Hints for Labs 1-3

Yanfei Kang

3/1/2020

Lab 1: Read and Write Data in R

You'll be working with swimming_pools.csv; it contains data on swimming pools in Brisbane, Australia (Source: data.gov.au). The file contains the column names in the first row. It uses a comma to separate values within rows.

1. **Try `read.csv()` and `read.table()` to import "swimming_pools.csv" as a data frame with the name `pools`.**

```
pools <- read.csv("~/iCloud/Teaching/BSC/Labs/data/swimming_pools.csv", header = TRUE)
head(pools)
```

```
##                           Name                                    Address
## 1 Acacia Ridge Leisure Centre         1391 Beaudesert Road, Acacia Ridge
## 2             Bellbowrie Pool               Sugarwood Street, Bellbowrie
## 3                 Carole Park Cnr Boundary Road and Waterford Road Wacol
## 4 Centenary Pool (inner City)           400 Gregory Terrace, Spring Hill
## 5              Chermside Pool               375 Hamilton Road, Chermside
## 6 Colmslie Pool (Morningside)               400 Lytton Road, Morningside
##    Latitude Longitude
## 1 -27.58616  153.0264
## 2 -27.56547  152.8911
## 3 -27.60744  152.9315
## 4 -27.45537  153.0251
## 5 -27.38583  153.0351
## 6 -27.45516  153.0789
```

2. **Try `write.table()`, `dput()`, and `save()` functions to write `pools` to files.**

```
write.csv(pools, "~/Desktop/swimming_pools.csv")
```

A variety of ways are available to store data. There is an intermediate format that is textual, but not as simple as something like CSV. For example, one can create a more descriptive representation of an R object by using the dput() or dump() functions.

```
# dput(pools) Send 'dput' output to a file
dput(pools, file = "~/Desktop/pools.R")
## Read in 'dput' output from a file
pools <- dget("~/Desktop/pools.R")
```

The complement to the textual format is the binary format. The key functions for converting R objects into a binary format are `save()` and `save.image()`.

```
save(pools, file = "~/Desktop/pools.Rdata")
load("~/Desktop/pools.Rdata")
```

3. **Restart R and read your saved data in R.**

4. **Practice subsetting of a data frame.**

Lab 2: `dplyr`

You'll be working with the `airquality` in the R package `datasets`. Bear in mind `%>%`.

1. Please return all the rows where Temp is larger than 80 and Month is after May.

```
library(dplyr)
filter(airquality, Temp > 80 & Month > 5) %>% head()
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    NA     186  9.2   84     6   4
## 2    NA     220  8.6   85     6   5
## 3    29     127  9.7   82     6   7
## 4    NA     273  6.9   87     6   8
## 5    71     291 13.8   90     6   9
## 6    39     323 11.5   87     6  10
```

2. Please add a new column that displays the temperature in Celsius.

```
airquality <- mutate(airquality, celtemp = (Temp - 32)/1.8)
```

3. Calculate the mean temperature in each month.

```
airquality %>% group_by(Month) %>% summarise(mean_temp = mean(Temp))
```

```
## # A tibble: 5 x 2
##   Month mean_temp
##   <int>     <dbl>
## 1     5      65.5
## 2     6      79.1
## 3     7      83.9
## 4     8      84.0
## 5     9      76.9
```

4. Remove all the data corresponding to Month = 5, group the data by month, and then find the mean of the temperature each month.

```
airquality %>% filter(Month != 5) %>% group_by(Month) %>% summarise(mean_temp = mean(Temp))
```

```
## # A tibble: 4 x 2
##   Month mean_temp
##   <int>     <dbl>
## 1     6      79.1
## 2     7      83.9
## 3     8      84.0
## 4     9      76.9
```

Lab3: functions

In this lab, you will use the temperature data in four cities: Melbourne, Sydney, Brisbane and Cairns. You can download them from https://yanfei.site/docs/sc/data/temp.zip.

1. Please make a function `load.file()` to read a .csv file and transform the first column (a character representing date and time) using `as.POSIXlt` into R time format.

```
load.file <- function(file_path) {
    temp_data <- read.csv(file_path)
    temp_data$time <- as.POSIXlt(temp_data$time)
```

```
    return(temp_data)
}
```

2. Then apply `load.file()` to each filename using `lapply()`.

```
mypath <- "~/iCloud/Teaching/BSC/Labs/data/temp/"
files <- paste0(file.path(mypath), list.files(mypath))
temp <- lapply(files, load.file)
names(temp) <- c("Brisbane", "Cairns", "Melbourne", "Sydney")
```

3. How many rows of data are there for each city?

```
sapply(temp, nrow)
```

```
##  Brisbane    Cairns Melbourne    Sydney
##        99        80        97        99
```

4. What is the hottest temperature recorded by city?

```
temp %>% lapply(select, temp.max) %>% sapply(max)
```

```
##  Brisbane    Cairns Melbourne    Sydney
##     23.89     35.00     13.33     23.33
```

5. Estimate the autocorrelation function for each city.

```
temp %>% lapply(select, temp) %>% lapply(acf, plot = FALSE)
```

```
## $Brisbane
##
## Autocorrelations of series 'X[[i]]', by lag
##
##      0      1      2      3      4      5      6      7      8      9
##  1.000  0.951  0.838  0.685  0.512  0.340  0.175  0.027 -0.095 -0.191
##     10     11     12     13     14     15     16     17     18     19
## -0.261 -0.308 -0.334 -0.341 -0.327 -0.286 -0.218 -0.121  0.001  0.137
##
## $Cairns
##
## Autocorrelations of series 'X[[i]]', by lag
##
##      0      1      2      3      4      5      6      7      8      9
##  1.000  0.892  0.709  0.517  0.316  0.124 -0.050 -0.187 -0.295 -0.369
##     10     11     12     13     14     15     16     17     18     19
## -0.401 -0.413 -0.415 -0.406 -0.383 -0.324 -0.233 -0.102  0.042  0.171
##
## $Melbourne
##
## Autocorrelations of series 'X[[i]]', by lag
##
##      0      1      2      3      4      5      6      7      8      9
##  1.000  0.945  0.836  0.687  0.530  0.376  0.245  0.129  0.033 -0.046
##     10     11     12     13     14     15     16     17     18     19
## -0.106 -0.151 -0.178 -0.186 -0.177 -0.137 -0.081 -0.006  0.081  0.191
##
## $Sydney
##
## Autocorrelations of series 'X[[i]]', by lag
```

```
##
##      0      1      2      3      4      5      6      7      8      9
##  1.000  0.943  0.823  0.658  0.465  0.273  0.094 -0.062 -0.192 -0.291
##     10     11     12     13     14     15     16     17     18     19
## -0.365 -0.412 -0.438 -0.438 -0.412 -0.360 -0.276 -0.166 -0.036  0.107
```

Quiz in the 2nd lecture

```
FindRoots <- function(a = 1, b = 1, c = 0) {
  ## Computes the roots of a quadratic equation.
  ## Args:
  ##   a: The coeffienct of x^2. Default is 1.
  ##   b: The coeffienct of x. Default is 1.
  ##   c: The constant. Default is 0.
  ## Returns:
  ##   Roots.
  if (b ^ 2 - 4 * a * c < 0) {
    return("No real roots")
  } else {
    root1 <- (-b + sqrt(b ^ 2 - 4 * a * c)) / (2 * a)
    root2 <- (-b - sqrt(b ^ 2 - 4 * a * c)) / (2 * a)
    roots <- c(root1, root2)
    return(roots)
  }
}

## some tests
FindRoots()
```

```
## [1]  0 -1
```

```
FindRoots(1, 2, 0)
```

```
## [1]  0 -2
```

```
FindRoots(1, 2, 3)
```

```
## [1] "No real roots"
```

Notes

1. R coding styles.
2. Avoid messy code. How?
3. Remember to include a return in a function. Before using a function, it has to be in your R environment.
4. Learn to read the error messages from R. You will learn how to debug in R.
5. `getwd()` and `setwd()`.