FinalExam-TimeSeriesAnalysis-LR

Lince Rumainum

May 1, 2019

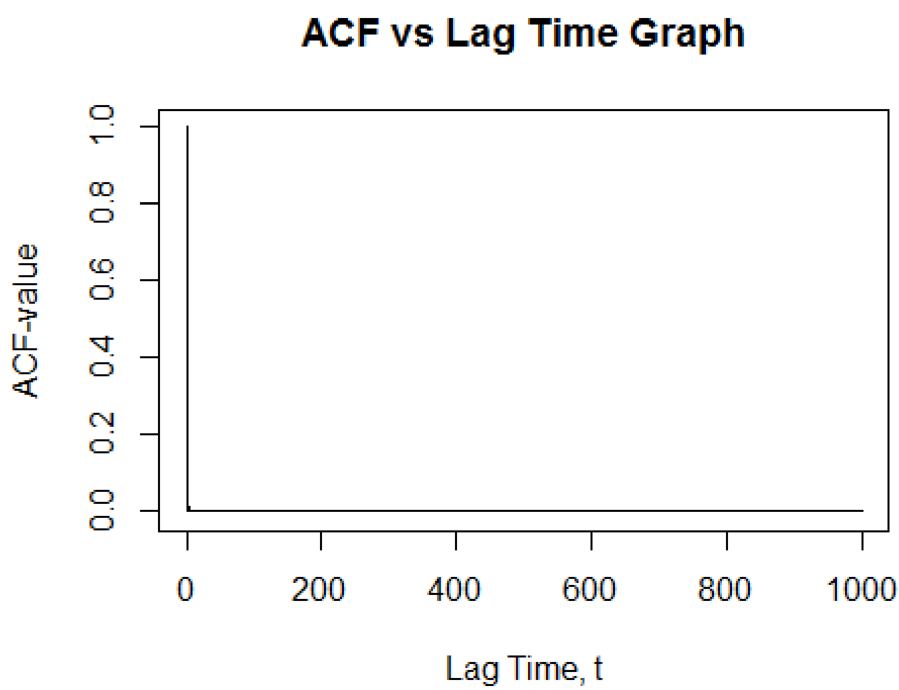
R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

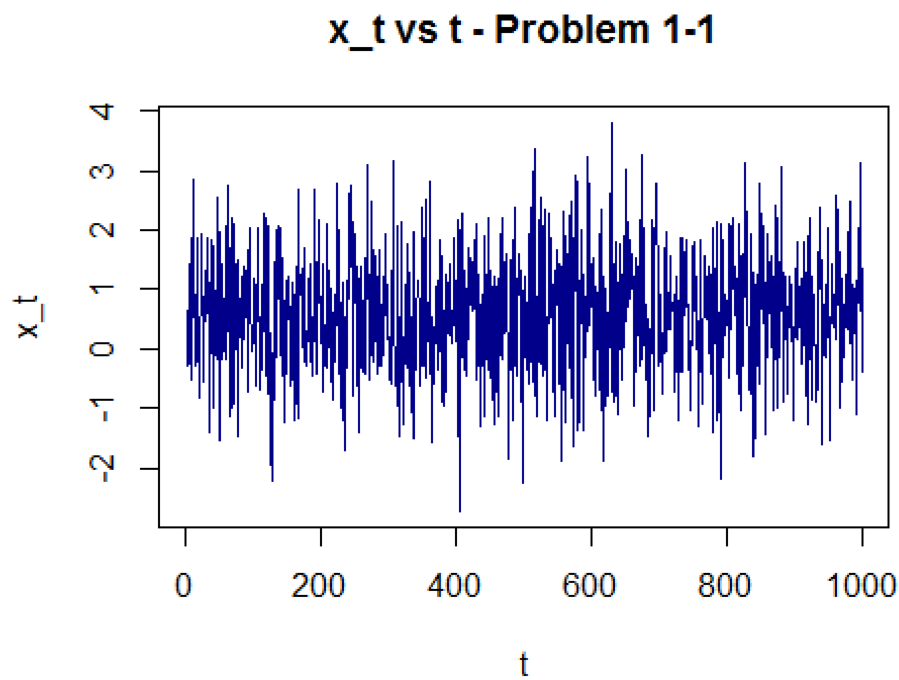
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
#--------------------------------------------------------------------
# Problem 1-1
# from equation x_t = 0.5 + 0.1 x_t-1 + epsilon_t
# we calculated ACF value from the derived function of AR(1) model
# and then plot those values
#--------------------------------------------------------------------
# Create time sequences from 0 to 1000
mainDf1_1 <- data.frame(t = seq(1, 1000, by = 1))
mainDf1_1$epsilon_t <- rnorm(1000, mean = 0, sd = 1)
#mainDf1_1$x_t[1] <- 0.5 +  mainDf1_1$epsilon_t[1]
mainDf1_1$x_t[1] <- 0

# set constants for phi
# from equation x_t = 0.5 + 0.1 x_t-1 + epsilon_t
phi <- 0.1

# calculate and plot ACF-value from given constants
# ACF value at lag = 0
rho_0 <- 1

# put those calculated values to data frame
# and then plot a line for the current acf data
mainDf1_1$rho_t[1] <- rho_0
x <- c(mainDf1_1$t[1],mainDf1_1$t[1])
y <- c(0,mainDf1_1$rho_t[1])
plot(x,y, type = "l", main="ACF vs Lag Time Graph", xlab="Lag Time, t",
ylab="ACF-value",
     xlim=c(0,length(mainDf1_1$t)), ylim=c(-0.01, max(mainDf1_1$rho_t)))

# Create the rest of the table for rho_t
for(i in 2:length(mainDf1_1$t)){
```

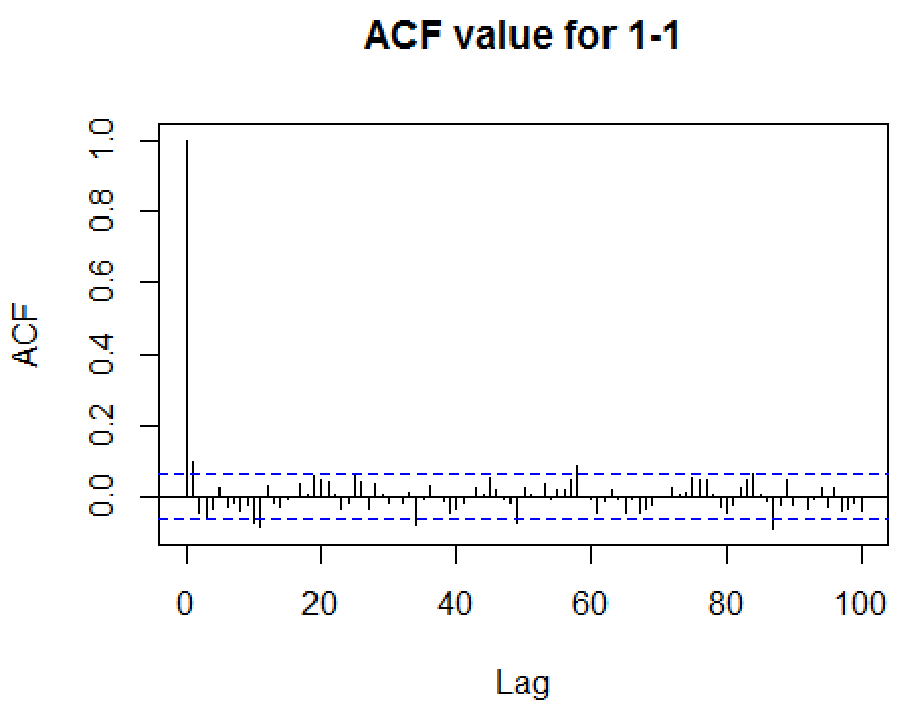
```r
  # Calculate data of rho_t
  # ACF value at current lag t value
  mainDf1_1$rho_t[i] <- phi ^ (i-1)
  # plot a line for current acf data
  x <- c(mainDf1_1$t[i],mainDf1_1$t[i])
  y <- c(0,mainDf1_1$rho_t[i])
  lines(x,y, type = "l")
}
```



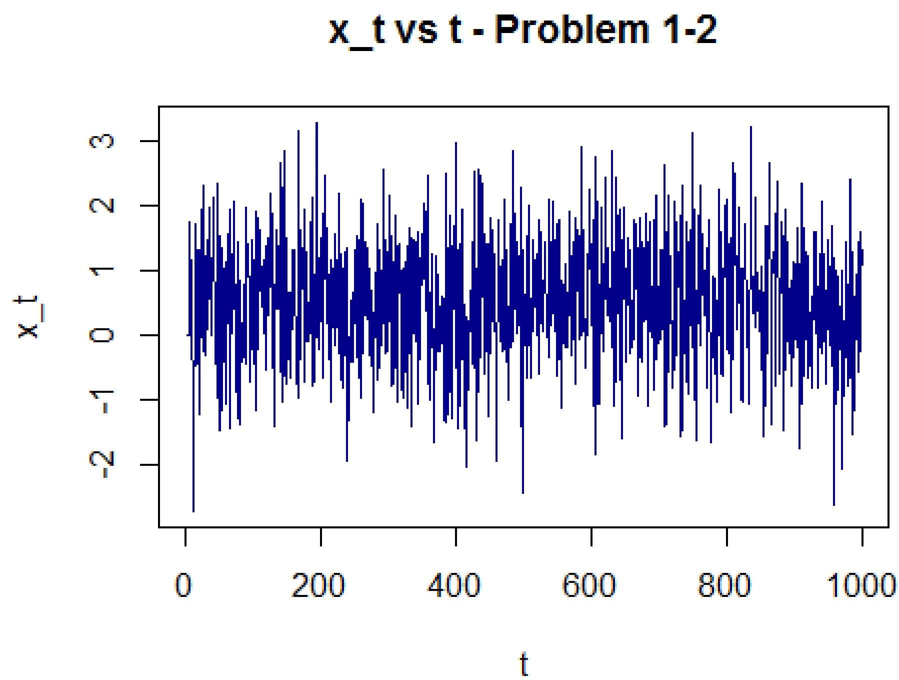

```r
# Create the rest of the table for x_t
for(i in 2:length(mainDf1_1$t)){
  # Calculate data of x_t
  mainDf1_1$x_t[i] <- 0.5 + (0.1 * mainDf1_1$x_t[i-1]) +
mainDf1_1$epsilon_t[i]
}

# plot x_t vs t
plot(mainDf1_1$t, mainDf1_1$x_t, type = "l", col= colors()[30], main="x_t vs
t - Problem 1-1", xlab="t", ylab="x_t")
```



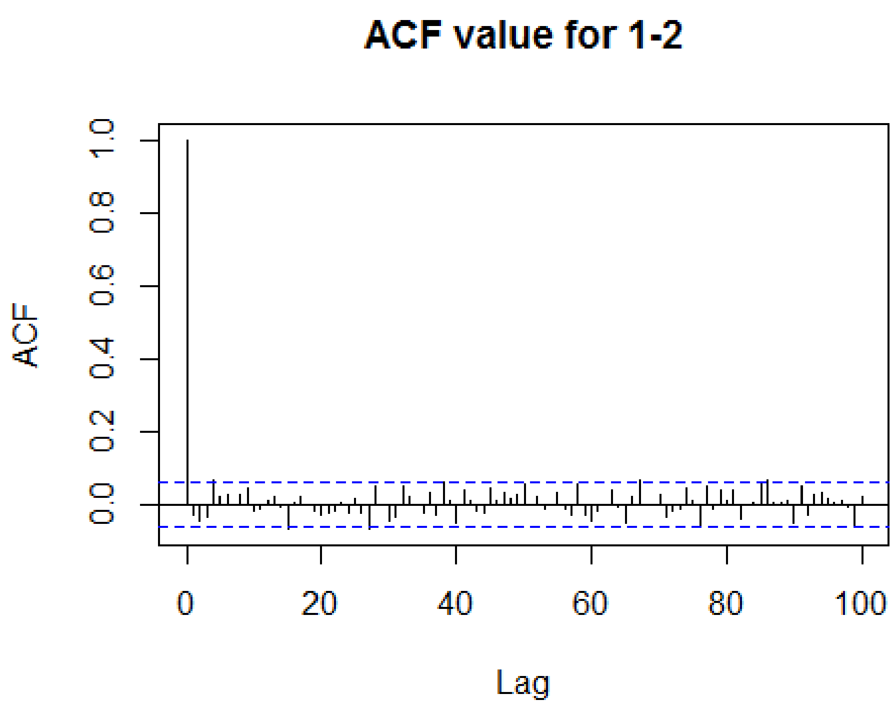

```
df11_acf = acf(mainDf1_1$x_t, lag.max = 100, type = c("correlation"), plot =
TRUE, main = "ACF value for 1-1", na.action = na.contiguous)
```



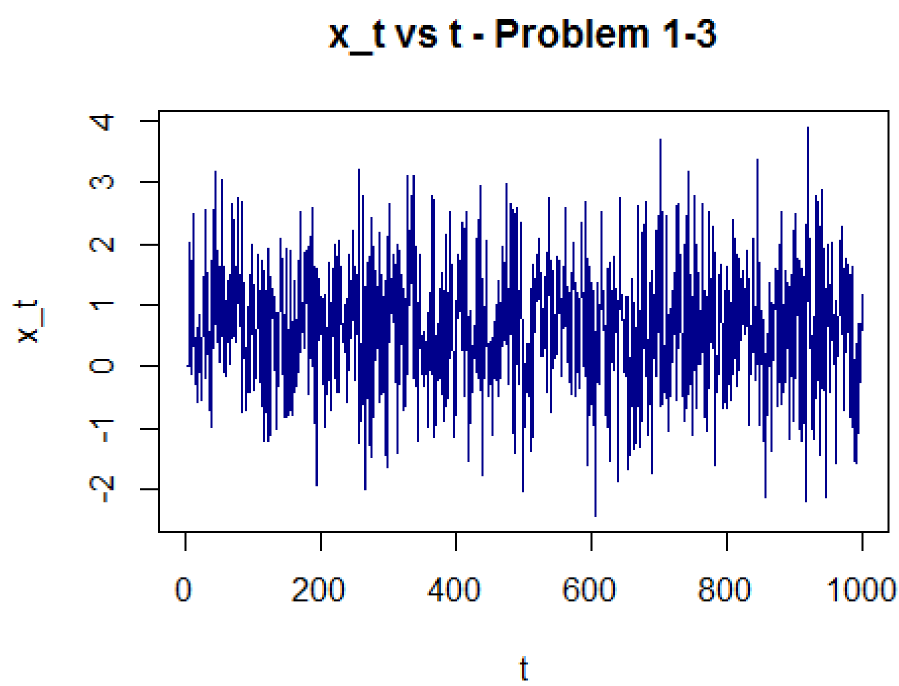
```r
df11_acf$acf[1] # ACF at j = 0

## [1] 1

df11_acf$acf[2] # ACF at j = 1

## [1] 0.09829083

df11_acf$acf[3] # ACF at j = 2

## [1] -0.04375581

#-----------------------------------------------------------------
# theorical ACF:
# at j = 0, rho(0) = 1
# at j = 1, rho(1) = 0.1
# at j = 2, rho(2) = 0.01
# END OF PROBLEM #1-1
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# Problem 1-2
# from equation x_t = 0.5 + 0.1 x_t-4 + epsilon_t
# we calculated ACF value from the derived function of AR(4) model
# and then plot those values
#-----------------------------------------------------------------
# Create time sequences from 0 to 1000
mainDf1_2 <- data.frame(t = seq(1, 1000, by = 1))
mainDf1_2$epsilon_t <- rnorm(1000, mean = 0, sd = 1)
#mainDf1_2$x_t[1] <- 0.5 +  mainDf1$epsilon_t[1]
mainDf1_2$x_t[1:4] <- 0

# Create the rest of the table for x_t
for(i in 5:length(mainDf1_2$t)){
  # Calculate data of x_t
  mainDf1_2$x_t[i] <- 0.5 + (0.1 * mainDf1_2$x_t[i-4]) +
mainDf1_2$epsilon_t[i]
}

# plot x_t vs t
plot(mainDf1_2$t, mainDf1_2$x_t, type = "l", col= colors()[30], main="x_t vs
t - Problem 1-2", xlab="t", ylab="x_t")
```




```
df12_acf = acf(mainDf1_2$x_t, lag.max = 100, type = c("correlation"), plot =
TRUE,  main = "ACF value for 1-2", na.action = na.contiguous, demean = TRUE)
```

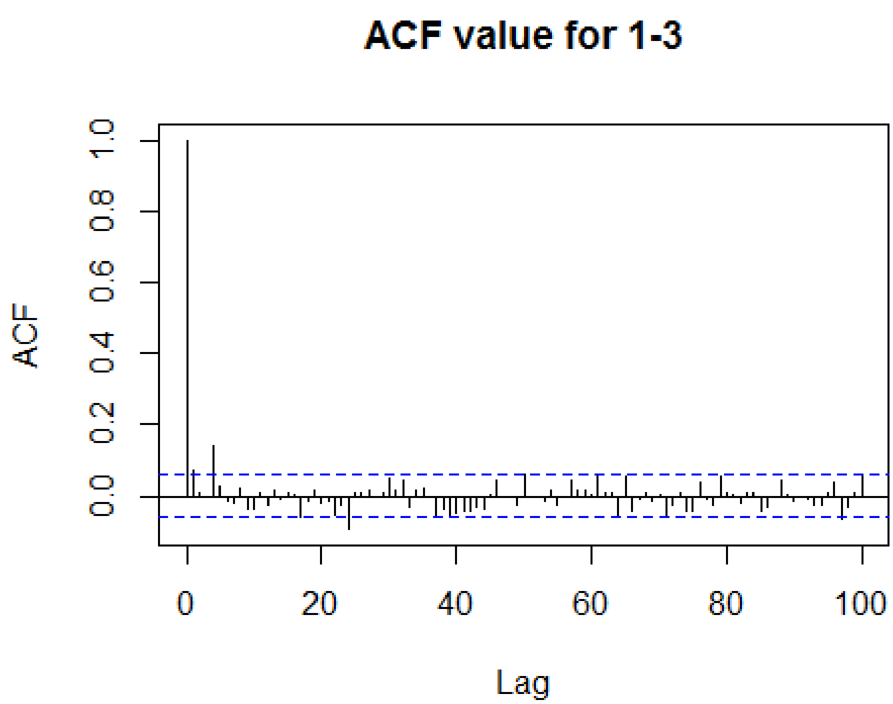


```r
df12_acf$acf[1] # ACF at j = 0

## [1] 1

df12_acf$acf[5] # ACF at j = 4

## [1] 0.0679824

#-------------------------------------------------------------------
# theorical ACF:
# at j = 0, rho(0) = 1
# at j = 4, rho(4) = 0.1
# END OF PROBLEM #1-2
#-------------------------------------------------------------------


#-------------------------------------------------------------------
# Problem 1-3
# from equation x_t = 0.5 + 0.1 x_t-1 + 0.1 x_t-4 + epsilon_t
# we calculated ACF value from the derived function of AR(1) model
# and then plot those values
#-------------------------------------------------------------------
# Create time sequences from 0 to 1000
mainDf1_3 <- data.frame(t = seq(1, 1000, by = 1))
mainDf1_3$epsilon_t <- rnorm(1000, mean = 0, sd = 1)
#mainDf1_3$x_t[1] <- 0.5 +  mainDf1_3$epsilon_t[1]
mainDf1_3$x_t[1:4] <- 0

# Create the rest of the table for x_t
for(i in 5:length(mainDf1_3$t)){
  # Calculate data of x_t
  mainDf1_3$x_t[i] <- 0.5 + (0.1 * mainDf1_3$x_t[i-1]) + (0.1 *
mainDf1_3$x_t[i-4]) +  mainDf1_3$epsilon_t[i]
}

# plot x_t vs t
plot(mainDf1_3$t, mainDf1_3$x_t, type = "l", col= colors()[30], main="x_t vs
t - Problem 1-3", xlab="t", ylab="x_t")
```



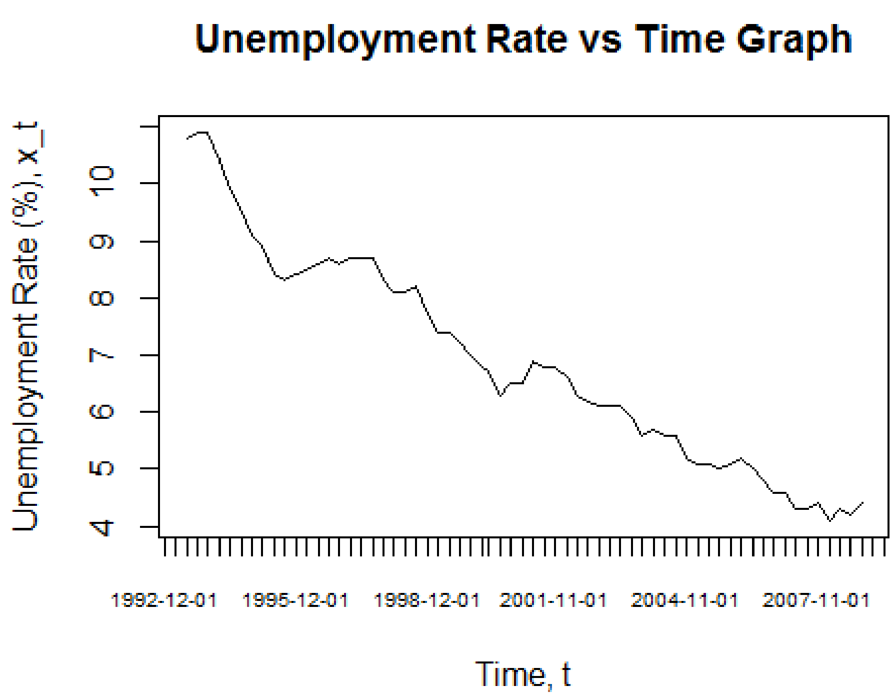

```
# plot the acf
df13_acf = acf(mainDf1_3$x_t, lag.max = 100, type = c("correlation"), plot =
TRUE,  main = "ACF value for 1-3", na.action = na.contiguous)
df13_acf$acf[1] # ACF at j = 0

## [1] 1

df13_acf$acf[2] # ACF at j = 1

## [1] 0.07127856

df13_acf$acf[5] # ACF at j = 4

## [1] 0.1422423

#------------------------------------------------------------------
# theorical ACF:
# at j = 0, rho(0) = 1
# at j = 1, rho(1) = 0.1011236
# at j = 4, rho(4) = 0.1011236
# END OF PROBLEM #1-3
#------------------------------------------------------------------



#---------------------------------------------------------------------------
-------------------------------------
# Problem #2
# Download three time series from the website
```

```
# that contain trend and seasonality

# data URLs:
# https://www.rba.gov.au/statistics/xls/unemployment-rate-by-horizon.xls
# http://www.bom.gov.au/tmp/cdio/IDCJAC0010_066196_2017.pdf - data no longer
available
# http://www.bom.gov.au/tmp/cdio/IDCJAC0010_066196_2016.pdf

# outside sources: (reading xls and pdf file)
# https://stackoverflow.com/questions/41368628/read-excel-file-from-a-url-
using-the-readxl-package
# https://datascienceplus.com/extracting-tables-from-pdfs-in-r-using-the-
tabulizer-package/
#---------------------------------------------------------------------------
-------------------------------------
#list of libraries needed for problem #2
#install.packages("httr")
#install.packages('tabulizer')
#install.packages('forecast')
library(DataCombine) # for slide function, i.e.: x_t-1
library(mgcv)

## Loading required package: nlme

## This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.
```



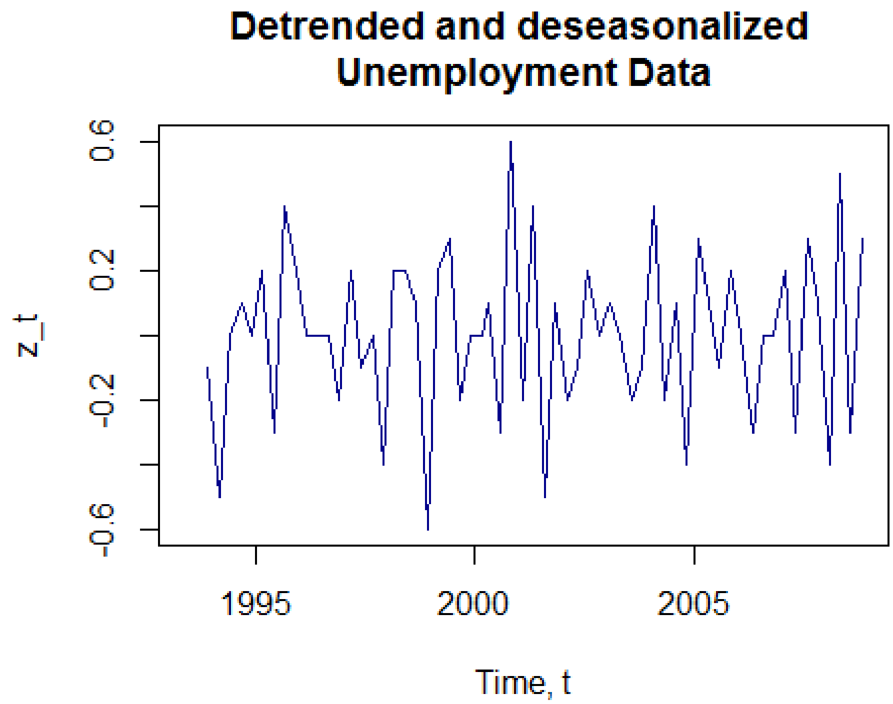
```r
library(readxl) # for reading excel file
library(httr)
packageVersion("readxl")

## [1] '1.3.0'

library(tabulizer) # for reading pdf file
library(dplyr)

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:nlme':
##
##     collapse

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

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

library(forecast)

## Warning: package 'forecast' was built under R version 3.5.3

##
## Attaching package: 'forecast'

## The following object is masked from 'package:nlme':
##
##     getResponse

# First time series is the unemployment rate data from the Australian
government website
dataURL <- "https://www.rba.gov.au/statistics/xls/unemployment-rate-by-
horizon.xls"
GET(dataURL, write_disk(tf <- tempfile(fileext = ".xls")))

## Response [https://www.rba.gov.au/statistics/xls/unemployment-rate-by-
horizon.xls]
##   Date: 2019-05-10 04:30
##   Status: 200
##   Content-Type: application/octet-stream
##   Size: 243 kB
## <ON DISK>
C:\Users\Lince\AppData\Local\Temp\RtmpoBhfTE\file901c54cb2d2f.xls

df_unemployment <- read_excel(tf, sheet = 3, col_names = TRUE, col_types =
NULL, na = "", skip = 1)
```

```
## New names:
## * `t-2` -> `t-2..11`
## * `t-2` -> `t-2..12`

df_unemployment$`Forecast date` <- as.Date(df_unemployment$`Forecast date`)

# Get unemployment data from June 1993 to November 2008
t <- df_unemployment$`Forecast date`[18:80]
unemploymentRate <- df_unemployment$t[18:80]

# Plot the original series, x_t
plot(t,unemploymentRate, type = "l", main="Unemployment Rate vs Time Graph",
xaxt = "n",
     xlab="Time, t", ylab="Unemployment Rate (%), x_t")
# Fix the x-axis so it would show dates
axis(1, df_unemployment$`Forecast date`, format(df_unemployment$`Forecast
date`), cex.axis = 0.6)
```

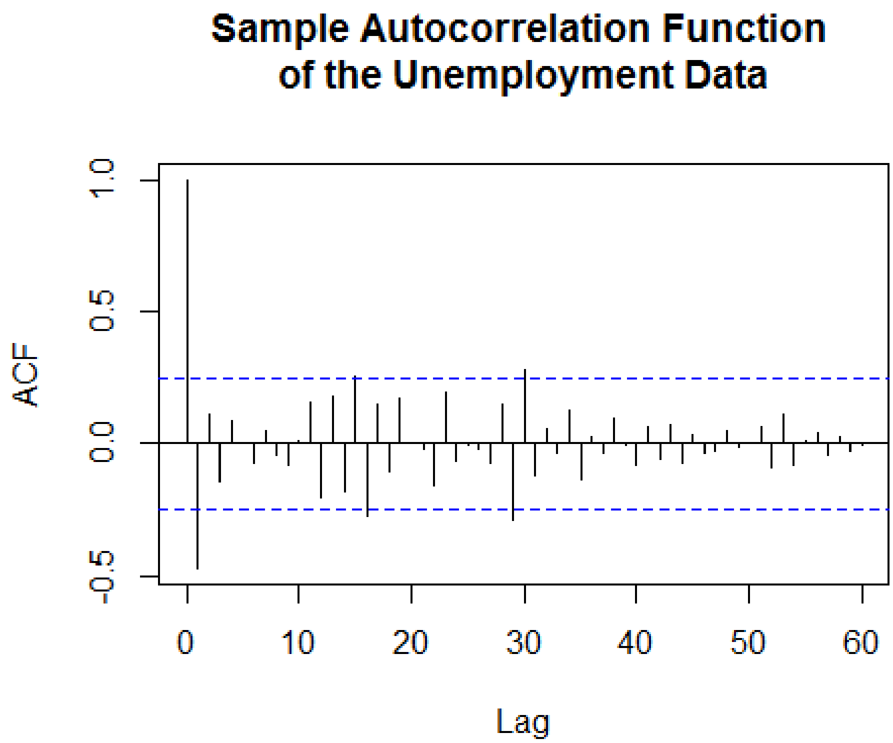



```
# Create a new dataframe without the unnecessary data
df_unemployment_main <- data.frame(t)
for (i in 1:length(t)){
  df_unemployment_main$x_t[i] <- unemploymentRate[i]
}

# removing the seasonality in x_t
df_unemployment_main<- slide(df_unemployment_main,"x_t", "t",
NewVar="xtLag1", slideBy = -1)
```

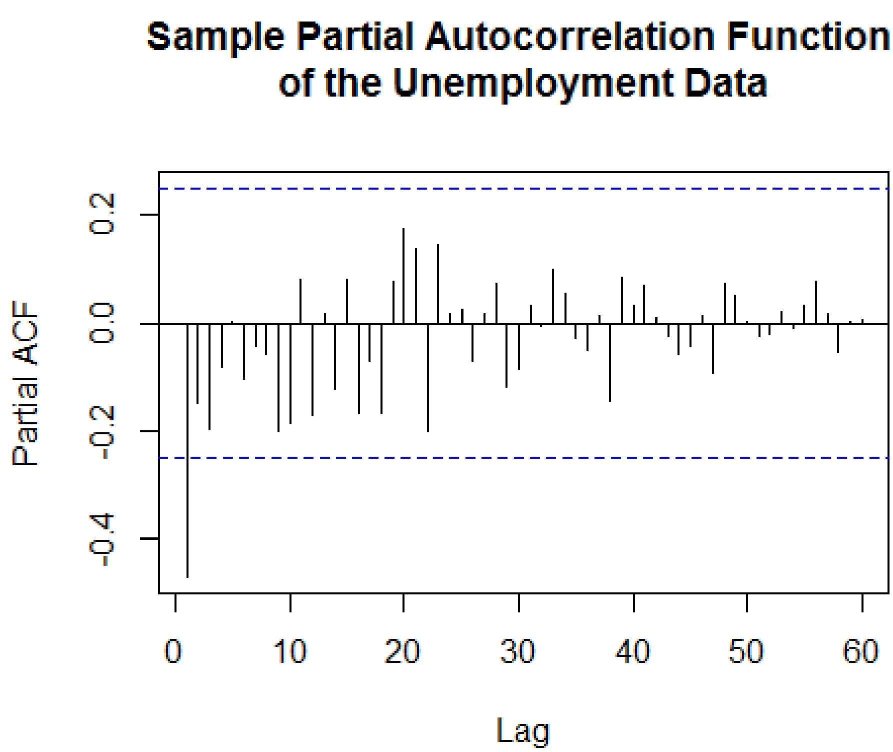
```
##
## Lagging x_t by 1 time units.

# removing the seasonality from x_t
for (i in 1:length(t)){
  if (!is.na(df_unemployment_main$x_t[i]) &
!is.na(df_unemployment_main$xtLag1[i])){
    df_unemployment_main$y_t[i] <- df_unemployment_main$x_t[i] -
df_unemployment_main$xtLag1[i]
  }
  else{
    df_unemployment_main$y_t[i] <- NA
  }
}

# plot yt vs t
y_t <- df_unemployment_main$y_t
t <- df_unemployment_main$t

# Plot z_t after removing the trend in y_t

df_unemployment_main <- slide(df_unemployment_main,"y_t", "t",
NewVar="ytLag1", slideBy = -1)

##
## Lagging y_t by 1 time units.

# removing the trend in y_t
for (i in 1:length(t)){
  df_unemployment_main$z_t[i] <- df_unemployment_main$y_t[i] -
df_unemployment_main$ytLag1[i]
}

# plot zt vs t
z_t <- df_unemployment_main$z_t
t <- df_unemployment_main$t
plot(t, z_t, type = "l", col= colors()[30], main="Detrended and
deseasonalized \nUnemployment Data", xlab="Time, t", ylab="z_t")
```




```
#Get the ACF and PACF
zt_acf <- df_unemployment_main$z_t[3:length(t)]
acf (zt_acf, lag.max = length(t), type=c("correlation"),plot=TRUE, main =
"Sample Autocorrelation Function \nof the Unemployment Data")
```

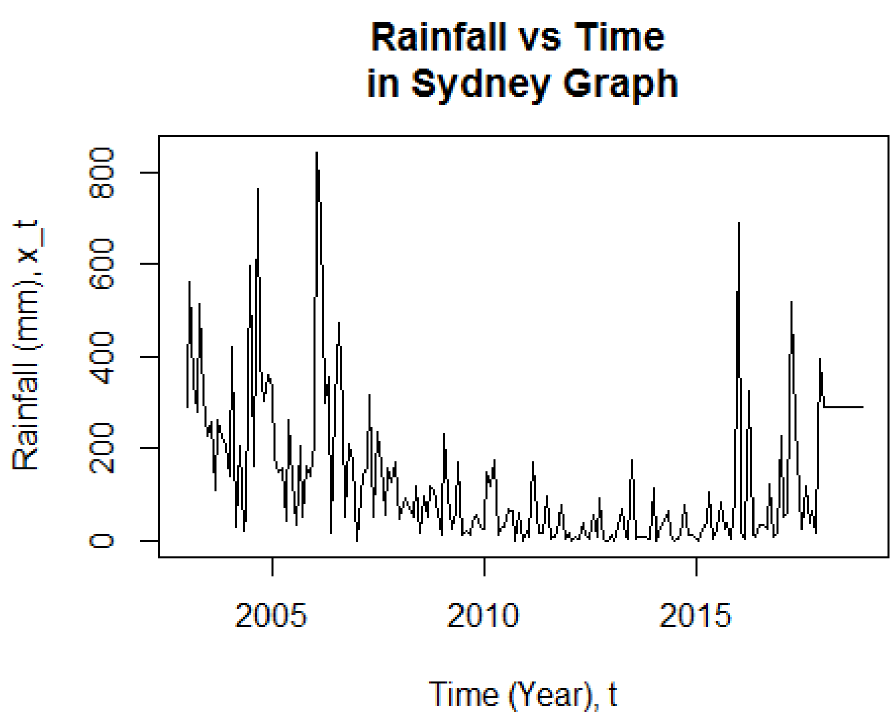



```
pacf (zt_acf, lag = length(zt_acf) - 1, plot = TRUE, main = "Sample Partial
Autocorrelation Function \nof the Unemployment Data")
```

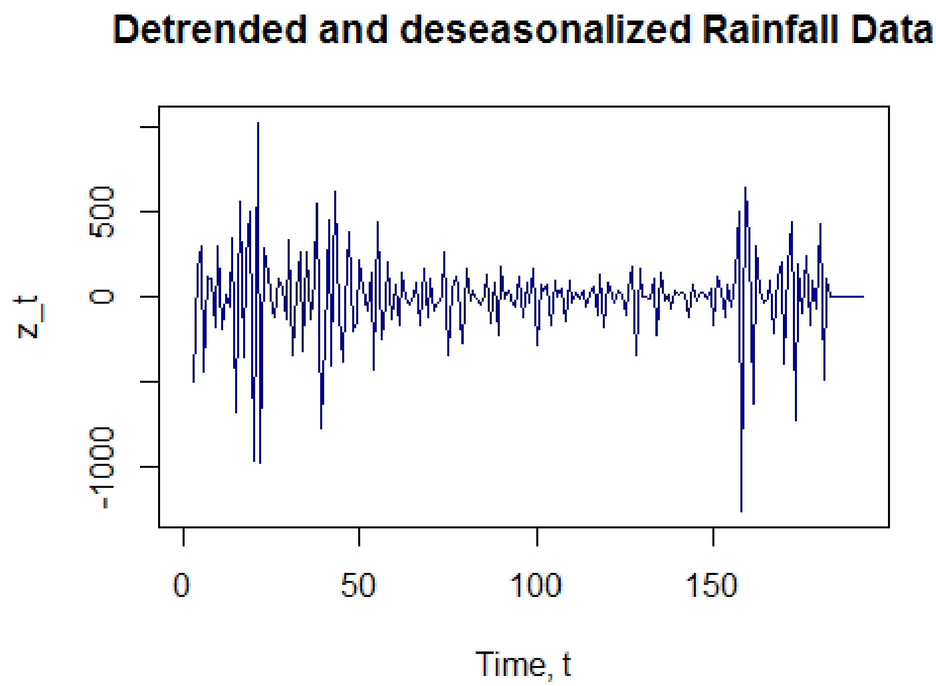


```r
# estimate model and calculate aic & aicc
forecast::auto.arima(df_unemployment_main$x_t, d = NA, D = NA, max.p = 5,
max.q = 5, max.P = 2, max.Q = 2, max.order = 5, max.d = 2, max.D = 1, start.p
= 2, start.q = 2, start.P = 1, start.Q = 1, stationary = FALSE, seasonal =
TRUE, ic = c("aicc", "aic","bic"))

## Series: df_unemployment_main$x_t
## ARIMA(0,1,0) with drift
##
## Coefficients:
##         drift
##       -0.1032
## s.e.   0.0254
##
## sigma^2 estimated as 0.04065:  log likelihood=11.82
## AIC=-19.64   AICc=-19.43   BIC=-15.38

#---------------------------------------------------------------------------
---------------------------------------
# END OF UNEMPLOYMENT DATA
#---------------------------------------------------------------------------
---------------------------------------
# Second time series is the rainfall monthly data on 2013-2018 from the
Australian government website

# montly rainfall in Sidney
dataURL2 <- 'http://www.bom.gov.au/tmp/cdio/IDCJAC0001_033106.pdf'

# Extract the table
out <- extract_tables(dataURL2)

df_rainfall <- do.call(rbind, out[-length(out)])

# table data start at fourth row
df_rainfall <- as.data.frame(df_rainfall[4:nrow(df_rainfall)-1, ],
stringsAsFactors=F)
levels(droplevels((df_rainfall[,]))) # drop levels from the dataframe

## NULL

# Column names
headers <- c('Year', 'Jan', 'Feb', 'Mar', 'Apr', 'May','Jun',
             'Jul', 'Aug', 'Sep','Oct','Nov','Dec')

# Apply custom column names
names(df_rainfall) <- headers


# getting data from 2003 to 2018
startYear <- 2003
```

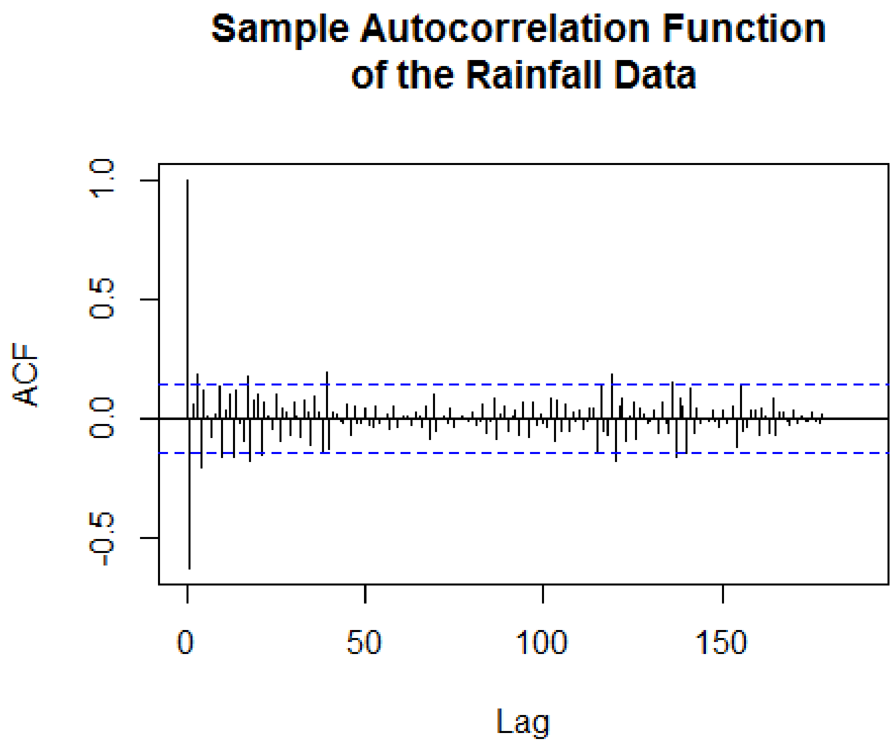
```
endYear <- 2018
totalYear <- endYear - startYear + 1
df_rainfall_main <- data.frame(t = seq(1, totalYear*12, by = 1))

#current row of rainfall for the new main dataframe
currRFRow <- 1

# goes from Jan column to Dec column
for(currCol in 2:13) {
  # iterate through row 2 to 17 and put the data into new main dataframe
  for (currRow in 2:nrow(df_rainfall)){
    df_rainfall_main$x_t[currRFRow] <- df_rainfall[currRow,currCol]
    currRFRow = currRFRow + 1
  }
}

t <- df_rainfall_main$t
rainfall <- df_rainfall_main$x_t

#convert the temperature data from character type to numeric type
df_rainfall_main$x_t = as.numeric(as.character(df_rainfall_main$x_t))

# Plot the original series, x_t
dates = seq(as.Date("2003-01-01"), by = "month", along = t)
plot(dates,rainfall, type = "l", main="Rainfall vs Time \nin Sydney Graph",
xlab="Time (Year), t", ylab="Rainfall (mm), x_t")
```



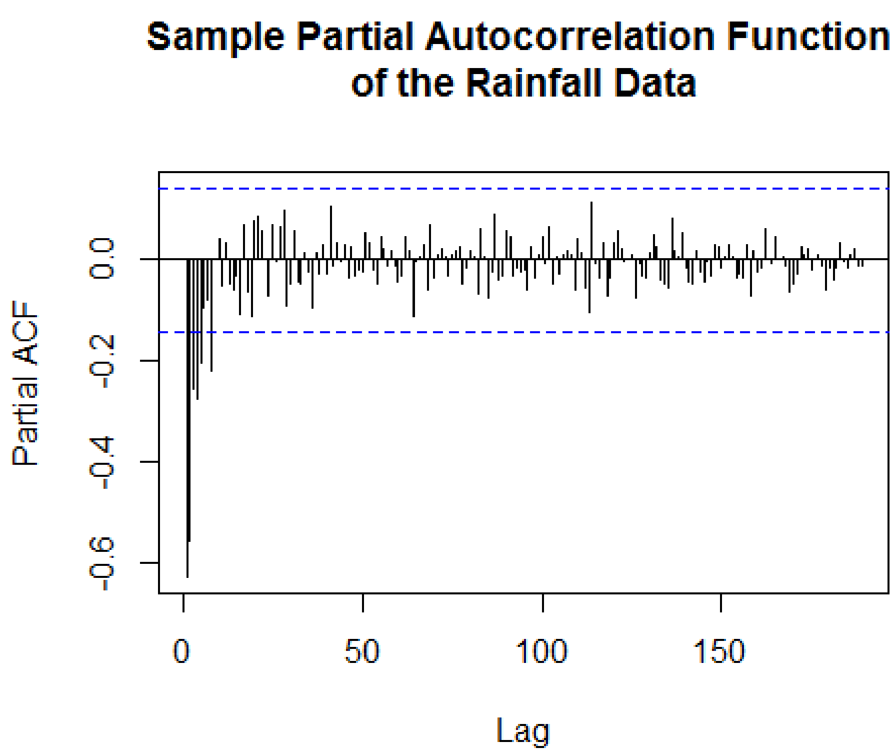
```r
# removing the seasonality in x_t
df_rainfall_main<- slide(df_rainfall_main,"x_t", "t", NewVar="xtLag1",
slideBy = -1)

##
## Lagging x_t by 1 time units.

# removing the trend in y_t
for (i in 1:length(t)){
  df_rainfall_main$y_t[i] <- df_rainfall_main$x_t[i] -
df_rainfall_main$xtLag1[i]
}
y_t <- df_rainfall_main$y_t
t <- df_rainfall_main$t

# Plot z_t after removing the trend in y_t
df_rainfall_main <- slide(df_rainfall_main,"y_t", "t", NewVar="ytLag1",
slideBy = -1)

##
## Lagging y_t by 1 time units.

# removing the trend in y_t
for (i in 1:length(t)){
  df_rainfall_main$z_t[i] <- df_rainfall_main$y_t[i] -
df_rainfall_main$ytLag1[i]
}

# plot zt vs t
z_t <- df_rainfall_main$z_t
t <- df_rainfall_main$t
plot(t, z_t, type = "l", col= colors()[30],main="Detrended and deseasonalized
Rainfall Data", xlab="Time, t", ylab="z_t")
```




```
#Get the ACF and PACF
zt_acf <- df_rainfall_main$z_t[3:length(t)]
acf (zt_acf, lag.max = length(t), type=c("correlation"),plot=TRUE, main =
"Sample Autocorrelation Function \nof the Rainfall Data")
```




```r
pacf (zt_acf, lag = length(zt_acf) - 1, plot = TRUE, main = "Sample Partial
Autocorrelation Function \nof the Rainfall Data")
```



```r
# estimate model and calculate aic & aicc
forecast::auto.arima(df_rainfall_main$x_t, d = NA, D = NA, max.p = 5, max.q =
5, max.P = 2, max.Q = 2, max.order = 5, max.d = 2, max.D = 1, start.p = 2,
start.q = 2, start.P = 1, start.Q = 1, stationary = FALSE, seasonal = TRUE,
ic = c("aicc", "aic","bic"))
```

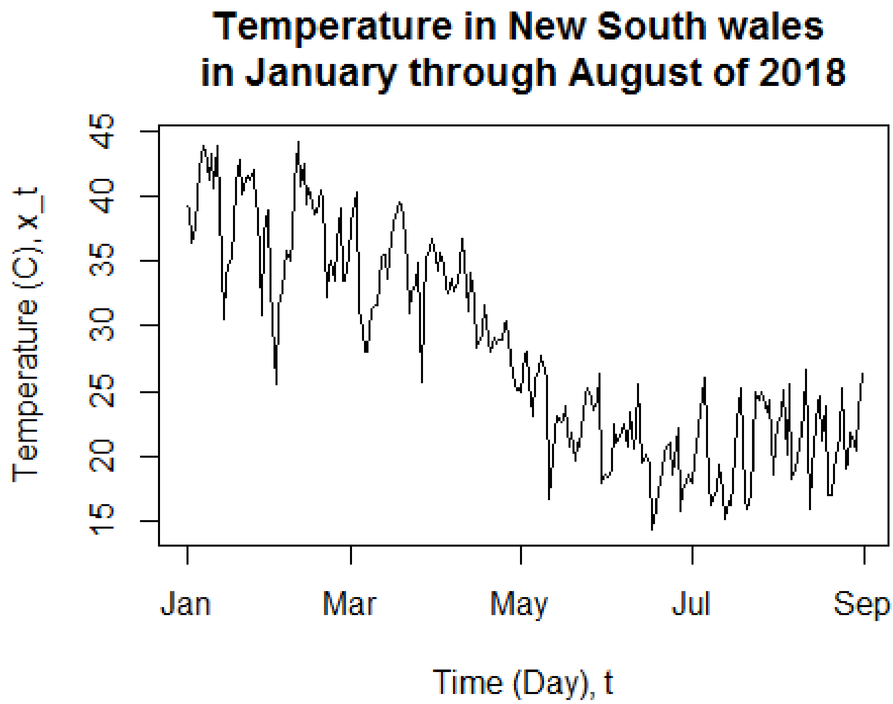
```
## Series: df_rainfall_main$x_t
## ARIMA(3,1,2)
##
## Coefficients:
##          ar1     ar2     ar3      ma1      ma2
##      -0.4172  0.1746  0.1942  -0.3087  -0.5501
## s.e.   0.2678  0.1017  0.0791   0.2678   0.2359
##
## sigma^2 estimated as 16448:  log likelihood=-1196.21
## AIC=2404.42   AICc=2404.88   BIC=2423.94
```

```r
#-------------------------------------------------------------------

#-------------------------------------------------------------------

# Third time series is the temperature data on 2018 in New South Wales
dataURL3 <- 'http://www.bom.gov.au/tmp/cdio/IDCJAC0010_052088_2018.pdf'

# Extract the table
out <- extract_tables(dataURL3)

df_temperature <- do.call(rbind, out[-length(out)])

# table data start at second row
df_temperature <- as.data.frame(df_temperature[2:nrow(df_temperature), ],
stringsAsFactors=F)
levels(droplevels((df_temperature[,]))) # drop levels from the dataframe
```

```
## NULL
```

```r
# Column names
headers <- c('2018', 'Jan', 'Feb', 'Mar', 'Apr', 'May','Jun',
             'Jul', 'Aug', 'Sep','Oct','Nov','Dec')

# Apply custom column names
names(df_temperature) <- headers

# Create a new dataframe without the unnecessary data
# Data from 1 January 2018 to 31 August 2018 (Day 1 to Day 243)
maxDay <- 243
df_temperature_main <- data.frame(t = seq(1, maxDay, by = 1))

#current row of temperature for the new main dataframe
currTempRow <- 1
```

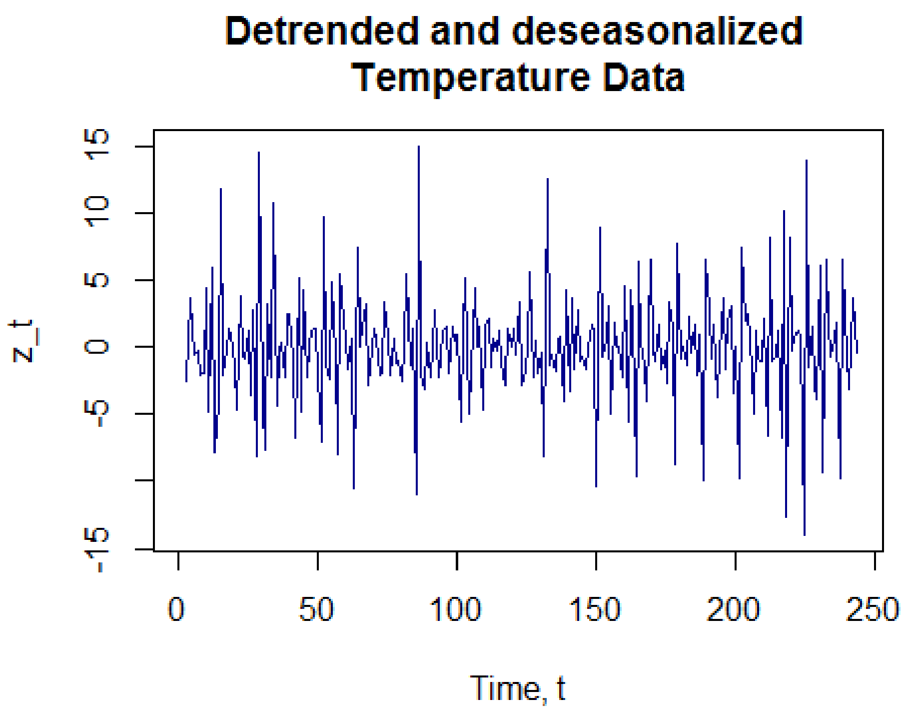
```
# goes from Jan column to Aug column
for(currCol in 2:9) {
  # iterate through row 1 to 31 and put the data into new main dataframe
  for (currRow in 1:nrow(df_temperature)){
    if ((df_temperature[currRow,currCol]!= "") & currRow <= 31){
      df_temperature_main$x_t[currTempRow] <- df_temperature[currRow,currCol]
      currTempRow = currTempRow + 1
    }
    else if ((currCol == 2 || currCol == 4 || currCol == 6 || currCol == 8 ||
currCol == 9)
             & (df_temperature[currRow,currCol] == "") & currRow <= 31){
      df_temperature_main$x_t[currTempRow]  <- NA
      currTempRow = currTempRow + 1
    }
    else if ((currCol == 5 || currCol == 7) &
(df_temperature[currRow,currCol] == "") & currRow <= 30){
      df_temperature_main$x_t[currTempRow]  <- NA
      currTempRow = currTempRow + 1
    }
    else if ((currCol == 3) & (df_temperature[currRow,currCol] == "") &
currRow <= 28){
      df_temperature_main$x_t[currTempRow]  <- NA
      currTempRow = currTempRow + 1
    }
  }
}

t <- df_temperature_main$t[1:maxDay]
temperature <- df_temperature_main$x_t[1:maxDay]

#convert the temperature data from character type to numeric type
df_temperature_main$x_t = as.numeric(as.character(df_temperature_main$x_t))

# Plot the original series, x_t
dates = seq(as.Date("2018-01-01"), by = "day", along = t)
plot(dates,temperature, type = "l", main="Temperature in New South wales \nin
January through August of 2018", xlab="Time (Day), t", ylab="Temperature (C),
x_t")
```



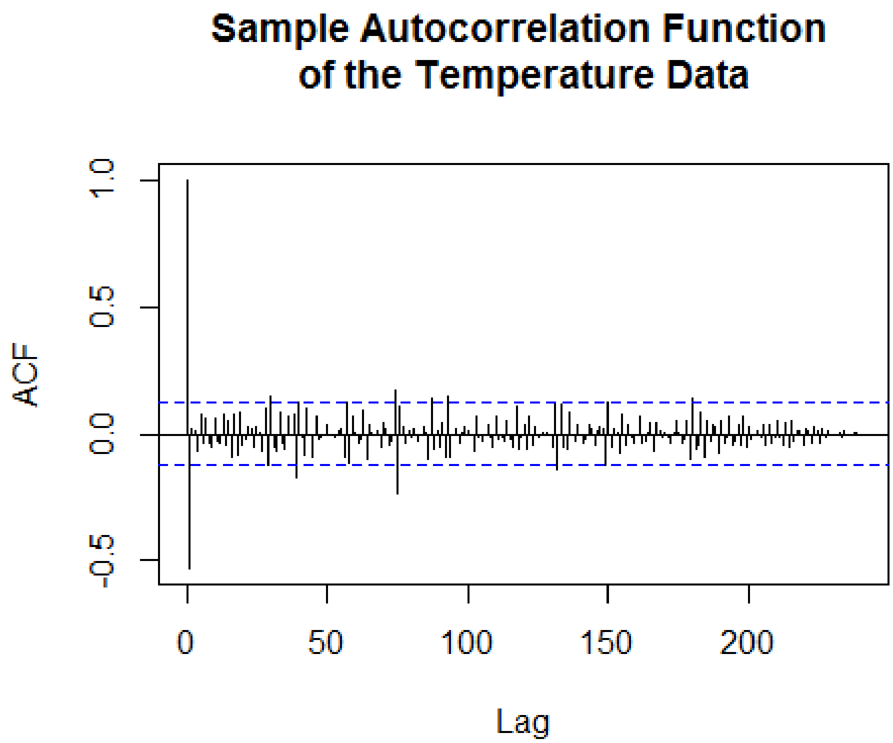

```
# Plot y_t after removing the seasonality in x_t
df_temperature_main<- slide(df_temperature_main,"x_t", "t", NewVar="xtLag1",
slideBy = -1)

##
## Lagging x_t by 1 time units.

# removing the seasonality from x_t
for (i in 1:length(t)){
  if (!is.na(df_temperature_main$x_t[i]) &
!is.na(df_temperature_main$xtLag1[i])){
    df_temperature_main$y_t[i] <- df_temperature_main$x_t[i] -
df_temperature_main$xtLag1[i]
  }
  else{
    df_temperature_main$y_t[i] <- NA
  }
}

y_t <- df_temperature_main$y_t
t <- df_temperature_main$t

# Plot z_t after removing the trend in y_t
df_temperature_main <- slide(df_temperature_main,"y_t", "t", NewVar="ytLag1",
slideBy = -1)
```

```
##
## Lagging y_t by 1 time units.
```

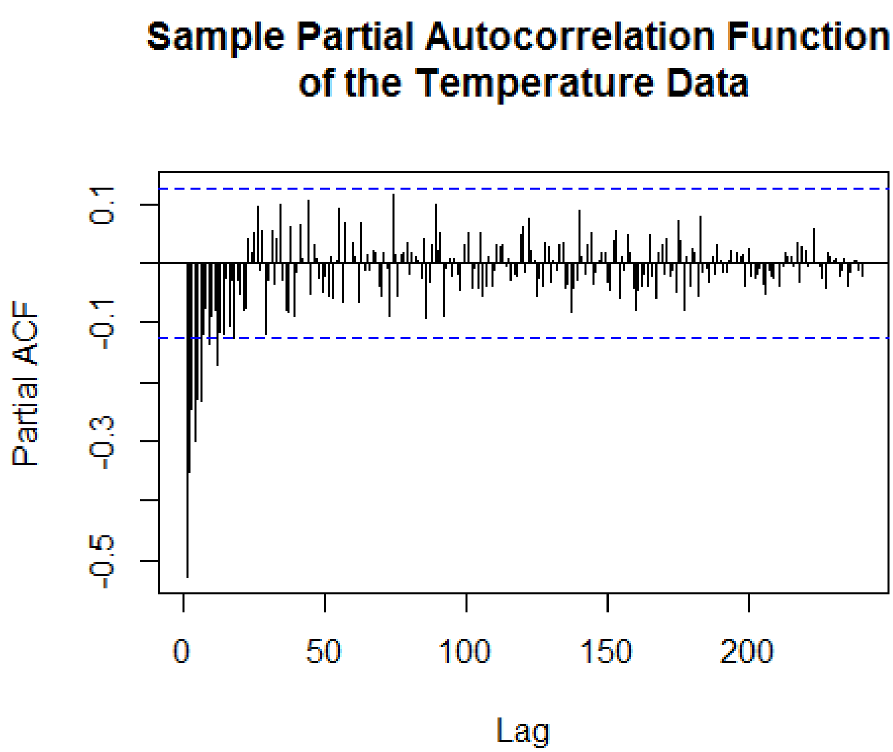
```
# removing the trend in y_t
for (i in 1:length(t)){
  df_temperature_main$z_t[i] <- df_temperature_main$y_t[i] -
df_temperature_main$ytLag1[i]
}

# plot zt vs t
z_t <- df_temperature_main$z_t
t <- df_temperature_main$t
plot(t, z_t, type = "l", col= colors()[30],main="Detrended and deseasonalized
\nTemperature Data", xlab="Time, t", ylab="z_t")
```




```
#Get the ACF and PACF
zt_acf <- df_temperature_main$z_t[3:length(t)]
acf (zt_acf, lag.max = length(t), type=c("correlation"),plot=TRUE, main =
"Sample Autocorrelation Function \nof the Temperature Data")
```




```r
pacf (zt_acf, lag = length(zt_acf) - 1, plot = TRUE, main = "Sample Partial
Autocorrelation Function \nof the Temperature Data")
```



```r
# estimate model and calculate aic & aicc
forecast::auto.arima(df_temperature_main$x_t, d = NA, D = NA, max.p = 5,
max.q = 5, max.P = 2, max.Q = 2, max.order = 5, max.d = 2, max.D = 1, start.p
= 2, start.q = 2, start.P = 1, start.Q = 1, stationary = FALSE, seasonal =
TRUE, ic = c("aicc", "aic","bic"))
```

```
## Series: df_temperature_main$x_t
## ARIMA(1,1,1) with drift
##
## Coefficients:
##          ar1      ma1    drift
##       0.6040  -0.9273  -0.0774
## s.e.  0.0638   0.0269   0.0338
##
## sigma^2 estimated as 7.473:  log likelihood=-585.63
## AIC=1179.26   AICc=1179.43   BIC=1193.22
```

```r
#------------------------------------------------------------------
# END OF PROBLEM #2
#------------------------------------------------------------------
```