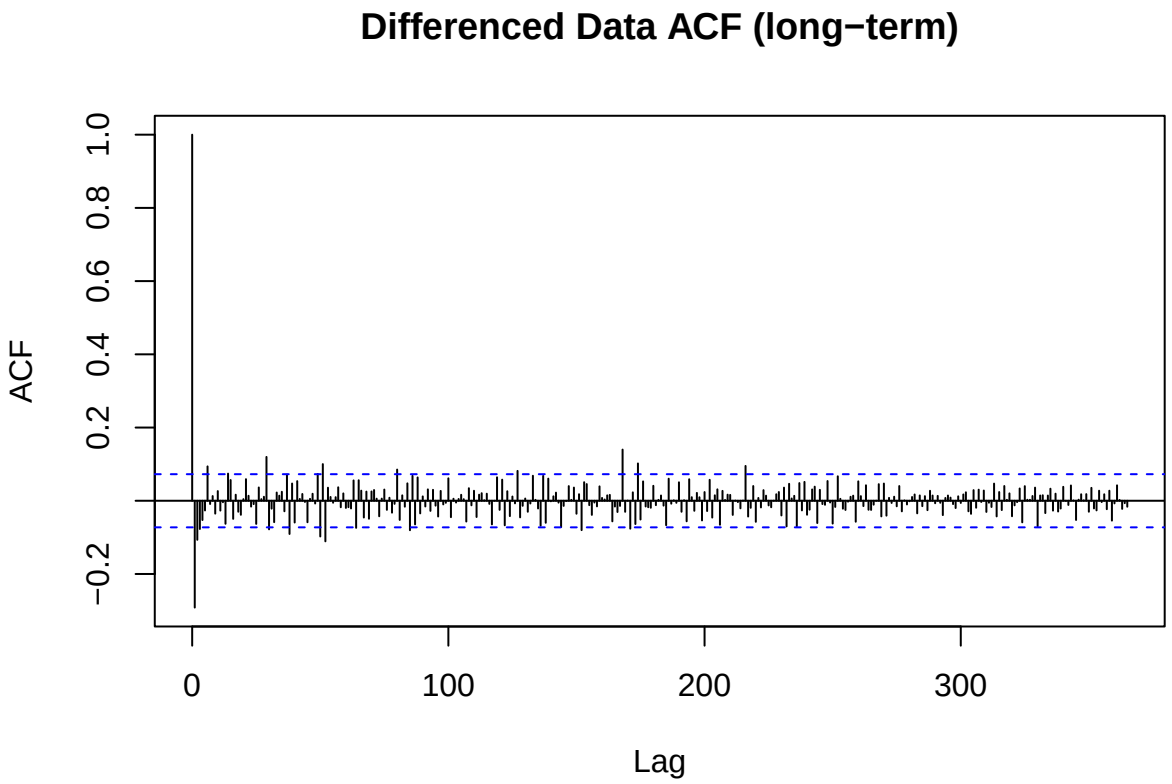

```
acf(diff.bike, main = 'Differenced Data ACF (short-term)', lag.max = 70)
```



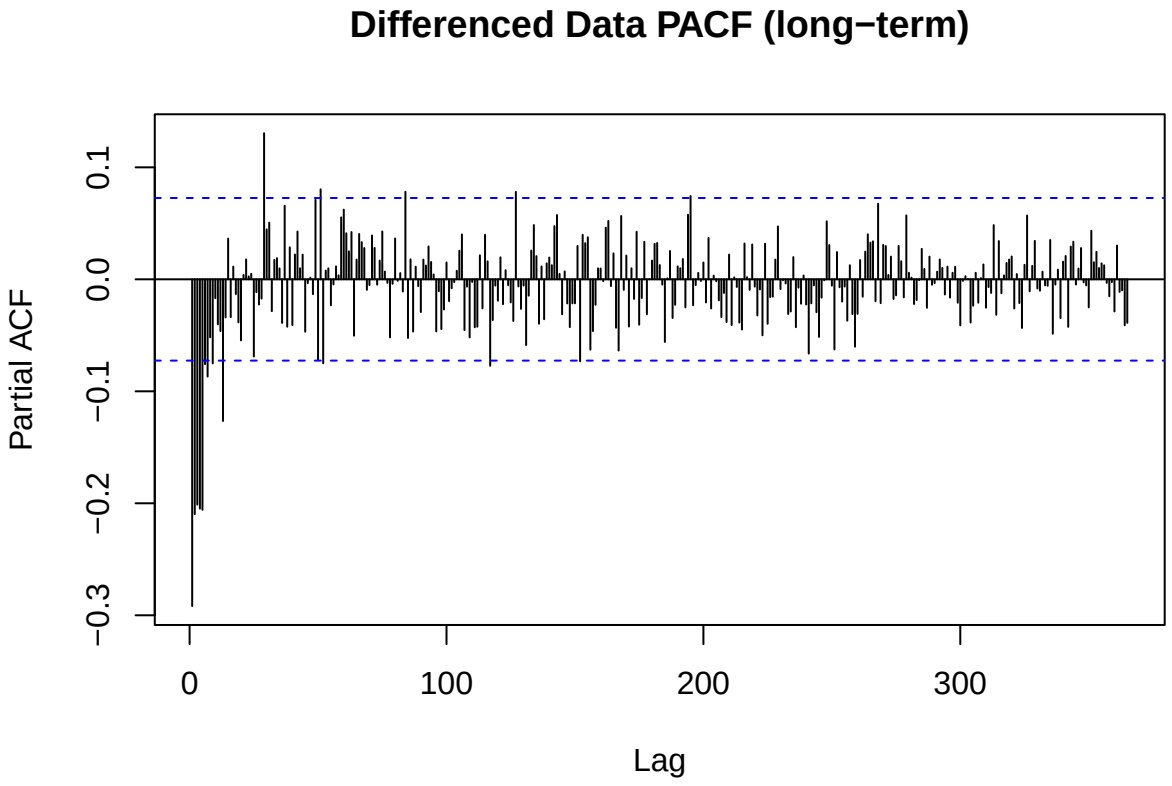

```
acf(diff.bike, type = 'partial', main = 'Differenced Data PACF (short-term)', lag.max = 70)
```




```
# long-term acf/pacf
acf(diff.bike, main = 'Differenced Data ACF (long-term)', lag.max = 365)
```




```
acf(diff.bike, type = 'partial', main = 'Differenced Data PACF (long-term)', lag.max = 365)
```




```
#par(mfrow = c(1, 1))
#pacf(diff.bike, main = 'Differenced Data PACF', xaxt='n')
```

```r
# looks like a MA(1) process

remove.outliers <- function(data) {
  q1 <- quantile(data, 0.25)
  q3 <- quantile(data, 0.75)
  iqr <- abs(q1 - q3)

  upper <- q3 + 1.5 * iqr
  lower <- q1 - 1.5 * iqr

  return(data[data <= upper & data >= lower])
  #return(data * (data <= upper & data >= lower))
}
```

```r
# params <- data.frame(p=0, d=0, q=0, P=0, D=0, Q=0)
ARMA.1.1 <- data.frame(name = 'ARMA(1,1)',
                       p=1, d=0, q=1, P=0, D=0, Q=0, S=0)
ARMA.2.2 <- data.frame(name = 'ARMA(2,2)',
                       p=2, d=0, q=2, P=0, D=0, Q=0, S=0)
ARIMA.212. <- data.frame(name = 'ARIMA(2,1,2)',
                         p=2, d=1, q=2, P=0, D=0, Q=0, S=0)
ARIMA.215. <- data.frame(name = 'ARIMA(2,1,5)',
                         p=2, d=1, q=5, P=0, D=0, Q=0, S=0)
ARIMA.315. <- data.frame(name = 'ARIMA(3,1,5)',
                         p=3, d=1, q=5, P=0, D=0, Q=0, S=0)
ARIMA.512. <- data.frame(name = 'ARIMA(5,1,2)',
                         p=5, d=1, q=2, P=0, D=0, Q=0, S=0)
ARIMA.513. <- data.frame(name = 'ARIMA(5,1,3)',
                         p=5, d=1, q=3, P=0, D=0, Q=0, S=0)
SARIMA.212.111.365 <- data.frame(name = 'SARIMA(2,1,2)x(1,1,1)_365',
                                 p=2, d=1, q=2, P=1, D=1, Q=1, S=365)
SARIMA..111.365 <- data.frame(name = 'SARIMA(2,1,2)x(1,1,1)_365',
                              p=0, d=0, q=0, P=1, D=1, Q=1, S=365)
```

```r
# fit a sarima process with data and parameters params
fit.sarima <- function(data, params) {
  model <- sarima(data,
                  p = params$p,
                  d = params$d,
                  q = params$q,
                  P = params$P,
                  D = params$D,
                  Q = params$Q,
                  S = params$S)

  # add fitted
  model$fit$fitted <- data - model$fit$residuals
  return(model)
}

# predict
pred.sarima <- function(data, params, pred.days) {
  pred.model <- sarima.for(data, n.ahead = pred.days, plot = FALSE,
                           p = params$p,
```

```
                            d = params$d,
                            q = params$q,
                            P = params$P,
                            D = params$D,
                            Q = params$Q,
                            S = params$S)
  return(pred.model)
}

# mse of the fit of a SARIMA process
# sarima.mse <- function(data, params) {
#   model <- fit.sarima(data, params)
#   mse <- mean(model$residuals^2)
# }

MSE <- function(residuals) {
  return(mean(residuals^2))
}

# calculate apse for the fit of sarima process on original data (non-differenced)
sarima.apse <- function(data, training.size, params) {
  n <- length(data)
  # assuming validation data come immediately after training data
  #data <- data[sample(1:n)]
  train.idx <- 1:training.size
  data.train <- data[train.idx]
  data.val <- data[-train.idx]
  model <- pred.sarima(data.train, params,
                       pred.days = length(data.val))
  result <- mean((model$pred - data.val)^2)
  return(result)
}

# plot fitted and predicted
plot.sarima.fit.pred <- function(name, data, fitted, pred, pred.se) {
  upper <- pred + qnorm(0.975) * pred.se
  lower <- pred - qnorm(0.975) * pred.se
  fit.pred <- c(fitted, pred)
  pred.range <- length(fitted) + 1:length(pred)
  plot(1:length(data), data, type = 'l',
       main = paste0('Fitted and Prediction: ', name),
       xlab = 'Time', ylab = 'Value',
       xlim = c(1, length(fit.pred)),
       ylim = c(min(lower), max(c(upper, data))))
  lines(1:length(fit.pred), fit.pred, col = 'red')
  lines(pred.range, upper, col = 'blue')
  lines(pred.range, lower, col = 'blue')
}

plot.sarima.residuals <- function(name, residuals, fitted) {
  # some code are from tutorial code ResidualDiagnostics-Rcodes.R
  #par(mfrow = c(2, 2))
  # qq plot
```

```
  car::qqPlot(
    residuals,
    pch = 16,
    col = adjustcolor("black", 0.7),
    xlab = "Normal (0,1)",
    ylab = "Residuals",
    main = paste0("Q-Q Plot: ", name)
  )

  acf(as.vector(residuals), main = paste0('Residual ACF: ', name),
      lag.max = 365)

  # residual vs fitted
  plot(as.vector(fitted), as.vector(residuals),
       main = paste0('Residuals vs Fitted: ', name),
       xlab = 'Fitted', ylab = 'Residuals')
  abline(h = 0, col = 'red' )

  # residual vs time
  plot(1:length(residuals), residuals, type = 'l', main = paste0('Residuals vs Time: ', name),
       xlab = 'Index', ylab = 'Residuals')
  abline(h = mean(residuals), col = 'red')
}


# Function for recovering differenced data:
# nabla is single differenced
recover.diff <- function(start, nabla) {
  #return(cumsum(c(start, nabla)))
  return(c(start, start + cumsum(nabla)))
}

# plot original data and recovered fit
plot.fit.pred.recover <- function(data, recovered.fit.pred) {
  #recover.diff(data[1], c(fit, pred))
  plot(dates, data, type = 'l', xlim = plot.range,
       main = 'Recovered Fitted and Prediction',
       xlab = 'Time', ylab = 'Counts')
  lines(pred.dates.full, recovered.fit.pred, col = 'red')
}

plot.ljun <- function(name, residuals, lag.max = 1) {
  tests <- c()
  for(i in 1:lag.max) {
    tests <- c(tests, Box.test(residuals, lag=i, type='Ljung-Box')$p.value)
  }
  plot(tests, main = paste0('Ljung-Box:', name), xlab='lags', ylab='p-value',
       ylim=c(0,0.1))
  abline(h=0.05, col='blue', lty=2)
}
```

entire model analysis:

```r
# analyze fits of sarima processes with provided data
# can be directly used on ARIMA with original data, or could work with
sarima.analysis <- function(data, params, print.summary = TRUE, plot = TRUE, residual.plot = TRUE) {
  name <- params$name
  fit.model <- fit.sarima(data, params)
  fitted <- fit.model$fit$fitted
  residuals <- fit.model$fit$residuals
  apse <- sarima.apse(data, as.integer(4/5 * length(data)), params)
  mse <- MSE(residuals)
  pred.model <- pred.sarima(data, params, 30)
  pred.val <- pred.model$pred
  pred.se <- pred.model$se
  normality <- shapiro.test(residuals)$p.value
  randomness <- randtests::runs.test(residuals)$p.value

  # residual analysis and plots
  # are they white noise?

  result <- data.frame(name = name, Pred.Err = pred.se[10],
                       AICc = fit.model$ICs['AICc'],
                       BIC = fit.model$ICs['BIC'],
                       apse = apse,
                       mse = mse,
                       normality = normality,
                       randomness = randomness)

  if(print.summary) {
    print(summary(fit.model))
    print(result)
  }

  if(plot) {
    plot.sarima.fit.pred(name, data, fitted, pred.val, pred.se)
  }

  if(residual.plot) {
    plot.sarima.residuals(name, residuals, fitted)
    plot.ljun(name, residuals, 40)
  }

  return(result)
}

boxcox.recover <- function(data, lambda) {
  if(lambda == 0) {
    return(exp(data))
  } else {
    return((data*lambda + 1)^(1/lambda))
  }
}

# sarima analysis with boxcox
sarima.analysis.boxcox <- function(data, params, lambda, print.summary = TRUE, plot = TRUE, residual.pl
```

```r
name <- params$name

data.copy <- data
if(lambda == 0) {
  data <- log(data)
} else {
  data <- (data^lambda - 1) / lambda
}

fit.model <- fit.sarima(data, params)
fitted <- fit.model$fit$fitted
residuals <- fit.model$fit$residuals

# transform back to original scale

#apse <- sarima.apse(data, as.integer(4/5 * length(data)), params)

# assuming validation data come immediately after training data
# predict on the final 1/5 data
training.size <- as.integer(4/5 * length(data))
train.idx <- 1:training.size
data.train <- data[train.idx]
data.val <- data.copy[-train.idx]
apse.model <- pred.sarima(data.train, params,
                      pred.days = length(data.val))
apse <- mean((boxcox.recover(apse.model$pred, lambda) - data.val)^2)

res.original.scale <- data.copy - boxcox.recover(fitted, lambda)
#mse <- MSE(residuals)
mse <- MSE(res.original.scale)
pred.model <- pred.sarima(data, params, 30)
pred.val <- pred.model$pred
pred.se <- pred.model$se
normality <- shapiro.test(residuals)$p.value
randomness <- randtests::runs.test(residuals)$p.value
#normality <- shapiro.test(res.original.scale)$p.value
#randomness <- randtests::runs.test(res.original.scale)$p.value

# residual analysis and plots
# are they white noise?

result <- data.frame(name = name, lambda = lambda, Pred.Err = pred.se[10],
                      AICc = fit.model$ICs['AICc'],
                      BIC = fit.model$ICs['BIC'],
                      apse = apse,
                      mse = mse,
                      normality = normality,
                      randomness = randomness)

if(print.summary) {
  print(summary(fit.model))
  print(result)
}
```

```
  if(plot) {
    plot.sarima.fit.pred(name, data, fitted, pred.val, pred.se)
  }

  if(residual.plot) {
    plot.sarima.residuals(name, residuals, fitted)
    plot.ljun(name, residuals, 40)
  }

  return(result)
}
```

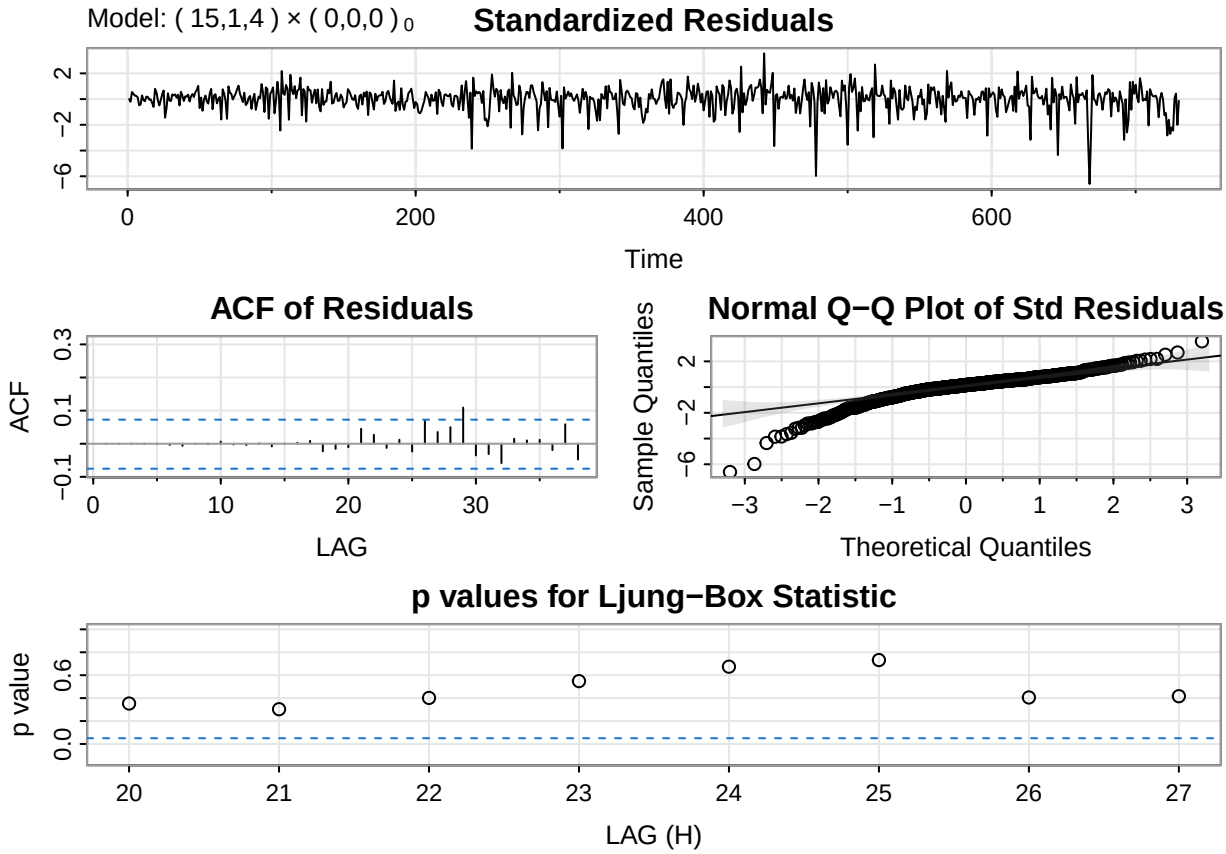
```
box.cox.training.data <- MASS::boxcox(bike.data$bikes ~ 1, plot=FALSE)
bike.lambda <- box.cox.training.data$x[which.max(box.cox.training.data$y)]

sarima.analysis.boxcox(bike.data$bikes,
                       data.frame(name = 'boxcox', p=15, d=1, q=4, P=0, D=0, Q=0, S=0),
                       bike.lambda)
```

```
## initial  value 6.153387
## iter   2 value 6.022579
## iter   3 value 6.009790
## iter   4 value 6.001592
## iter   5 value 6.001271
## iter   6 value 6.000912
## iter   7 value 6.000853
## iter   8 value 6.000813
## iter   9 value 6.000785
## iter  10 value 6.000772
## iter  11 value 6.000749
## iter  12 value 6.000686
## iter  13 value 6.000613
## iter  14 value 6.000491
## iter  15 value 6.000406
## iter  16 value 6.000377
## iter  17 value 6.000372
## iter  18 value 6.000366
## iter  19 value 6.000343
## iter  20 value 6.000321
## iter  21 value 6.000308
## iter  22 value 6.000306
## iter  23 value 6.000305
## iter  24 value 6.000304
## iter  25 value 6.000302
## iter  26 value 6.000296
## iter  27 value 6.000282
## iter  28 value 6.000260
## iter  29 value 6.000240
## iter  30 value 6.000235
## iter  31 value 6.000234
## iter  32 value 6.000234
## iter  32 value 6.000234
## iter  32 value 6.000234
## final  value 6.000234
```

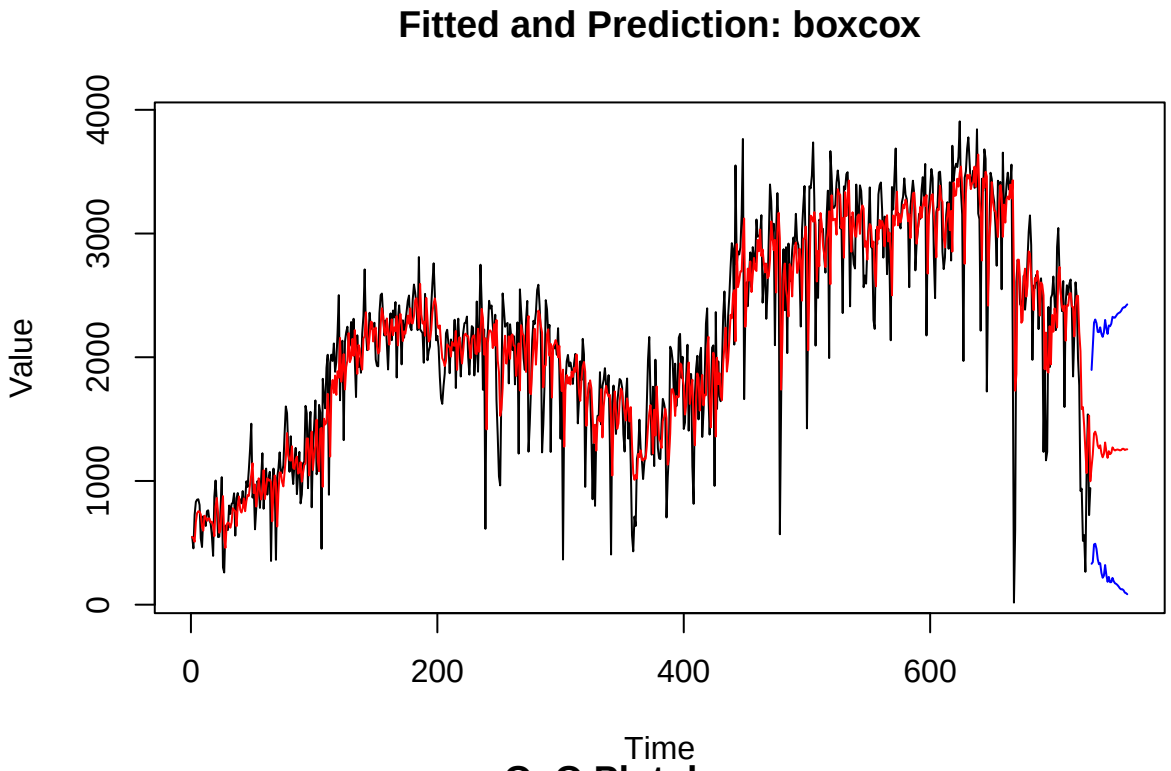
```
## converged
## initial  value 5.991530
## iter   2 value 5.991527
## iter   3 value 5.991525
## iter   4 value 5.991525
## iter   5 value 5.991525
## iter   5 value 5.991525
## iter   5 value 5.991525
## final  value 5.991525
## converged
```

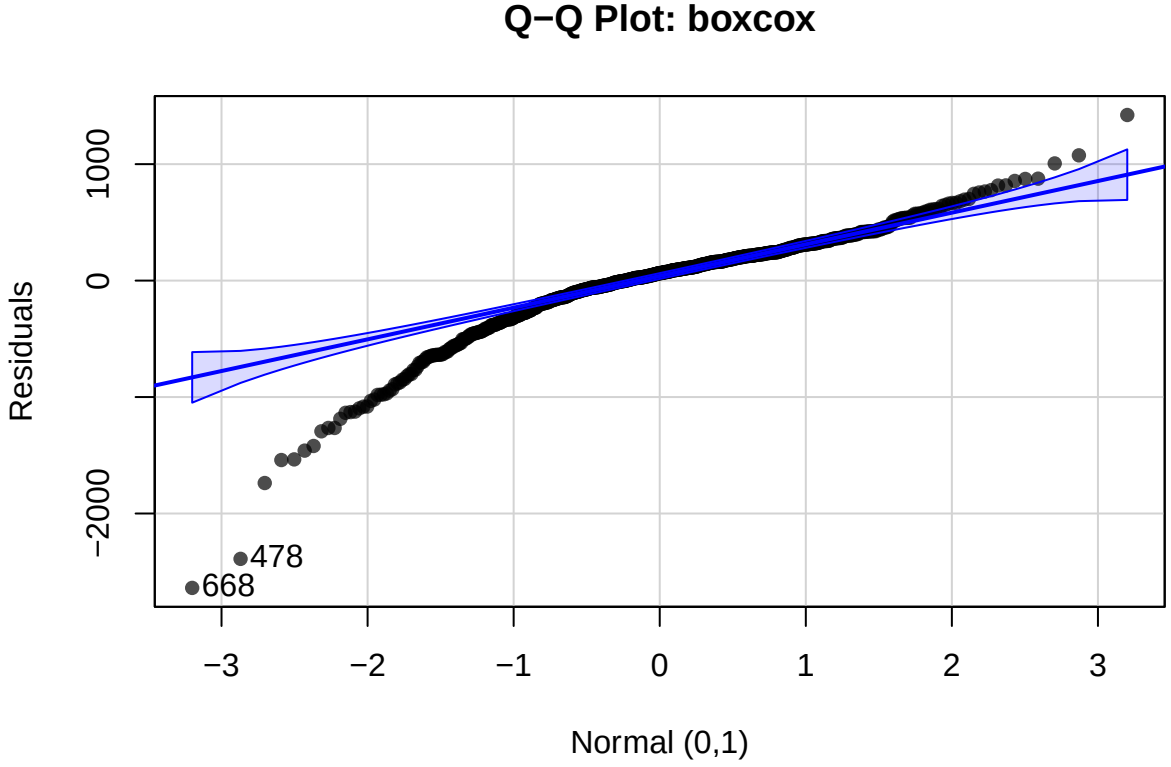
```
## Warning in sqrt(diag(fitit$var.coef)): NaNs produced
## Warning in sqrt(diag(fitit$var.coef)): NaNs produced
```

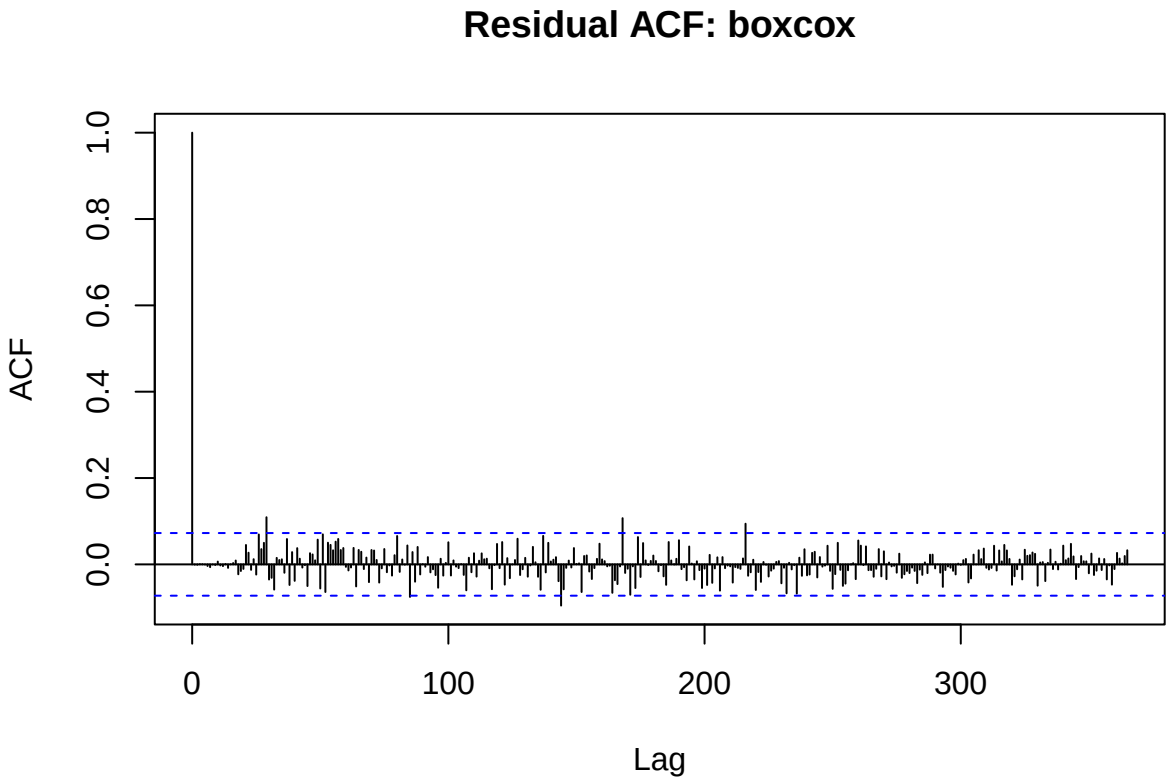
```
## <><><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE t.value p.value
## ar1       -0.4521    NaN     NaN     NaN
## ar2       -0.2741 0.4778 -0.5736  0.5664
## ar3        0.0474    NaN     NaN     NaN
## ar4       -0.1209 0.1162 -1.0406  0.2984
## ar5       -0.0901 0.1571 -0.5733  0.5666
## ar6        0.0295 0.1422  0.2077  0.8355
## ar7        0.0076 0.1021  0.0745  0.9406
## ar8       -0.0031 0.0813 -0.0382  0.9695
## ar9       -0.0670 0.0772 -0.8675  0.3859
## ar10      -0.0153 0.0792 -0.1933  0.8468
## ar11      -0.0507 0.0747 -0.6795  0.4970
## ar12      -0.0487 0.0626 -0.7782  0.4367
## ar13      -0.0851 0.0739 -1.1526  0.2495
## ar14       0.0255 0.0753  0.3390  0.7347
## ar15       0.0641 0.0509  1.2585  0.2086
## ma1       -0.0664 0.0116 -5.7236  0.0000
## ma2       -0.1642 0.0211 -7.7893  0.0000
## ma3       -0.3869    NaN     NaN     NaN
## ma4       -0.0010    NaN     NaN     NaN
## constant   0.7890 2.8071  0.2811  0.7787
##
## sigma^2 estimated as 159756.1 on 709 degrees of freedom
##
## AIC = 14.87854  AICc = 14.88017  BIC = 15.01081
##
```

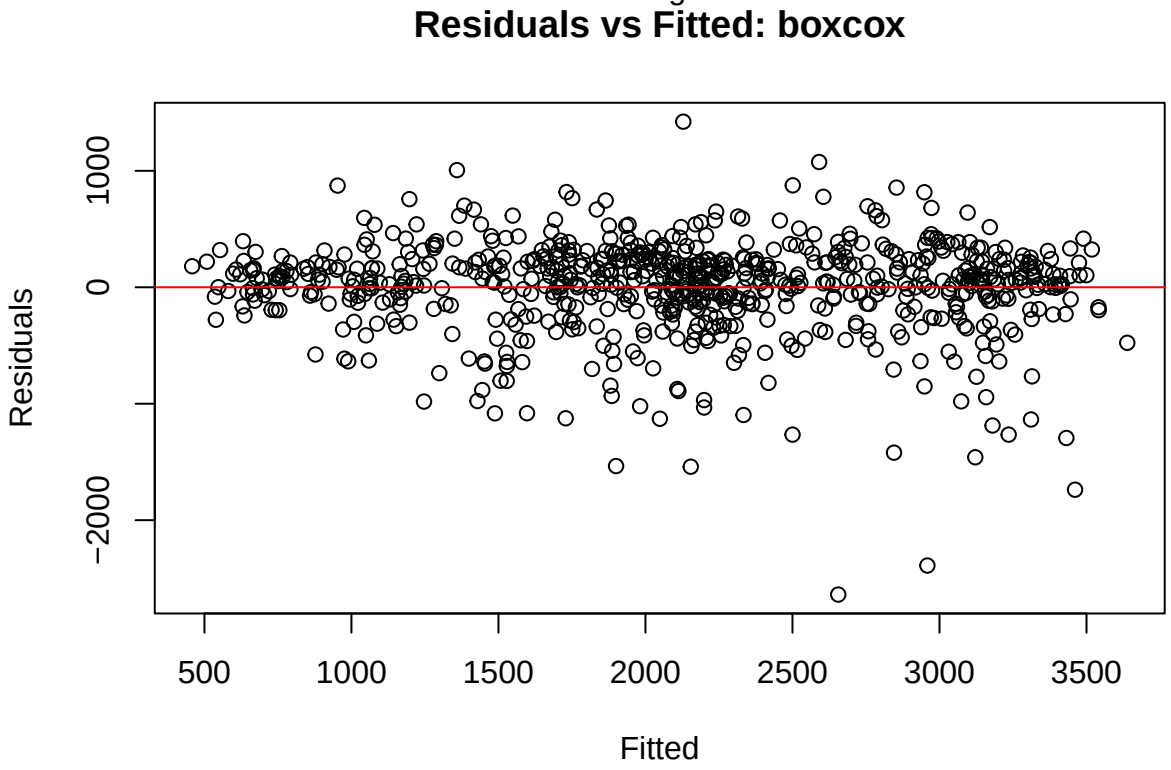


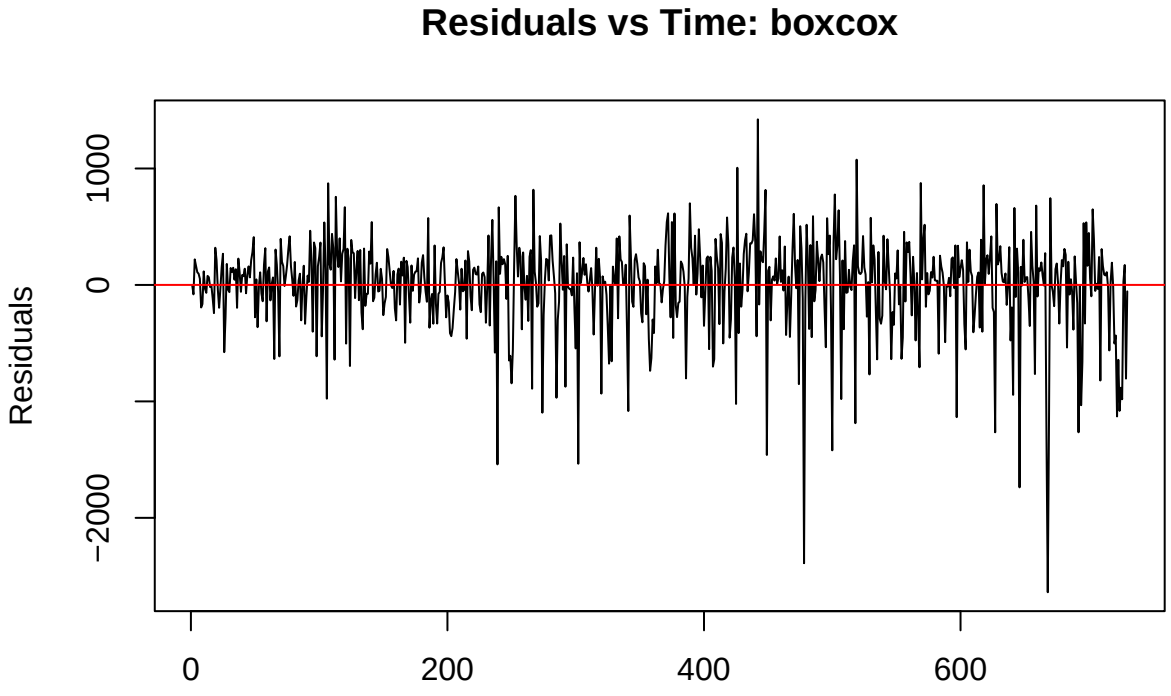

```
##                   Length Class  Mode
## fit                15     Arima  list
## sigma2              1     -none- numeric
## degrees_of_freedom  1     -none- numeric
## t.table            80     -none- numeric
## ICs                 3     -none- numeric
##        name lambda Pred.Err     AICc      BIC    apse    mse    normality
## AICc boxcox    0.9 496.7541 14.88017 15.01081 8054157 830202 2.988681e-21
##      randomness
## AICc  0.6040962
```

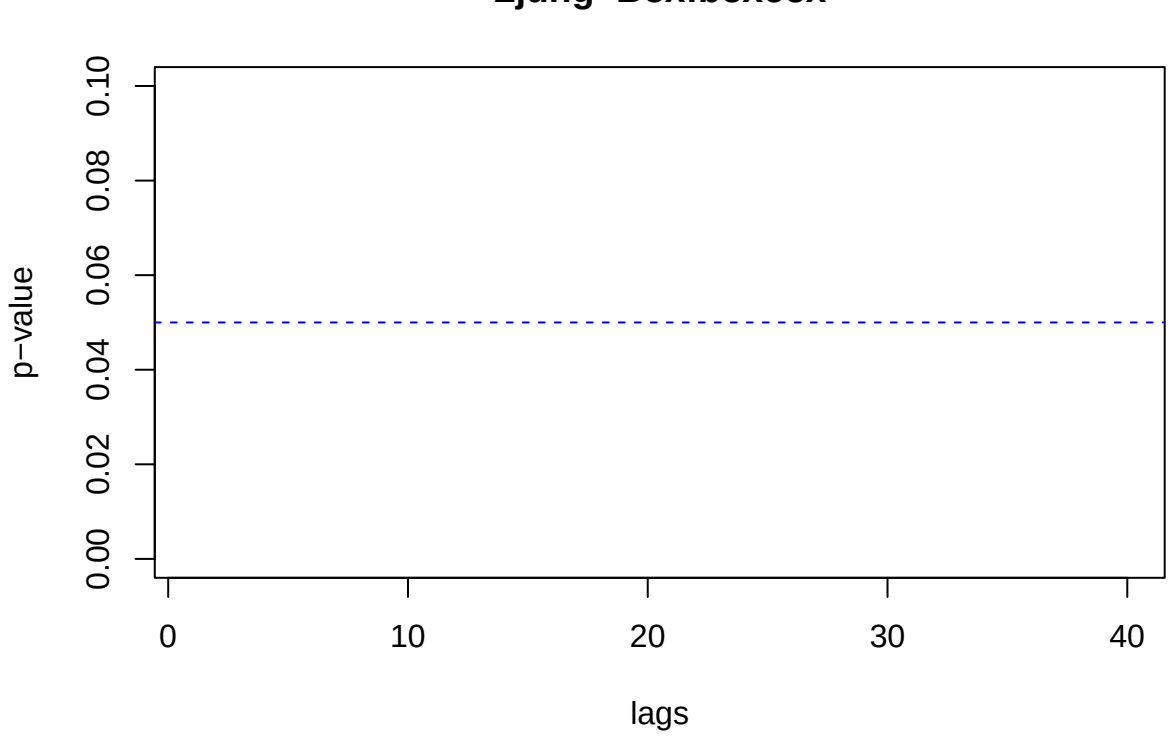

Fitted and Prediction: boxcox
Value
0
1000
2000
3000
4000
0
200
400
600
Time


Q−Q Plot: boxcox
Residuals
1000
0
−2000
478
668
−3
−2
−1
0
1
2
3
Normal (0,1)


Residual ACF: boxcox
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
Lag


Residuals vs Fitted: boxcox
Residuals
1000
0
−2000
500
1000
1500
2000
2500
3000
3500
Fitted



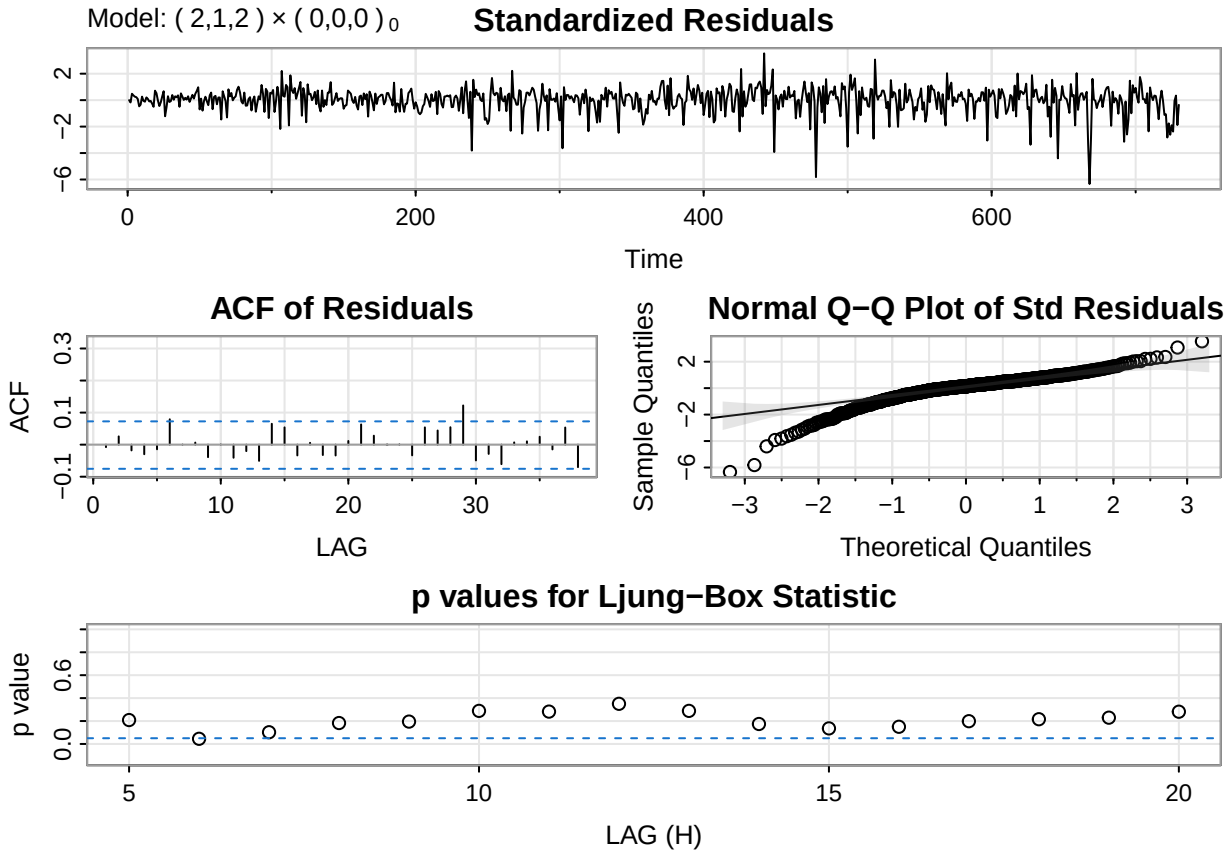



```
##        name lambda Pred.Err     AICc      BIC    apse    mse    normality
## AICc boxcox    0.9 496.7541 14.88017 15.01081 8054157 830202 2.988681e-21
##      randomness
## AICc  0.6040962
```

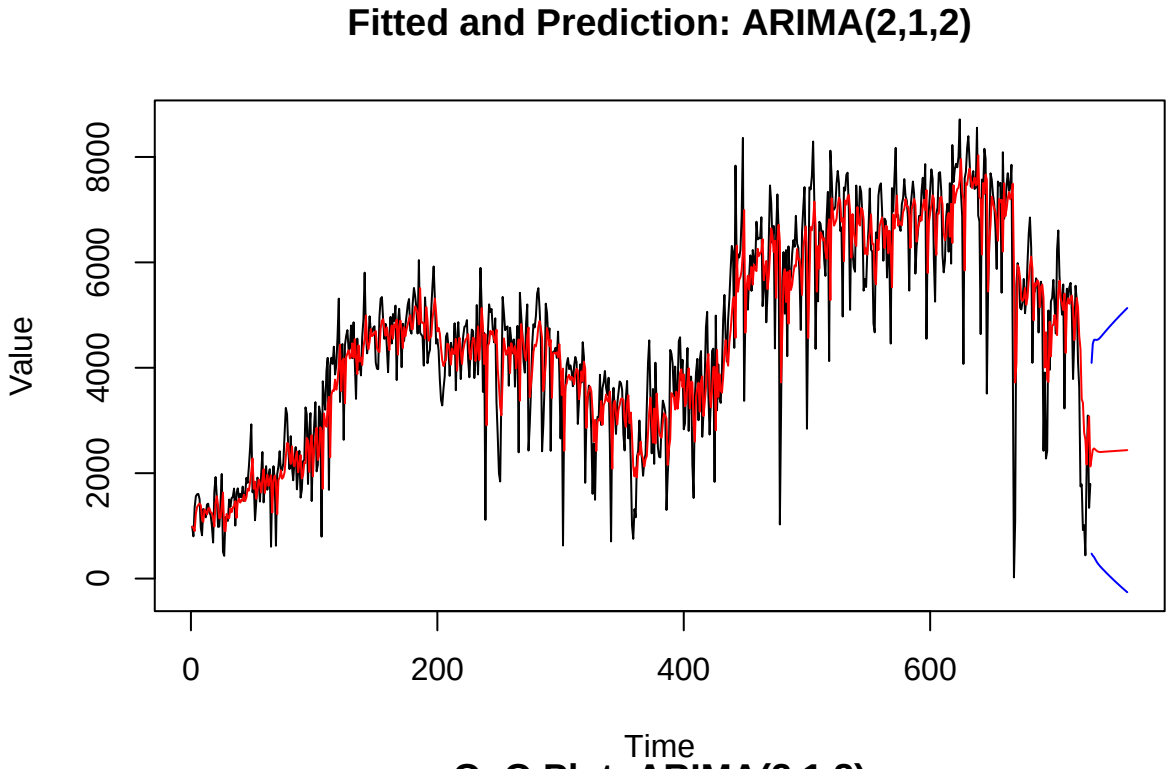
```r
sarima.performances <- sarima.analysis(bike.data$bikes, ARIMA.212.)
```

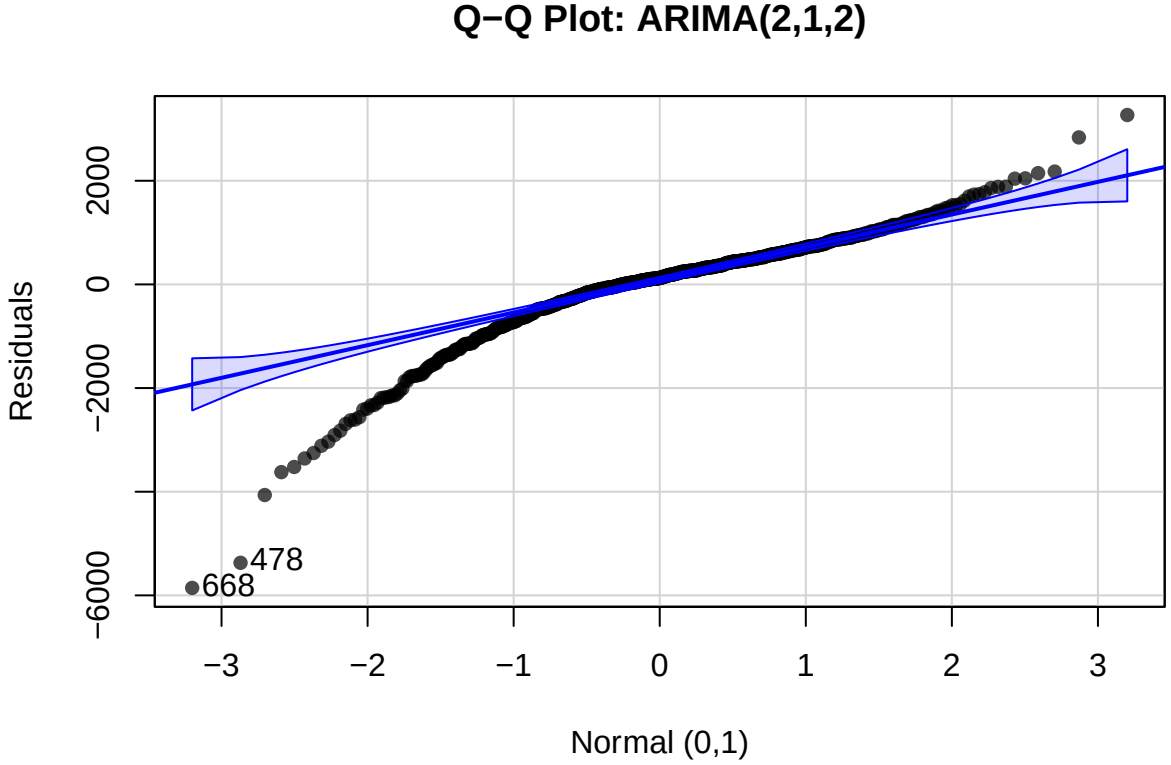
```
## initial  value 6.971617
## iter   2 value 6.886261
## iter   3 value 6.863559
## iter   4 value 6.842116
## iter   5 value 6.837298
## iter   6 value 6.834196
## iter   7 value 6.832685
## iter   8 value 6.832090
## iter   9 value 6.831573
## iter  10 value 6.831312
## iter  11 value 6.831297
## iter  12 value 6.831295
## iter  13 value 6.831295
## iter  14 value 6.831294
## iter  15 value 6.831294
## iter  16 value 6.831293
## iter  17 value 6.831292
## iter  18 value 6.831282
## iter  19 value 6.831260
## iter  20 value 6.831178
## iter  21 value 6.831061
## iter  22 value 6.830914
## iter  23 value 6.830884
## iter  24 value 6.830854
## iter  25 value 6.830834
## iter  26 value 6.830833
## iter  27 value 6.830829
## iter  28 value 6.830819
## iter  29 value 6.830791
## iter  30 value 6.830717
## iter  31 value 6.830646
## iter  32 value 6.830528
## iter  33 value 6.830490
## iter  34 value 6.830453
## iter  35 value 6.830432
## iter  36 value 6.830417
## iter  37 value 6.830416
## iter  38 value 6.830416
## iter  39 value 6.830410
## iter  40 value 6.830402
## iter  41 value 6.830391
## iter  42 value 6.830386
## iter  43 value 6.830385
## iter  43 value 6.830385
## final  value 6.830385
## converged
## initial  value 6.829283
## iter   2 value 6.829275
## iter   3 value 6.829273
## iter   4 value 6.829265
## iter   5 value 6.829257
## iter   6 value 6.829247
```

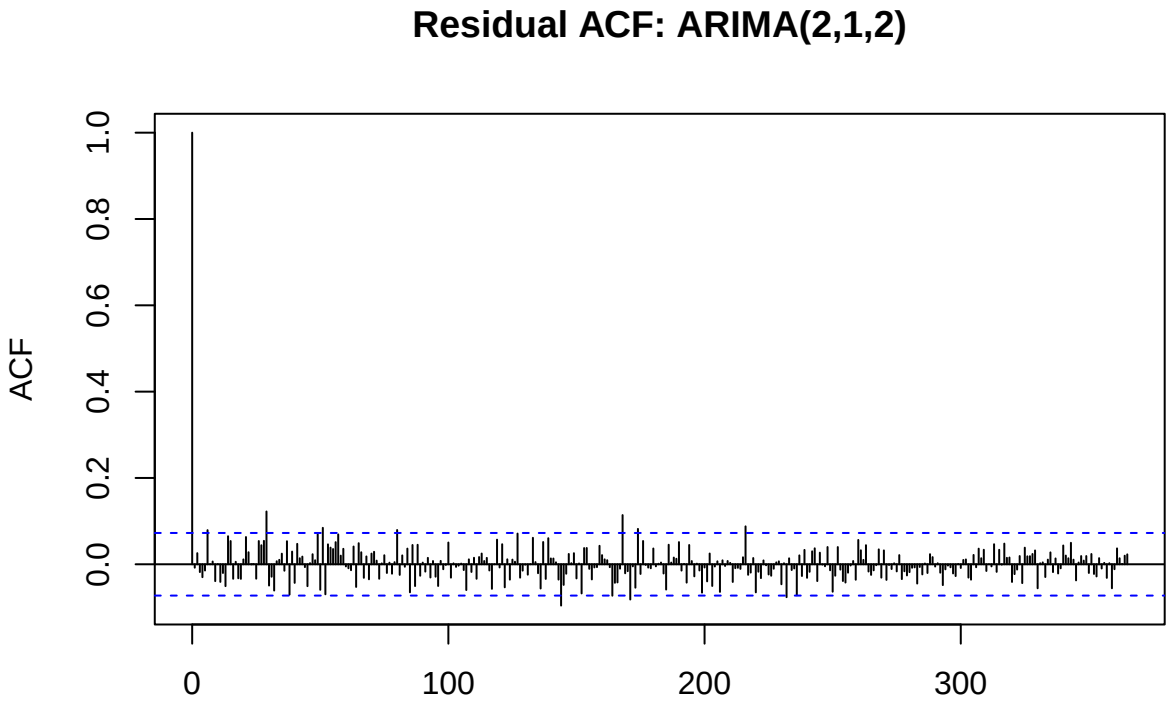
```
## iter   7 value 6.829202
## iter   8 value 6.829126
## iter   9 value 6.829034
## iter  10 value 6.828920
## iter  11 value 6.828822
## iter  12 value 6.828807
## iter  13 value 6.828806
## iter  14 value 6.828805
## iter  15 value 6.828805
## iter  16 value 6.828805
## iter  16 value 6.828805
## iter  16 value 6.828805
## final  value 6.828805
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE t.value p.value
## ar1        0.8981 0.2728  3.2925  0.0010
## ar2       -0.2538 0.0957 -2.6518  0.0082
## ma1       -1.4137 0.2752 -5.1370  0.0000
## ma2        0.4808 0.2425  1.9825  0.0478
## constant   1.6031 6.5125  0.2462  0.8056
##
## sigma^2 estimated as 852811.2 on 724 degrees of freedom
##
## AIC = 16.51195  AICc = 16.51206  BIC = 16.54974
##
```

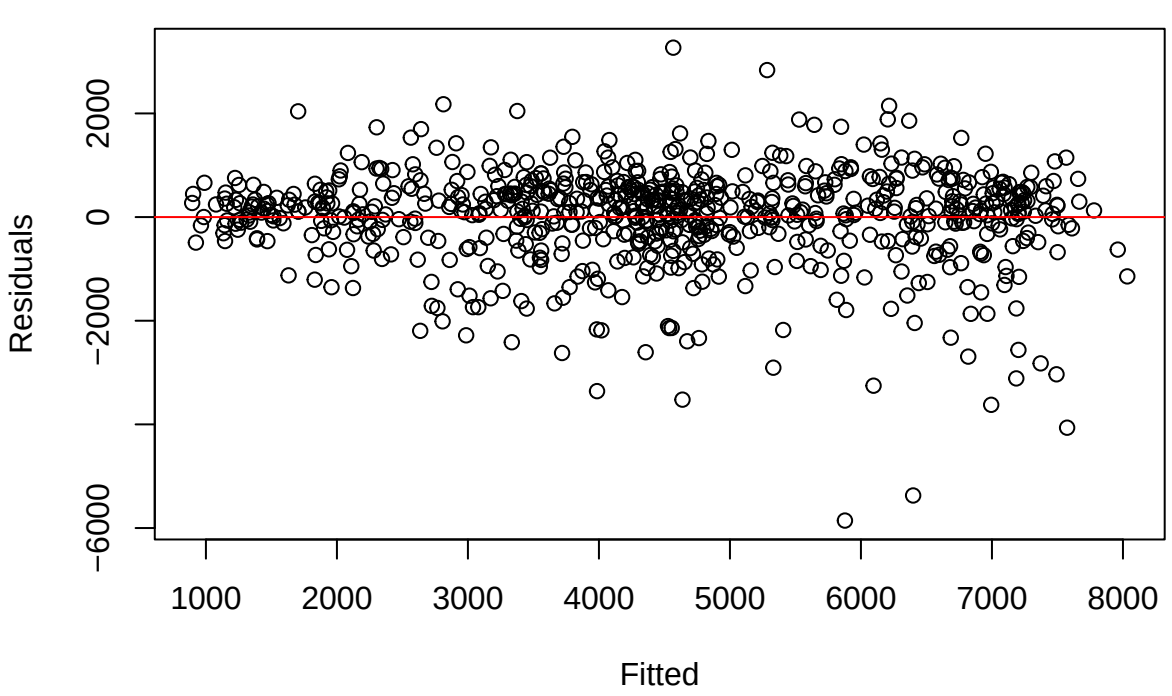


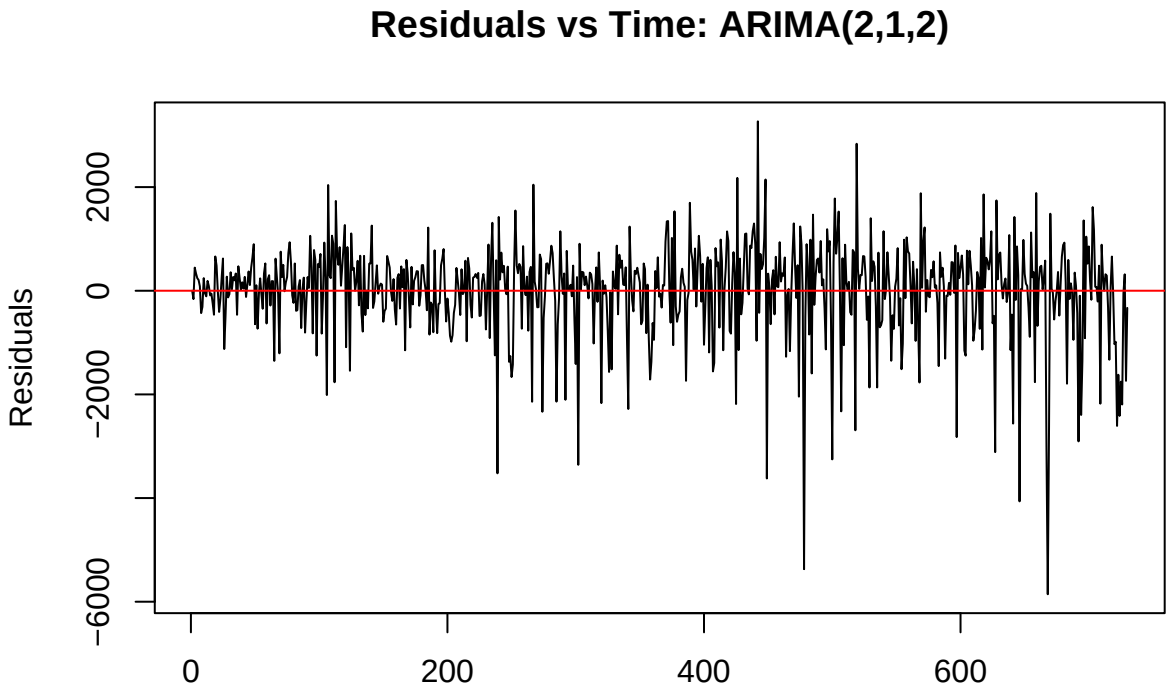

```
##                   Length Class  Mode
## fit                15     Arima  list
## sigma2              1     -none- numeric
## degrees_of_freedom  1     -none- numeric
## t.table            20     -none- numeric
## ICs                 3     -none- numeric
##              name Pred.Err     AICc      BIC    apse      mse    normality
## AICc ARIMA(2,1,2) 1134.172 16.51206 16.54974 7815633 851642.9 5.789628e-21
##      randomness
## AICc  0.5049847
```

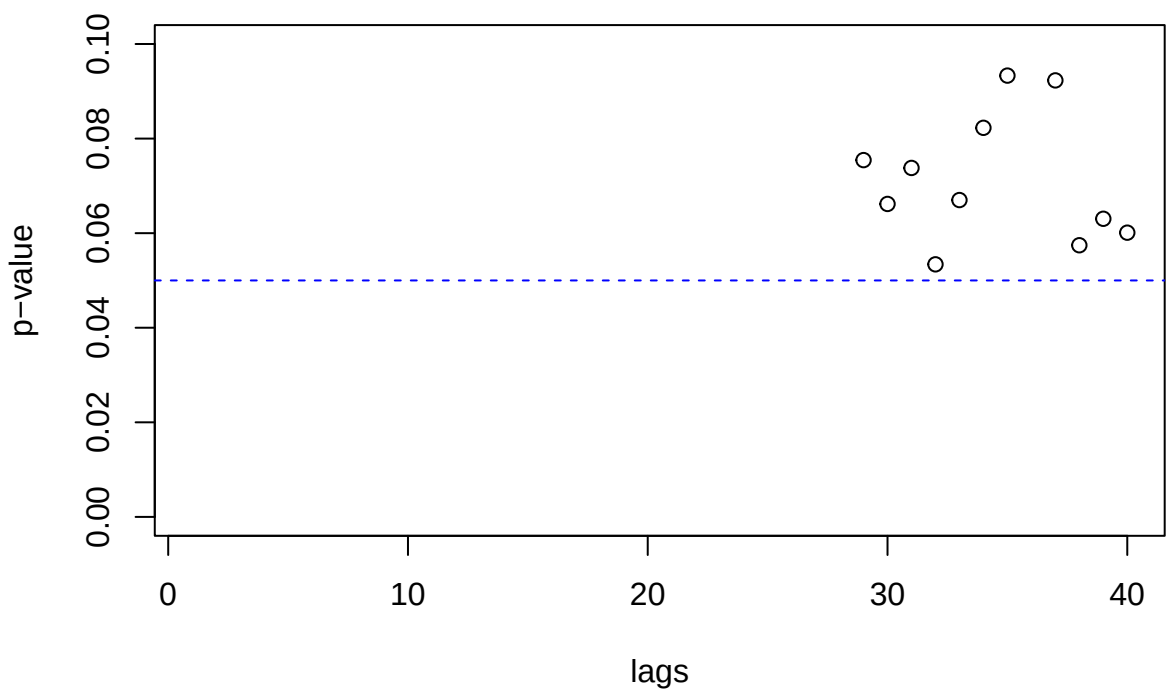

Fitted and Prediction: ARIMA(2,1,2)
Value
0
2000
4000
6000
8000
0
200
400
600
Time


Q−Q Plot: ARIMA(2,1,2)
Residuals
−6000
−2000
0
2000
478
668
−3
−2
−1
0
1
2
3
Normal (0,1)


Residual ACF: ARIMA(2,1,2)
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
Lag


Residuals vs Fitted: ARIMA(2,1,2)
Residuals
2000
0
−2000
−6000
1000
2000
3000
4000
5000
6000
7000
8000
Fitted



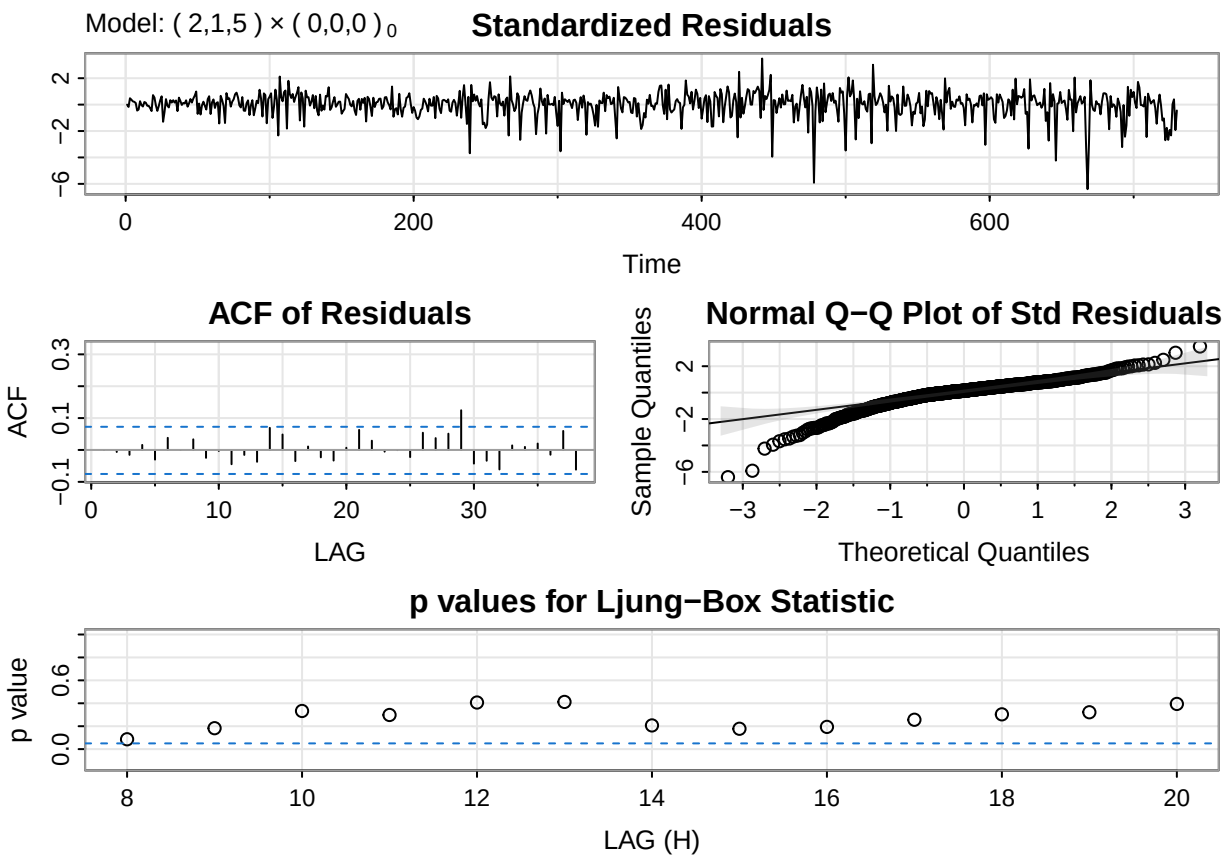



```
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, ARIMA.215.))
```

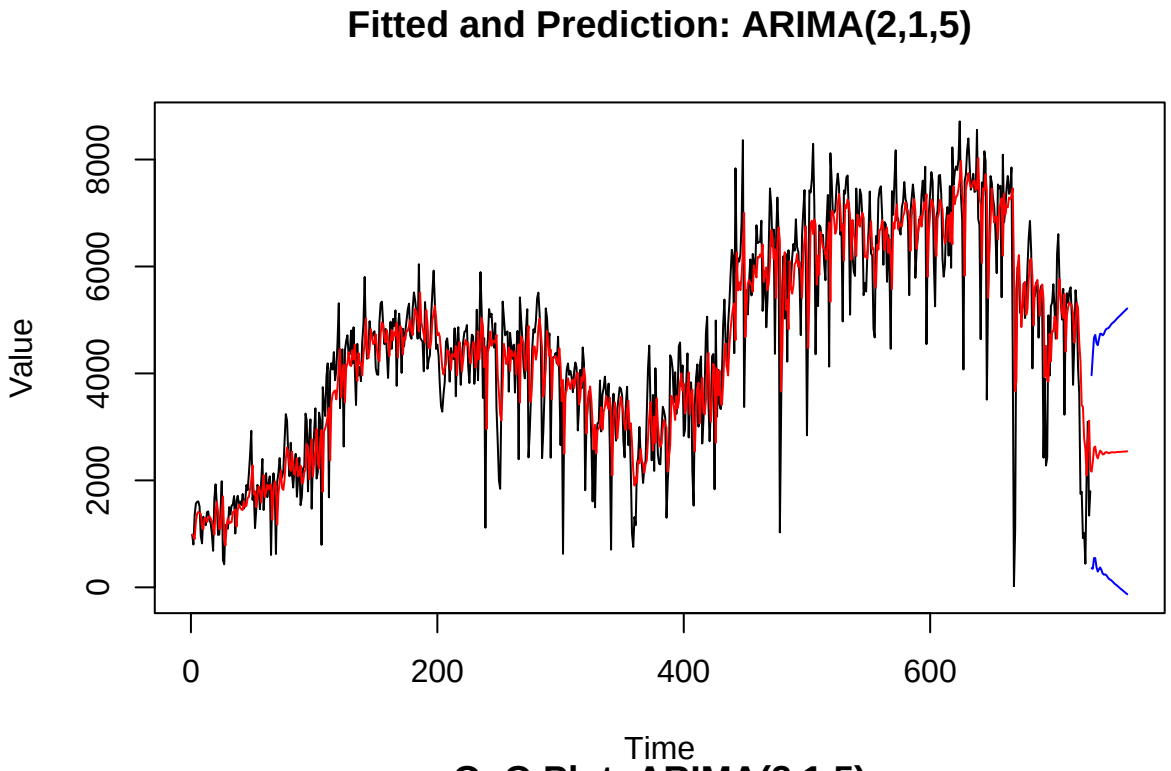
```
## initial  value 6.971617
## iter   2 value 6.858179
## iter   3 value 6.840680
```

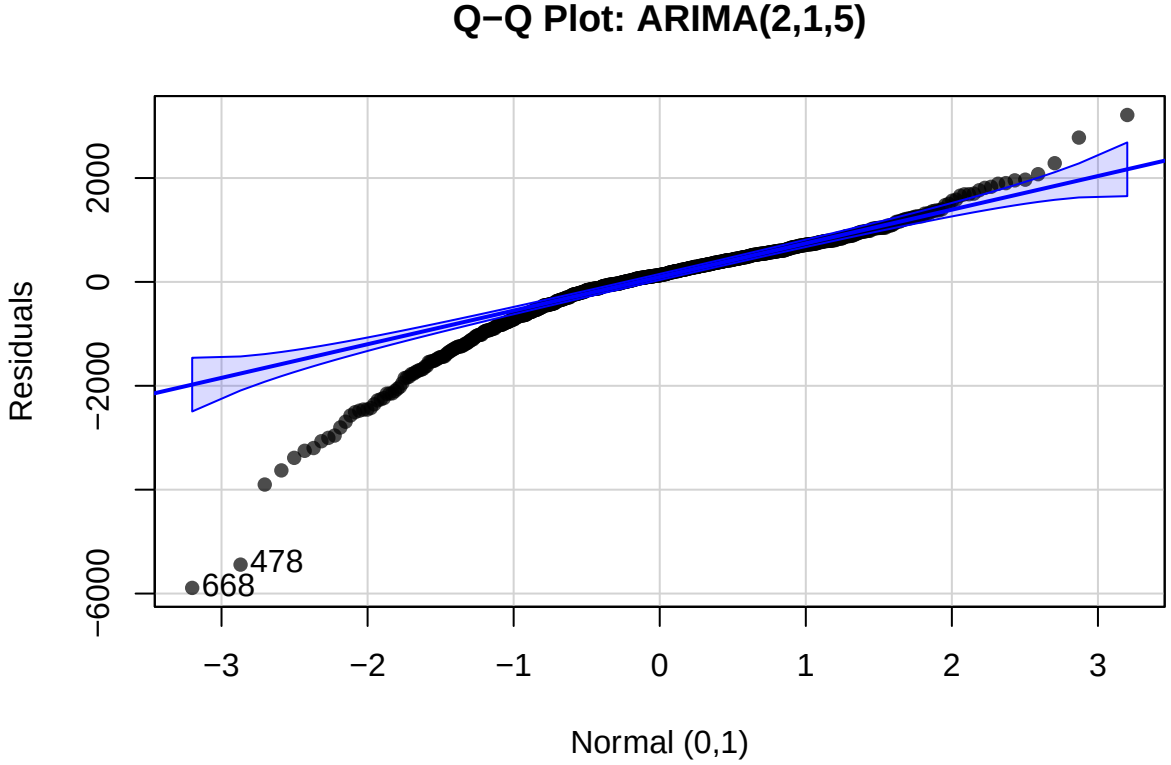
```
## iter   4 value 6.830024
## iter   5 value 6.829610
## iter   6 value 6.829566
## iter   7 value 6.829502
## iter   8 value 6.829485
## iter   9 value 6.829472
## iter  10 value 6.829465
## iter  11 value 6.829398
## iter  12 value 6.829228
## iter  13 value 6.828308
## iter  14 value 6.828231
## iter  15 value 6.827910
## iter  16 value 6.827261
## iter  17 value 6.826483
## iter  18 value 6.826239
## iter  19 value 6.826146
## iter  20 value 6.826041
## iter  21 value 6.826000
## iter  22 value 6.825972
## iter  23 value 6.825965
## iter  24 value 6.825962
## iter  25 value 6.825962
## iter  26 value 6.825961
## iter  27 value 6.825958
## iter  28 value 6.825954
## iter  29 value 6.825950
## iter  30 value 6.825948
## iter  31 value 6.825948
## iter  31 value 6.825948
## iter  31 value 6.825948
## final  value 6.825948
## converged
## initial  value 6.825177
## iter   2 value 6.825174
## iter   2 value 6.825174
## iter   2 value 6.825174
## final  value 6.825174
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE t.value p.value
## ar1        0.3254 0.2444  1.3312  0.1836
## ar2       -0.6160 0.2290 -2.6897  0.0073
## ma1       -0.8499 0.2489 -3.4149  0.0007
## ma2        0.5942 0.2269  2.6189  0.0090
## ma3       -0.3612 0.1414 -2.5546  0.0108
## ma4       -0.1573 0.0732 -2.1496  0.0319
## ma5        0.0100 0.0579  0.1724  0.8631
## constant   1.6900 6.2905  0.2687  0.7883
##
## sigma^2 estimated as 846597.9 on 721 degrees of freedom
##
## AIC = 16.51292  AICc = 16.51319  BIC = 16.5696
```

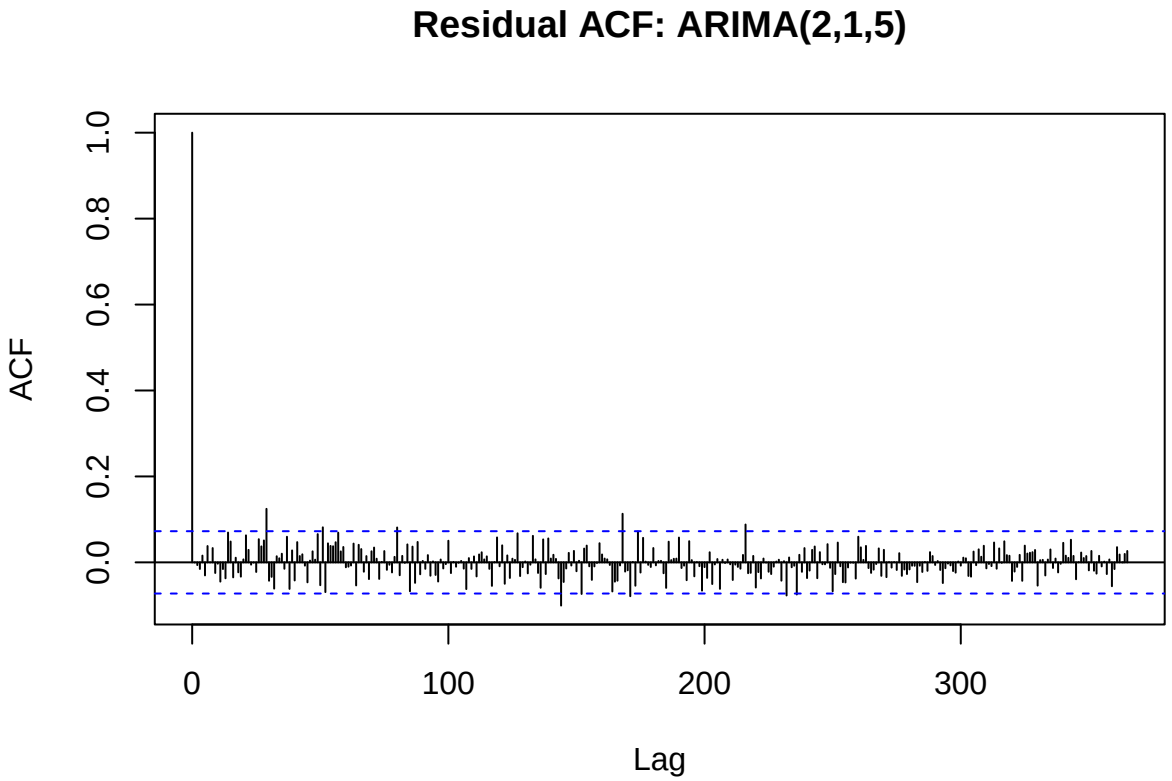
##

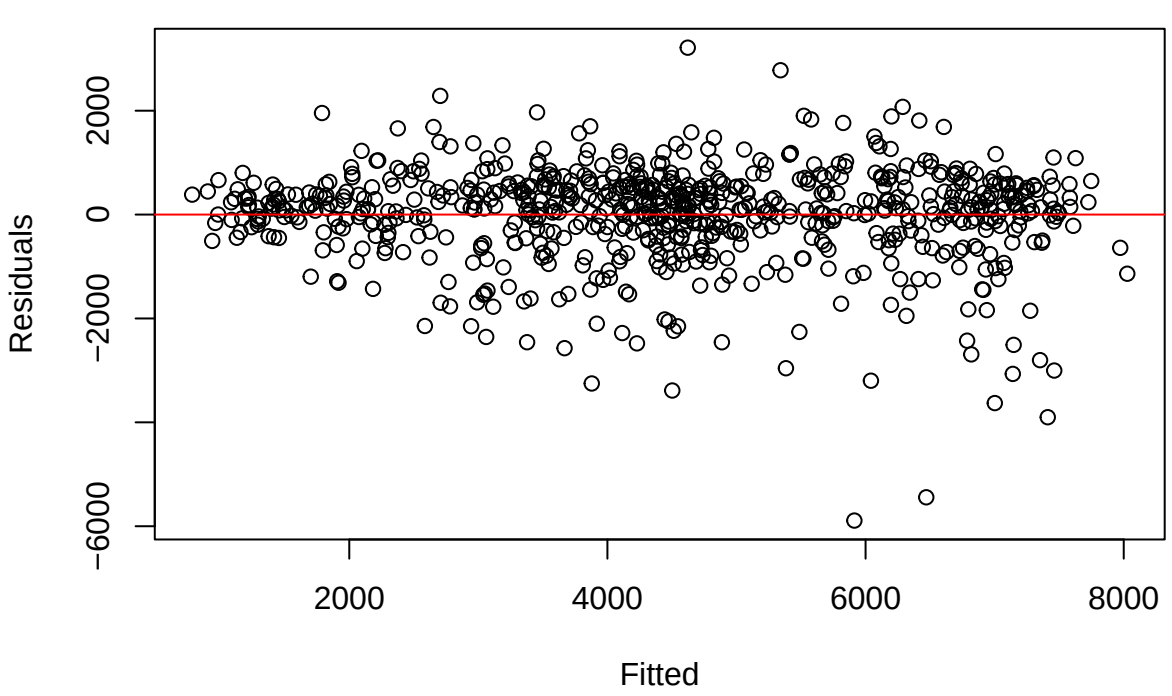


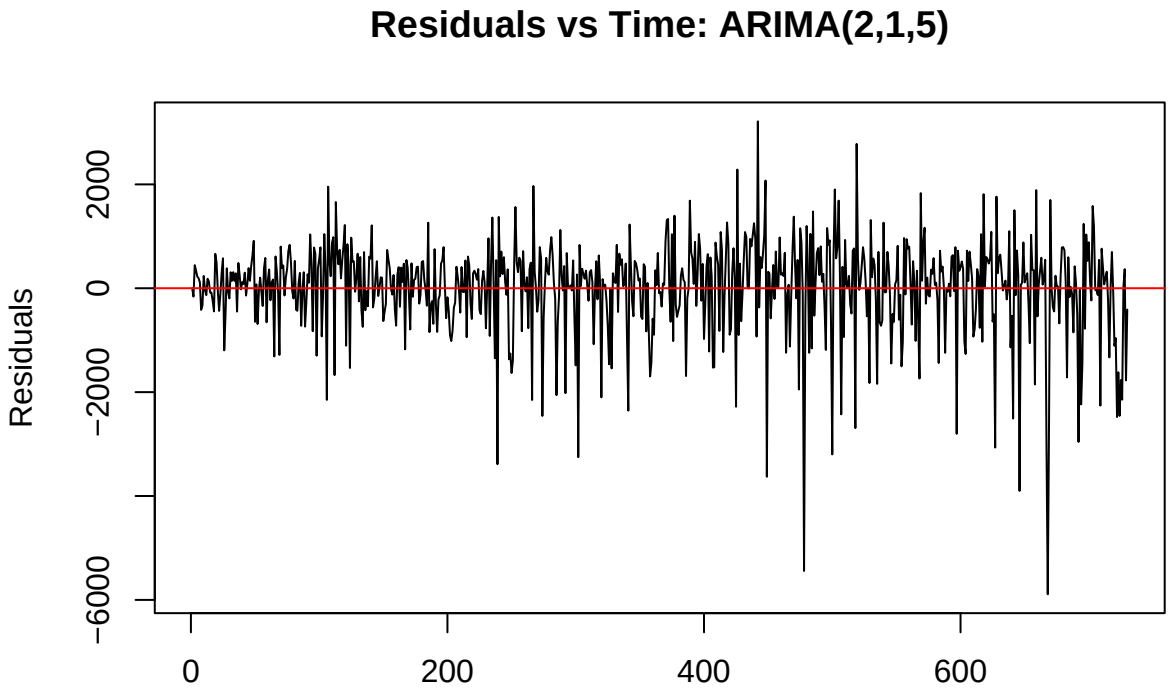

```
##                   Length Class  Mode
## fit                15     Arima  list
## sigma2              1     -none- numeric
## degrees_of_freedom  1     -none- numeric
## t.table            32     -none- numeric
## ICs                 3     -none- numeric
##              name Pred.Err     AICc     BIC    apse      mse    normality
## AICc ARIMA(2,1,5) 1135.682 16.51319 16.5696 7641713 845438.2 6.643426e-21
##      randomness
## AICc 0.06404685
```

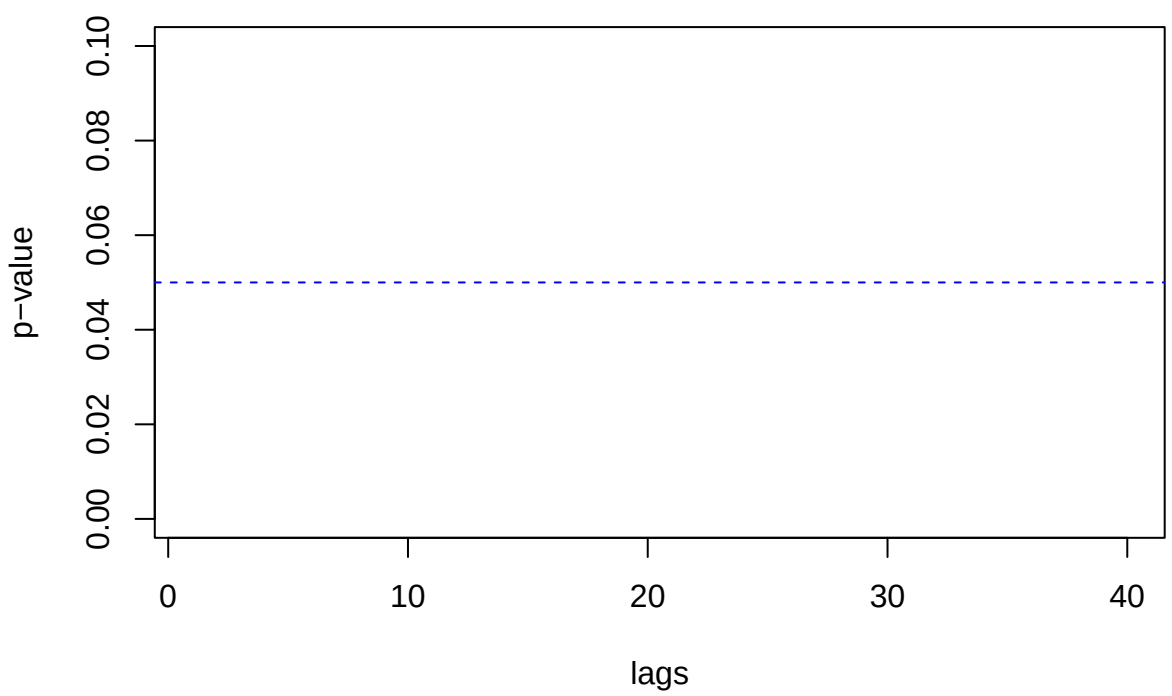

Fitted and Prediction: ARIMA(2,1,5)
Value
0
2000
4000
6000
8000
0
200
400
600
Time


Q−Q Plot: ARIMA(2,1,5)
Residuals
−6000
−2000
0
2000
478
668
−3
−2
−1
0
1
2
3
Normal (0,1)


Residual ACF: ARIMA(2,1,5)
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
Lag


Residuals vs Fitted: ARIMA(2,1,5)
Residuals
2000
0
−2000
−6000
2000
4000
6000
8000
Fitted



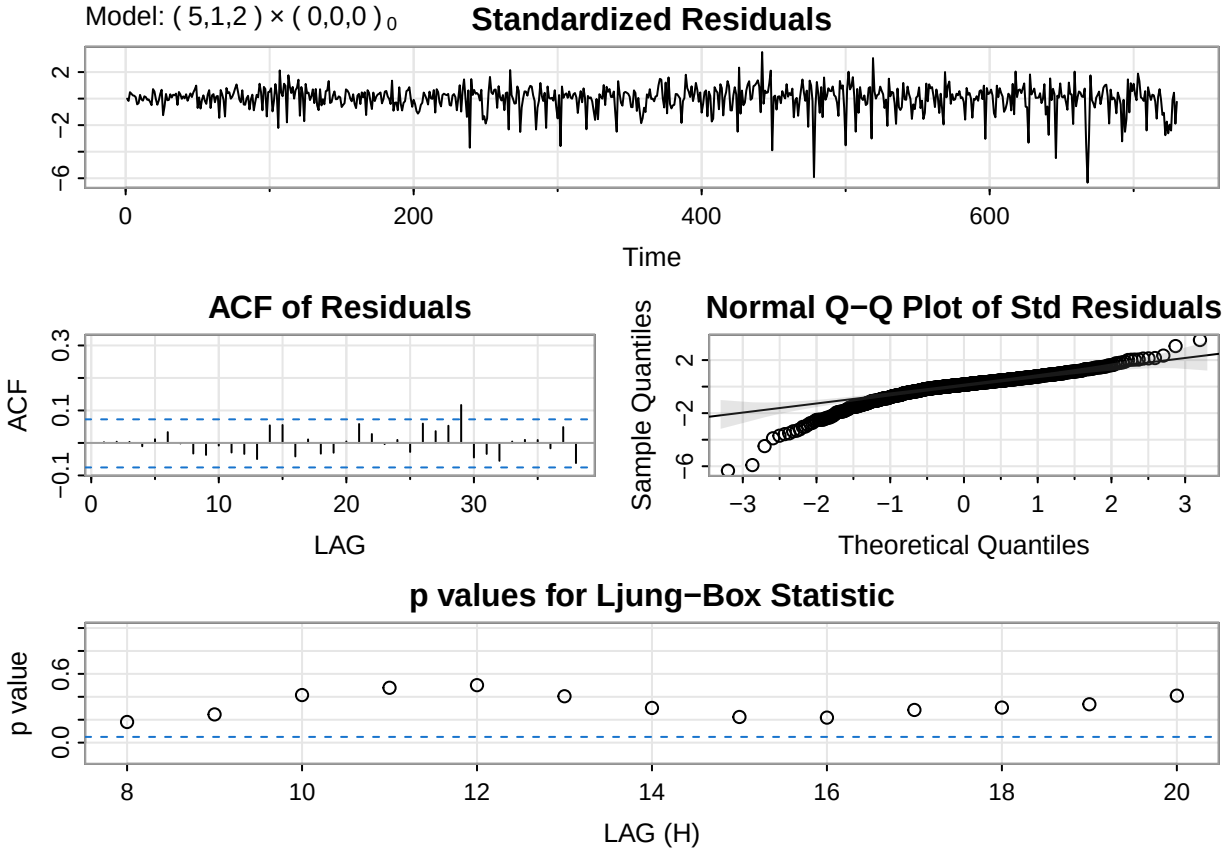



```
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, ARIMA.512.))
```

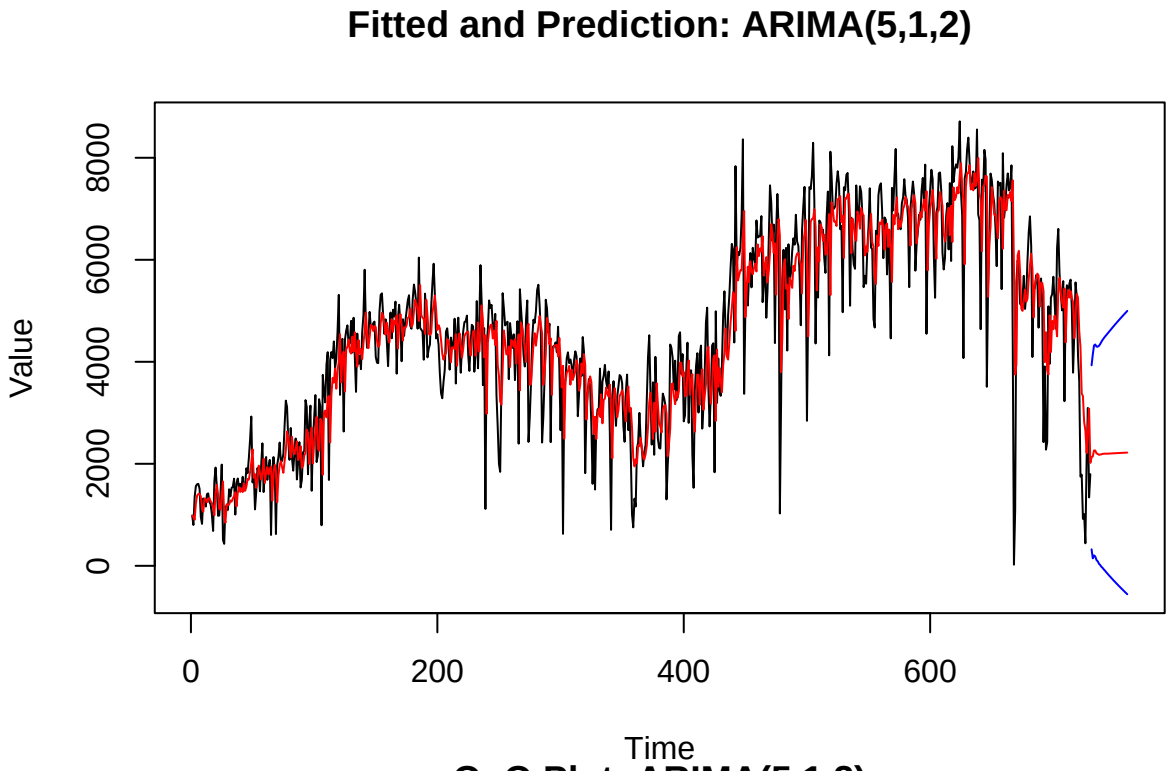
```
## initial  value 6.973657
## iter   2 value 6.866400
## iter   3 value 6.852132
```

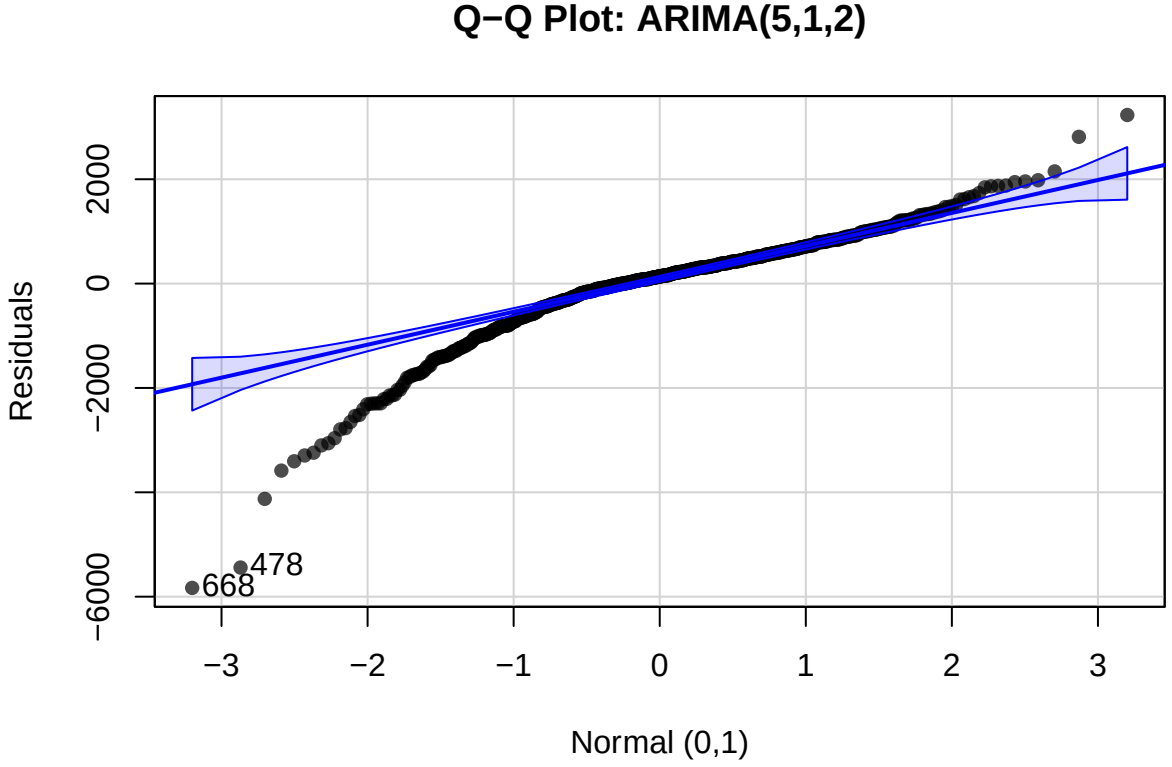
```
## iter   4 value 6.832648
## iter   5 value 6.831727
## iter   6 value 6.831197
## iter   7 value 6.830878
## iter   8 value 6.830191
## iter   9 value 6.829488
## iter  10 value 6.829136
## iter  11 value 6.829099
## iter  12 value 6.829087
## iter  13 value 6.829083
## iter  14 value 6.829036
## iter  15 value 6.828910
## iter  16 value 6.828832
## iter  17 value 6.828652
## iter  18 value 6.828417
## iter  19 value 6.828378
## iter  20 value 6.828362
## iter  21 value 6.828340
## iter  22 value 6.828329
## iter  23 value 6.828314
## iter  24 value 6.828272
## iter  25 value 6.828241
## iter  26 value 6.828229
## iter  27 value 6.828212
## iter  28 value 6.828180
## iter  29 value 6.828144
## iter  30 value 6.828108
## iter  31 value 6.828101
## iter  32 value 6.828100
## iter  32 value 6.828100
## iter  32 value 6.828100
## final  value 6.828100
## converged
## initial  value 6.825543
## iter   2 value 6.825541
## iter   3 value 6.825541
## iter   4 value 6.825541
## iter   5 value 6.825541
## iter   5 value 6.825540
## iter   5 value 6.825540
## final  value 6.825540
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE t.value p.value
## ar1       -0.5216 0.1406 -3.7084  0.0002
## ar2        0.1786 0.0929  1.9223  0.0550
## ar3       -0.1233 0.0583 -2.1154  0.0347
## ar4       -0.1161 0.0614 -1.8892  0.0593
## ar5       -0.0844 0.0469 -1.7976  0.0727
## ma1       -0.0007 0.1372 -0.0050  0.9960
## ma2       -0.6592 0.1244 -5.2994  0.0000
## constant   1.2996 7.0122  0.1853  0.8530
```

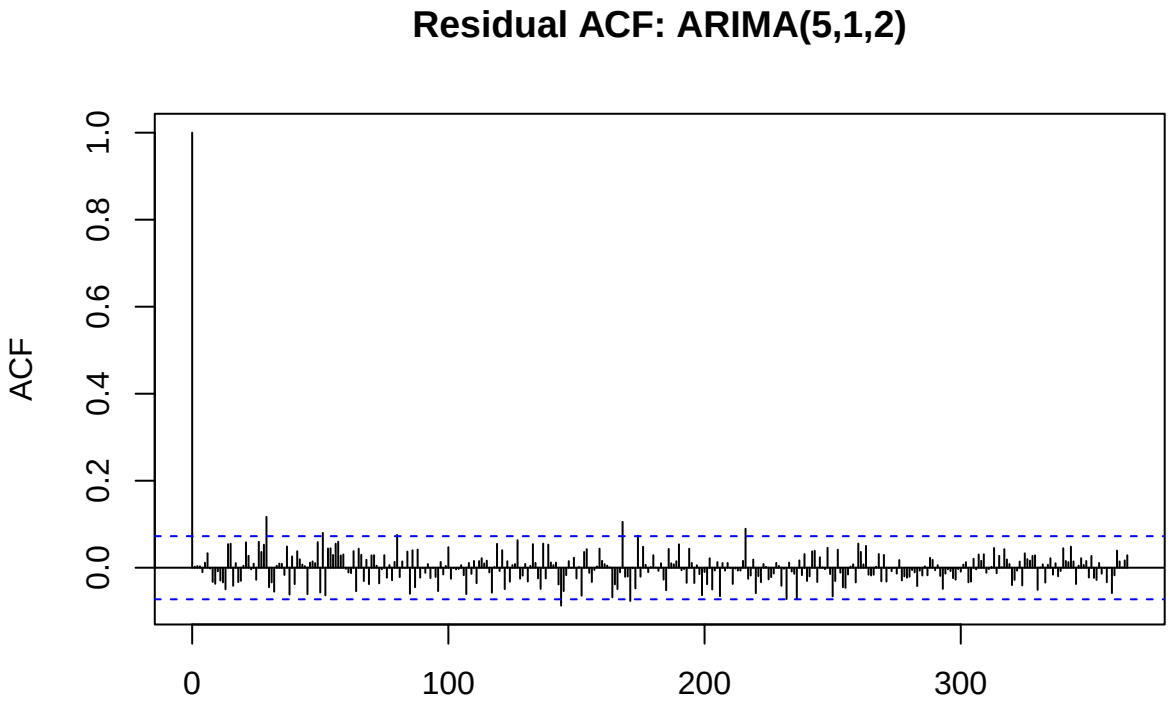
```
##
## sigma^2 estimated as 847245 on 721 degrees of freedom
##
## AIC = 16.51365  AICc = 16.51392  BIC = 16.57034
##
```

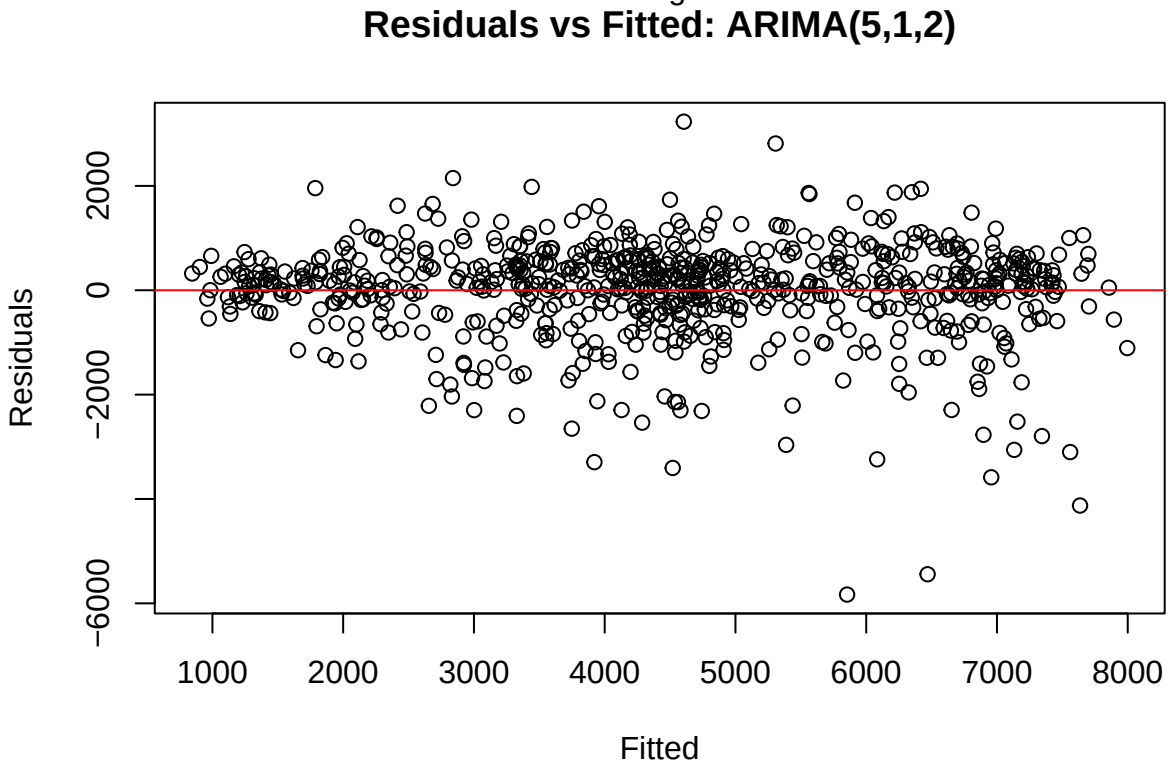


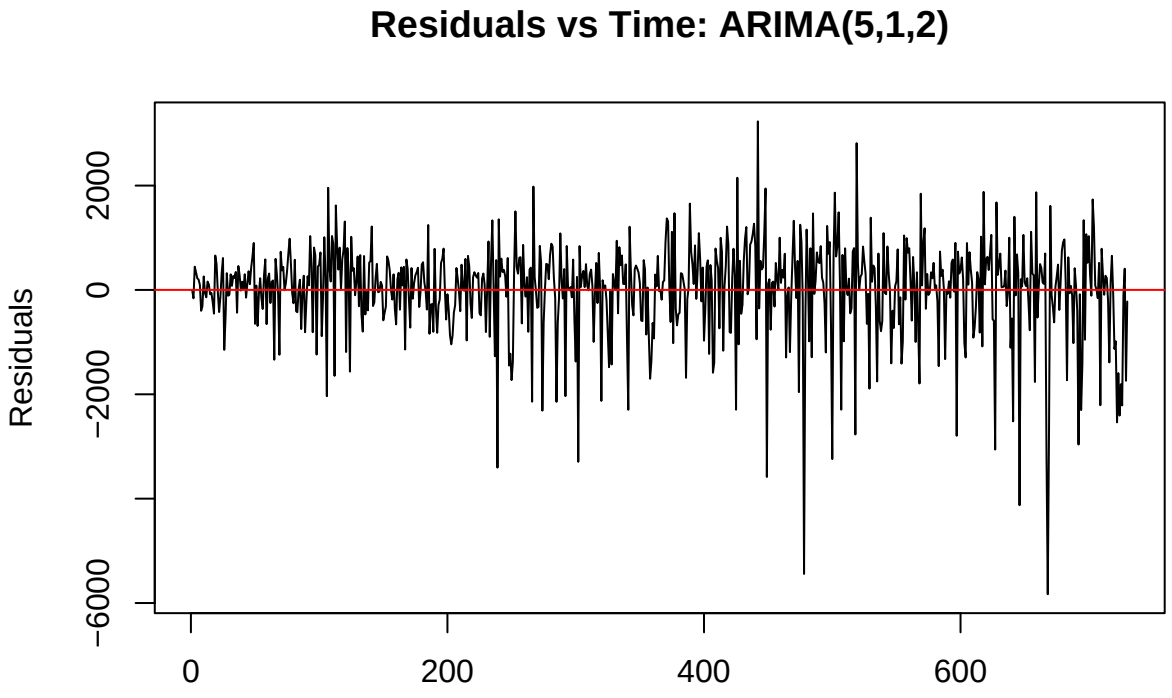

```
##                   Length Class  Mode
## fit                15     Arima  list
## sigma2              1     -none- numeric
## degrees_of_freedom  1     -none- numeric
## t.table            32     -none- numeric
## ICs                 3     -none- numeric
##              name Pred.Err     AICc      BIC    apse      mse    normality
## AICc ARIMA(5,1,2) 1140.781 16.51392 16.57034 7698798 846084.3 5.421009e-21
##      randomness
## AICc  0.6040962
```

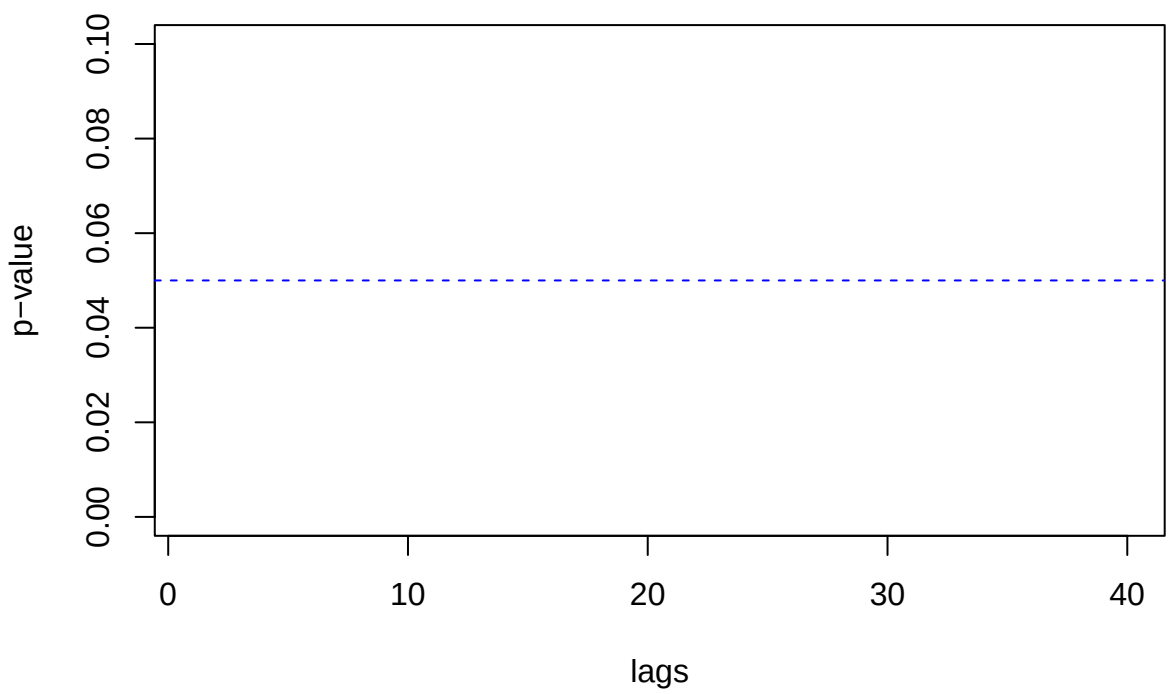

Fitted and Prediction: ARIMA(5,1,2)
Value
0
2000
4000
6000
8000
0
200
400
600
Time


Q−Q Plot: ARIMA(5,1,2)
Residuals
−6000
−2000
0
2000
478
668
−3
−2
−1
0
1
2
3
Normal (0,1)


Residual ACF: ARIMA(5,1,2)
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
Lag


Residuals vs Fitted: ARIMA(5,1,2)
Residuals
2000
0
−2000
−6000
1000
2000
3000
4000
5000
6000
7000
8000
Fitted






```
sarima.performances
```

```
##                name Pred.Err     AICc      BIC    apse      mse    normality
## AICc  ARIMA(2,1,2) 1134.172 16.51206 16.54974 7815633 851642.9 5.789628e-21
## AICc1 ARIMA(2,1,5) 1135.682 16.51319 16.56960 7641713 845438.2 6.643426e-21
## AICc2 ARIMA(5,1,2) 1140.781 16.51392 16.57034 7698798 846084.3 5.421009e-21
```

```
##       randomness
## AICc  0.50498467
## AICc1 0.06404685
## AICc2 0.60409620
```

Set eval=TRUE to run these code

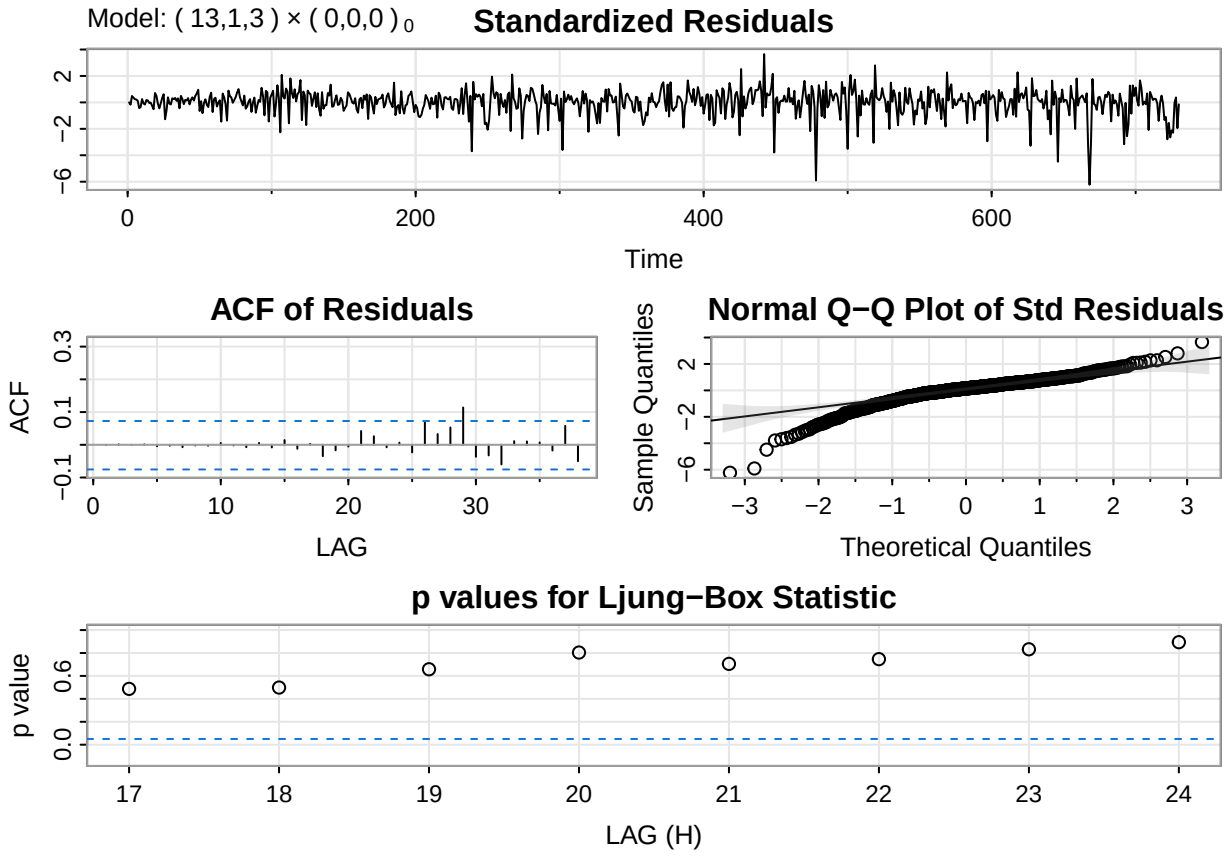
```
try.sarima <- data.frame(name = 'ARIMA(13,1,4)',
                         p=13, d=1, q=4, P=0, D=0, Q=0, S=0)
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, try.sarima))
try.sarima <- data.frame(name = 'ARIMA(15,1,4)',
                         p=15, d=1, q=4, P=0, D=0, Q=0, S=0)
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, try.sarima))
try.sarima <- data.frame(name = 'ARIMA(15,1,1)',
                         p=15, d=1, q=1, P=0, D=0, Q=0, S=0)
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, try.sarima))
try.sarima <- data.frame(name = 'ARIMA(2,1,29)',
                         p=2, d=1, q=29, P=0, D=0, Q=0, S=0)
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, try.sarima))
 try.sarima <- data.frame(name = 'ARIMA(13,1,29)',
                          p=13, d=1, q=29, P=0, D=0, Q=0, S=0)
 sarima.performances <- rbind(sarima.performances,
                              sarima.analysis(bike.data$bikes, try.sarima))
# try.sarima <- data.frame(name = 'ARIMA(29,1,4)',
#                          p=29, d=1, q=4, P=0, D=0, Q=0, S=0)
# sarima.performances <- rbind(sarima.performances,
#                              sarima.analysis(bike.data$bikes, try.sarima))
# try.sarima <- data.frame(name = 'ARIMA(29,1,1)',
#                          p=29, d=1, q=1, P=0, D=0, Q=0, S=0)
# sarima.performances <- rbind(sarima.performances,
#                              sarima.analysis(bike.data$bikes, try.sarima))
# try.sarima <- data.frame(name = 'SARIMA(29,1,4)x(0,0,0)_365',
#                          p=29, d=1, q=4, P=0, D=0, Q=0, S=365)
#sarima.performances <- rbind(sarima.performances,
#                              sarima.analysis(bike.data$bikes, try.sarima))
sarima.performances
```

```
try.sarima <- data.frame(name = 'ARIMA(13,1,6)',
                         p=13, d=1, q=6, P=0, D=0, Q=0, S=0)
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, try.sarima))
try.sarima <- data.frame(name = 'ARIMA(29,1,29)',
                         p=29, d=1, q=29, P=0, D=0, Q=0, S=0)
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, try.sarima))
sarima.performances
```

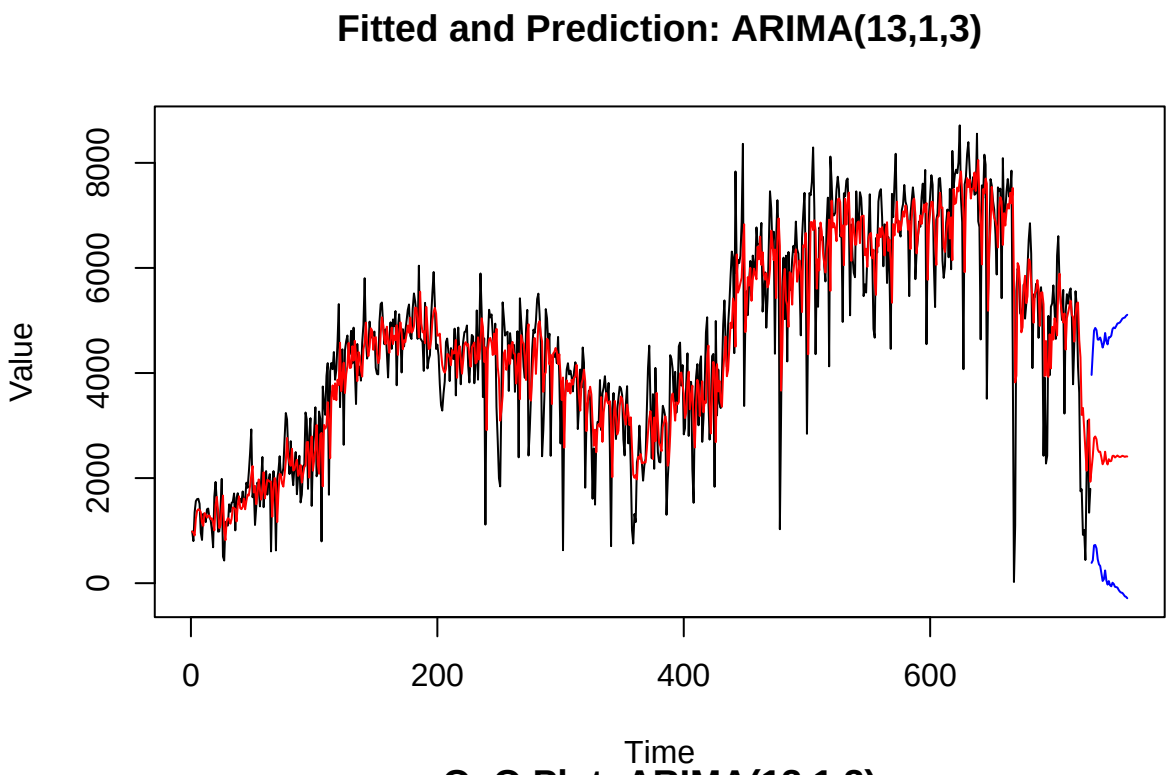
```
try.sarima <- data.frame(name = 'ARIMA(13,1,3)',
                         p=13, d=1, q=3, P=0, D=0, Q=0, S=0)
sarima.performances <- rbind(sarima.performances,
                             sarima.analysis(bike.data$bikes, try.sarima))
```

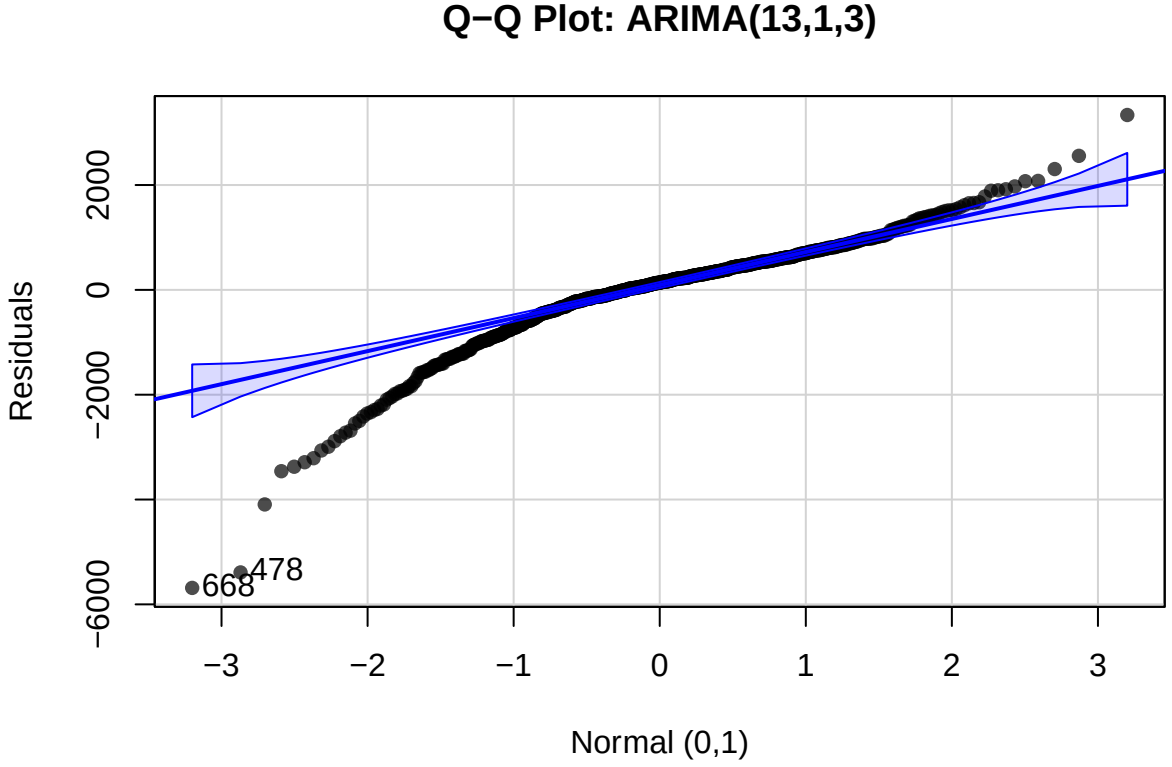
```
## initial  value 6.978816
## iter   2 value 6.855998
## iter   3 value 6.842111
## iter   4 value 6.829562
## iter   5 value 6.827986
## iter   6 value 6.827185
## iter   7 value 6.826889
## iter   8 value 6.826739
## iter   9 value 6.826667
## iter  10 value 6.826617
## iter  11 value 6.826472
## iter  12 value 6.826234
## iter  13 value 6.825833
## iter  14 value 6.825366
## iter  15 value 6.825084
## iter  16 value 6.824861
## iter  17 value 6.824762
## iter  18 value 6.824722
## iter  19 value 6.824718
## iter  20 value 6.824715
## iter  21 value 6.824712
## iter  22 value 6.824708
## iter  23 value 6.824697
## iter  24 value 6.824672
## iter  25 value 6.824641
## iter  26 value 6.824621
## iter  27 value 6.824616
## iter  28 value 6.824615
## iter  28 value 6.824615
## iter  28 value 6.824615
## final  value 6.824615
## converged
## initial  value 6.817111
## iter   2 value 6.817109
## iter   3 value 6.817108
## iter   4 value 6.817107
## iter   5 value 6.817107
## iter   5 value 6.817107
## iter   5 value 6.817107
## final  value 6.817107
## converged
## <><><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE t.value p.value
## ar1       -0.3545 0.2903 -1.2209  0.2225
## ar2       -0.5564 0.2144 -2.5947  0.0097
## ar3       -0.0617 0.2568 -0.2403  0.8102
## ar4       -0.2300 0.1653 -1.3913  0.1646
## ar5       -0.1897 0.1538 -1.2337  0.2177
## ar6       -0.0659 0.1454 -0.4532  0.6505
## ar7       -0.0777 0.1313 -0.5914  0.5544
## ar8       -0.0516 0.1051 -0.4906  0.6238
## ar9       -0.1160 0.0787 -1.4740  0.1409
```

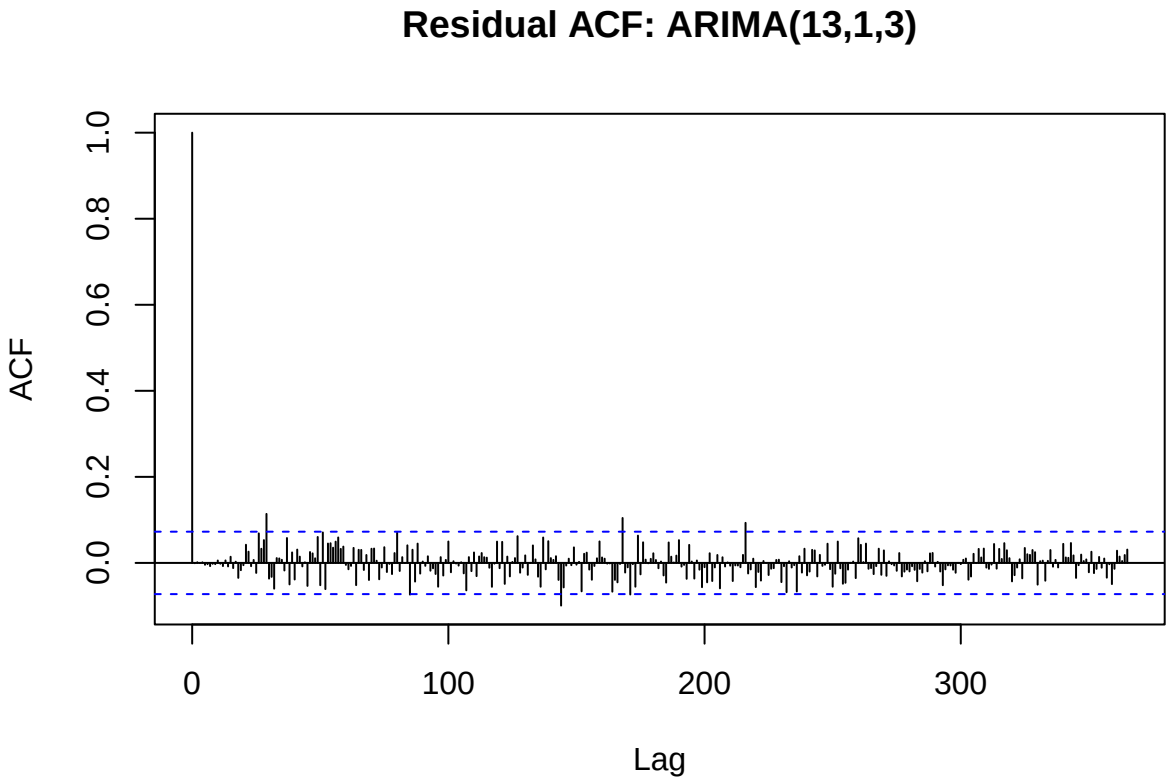
```
## ar10      -0.0596 0.0706 -0.8445  0.3987
## ar11      -0.0977 0.0639 -1.5286  0.1268
## ar12      -0.0755 0.0524 -1.4417  0.1498
## ar13      -0.1268 0.0421 -3.0093  0.0027
## ma1       -0.1676 0.2926 -0.5728  0.5670
## ma2        0.1708 0.2600  0.6568  0.5115
## ma3       -0.4095 0.2332 -1.7560  0.0795
## constant   1.5984 6.6135  0.2417  0.8091
##
## sigma^2 estimated as 832822.6 on 712 degrees of freedom
##
## AIC = 16.52147  AICc = 16.52265  BIC = 16.63485
##
```

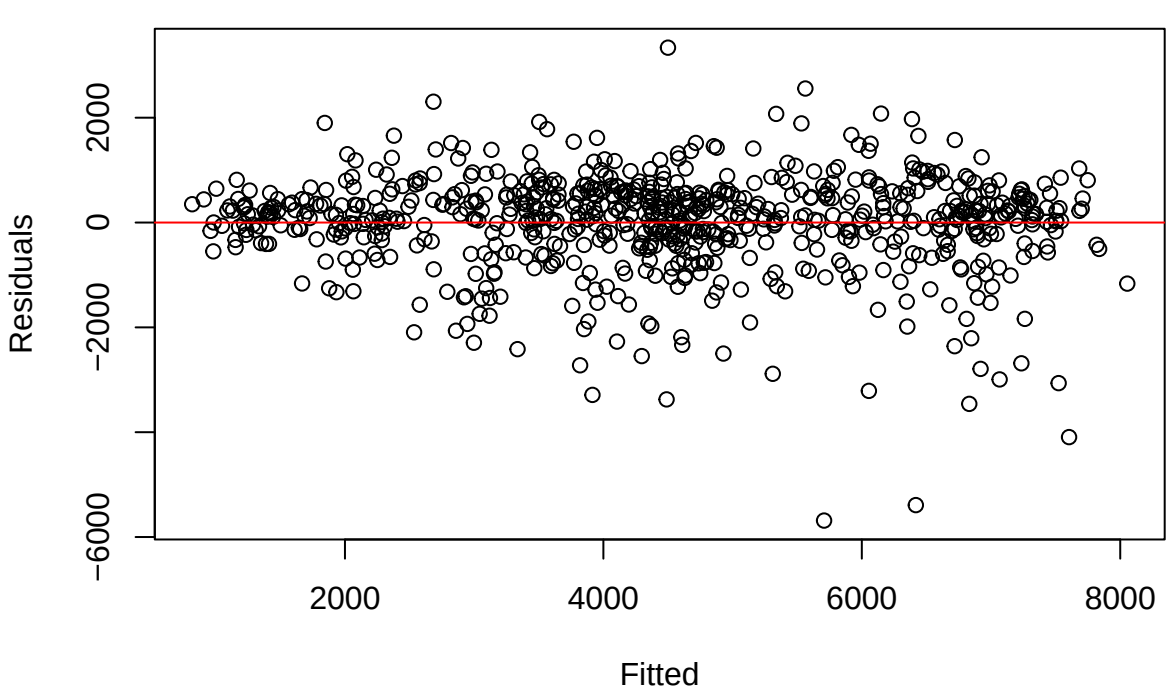


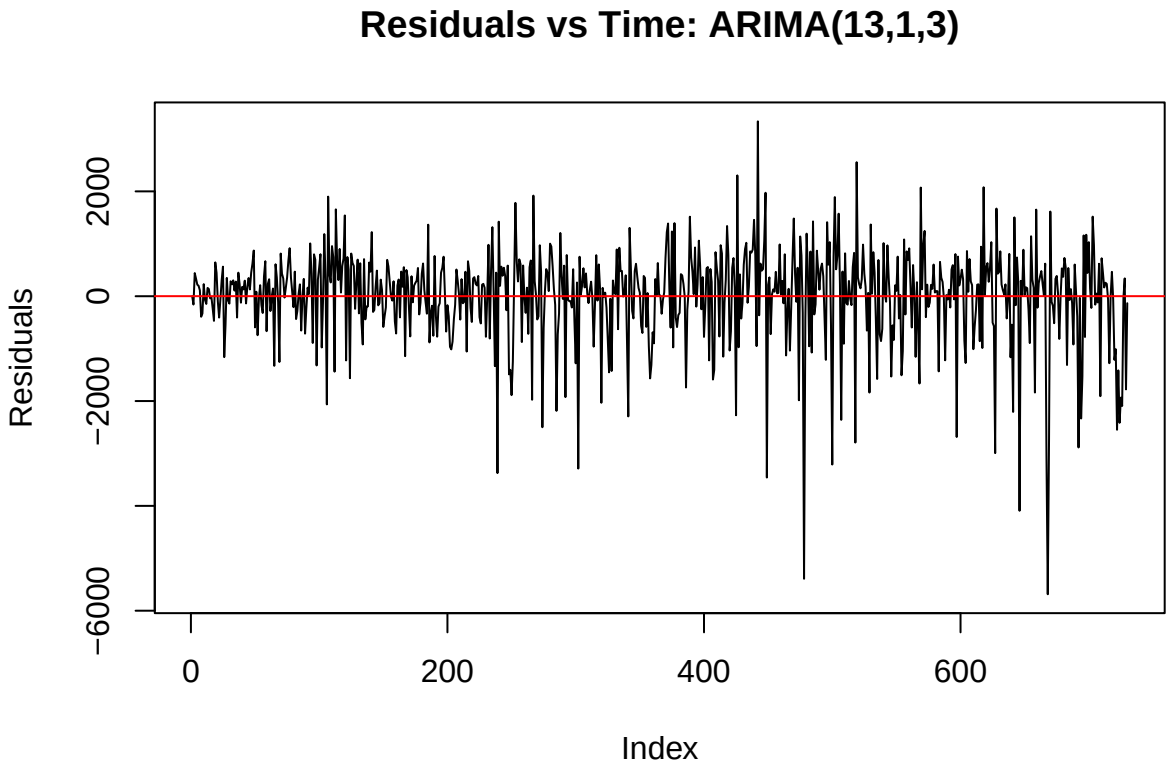

```
##                   Length Class  Mode
## fit                15     Arima  list
## sigma2              1     -none- numeric
## degrees_of_freedom  1     -none- numeric
## t.table            68     -none- numeric
## ICs                 3     -none- numeric
##               name Pred.Err     AICc      BIC    apse      mse    normality
## AICc ARIMA(13,1,3) 1133.426 16.52265 16.63485 7762009 831681.8 1.471649e-20
##      randomness
## AICc  0.6040962
```

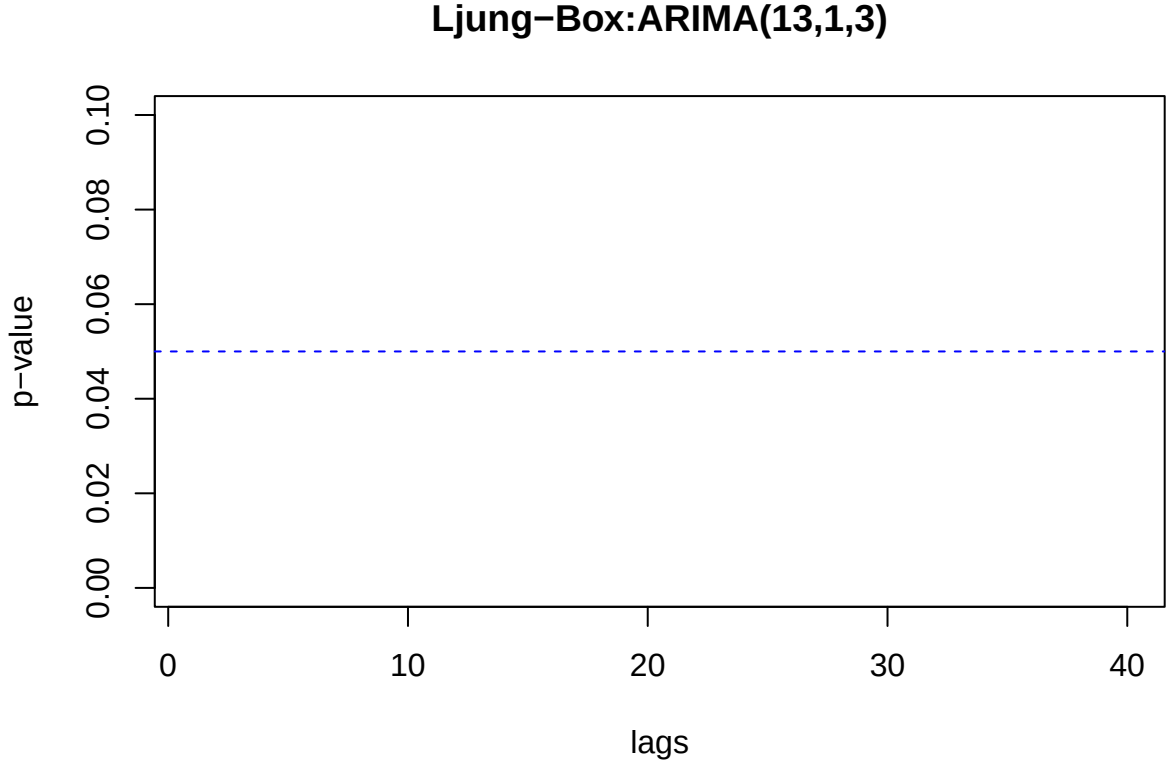

Fitted and Prediction: ARIMA(13,1,3)
Value
0
2000
4000
6000
8000
0
200
400
600
Time


Q−Q Plot: ARIMA(13,1,3)
Residuals
−6000
−2000
0
2000
668
478
−3
−2
−1
0
1
2
3
Normal (0,1)


Residual ACF: ARIMA(13,1,3)
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
Lag


Residuals vs Fitted: ARIMA(13,1,3)
Residuals
−6000
−2000
0
2000
2000
4000
6000
8000
Fitted






```
sarima.performances
```

```
##                name Pred.Err     AICc      BIC    apse      mse    normality
## AICc   ARIMA(2,1,2) 1134.172 16.51206 16.54974 7815633 851642.9 5.789628e-21
## AICc1  ARIMA(2,1,5) 1135.682 16.51319 16.56960 7641713 845438.2 6.643426e-21
## AICc2  ARIMA(5,1,2) 1140.781 16.51392 16.57034 7698798 846084.3 5.421009e-21
```

```
## AICc3 ARIMA(13,1,3) 1133.426 16.52265 16.63485 7762009 831681.8 1.471649e-20
##       randomness
## AICc  0.50498467
## AICc1 0.06404685
## AICc2 0.60409620
## AICc3 0.60409620
```

```r
write.csv(sarima.performances, 'sarima_models_comp.csv', row.names = FALSE)
```

Regression

Functions for Regression:

```
# confidence band for prediction
CI.wing.vec <- function(X.T.X.inv, sd, X.pred) {
  #X.T.X.inv <- solve(crossprod(X.train))
  quad <- rowSums((X.pred %*% X.T.X.inv) * X.pred)
  return(qnorm(0.975) * sd * sqrt(1+quad))
}

# give up weights
# Take a raw model and return true model along with matrices as data.frame
Fit.Model <- function(model, model.name, weights = FALSE) {
  # keep two versions: matrix X for (XTX)^{-1}, and data frame for lm
  matrix.fit <- model.matrix(model)
  matrix.train <- matrix.fit[bike.data$t, , drop = FALSE]
  matrix.pred <- matrix.fit[t.pred, , drop = FALSE]

  # data frame version
  X <- as.data.frame(matrix.fit[, -1, drop=FALSE])
  X.train <- X[bike.data$t,]
  data.train <- data.frame(bikes = bike.data$bikes, X.train)
  X.pred <- X[t.pred,]
  true.model <- lm(bikes ~ ., data = data.train)
  if(!isFALSE(weights)) {
    true.model <- lm(bikes ~ ., weights = weights, data = data.train)
  }

  return(list(name = model.name,
              model = true.model,
              X.train = X.train,
              data.train = data.train, # dataset for lm
              X.pred = X.pred,
              X.full = X,
              matrix.train = matrix.train,
              matrix.pred = matrix.pred,
              matrix.full = matrix.fit))
}

# APSE (5-fold CV)
apse.k <- 5 # 5 groups
apse.size <- num.obs / apse.k # 146 observations per group

# run k-fold cross-validation and gives the apse
APSE <- function(model, dataset, y.weights = FALSE) {
  # shuffle training data
  dataset <- dataset[sample(1:nrow(dataset)),]
  # prediction error for each test group
  apses <- c()
  for(i in 1:apse.k) {
    # indices for testing group
    testing_idx <- (i - 1) * apse.size + 1:apse.size
    # training data
    data.train <- dataset[-testing_idx,]
```

```
    weights.train <- y.weights[-testing_idx]
    data.test <- dataset[testing_idx,]
    #new.model <- update(model, data = data.train)
    new.model <- lm(bikes ~ ., data = data.train)
    if(!isFALSE(y.weights)) {
      new.model <- lm(bikes ~ ., data = data.train, weights = weights.train)
    }
    apse <- mean((data.test$bikes - predict(new.model, newdata=data.test))^2)
    apses <- c(apses, apse)
  }
  return(mean(apses))
}
```

For model analysis:

```
# Contains model fit, prediction, plot, and residual diagnostics.
# model.fit is a collection generated from Fit.Model()
model.summary <- function(model.fit,
                          print.summary=FALSE,
                          plot=FALSE,
                          residual.plot=FALSE) {
  model <- model.fit$model
  name <- model.fit$name
  X.train <- model.fit$X.train
  data.train <- model.fit$data.train
  X.pred <- model.fit$X.pred
  matrix.train <- model.fit$matrix.train
  matrix.pred <- model.fit$matrix.pred
  X.T.X.inv <- solve(crossprod(matrix.train))

  if(print.summary | plot | residual.plot) {
    print(paste0('--------> START SUMMARY of MODEL ', name))
  }

  # if True, print summary of model
  summ <- summary(model)
  if(print.summary) {
    print(summ)
  }

  model.pred <- predict(model, newdata = X.pred)

  # std dev
  model.sd <- sqrt(sum(model$residuals^2) / model$df.residual)
  model.apse <- APSE(model, data.train)
  CI.wing <- CI.wing.vec(X.T.X.inv, model.sd, matrix.pred)
  # prediction error at beginning of prediction (2013-01-09)
  pred.err <- CI.wing[10]
  # record p values of tests
  normality <- shapiro.test(model$residuals)$p.value
  randomness <- randtests::runs.test(model$residuals)$p.value

  # fitted data and prediction
  if(plot) {
```

```
  # original data
  plot(dates, Bike, type = 'l', xlim = plot.range,
       main = paste0('Fitted and Prediction: ', name),
       xlab = 'Time', ylab = 'Counts',
       ylim = c(0, max(c(model.pred + CI.wing, bike.data$bikes))))

  # fitted
  lines(dates, model$fitted.values, col = 'red')

  # prediction
  lines(dates.pred, model.pred, col = 'red')

  # prediction interval
  lines(dates.pred, model.pred + CI.wing, col = 'blue')
  lines(dates.pred, model.pred - CI.wing, col = 'blue')
  legend('topleft', legend = c('Fitted&Pred', 'Pred Interval'),
         col = c('red', 'blue'),
         lty = 1)
}

# residual diagnostics
if(residual.plot) {
  H <- matrix.train %*% X.T.X.inv %*% t(matrix.train) # projection matrix
  res.band <- 1.96 * model.sd * (1 - diag(H))
  super.residual.plot(name, model$residuals, model$fitted.values, res.band)
}


results <- data.frame(Model = name,
                      ResErr = model.sd,
                      PredErr = pred.err,
                      AICc = AICc(model),
                      BIC = BIC(model),
                      adjR2 = summ$adj.r.squared,
                      APSE = model.apse,
                      Normality = normality,
                      Randomness = randomness)

if(print.summary) {
  print('Summary:')
  print(results)
}

if(print.summary | plot | residual.plot) {
  print(paste0('--------> END SUMMARY of MODEL ', name))
}

# save information into records
# if (!isFALSE(records)) {
#   records[name, ] <- results
#   print(records)
# } not working in R since data frame is passed by copies
```

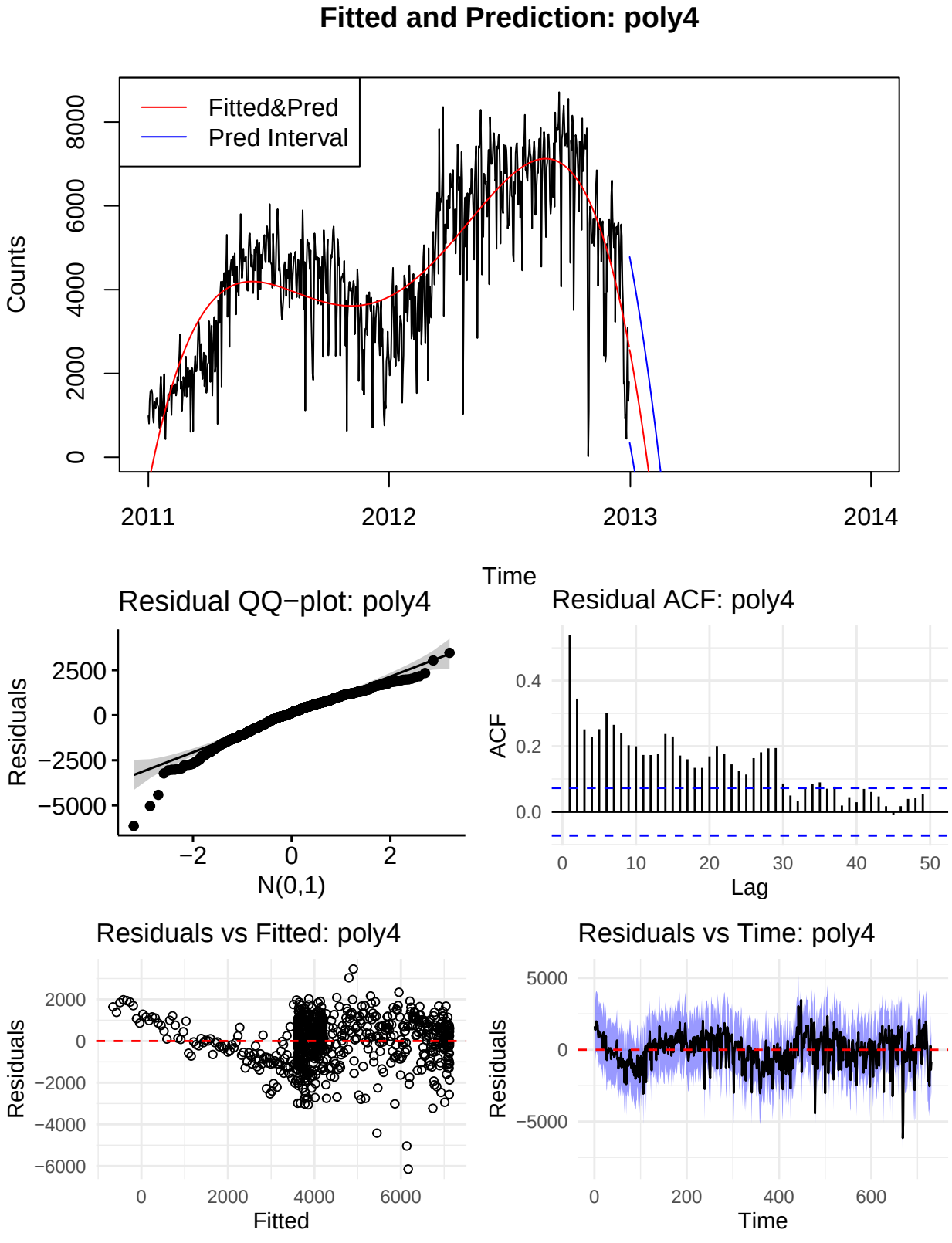
```
  # return a list of model analysis results
  return(results)
}
```

Regression Processes

```
data.weights <- function(y) {
  y <- as.vector(y)
  n <- length(y)
  w <- numeric(n)
  w[1] <- 1 / abs(y[1]-y[2])
  w[n] <- 1 / abs(y[n]-y[n-1])
  for(i in 2:(n-1)) {
    # average abs diff between its two adjacent neighbors
    diff <- mean(abs(y[i]-y[i-1]), abs(y[i]-y[i+1]))
    w[i] <- 1 / diff^2
  }
  return(w)
}

bike.weights <- data.weights(Bike)
```

```
#Fit a baseline model (deg 4 polynomial)
poly4 <- poly(t.full, degree = 4)
model.poly4 <- lm(y.dummy ~ poly4)
FIT <- Fit.Model(model.poly4, 'poly4')
# FIT$model
# FIT$data.train
# APSE(FIT$model, FIT$data.train)
poly4.summary <- model.summary(FIT, print.summary = TRUE, plot = TRUE, residual.plot = TRUE)
```

```
## [1] "--------> START SUMMARY of MODEL poly4"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6146.7  -663.1   132.4   752.8  3460.8
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -13978.3      827.1  -16.90   <2e-16 ***
## poly41      -853080.3    38666.4  -22.06   <2e-16 ***
## poly42      -770283.7    32518.4  -23.69   <2e-16 ***
## poly43      -437234.5    18931.2  -23.10   <2e-16 ***
## poly44      -157420.6     6926.9  -22.73   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1113 on 725 degrees of freedom
## Multiple R-squared:  0.672,  Adjusted R-squared:  0.6702
## F-statistic: 371.4 on 4 and 725 DF,  p-value: < 2.2e-16
```




```
## [1] "Summary:"
##     Model   ResErr  PredErr     AICc      BIC     adjR2    APSE    Normality
## 740 poly4 1112.593 2230.952 12319.84 12347.29 0.6702201 1248542 3.640645e-12
```

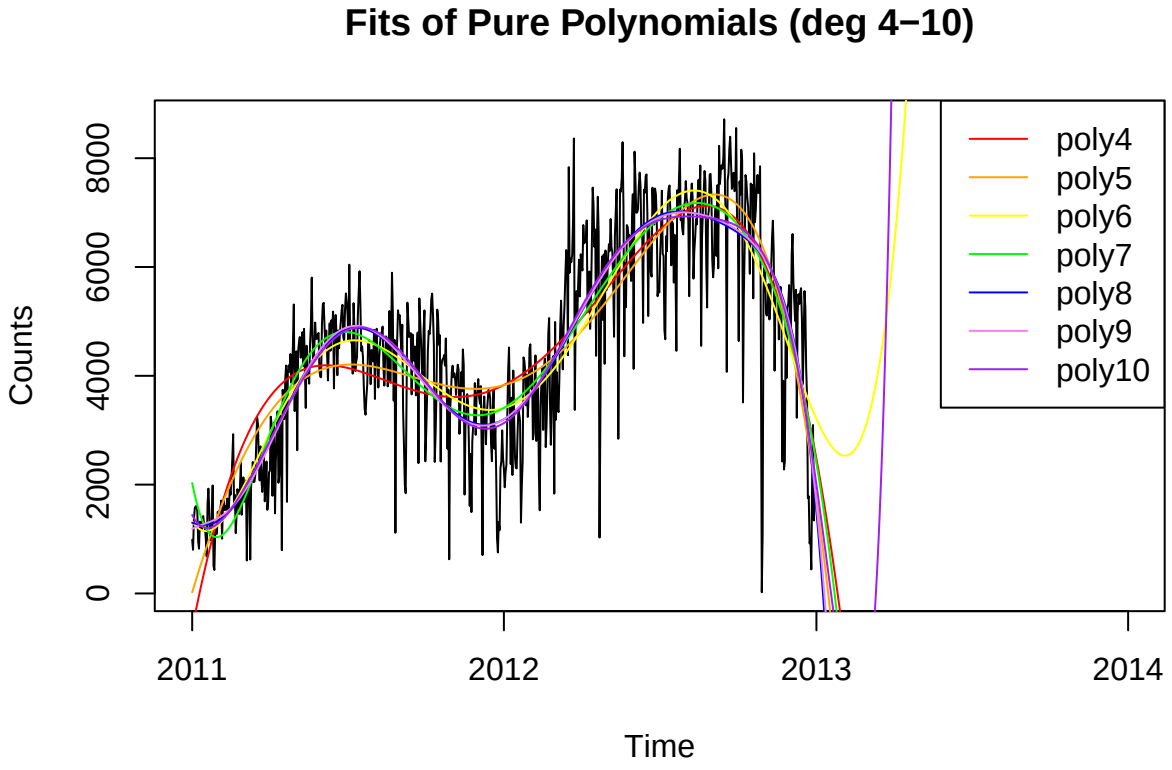
```
##       Randomness
## 740 7.531584e-39
## [1] "--------> END SUMMARY of MODEL poly4"
```

```
poly4.summary
```

```
##     Model   ResErr  PredErr     AICc      BIC     adjR2    APSE    Normality
## 740 poly4 1112.593 2230.952 12319.84 12347.29 0.6702201 1248542 3.640645e-12
##       Randomness
## 740 7.531584e-39
```

Pure Polynomials:

```
# 5-fold CV for fitting pure-polynomials
# performances for all models
performances <- poly4.summary

# consider 6 degrees
# plot all fitted values and predictions
# names and color rainbow for fit of each degree
poly.names <- c('poly4', 'poly5', 'poly6', 'poly7', 'poly8', 'poly9', 'poly10')
poly.colors <- c("red", "orange", "yellow", "green", "blue", "violet", "purple")
plot(dates, Bike, type = 'l',
     xlim=plot.range, main = 'Fits of Pure Polynomials (deg 4-10)',
     xlab = 'Time', ylab = 'Counts')
for(deg in 4:10) {
  # polynomials
  P <- poly(t.full, degree = deg, raw = FALSE)
  model <- lm(y.dummy ~ P)
  model.fit <- Fit.Model(model, poly.names[deg - 3])
  performances <- rbind(performances, model.summary(model.fit))
  # both fit and prediction
  fit.pred <- predict(model.fit$model, model.fit$X.full)
  lines(pred.dates.full, fit.pred, col = poly.colors[deg - 3])
}
legend('topright',
       legend = poly.names,
       col = poly.colors,
       lty = 1)
```




```
performances <- performances[-1,]
performances
```

```
##       Model    ResErr  PredErr     AICc      BIC     adjR2      APSE
## 7401  poly4 1112.5934 2230.952 12319.84 12347.29 0.6702201 1251259.4
## 7402  poly5 1093.6339 2224.864 12295.78 12327.78 0.6813637 1207510.1
## 7403  poly6 1028.0707 2136.464 12206.56 12243.10 0.7184230 1067152.5
## 7404  poly7 1011.1478 2167.197 12183.36 12224.45 0.7276167 1034614.1
## 7405  poly8  994.3370 2226.979 12159.93 12205.56 0.7365984  999267.2
## 7406  poly9  994.6156 2367.755 12161.39 12211.54 0.7364508 1007148.8
## 7407 poly10  993.4027 2566.486 12160.66 12215.34 0.7370932 1003197.3
##         Normality   Randomness
## 7401 3.640645e-12 7.531584e-39
## 7402 2.217222e-12 2.009948e-41
## 7403 4.779123e-14 2.907225e-22
## 7404 6.434284e-15 1.523826e-23
## 7405 2.576609e-16 2.907225e-22
## 7406 2.815881e-16 1.229119e-21
## 7407 7.622537e-17 7.321918e-25
```

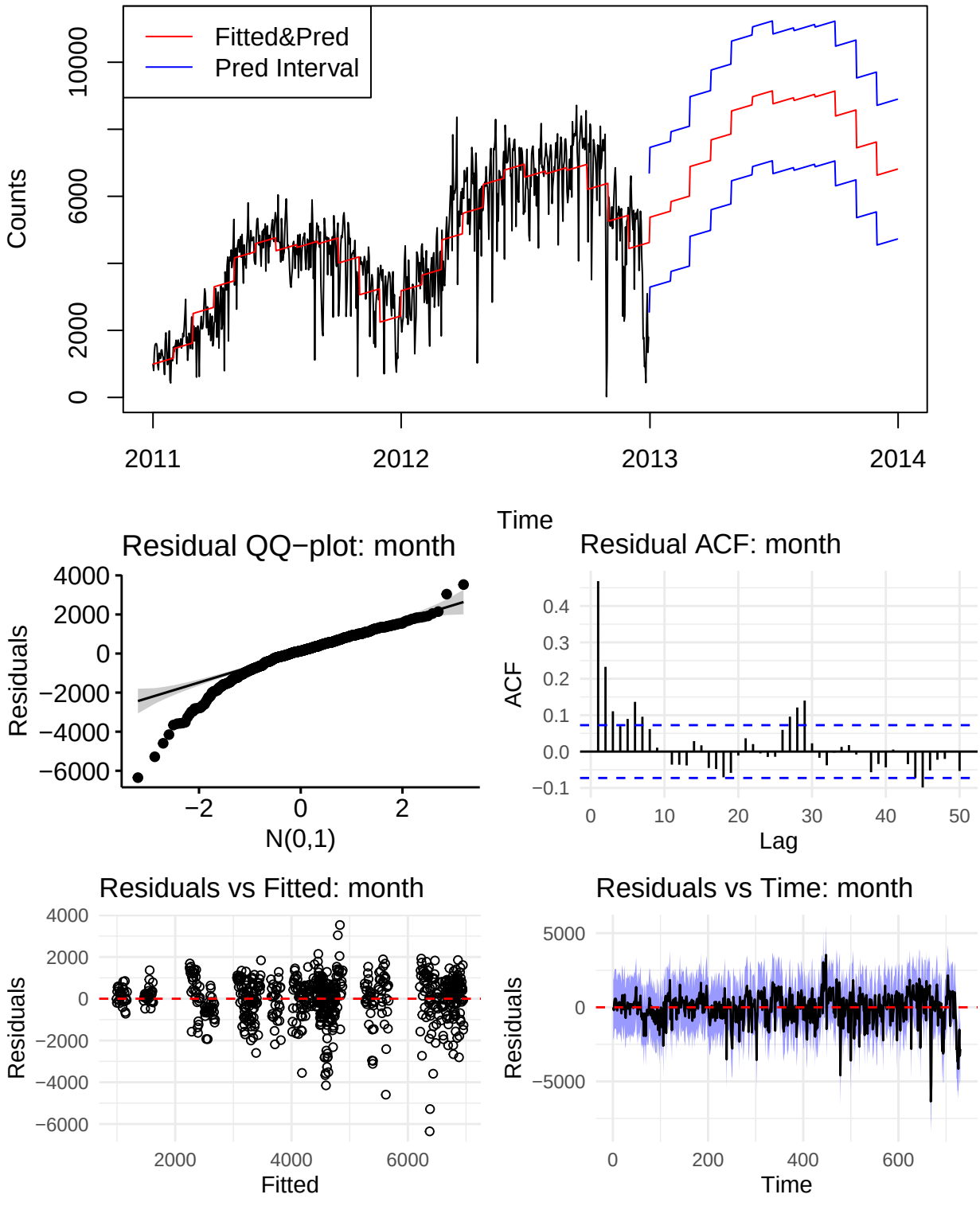
```
# having too many parameters, and the model fit and prediction power are not improving significantly as
# poly6 seems to have the best prediction power, but still violating residual normality, randomness and
# It is difficult, or at least inefficient for polynomials to predict
# Pure polynomial is not a good choice. Alost none of them are capturing the increasing trend
```

Polynomials + monthly seasonality

```
to.month <- function(time) {
  return(format(time, '%B'))
}
```

```r
months <- to.month(pred.dates.full)
months <- factor(months)
#months

data.months <- data.frame(y = y.dummy,
                          poly = t.full,
                          months = months)
# model with monthly effects (for seasonality)
model.months <- lm(y ~ ., data = data.months)
fit.months <- Fit.Model(model.months, 'month')
performances <- rbind(performances, model.summary(fit.months,
                                                  print.summary = TRUE,
                                                  plot = TRUE,
                                                  residual.plot = TRUE))
```

```
## [1] "--------> START SUMMARY of MODEL month"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6352.7  -432.7   133.9   635.4  3530.3
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2754.6831   148.5424  18.545  < 2e-16 ***
## poly                5.9973     0.2119  28.298  < 2e-16 ***
## monthsAugust      444.8519   191.6648   2.321 0.020567 *
## monthsDecember  -2516.9009   197.3992 -12.750  < 2e-16 ***
## monthsFebruary  -1487.4934   194.3276  -7.655 6.28e-14 ***
## monthsJanuary   -1768.8056   190.8539  -9.268  < 2e-16 ***
## monthsJuly        530.0258   190.8859   2.777 0.005635 **
## monthsJune        921.6322   191.8845   4.803 1.90e-06 ***
## monthsMarch      -609.7247   190.0083  -3.209 0.001392 **
## monthsMay         681.9570   190.0083   3.589 0.000354 ***
## monthsNovember  -1521.1358   196.7475  -7.731 3.60e-14 ***
## monthsOctober    -386.1761   193.8397  -1.992 0.046723 *
## monthsSeptember   364.0320   194.1752   1.875 0.061232 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1049 on 717 degrees of freedom
## Multiple R-squared:  0.7119, Adjusted R-squared:  0.7071
## F-statistic: 147.6 on 12 and 717 DF,  p-value: < 2.2e-16
```




```
## [1] "Summary:"
##     Model   ResErr  PredErr     AICc      BIC     adjR2    APSE    Normality
## 740 month 1048.607 2083.914 12241.74 12305.45 0.7070615 1126889 1.088674e-18
```

```
##       Randomness
## 740 6.728629e-23
## [1] "--------> END SUMMARY of MODEL month"
```

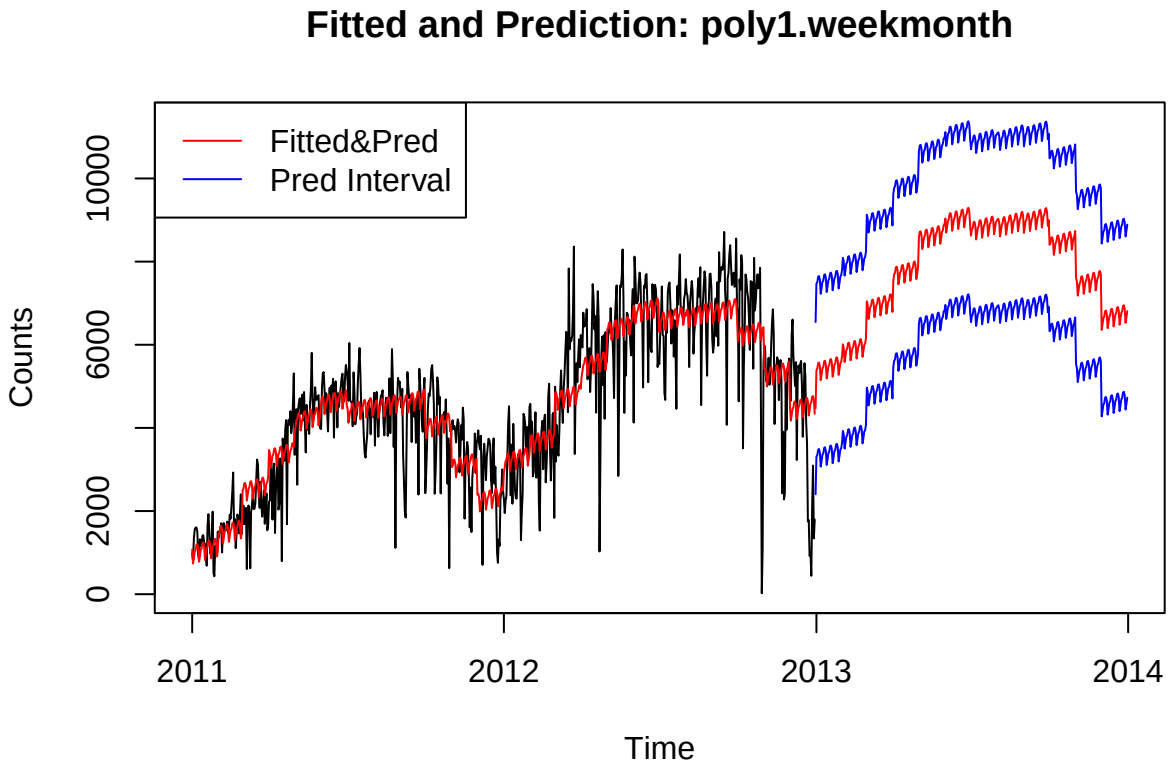
```
#performances[nrow(performances), ]
# poor residuals
```

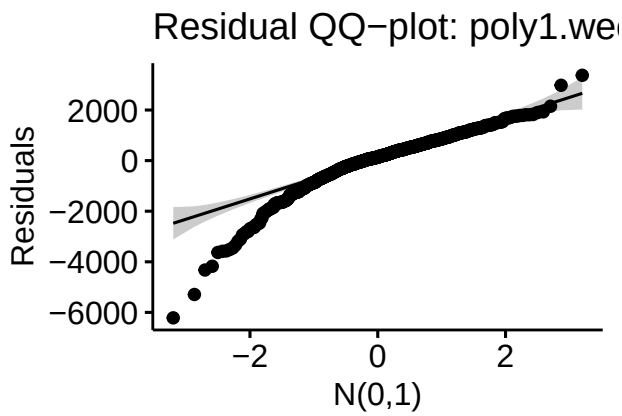
Polynomial + weekly and monthly seasonality

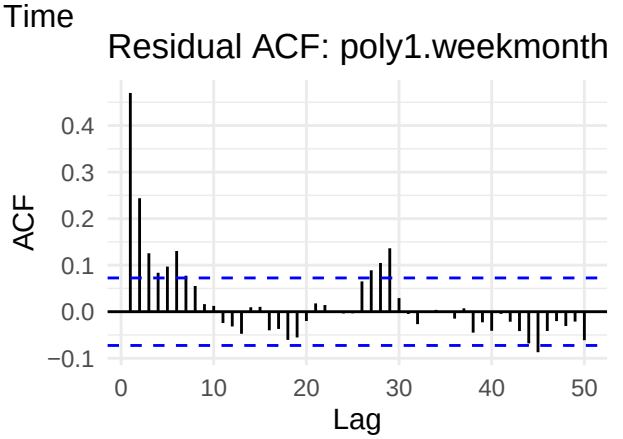
```
weekdays <- factor(t.full %% 7)
data.weekmonth <- data.frame(y = y.dummy,
                        X = t.full,
                        month = months,
                        week = weekdays)
model.weekmonth <- lm(y ~ ., data = data.weekmonth)
fit.weekmonth <- Fit.Model(model.weekmonth, 'poly1.weekmonth')
performances <- rbind(performances, model.summary(fit.weekmonth,
                                                  print.summary = TRUE,
                                                  plot = TRUE,
                                                  residual.plot = TRUE))
```

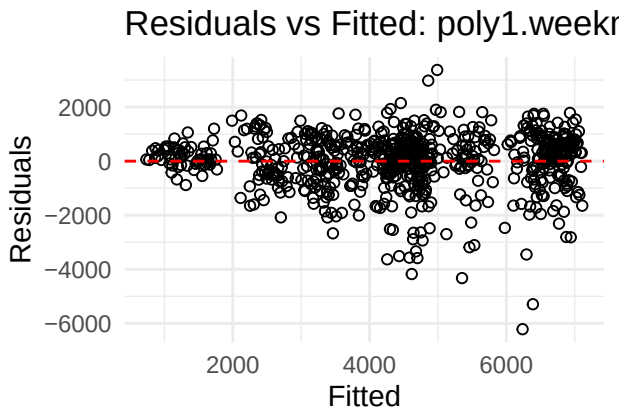
```
## [1] "--------> START SUMMARY of MODEL poly1.weekmonth"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6215.7  -450.0   135.9   632.7  3373.1
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2922.8911   174.9716  16.705  < 2e-16 ***
## X                  5.9997     0.2107  28.479  < 2e-16 ***
## monthAugust      437.8358   190.5768   2.297 0.021884 *
## monthDecember  -2523.5976   196.2450 -12.859  < 2e-16 ***
## monthFebruary  -1490.8531   193.1984  -7.717 4.05e-14 ***
## monthJanuary   -1759.2633   189.7415  -9.272  < 2e-16 ***
## monthJuly        534.1092   189.7579   2.815 0.005018 **
## monthJune        911.5014   190.7717   4.778 2.15e-06 ***
## monthMarch      -621.8757   188.9305  -3.292 0.001046 **
## monthMay         682.6947   188.9305   3.613 0.000323 ***
## monthNovember  -1530.4535   195.6195  -7.824 1.86e-14 ***
## monthOctober    -382.1524   192.7092  -1.983 0.047746 *
## monthSeptember   358.6839   193.0331   1.858 0.063561 .
## week1            -94.8902   144.2324  -0.658 0.510817
## week2           -438.7138   144.2658  -3.041 0.002445 **
## week3           -310.8242   144.6281  -2.149 0.031961 *
## week4           -166.2160   144.6056  -1.149 0.250760
## week5           -138.1960   144.5947  -0.956 0.339525
## week6            -12.4782   144.5715  -0.086 0.931243
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1042 on 711 degrees of freedom
```

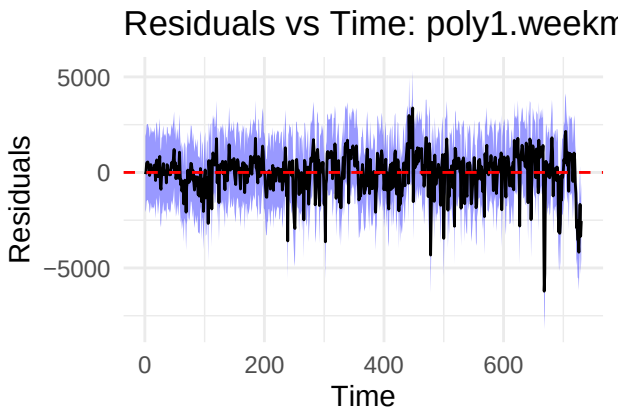
```
## Multiple R-squared:  0.7177, Adjusted R-squared:  0.7105
## F-statistic: 100.4 on 18 and 711 DF,  p-value: < 2.2e-16
```












```
## [1] "Summary:"
```

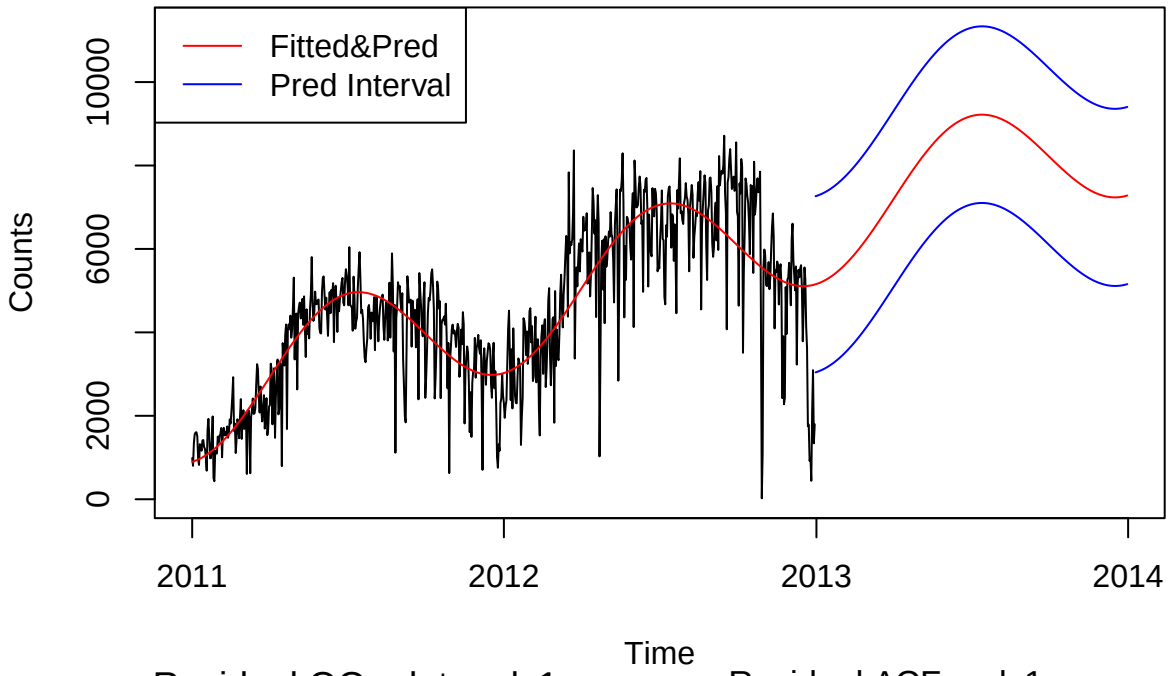
```
##               Model   ResErr  PredErr     AICc      BIC    adjR2    APSE
## 740 poly1.weekmonth 1042.357 2080.063 12239.47 12330.15 0.710543 1142921
##        Normality   Randomness
## 740 6.263575e-19 1.553463e-25
## [1] "--------> END SUMMARY of MODEL poly1.weekmonth"
```

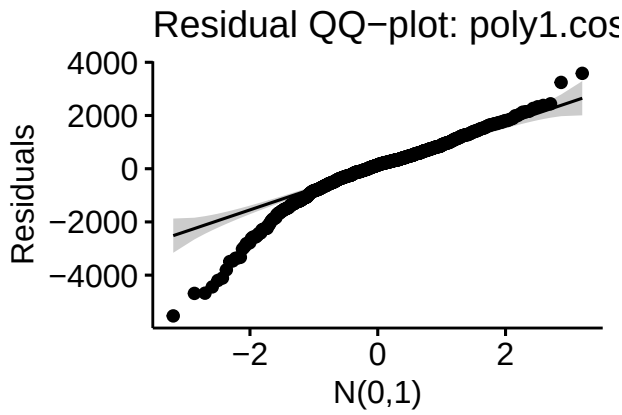
```
#performances[nrow(performances), ]
# weekdays are not significant, but some months are
```

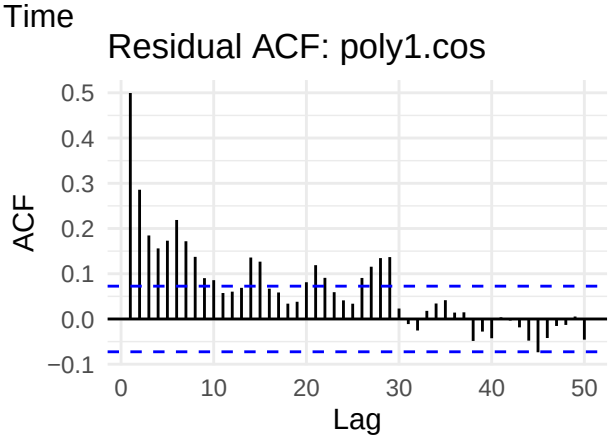
Polynomials + annual seasonality

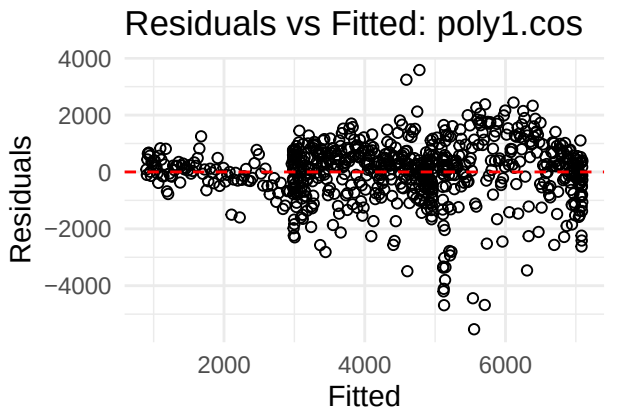
```
# consider a cos function with period 365
data.cos <- data.frame(y = y.dummy,
                       X = t.full,
                       season = cos((2*pi/365) * t.full))
model.cos <- lm(y ~ ., data = data.cos)
fit.cos <- Fit.Model(model.cos, 'poly1.cos')
performances <- rbind(performances, model.summary(fit.cos,
                                                  print.summary = TRUE,
                                                  plot = TRUE,
                                                  residual.plot = TRUE))
```

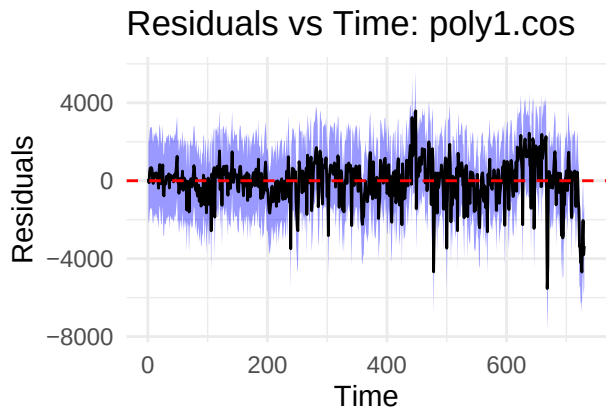
```
## [1] "--------> START SUMMARY of MODEL poly1.cos"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5531.8  -477.6   138.8   610.8  3584.4
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2376.1751    79.4886   29.89   <2e-16 ***
## X               5.8293     0.1884   30.94   <2e-16 ***
## season      -1484.6492    56.1492  -26.44   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1073 on 727 degrees of freedom
## Multiple R-squared:  0.6943, Adjusted R-squared:  0.6934
## F-statistic: 825.5 on 2 and 727 DF,  p-value: < 2.2e-16
```









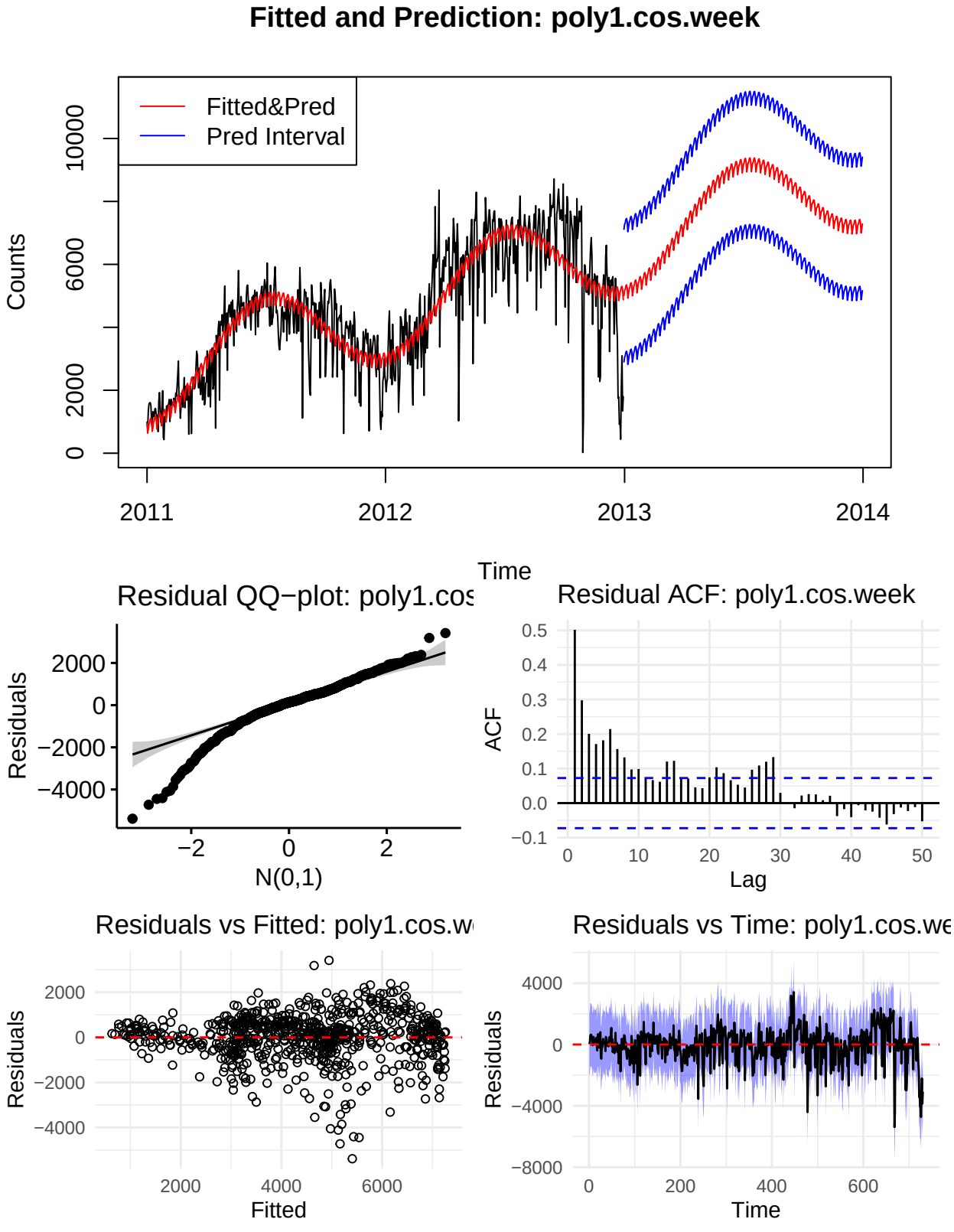



```
## [1] "Summary:"
##         Model   ResErr  PredErr     AICc      BIC     adjR2    APSE
## 740 poly1.cos 1072.724 2111.242 12264.52 12282.83 0.6934318 1154837
```

```
##        Normality   Randomness
## 740 1.377683e-16 2.907225e-22
## [1] "--------> END SUMMARY of MODEL poly1.cos"
```

```
#performances[nrow(performances), ]
data.cos.week <- data.frame(y = y.dummy,
                            X = t.full,
                            cos = cos((2*pi/365) * t.full),
                            week = weekdays)
model.cos.week <- lm(y ~ ., data = data.cos.week)
fit.cos.week <- Fit.Model(model.cos.week, 'poly1.cos.week')
performances <- rbind(performances, model.summary(fit.cos.week,
                                                  print.summary = TRUE,
                                                  plot = TRUE,
                                                  residual.plot = TRUE))
```

```
## [1] "--------> START SUMMARY of MODEL poly1.cos.week"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5385.9  -431.0   136.0   588.6  3416.7
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2544.6659   125.2655  20.314  < 2e-16 ***
## X               5.8273     0.1874  31.094  < 2e-16 ***
## cos         -1484.0593    55.8526 -26.571  < 2e-16 ***
## week1        -111.0030   147.6204  -0.752  0.45233
## week2        -438.5462   147.6199  -2.971  0.00307 **
## week3        -313.3854   147.9733  -2.118  0.03453 *
## week4        -162.1466   147.9725  -1.096  0.27354
## week5        -130.0989   147.9719  -0.879  0.37958
## week6         -17.2039   147.9715  -0.116  0.90747
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1067 on 721 degrees of freedom
## Multiple R-squared:    0.7,  Adjusted R-squared:  0.6967
## F-statistic: 210.3 on 8 and 721 DF,  p-value: < 2.2e-16
```

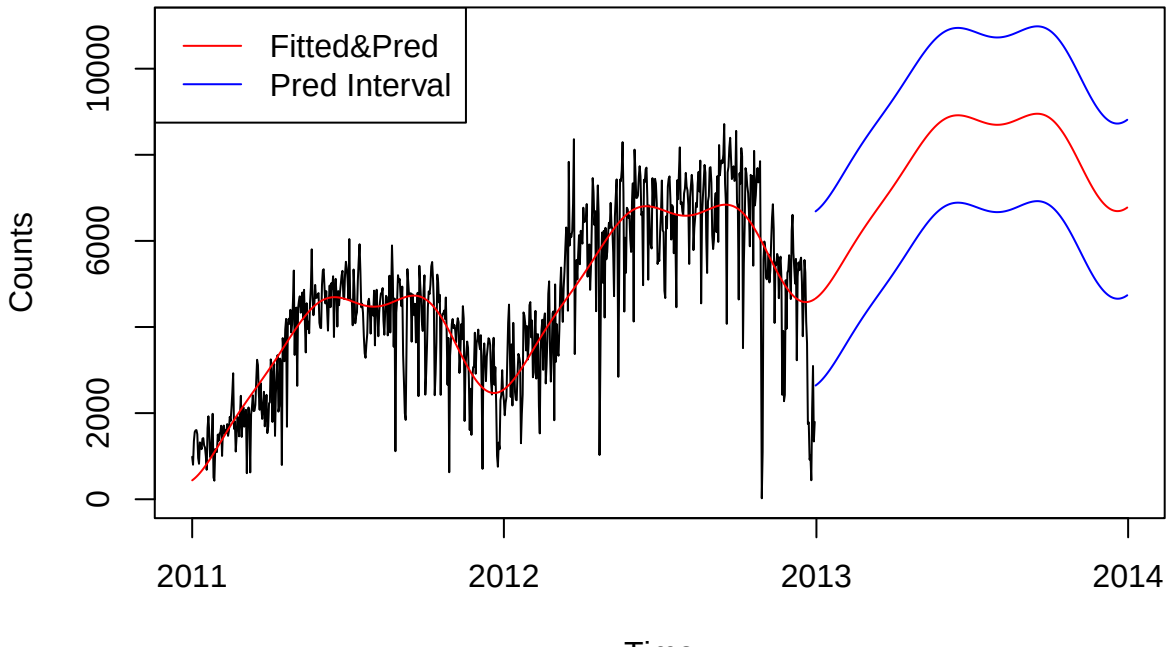


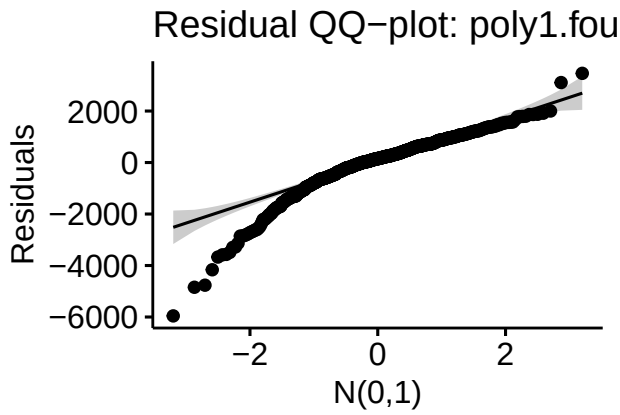

```
## [1] "Summary:"
##              Model   ResErr  PredErr     AICc      BIC     adjR2    APSE
## 740 poly1.cos.week 1067.037 2108.634 12262.96 12308.58 0.6966736 1157433
```

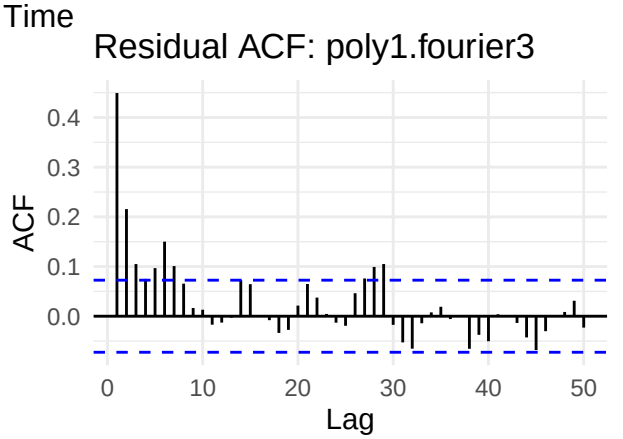
```
##        Normality   Randomness
## 740 1.334117e-16 6.728629e-23
## [1] "--------> END SUMMARY of MODEL poly1.cos.week"
```

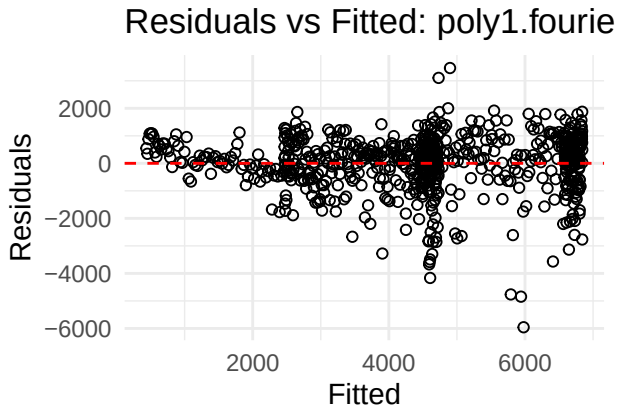
```
# try fourier series with degree 3
data.fourier <- data.frame(y = y.dummy,
                           X = t.full,
                           season = fourier(ts(t.full, frequency = 365), K = 3))
model.fourier <- lm(y ~ ., data = data.fourier)
fit.fourier <- Fit.Model(model.fourier, 'poly1.fourier3')
performances <- rbind(performances, model.summary(fit.fourier,
                                                  print.summary = TRUE,
                                                  plot = TRUE,
                                                  residual.plot = TRUE))
```

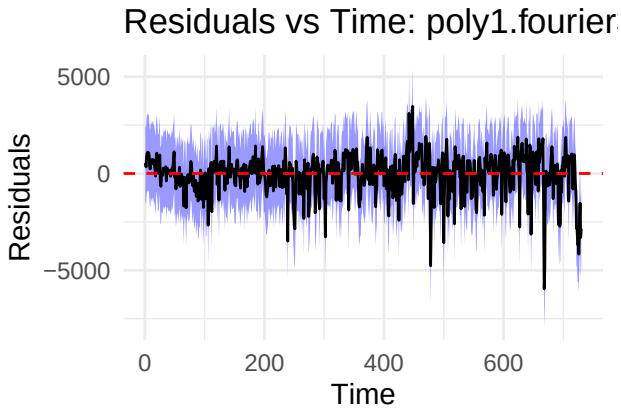
```
## [1] "--------> START SUMMARY of MODEL poly1.fourier3"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5957.4  -459.6   153.4   638.4  3463.6
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2391.748     83.004  28.815  < 2e-16 ***
## X                 5.787      0.202  28.644  < 2e-16 ***
## season.S1.365   -71.701     58.530  -1.225   0.2210
## season.C1.365 -1484.607     53.619 -27.688  < 2e-16 ***
## season.S2.365   -73.323     54.887  -1.336   0.1820
## season.C2.365  -388.670     53.619  -7.249 1.08e-12 ***
## season.S3.365   214.962     54.186   3.967 8.00e-05 ***
## season.C3.365   -93.199     53.619  -1.738   0.0826 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1024 on 722 degrees of freedom
## Multiple R-squared:  0.7231, Adjusted R-squared:  0.7204
## F-statistic: 269.4 on 7 and 722 DF,  p-value: < 2.2e-16
```









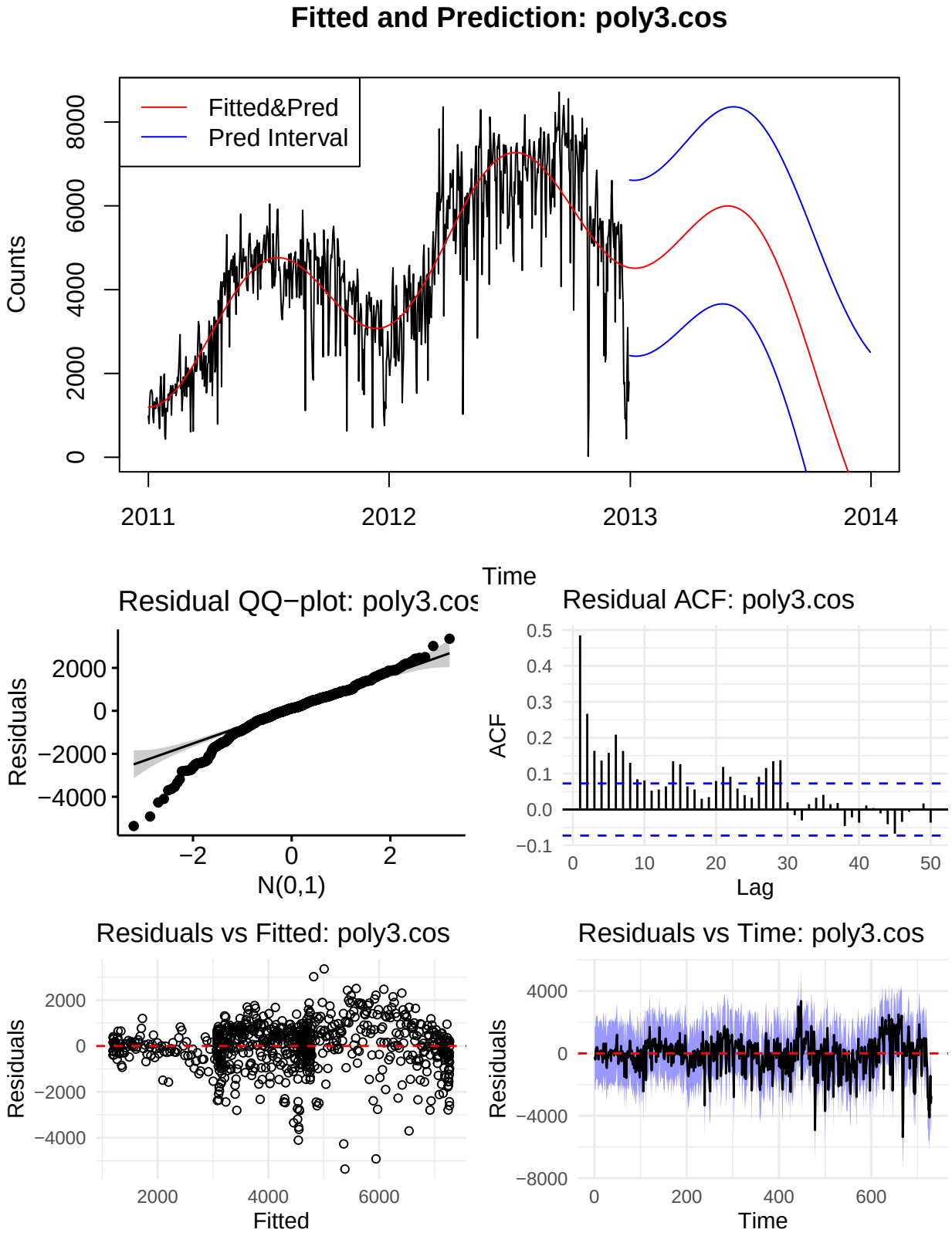



```
## [1] "Summary:"
##              Model   ResErr PredErr     AICc      BIC     adjR2    APSE
## 740 poly1.fourier3 1024.378 2024.53 12202.34 12243.43 0.7204423 1058038
```

```
##        Normality  Randomness
## 740 2.042864e-19 7.19282e-24
## [1] "--------> END SUMMARY of MODEL poly1.fourier3"
```

```
#performances[nrow(performances), ]


# try adding polynomial degrees
data.cos <- data.frame(y = y.dummy,
                       X = poly(t.full, degree = 3, raw = FALSE),
                       season = cos((2*pi/365) * t.full))
model.cos <- lm(y ~ ., data = data.cos)
fit.cos <- Fit.Model(model.cos, 'poly3.cos')
performances <- rbind(performances, model.summary(fit.cos,
                                                  print.summary = TRUE,
                                                  plot = TRUE,
                                                  residual.plot = TRUE))
```

```
## [1] "--------> START SUMMARY of MODEL poly3.cos"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5360.6  -450.4   117.1   639.1  3355.0
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4273.91     268.24  15.933  < 2e-16 ***
## X.1           2681.94   11999.94   0.223    0.823
## X.2         -45174.86    9164.19  -4.929 1.02e-06 ***
## X.3         -19609.75    4380.79  -4.476 8.82e-06 ***
## season       -1454.57      56.97 -25.534  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1056 on 725 degrees of freedom
## Multiple R-squared:  0.7043, Adjusted R-squared:  0.7027
## F-statistic: 431.7 on 4 and 725 DF,  p-value: < 2.2e-16
```

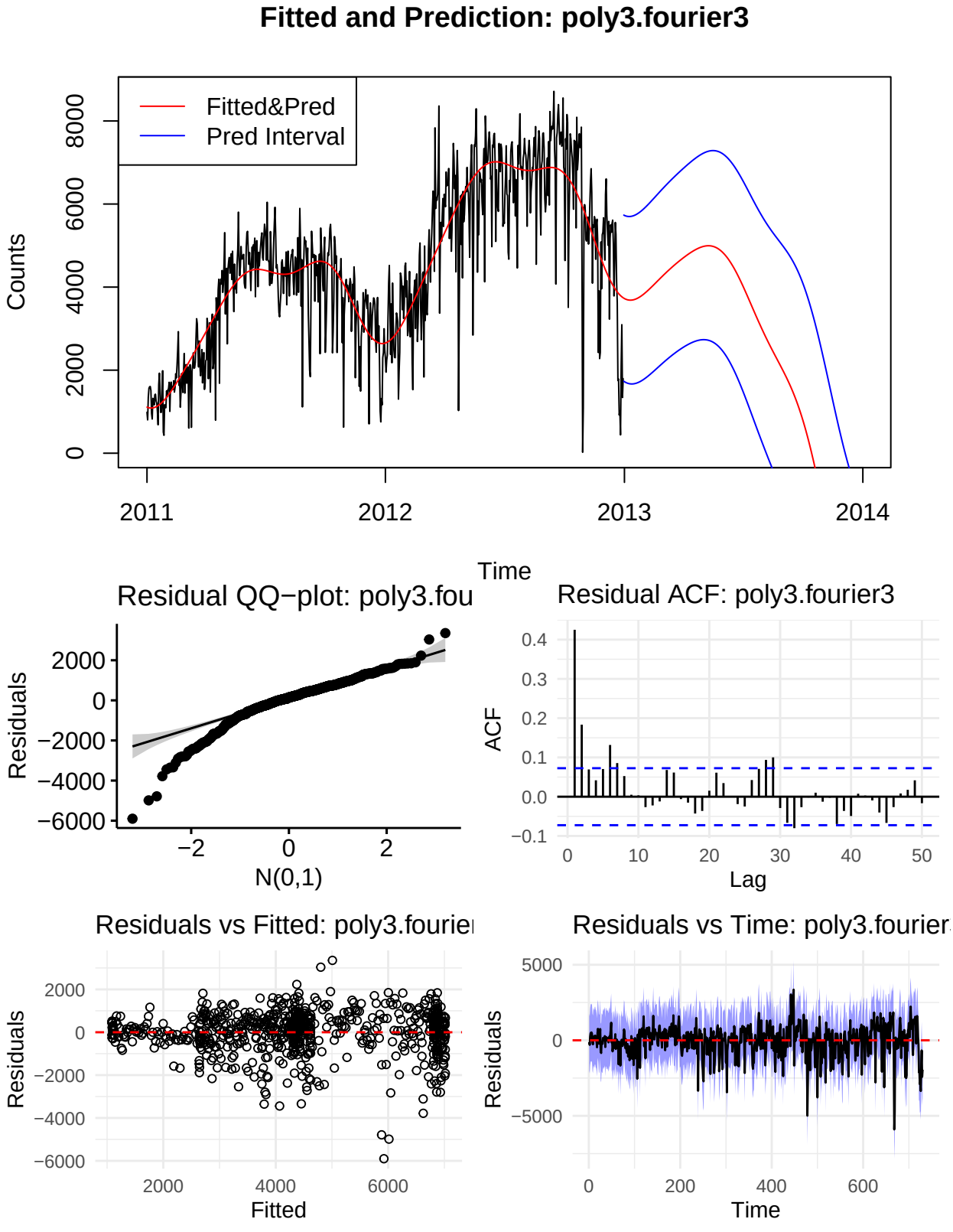



```
## [1] "Summary:"
##         Model   ResErr  PredErr     AICc      BIC     adjR2    APSE   Normality
## 740 poly3.cos 1056.427 2098.866 12244.22 12271.66 0.7026757 1123724 6.36322e-15
```

```
##       Randomness
## 740 2.907225e-22
## [1] "--------> END SUMMARY of MODEL poly3.cos"
```

```
#performances[nrow(performances), ]

# poly 3, fourier 3
data.poly3fourier <- data.frame(y = y.dummy,
                           X = poly(t.full, degree = 3, raw = FALSE),
                           season = fourier(ts(t.full, frequency = 365), K = 3))
model.poly3fourier <- lm(y ~ ., data = data.poly3fourier)
fit.poly3fourier <- Fit.Model(model.poly3fourier, 'poly3.fourier3')
performances <- rbind(performances, model.summary(fit.poly3fourier,
                                                   print.summary = TRUE,
                                                   plot = TRUE,
                                                   residual.plot = TRUE))
```

```
## [1] "--------> START SUMMARY of MODEL poly3.fourier3"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5902.0  -399.8   141.5   614.1  3353.5
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     3738.55     308.73  12.110  < 2e-16 ***
## X.1           -22477.76   13997.91  -1.606 0.108758
## X.2           -61565.85   10255.66  -6.003 3.07e-09 ***
## X.3           -28634.19    4985.89  -5.743 1.37e-08 ***
## season.S1.365   -244.26      64.63  -3.780 0.000170 ***
## season.C1.365  -1460.37      53.99 -27.050  < 2e-16 ***
## season.S2.365   -190.26      57.38  -3.316 0.000959 ***
## season.C2.365   -380.97      52.50  -7.257 1.03e-12 ***
## season.S3.365    133.23      54.83   2.430 0.015356 *
## season.C3.365    -88.56      52.42  -1.689 0.091564 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1001 on 720 degrees of freedom
## Multiple R-squared:  0.7363, Adjusted R-squared:  0.733
## F-statistic: 223.4 on 9 and 720 DF,  p-value: < 2.2e-16
```




```
## [1] "Summary:"
##              Model   ResErr  PredErr     AICc      BIC     adjR2    APSE
## 740 poly3.fourier3 1001.049 2013.981 12170.8 12220.96 0.7330305 1014262
```

```
##        Normality   Randomness
## 740 1.747467e-18 8.153608e-20
## [1] "--------> END SUMMARY of MODEL poly3.fourier3"
```

```
#performances[nrow(performances), ]
```

Summary:

```
# model performances
performances
```

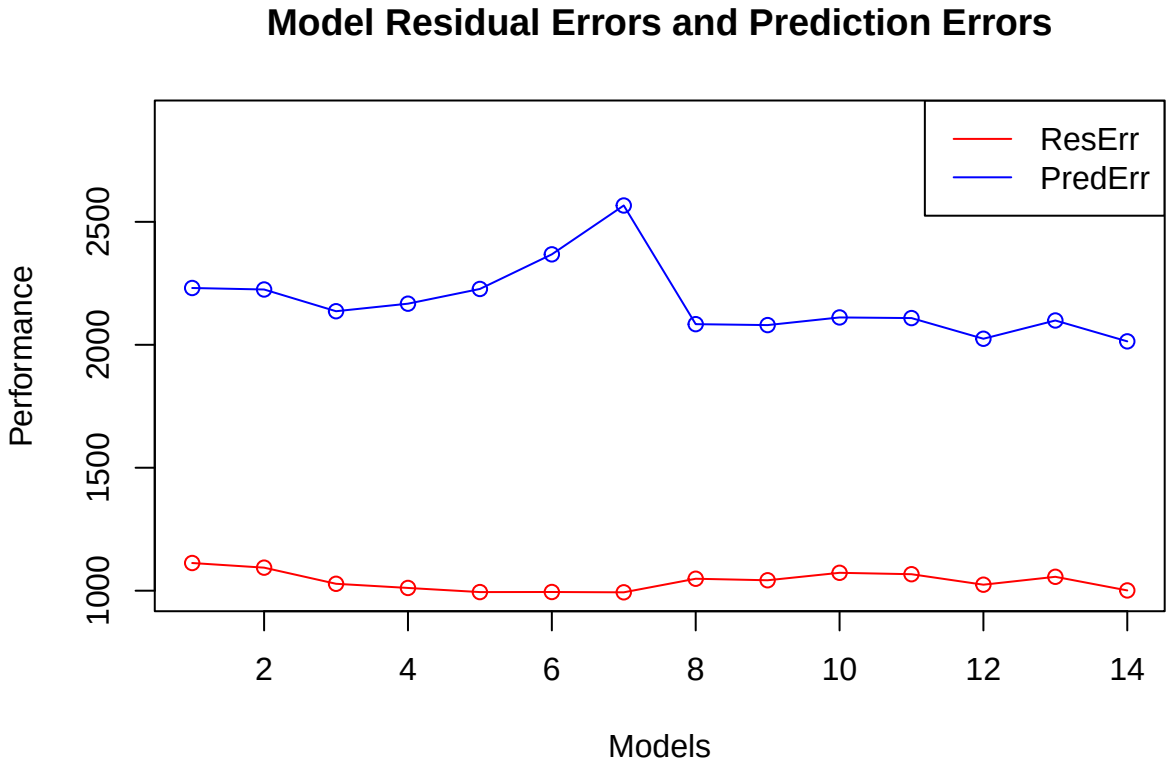
```
##                Model    ResErr  PredErr     AICc      BIC     adjR2      APSE
## 7401           poly4 1112.5934 2230.952 12319.84 12347.29 0.6702201 1251259.4
## 7402           poly5 1093.6339 2224.864 12295.78 12327.78 0.6813637 1207510.1
## 7403           poly6 1028.0707 2136.464 12206.56 12243.10 0.7184230 1067152.5
## 7404           poly7 1011.1478 2167.197 12183.36 12224.45 0.7276167 1034614.1
## 7405           poly8  994.3370 2226.979 12159.93 12205.56 0.7365984  999267.2
## 7406           poly9  994.6156 2367.755 12161.39 12211.54 0.7364508 1007148.8
## 7407          poly10  993.4027 2566.486 12160.66 12215.34 0.7370932 1003197.3
## 740            month 1048.6067 2083.914 12241.74 12305.45 0.7070615 1126889.3
## 7408  poly1.weekmonth 1042.3567 2080.063 12239.47 12330.15 0.7105430 1142920.9
## 7409        poly1.cos 1072.7238 2111.242 12264.52 12282.83 0.6934318 1154837.3
## 74010  poly1.cos.week 1067.0371 2108.634 12262.96 12308.58 0.6966736 1157432.9
## 74011  poly1.fourier3 1024.3777 2024.530 12202.34 12243.43 0.7204423 1058038.5
## 74012       poly3.cos 1056.4272 2098.866 12244.22 12271.66 0.7026757 1123723.6
## 74013  poly3.fourier3 1001.0486 2013.981 12170.80 12220.96 0.7330305 1014262.4
##          Normality   Randomness
## 7401  3.640645e-12 7.531584e-39
## 7402  2.217222e-12 2.009948e-41
## 7403  4.779123e-14 2.907225e-22
## 7404  6.434284e-15 1.523826e-23
## 7405  2.576609e-16 2.907225e-22
## 7406  2.815881e-16 1.229119e-21
## 7407  7.622537e-17 7.321918e-25
## 740   1.088674e-18 6.728629e-23
## 7408  6.263575e-19 1.553463e-25
## 7409  1.377683e-16 2.907225e-22
## 74010 1.334117e-16 6.728629e-23
## 74011 2.042864e-19 7.192820e-24
## 74012 6.363220e-15 2.907225e-22
## 74013 1.747467e-18 8.153608e-20
```

```
models <- as.factor(as.vector(performances[, 'Model']))
models <- data.frame(index = 1:length(models), name = models)
models
```

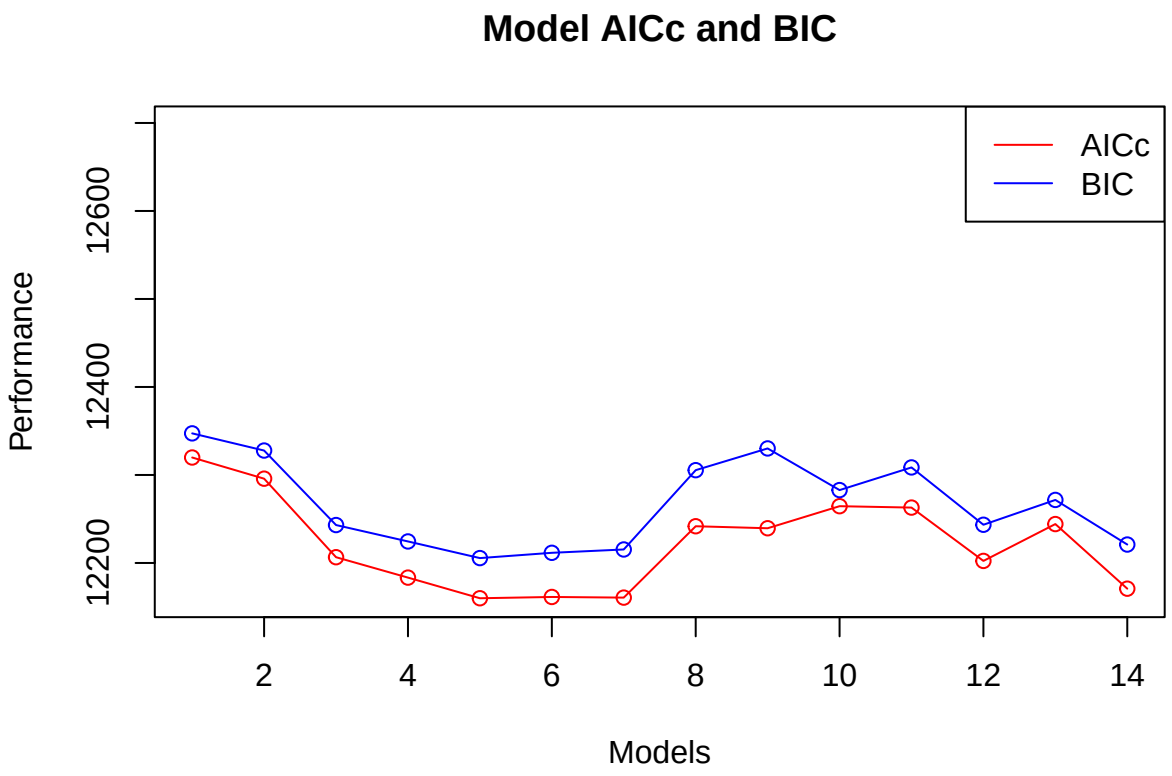
```
##    index            name
## 1      1           poly4
## 2      2           poly5
## 3      3           poly6
## 4      4           poly7
## 5      5           poly8
## 6      6           poly9
## 7      7          poly10
## 8      8           month
## 9      9 poly1.weekmonth
```

```
## 10    10       poly1.cos
## 11    11  poly1.cos.week
## 12    12  poly1.fourier3
## 13    13       poly3.cos
## 14    14  poly3.fourier3
```

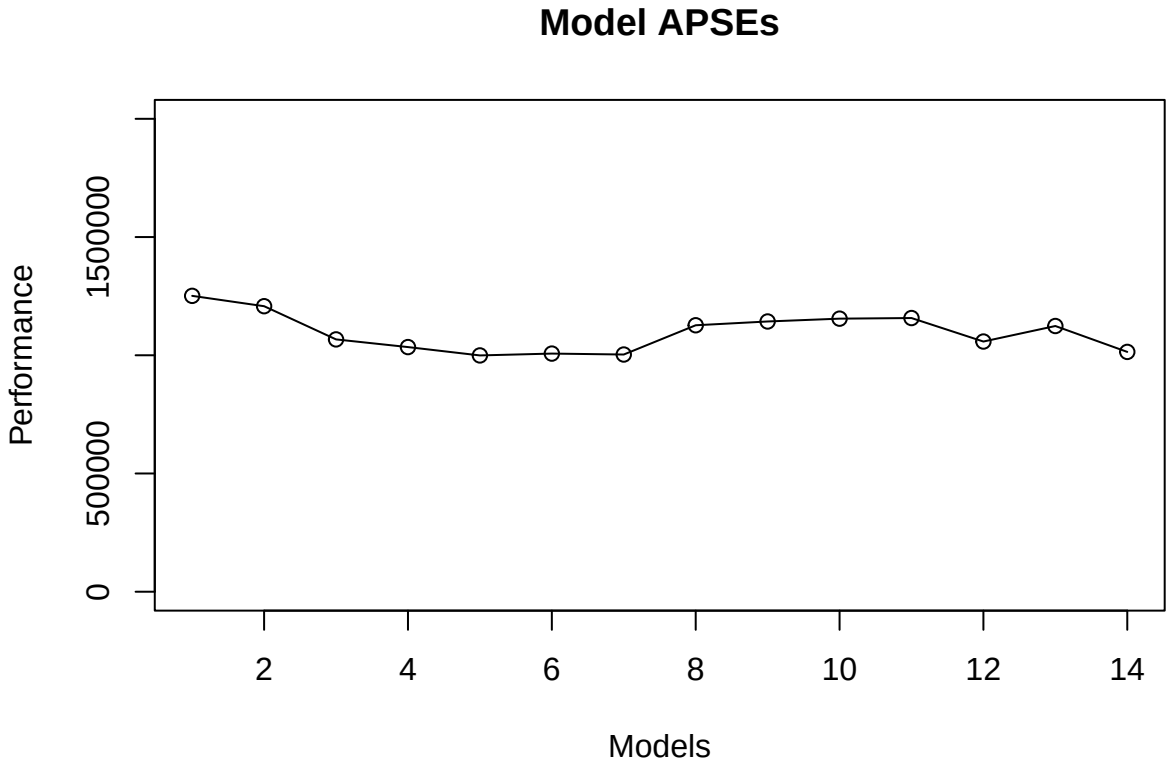
```
# plot performances
# residual and prediction errors
ResErr <- as.vector(performances[, 'ResErr'])
PredErr <- as.vector(performances[, 'PredErr'])
plot(models$index, ResErr, type = 'o', col = 'red',
     main = 'Model Residual Errors and Prediction Errors',
     xlab = 'Models', ylab = 'Performance',
     ylim = c(min(c(ResErr, PredErr)),
              max(c(ResErr, PredErr))+350))
lines(models$index, PredErr, type = 'o', col = 'blue')

legend('topright', legend = c('ResErr', 'PredErr'), col = c('red', 'blue'), lty = 1)
```



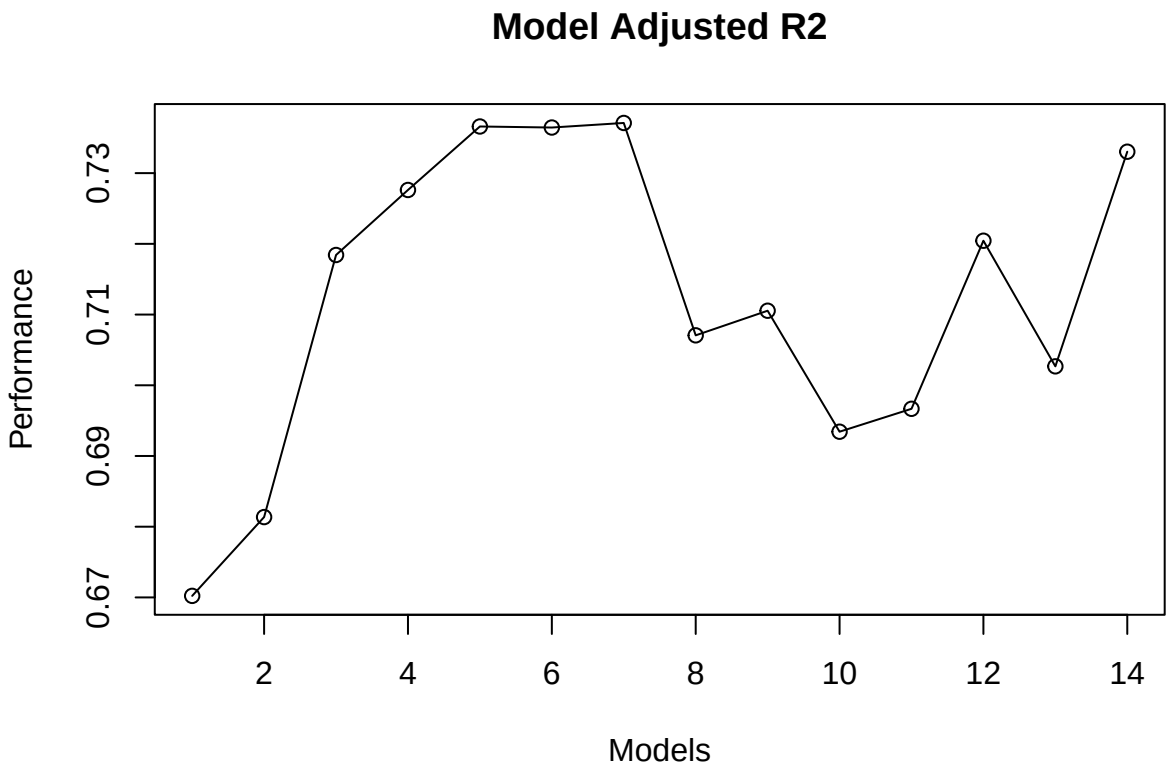

```
# AICc and BIC
aicc <- as.vector(performances[, 'AICc'])
bic <- as.vector(performances[, 'BIC'])
plot(models$index, aicc, type = 'o', col = 'red',
     main = 'Model AICc and BIC',
     xlab = 'Models', ylab = 'Performance',
     ylim = c(min(c(aicc, bic)), max(c(aicc, bic))+350))
lines(models$index, bic, type = 'o', col = 'blue',
     main = 'Model Residual Errors and Prediction Errors',
     xlab = 'Models', ylab = 'Performance')
legend('topright', legend = c('AICc', 'BIC'), col = c('red', 'blue'), lty = 1)
```




```
apses <- as.vector(performances[, 'APSE'])
plot(models$index, apses, type = 'o',
     main = 'Model APSEs',
     xlab = 'Models', ylab = 'Performance', ylim = c(0, 2e6))
```



```r
# adj R2
adjR2 <- as.vector(performances[, 'adjR2'])
plot(models$index, adjR2, type = 'o', xlab = 'Models', ylab = 'Performance',
     main = 'Model Adjusted R2')
```



Model 10: poly1cos is a relatively good model.

```r
# write data
#write.csv(performances, 'model_comparison.csv', row.names = FALSE)
```

Model without outliers:

```r
# indices of non-outliers
no.outlier.indices <- function(y, threshold) {
  y <- as.vector(y)
  n <- length(y)
  w <- numeric(n)
  w[1] <- abs(y[1]-y[2])
  w[n] <- abs(y[n]-y[n-1])
  for(i in 2:(n-1)) {
    # average abs diff between its two adjacent neighbors
    diff <- mean(abs(y[i]-y[i-1]), abs(y[i]-y[i+1]))
    w[i] <- diff
  }
  return(as.logical(w < threshold))
}

recover.boxcox <- function(bc.data, lambda) {
  if(lambda == 0) {
    return(exp(data))
  } else {
```

```
    return((lambda * bc.data + 1)^(1/lambda))
  }
}

# run k-fold cross-validation with boxcox
APSE.bc <- function(model, dataset, lambda) {
  # shuffle training data
  dataset <- dataset[sample(1:nrow(dataset)),]
  # prediction error for each test group
  apses <- c()
  for(i in 1:apse.k) {
    # indices for testing group
    testing_idx <- (i - 1) * apse.size + 1:apse.size
    # training data
    data.train <- dataset[-testing_idx,]
    data.test <- dataset[testing_idx,]
    #new.model <- update(model, data = data.train)
    new.model <- lm(bikes ~ ., data = data.train)
    # recover to original scale
    pred <- recover.boxcox(predict(new.model, newdata=data.test), lambda)
    data.val <- recover.boxcox(data.test$bikes, lambda)
    apse <- mean((data.val - pred)^2)
    print(apse)
    apses <- c(apses, apse)
  }
  return(mean(apses))
}
```

```
# super function for model with/without outliers, boxcox, or weights
model.analysis <- function(data, name, train.idx = FALSE, y.weights = FALSE,
                           box.cox.lambda = FALSE,
                           print.summary=FALSE,
                           plot=FALSE,
                           residual.plot=FALSE) {
  # extract training and prediction data
  data.train <- data.frame(bikes = Bike, X = data[bike.data$t, ])
  data.pred <- data.frame(X = data[t.pred, ])



  # for matrix
  matrix.train <- cbind(rep(1, nrow(data.train)), as.matrix(data.train[, -1]))
  matrix.pred <- cbind(rep(1, nrow(data.pred)), as.matrix(data.pred))
  X.T.X.inv <- solve(crossprod(matrix.train))



  # remove outliers
  # train.idx is the index of non-outlier data points
  dates.plot <- dates
  if(!isFALSE(train.idx)) {
    data.train <- data.train[train.idx, ]
    dates.plot <- dates.plot[train.idx]
    matrix.train <- matrix.train[train.idx,, drop=FALSE]
```

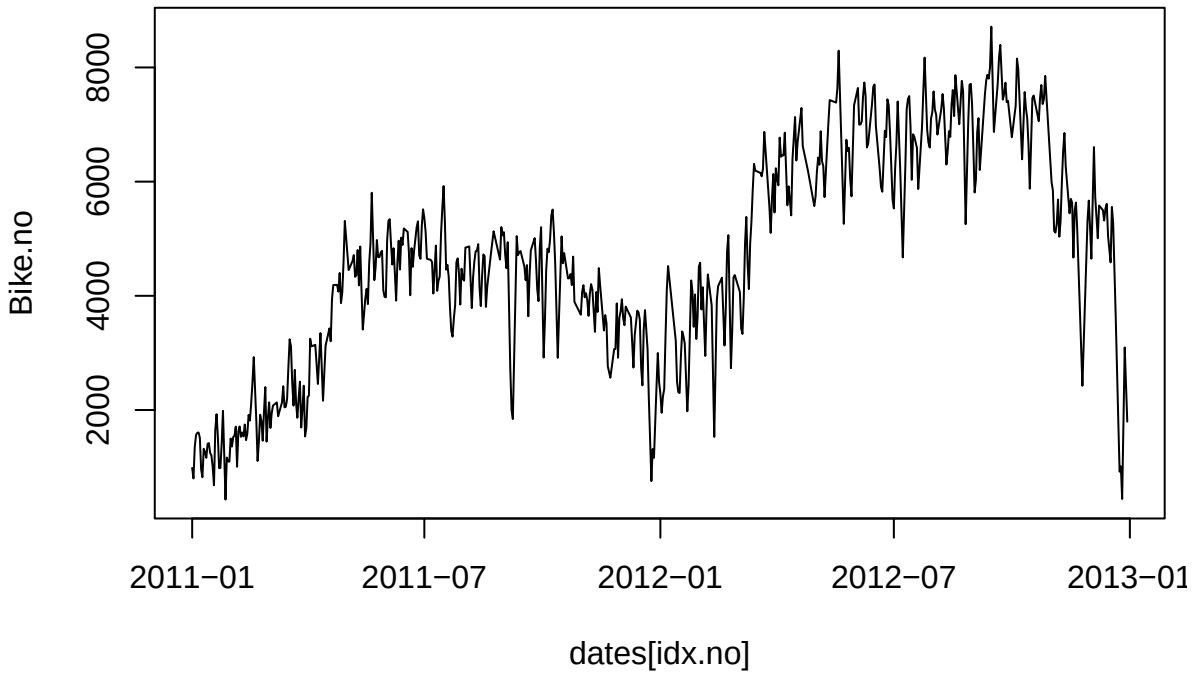
```
  X.T.X.inv <- solve(crossprod(matrix.train))
  if(!isFALSE(y.weights)) {
    y.weights <- y.weights[train.idx]
  }
}

lambda.store <- list(lambda = FALSE)
# train model
if(!isFALSE(box.cox.lambda)) {
  boxcox.modeled.data <- MASS::boxcox(model, lambda = box.cox.lambda,
                                      plotit = FALSE)
  # find optimal lambda
  lambda <- boxcox.modeled.data$x[which.max(boxcox.modeled.data$y)]
  print(lambda)
  if(lambda == 0) {
    data.train$bikes <- log(data.train$bikes)
  } else {
    data.train$bikes <- (data.train$bikes^lambda - 1) / lambda
  }
  lambda.store$lambda <- lambda
  }

model <- lm(bikes ~ ., data = data.train)
if(!isFALSE(y.weights)) {
  model <- lm(bikes ~ ., data = data.train, weights = y.weights)
}



# extract results
# prediction over the whole period (also fills in the gaps of outliers)
fit.val <- predict(model, newdata = data.train)
pred.val <- predict(model, newdata = data.pred)

# residual std. dev.
resi.sd <- sqrt(sum(model$residuals^2) / model$df.residual)
CI.wing <- CI.wing.vec(X.T.X.inv, resi.sd, matrix.pred)
pred.err <- CI.wing[10]
apse <- APSE(model, data.train, y.weights)
# if(!isFALSE(box.cox.lambda)) {
#   apse <- APSE.bc(model, data.train, lambda.store$lambda)
# }

summ <- summary(model)
normality <- shapiro.test(model$residuals)$p.value
randomness <- randtests::runs.test(model$residuals)$p.value


# plots
if(print.summary | plot | residual.plot) {
  print(paste0('--------> START SUMMARY of MODEL', name))
  if(!isFALSE(box.cox.lambda)) {
    print('ALL DATA ARE BOX-COX TRANSFORMED')
```

```
  }
}

if(print.summary) {
  print(summ)
  if(!isFALSE(box.cox.lambda)) {
    if(length(box.cox.lambda) > 1) {
      print(paste0('Options of lambdas: ', min(box.cox.lambda), ' ~ ',
                   max(box.cox.lambda)))
    } else {
      print(paste0('lambda: ', box.cox.lambda))
    }
  }
}

if(plot) {
  if(!isFALSE(box.cox.lambda)) {
    print('ALL DATA ARE BOX-COX TRANSFORMED')
  }

  CI.upper <- pred.val + CI.wing
  CI.lower <- pred.val - CI.wing
  plot(dates.plot, data.train$bikes, type ='l', xlim = plot.range,
       main = paste0('Fitted and Prediction: ', name),
       xlab = 'Time', ylab ='Counts',
       ylim = c(min(c(CI.lower, data.train$bikes)),
                max(c(CI.upper, data.train$bikes))))
  lines(dates.plot, fit.val, col = 'red')
  lines(dates.pred, pred.val, col = 'red')
  lines(dates.pred, CI.upper, col = 'blue')
  lines(dates.pred, CI.lower, col = 'blue')
  legend('topleft', legend = c('Fitted&Pred', 'Pred Interval'),
         col = c('red', 'blue'),
         lty = 1)
}

if(residual.plot) {
  H <- matrix.train %*% X.T.X.inv %*% t(matrix.train)
  # projection matrix
  res.band <- 1.96 * resi.sd * (1- diag(H))
  super.residual.plot(name, model$residuals, model$fitted.values, res.band)
}

results <- data.frame(Model = name,
                ResErr = resi.sd,
                PredErr = pred.err,
                AICc = AICc(model),
                BIC = BIC(model),
                adjR2 = summ$adj.r.squared,
                APSE = apse,
                Normality = normality,
                Randomness = randomness)
if(!isFALSE(box.cox.lambda)) {
```

```
    results$lambda <- lambda.store$lambda
  } else {
    results$lambda <- NA
  }

  if(print.summary) {
    print('Summary:')
    print(results)
  }

  return(results)
}
```

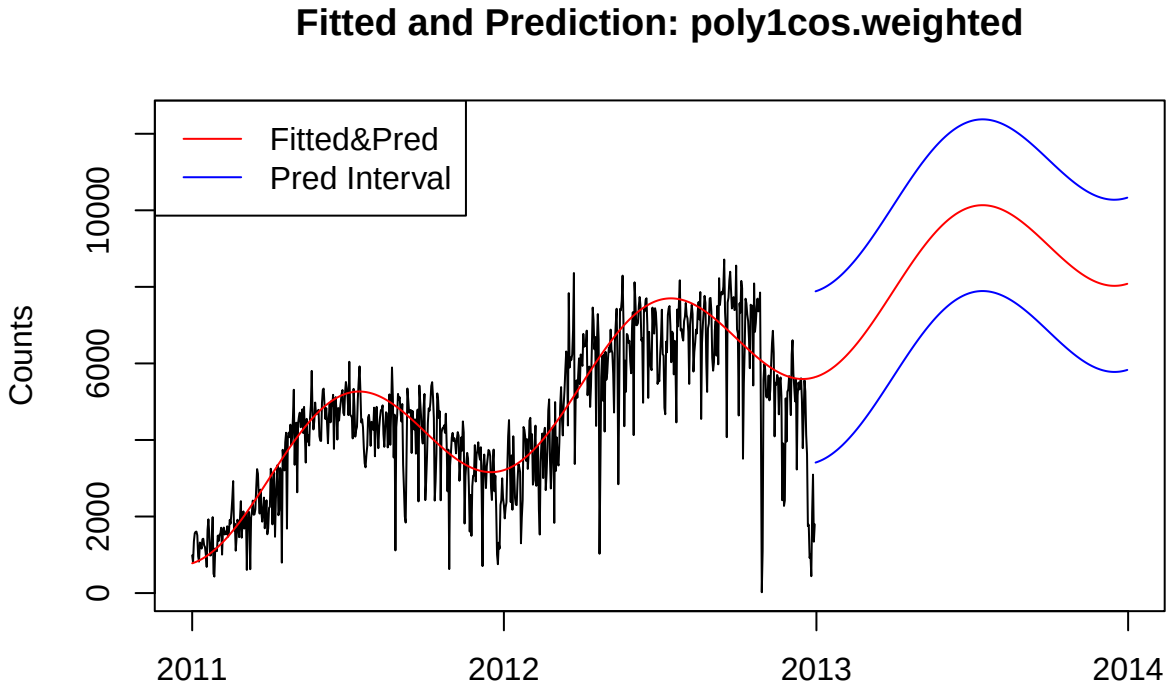
```
# A little try of weighted model:
weighted.data <- data.frame(y=bike.data$bikes, t=bike.data$t,
                            cos=cos(2*pi/365*bike.data$t),
                            weights=bike.weights)
weighted.model <- lm(y ~ t+cos, weights = weights, data=weighted.data)
summary(weighted.model)
plot(bike.data$bikes, type='l', main = 'weighted model')
lines(weighted.model$fitted.values, col='red')

joint.data <- data.frame(y=bike.data$bikes, t=bike.data$t,
                            cos=cos(2*pi/365*bike.data$t),
                            sin=sin(2*pi/365*bike.data$t))
joint.model <- lm(y ~ t+cos+sin+t:cos+t:sin, data=joint.data)
summary(joint.model)
plot(bike.data$bikes, type='l', main = 'weighted model')
lines(joint.model$fitted.values, col='red')

data.joint.fourier <- data.frame(y = y.dummy,
                           X = poly(t.full, degree = 3, raw = FALSE),
                           season = fourier(ts(t.full, frequency = 365), K = 3))
```

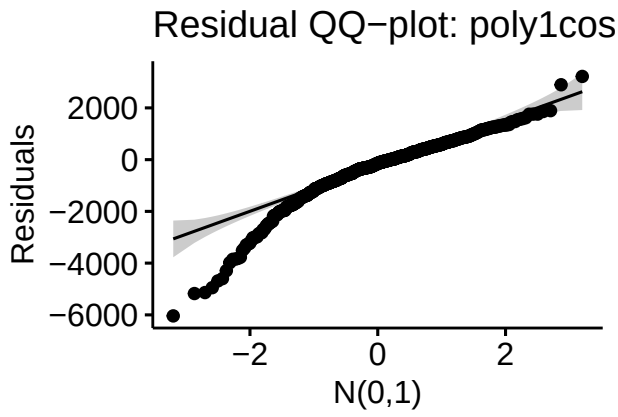
```
# data without outliers
idx.no <- no.outlier.indices(Bike, 1000)
Bike.no <- Bike[idx.no]
plot(dates[idx.no], Bike.no, type = 'l')
```

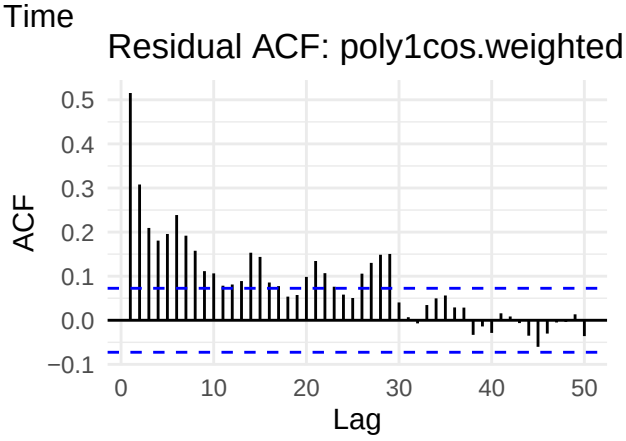


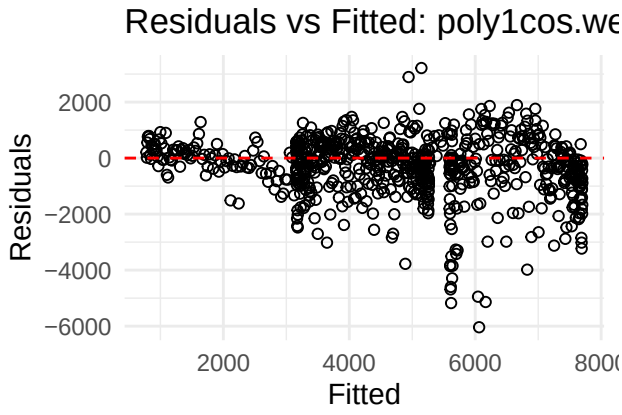

```
#weighted.data <- data.frame(y=bike.data$bikes, t=bike.data$t,
#                             cos=cos(2*pi/365*bike.data$t))

# design matrix for model
poly1cos.no.data <- data.frame(t = t.full, cos = cos(2*pi/365*t.full))

# THE SELECTED REGRESSION MODEL IN THE REPORT
weighted.model <- model.analysis(poly1cos.no.data, name = 'poly1cos.weighted',
                                 y.weights = bike.weights,
                                 print.summary = TRUE,
                                 plot = TRUE, residual.plot = TRUE)
```

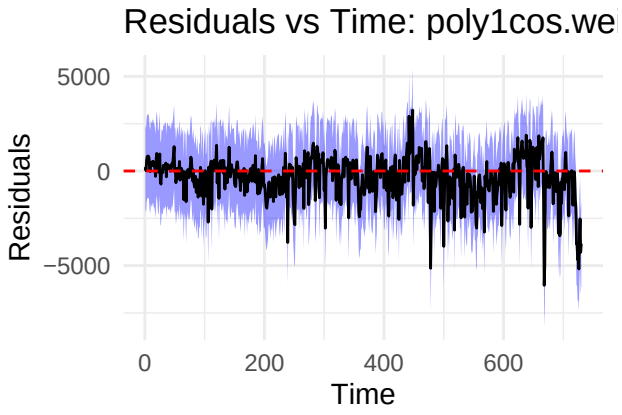
```
## [1] "--------> START SUMMARY of MODELpoly1cos.weighted"
##
## Call:
## lm(formula = bikes ~ ., data = data.train, weights = y.weights)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -180.186   -1.253   -0.256    0.813  277.607
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.387e+03  2.707e+01   88.17   <2e-16 ***
## X.t          6.669e+00  7.451e-02   89.50   <2e-16 ***
## X.cos       -1.615e+03  1.829e+01  -88.32   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.27 on 727 degrees of freedom
## Multiple R-squared:  0.9554, Adjusted R-squared:  0.9553
## F-statistic:  7788 on 2 and 727 DF,  p-value: < 2.2e-16
```









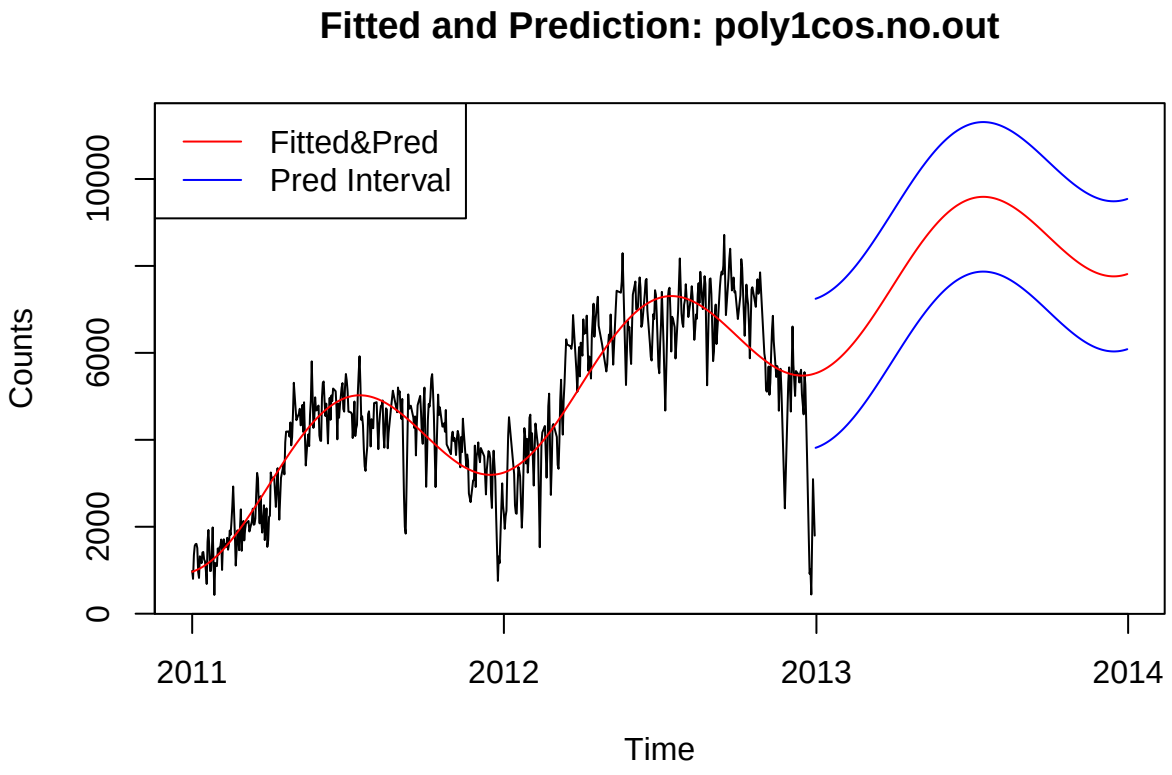


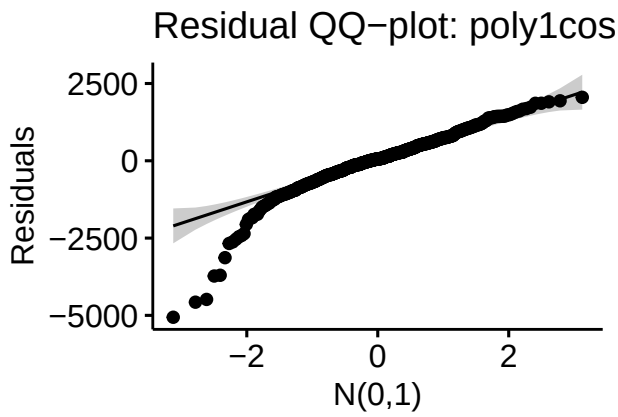

```
## [1] "Summary:"
##                  Model   ResErr  PredErr     AICc      BIC     adjR2    APSE
## 740 poly1cos.weighted 1136.516 2236.792 15005.54 15023.86 0.9552838 1460534
```

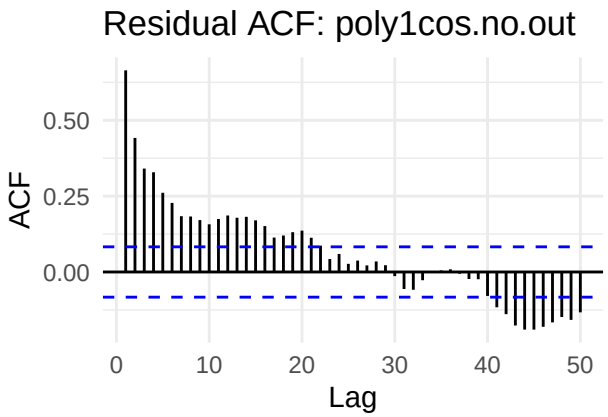
```
##        Normality   Randomness lambda
## 740 1.762354e-18 1.576682e-24     NA
```

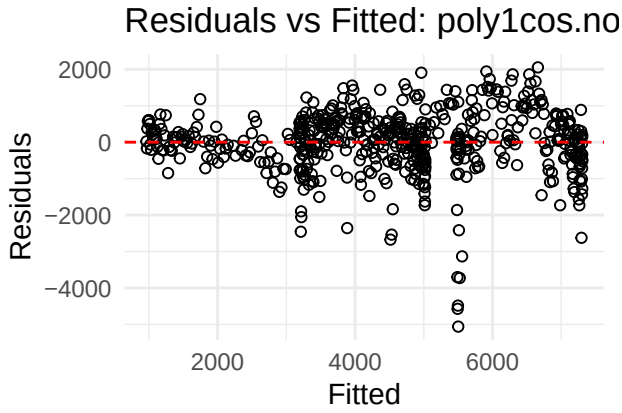
```
# fit model with no outliers and no box-cox transformation
result1 <- model.analysis(poly1cos.no.data, name = 'poly1cos.no.out',
                          train.idx = idx.no,
                          box.cox.lambda = FALSE,
                          print.summary = TRUE, plot = TRUE, residual.plot = TRUE)
```

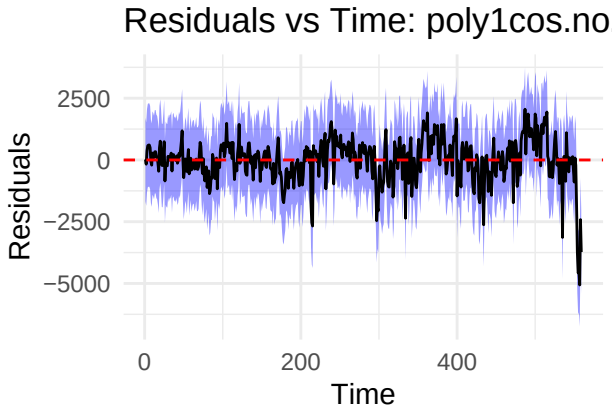
```
## [1] "--------> START SUMMARY of MODELpoly1cos.no.out"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5060.8  -409.8    58.9   524.4  2052.7
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2397.1710    70.3127   34.09   <2e-16 ***
## X.t             6.2541     0.1727   36.22   <2e-16 ***
## X.cos       -1439.2856    51.1133  -28.16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 869.7 on 556 degrees of freedom
## Multiple R-squared:  0.7949, Adjusted R-squared:  0.7941
## F-statistic:  1077 on 2 and 556 DF,  p-value: < 2.2e-16
```









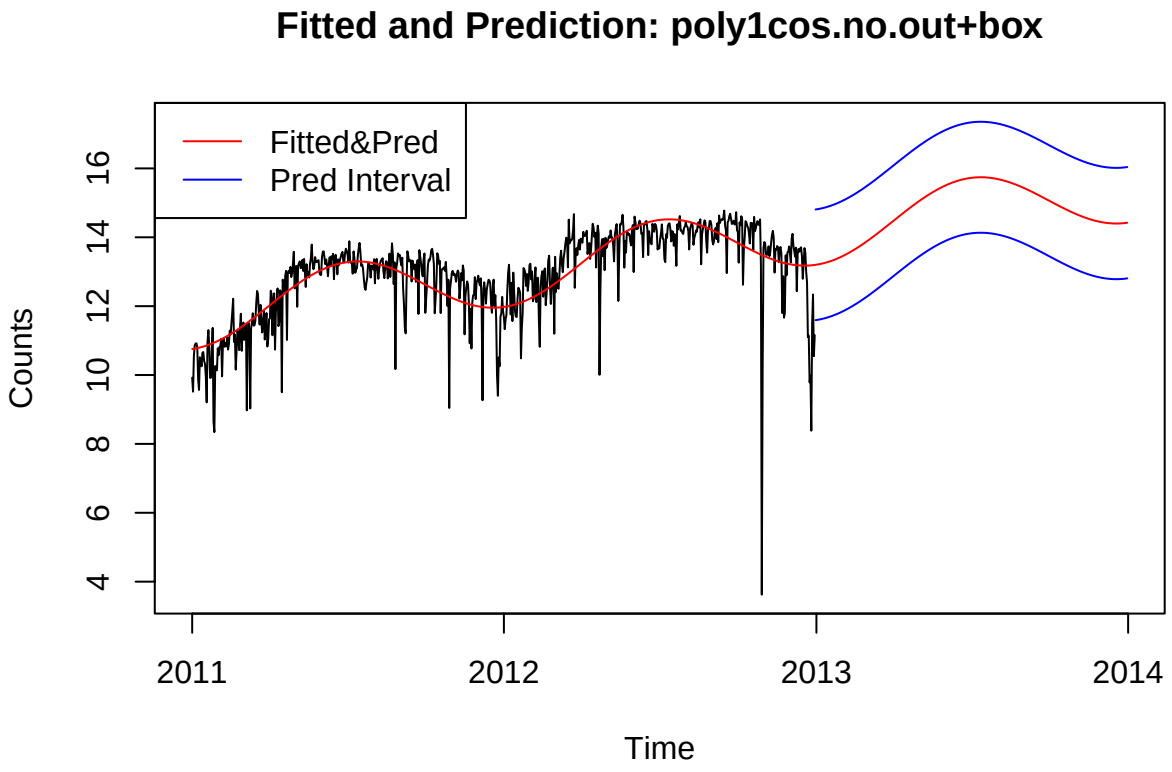


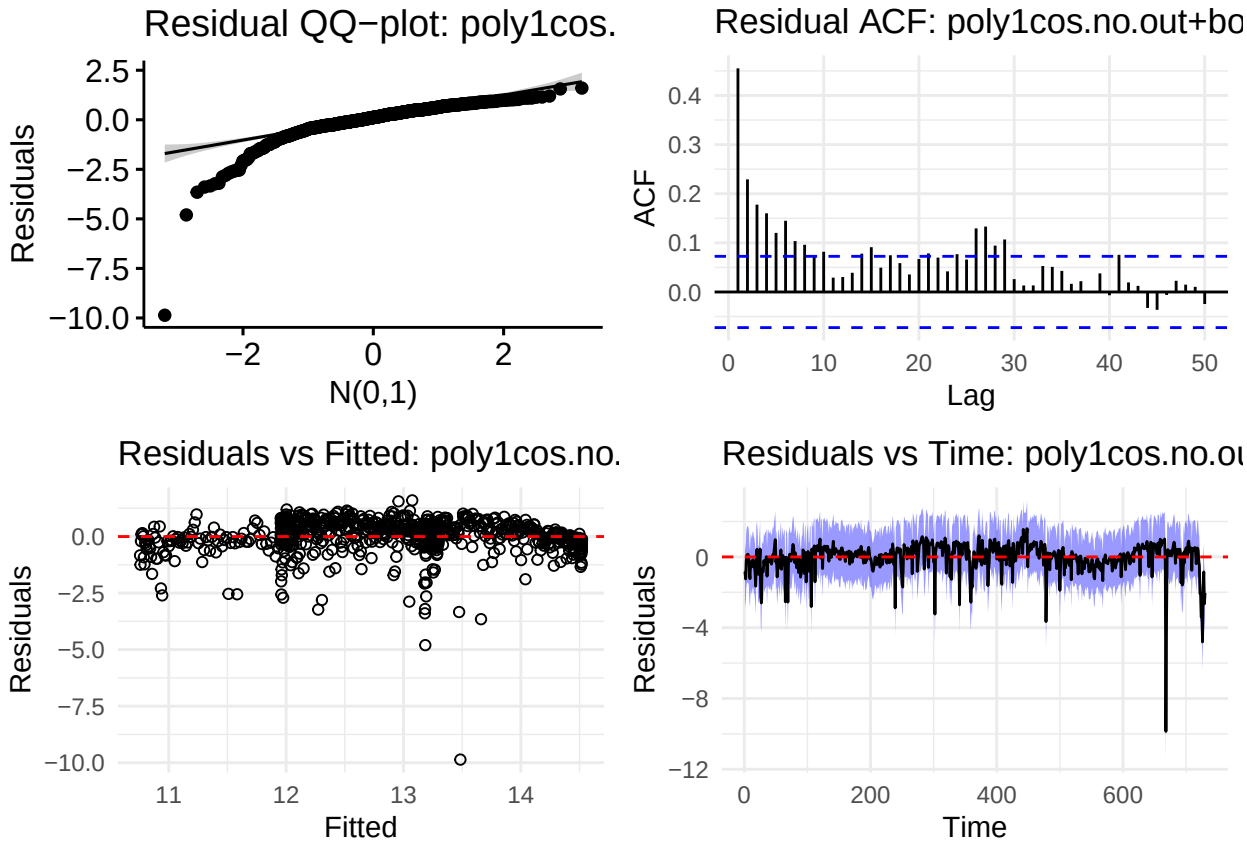

```
## [1] "Summary:"
##               Model   ResErr  PredErr     AICc      BIC     adjR2 APSE
## 740 poly1cos.no.out 869.7477 1714.381 9158.289 9175.521 0.7941418   NA
##        Normality   Randomness lambda
## 740 1.635724e-16 1.939537e-25     NA
```

```
# fit model with outliers but box-cox transformed
result2 <- model.analysis(poly1cos.no.data, name = 'poly1cos.no.out+box',
                          train.idx = FALSE,
                          box.cox.lambda = seq(0.1, 8, 0.05),
                          print.summary = TRUE, plot = TRUE, residual.plot = TRUE)
```

```
## [1] 0.1
## [1] "--------> START SUMMARY of MODELpoly1cos.no.out+box"
## [1] "ALL DATA ARE BOX-COX TRANSFORMED"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.8629 -0.2721  0.1146  0.4923  1.5989
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 11.7096947  0.0605018  193.54   <2e-16 ***
## X.t          0.0033491  0.0001434   23.35   <2e-16 ***
## X.cos       -0.9575218  0.0427373  -22.41   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8165 on 727 degrees of freedom
## Multiple R-squared:  0.5895, Adjusted R-squared:  0.5883
## F-statistic:   522 on 2 and 727 DF,  p-value: < 2.2e-16
##
## [1] "Options of lambdas: 0.1 ~ 8"
## [1] "ALL DATA ARE BOX-COX TRANSFORMED"
```



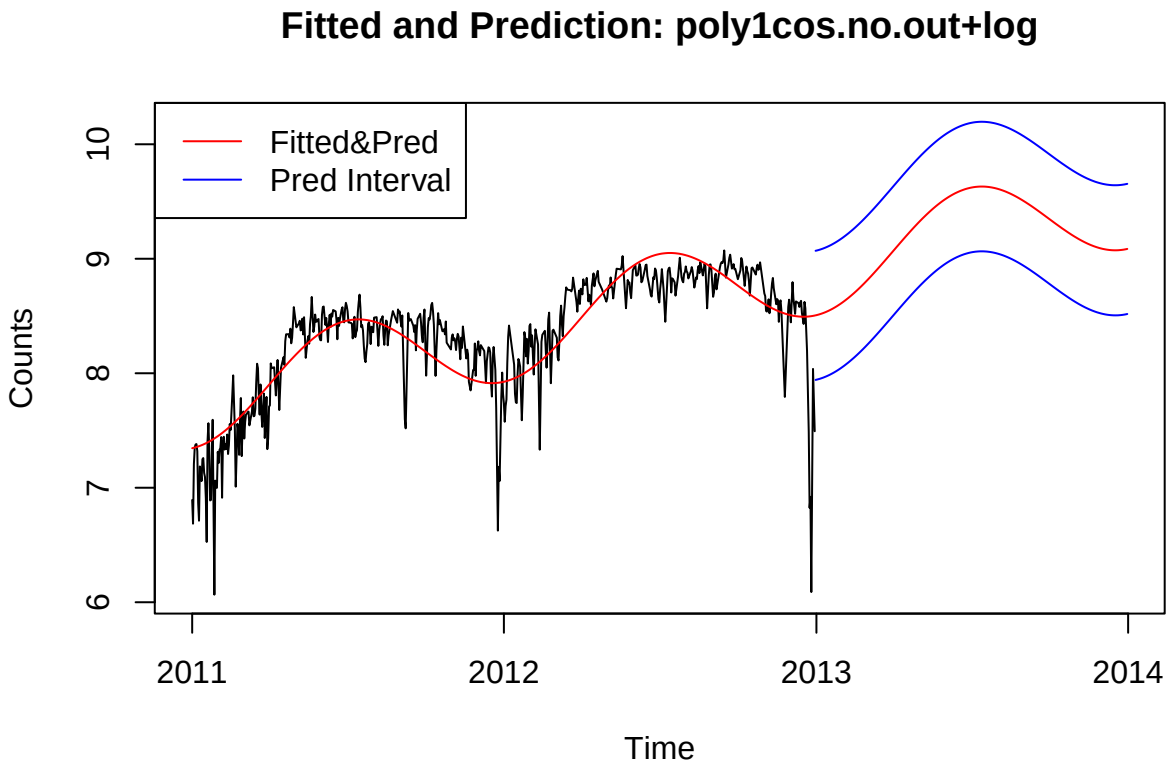


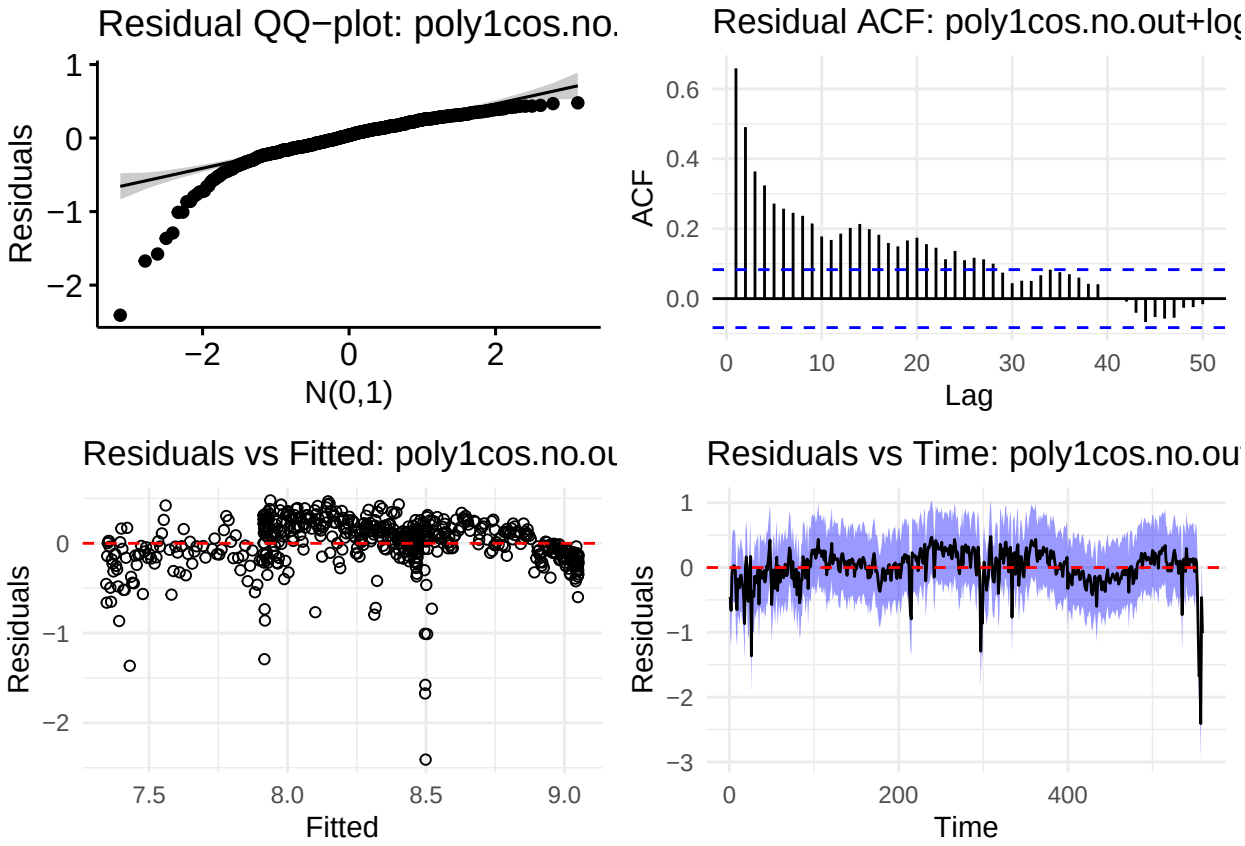

```
## [1] "Summary:"
##                   Model    ResErr  PredErr   AICc      BIC     adjR2      APSE
## 740 poly1cos.no.out+box 0.8164911 1.606947 1780.7 1799.017 0.5883468 0.6665829
##        Normality   Randomness lambda
## 740 9.969548e-31 5.080845e-32    0.1
```

```
# fit model with no outliers but log transformation
result3 <- model.analysis(poly1cos.no.data, name = 'poly1cos.no.out+log',
                          train.idx = idx.no,
                          box.cox.lambda = 0,
                          print.summary = TRUE, plot = TRUE, residual.plot = TRUE)
```

```
## [1] 0
## [1] "--------> START SUMMARY of MODELpoly1cos.no.out+log"
## [1] "ALL DATA ARE BOX-COX TRANSFORMED"
##
## Call:
## lm(formula = bikes ~ ., data = data.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.40956 -0.12121  0.04144  0.17412  0.47858
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.7566579  0.0231303  335.35   <2e-16 ***
## X.t          0.0015896  0.0000568   27.99   <2e-16 ***
## X.cos       -0.4130774  0.0168144  -24.57   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2861 on 556 degrees of freedom
## Multiple R-squared:  0.7187, Adjusted R-squared:  0.7176
## F-statistic: 710.1 on 2 and 556 DF,  p-value: < 2.2e-16
##
## [1] "lambda: 0"
## [1] "ALL DATA ARE BOX-COX TRANSFORMED"
```






```
## [1] "Summary:"
##                    Model    ResErr   PredErr     AICc      BIC     adjR2 APSE
## 740 poly1cos.no.out+log 0.2861146 0.5639674 192.4139 209.6463 0.7176451   NA
##        Normality   Randomness lambda
## 740 1.318086e-23 1.380601e-47      0
```

```
# data.poly8.cos <- data.frame(X = poly(t.full, degree = 4, raw = FALSE),
#                              cos = cos(2*pi/365*t.full))
# poly8.cos <- model.analysis(data.poly8.cos, name = 'poly8.cos',
#                             print.summary = TRUE, plot = TRUE,
#                             residual.plot = TRUE)

performance.no <- rbind(result1, result2)
performance.no <- rbind(performance.no, result3)
performance.no
```

```
##                    Model      ResErr      PredErr      AICc       BIC     adjR2
## 740      poly1cos.no.out 869.7477362 1714.3806941 9158.2886 9175.5210 0.7941418
## 7401 poly1cos.no.out+box   0.8164911    1.6069467 1780.6999 1799.0169 0.5883468
## 7402 poly1cos.no.out+log   0.2861146    0.5639674  192.4139  209.6463 0.7176451
##           APSE    Normality   Randomness lambda
## 740         NA 1.635724e-16 1.939537e-25     NA
## 7401 0.6665829 9.969548e-31 5.080845e-32    0.1
## 7402        NA 1.318086e-23 1.380601e-47    0.0
```

```
#write.csv(performance.no, 'regression_model_no_out_comparison.csv', row.names=FALSE)
```

```r
# fake model for obtaining full matrix (training + prediction)
model <- lm(y.dummy ~ poly(t.full, degree=3) + cos(2*pi/365*t.full))
X <- model.matrix(model) # extract matrix
X <- as.data.frame(X) # convert to data frame for easier prediction
weights <- c()
for(t in 1:num.obs) {
  portion <- t / 365 # number of cycles
  lower <- floor(portion)
  upper <- ceiling(portion)
  weights <- c(weights, min(1000, 1/min(abs(portion-lower), abs(portion-upper))))
}
#weights
model <- lm(Bike ~ ., data=X[bike.data$t,]) # fit real model
summary(model)
pred <- predict(model, newdata=X[t.pred,]) # predict with prediction matrix
pred <- as.vector(pred)
fitted <- c(model$fitted.values, pred)
#plot(pred.dates.full, fitted , col='red', lwd='1')
plot(dates, Bike, xlim=plot.range)
lines(model$fitted.values, col='red')
lines(dates.pred, pred, col='red')
width <- qnorm(0.975) * sqrt(sum(model$residuals^2)/model$df.residual) * sqrt(1 + rowSums((data.matrix(
sum(model$residuals^2)/model$df.residual
#width
lines(dates.pred, pred+width, col='blue')
lines(dates.pred, pred-width, col='blue')


data.no <- data.frame(y = Bike[bike.no], t = bike.data$t[bike.no], cos = cos(2*pi/365 * bike.data$t[bik
model.no <- lm(y ~ t + cos, data = data.no)
summary(model.no)
plot(Bike, type = 'l')
lines(predict(model.no, data.frame(t = t.full, cos = cos(2 * pi / 365 * t.full))), col = 'red')
plot(model.no)
shapiro.test(model.no$residuals)
```