Schweizer Paraplegiker Forschung

Exercise: R functions

Workshop – Introduction to R

SOLUTION

1 Importing data

a) Download the excel file **cardata.xlsx** from ?? and put it in the same folder where you have stored your R script.

b) Although an excel file can be read directly into R, we will first see how to turn it into the simpler "csv" (comma separated values) format to use in R. Open the excel file and turn it into a .csv file by going to "File > Save As > Browse" and choosing "CSV (Comma delimited)" as the format before saving it. Open the newly created .csv file in a simple text editor (e.g. Notepad on Windows) and look at it.

c) Check what your current working directory is.

```
getwd()
```

d) Set the working directory to the folder where the .csv file is located.

```
setwd("C:\\Users\\rothacher_y\\Documents")
```

e) Read in the .csv file to R by using the `read.csv` function. Depending on the used separator sign you might need to adapt your command (see slides). Store the imported data under the name `cardata` and look at this object, what do you see?

```
cardata <- read.csv("cardata.csv", sep=';')
cardata
```

```
##                    car  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1            Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 2        Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 3           Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 4       Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 5    Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## 6              Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 7           Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 8            Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 9             Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 10            Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 11           Merc 280C 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## 12          Merc 450SE 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 13          Merc 450SL 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## 14         Merc 450SLC 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## 15  Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## 16 Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## 17   Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
```

```
## 18            Fiat 128 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 19         Honda Civic 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 20      Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 21       Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 22    Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## 23         AMC Javelin 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## 24          Camaro Z28 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## 25    Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 26           Fiat X1-9 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 27       Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 28        Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 29      Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## 30        Ferrari Dino 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## 31       Maserati Bora 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## 32          Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

The data has been imported as a data frame. The data contains information on 32 different cars. This data is actually also available as a built-in data set in R and can be called simply by typing the name mtcars.

f) As mentioned, we can also read in an excel file directly. To do this we need to first install and load an additional R-package. Install and load the `readxl` package to make its contents available.

```
install.packages("readxl")   # Downloads and installs the package
library(readxl)   # Activates the package
```

g) Now you can read in the excel file using the following command. Look at the resulting data frame.

```
dat <- as.data.frame(read_excel("cardata.xlsx"))
```

```
dat

##                    car  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1            Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 2        Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 3           Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 4       Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 5    Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## 6              Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 7           Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 8            Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 9             Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 10            Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 11           Merc 280C 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## 12          Merc 450SE 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 13          Merc 450SL 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## 14         Merc 450SLC 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## 15  Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## 16 Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## 17   Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 18            Fiat 128 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 19         Honda Civic 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 20      Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 21       Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 22    Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
```

```
## 23         AMC Javelin 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## 24          Camaro Z28 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## 25    Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 26           Fiat X1-9 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 27       Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 28        Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 29      Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## 30        Ferrari Dino 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## 31       Maserati Bora 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## 32          Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

The data has again been imported as a data frame.

2 Ready-to-use functions

a) In this first exercise we will work with the `mpg` variable of the `cardata` data set. It contains the miles per gallon value for each car. The following list contains a selection of functions which can be applied to vectors. Try them out on the `mpg` column.

- `mean` to calculate the mean value

```
v <- cardata$mpg
v

##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4

mean(v)

## [1] 20.09062
```

- `sd` to calculate the sample standard deviation and `var` for the sample variance.

```
sd(v)

## [1] 6.026948

var(v)

## [1] 36.3241
```

- `sum` to add all elements of a vector together.

```
sum(v)

## [1] 642.9
```

- `min` and `max` to get the smallest and largest element of a vector. `range` returns these values in one vector.

```r
min(v)
```

```
## [1] 10.4
```

```r
max(v)
```

```
## [1] 33.9
```

```r
range(v)
```

```
## [1] 10.4 33.9
```

- `length` to show how many elements a vector contains.

```r
length(v)
```

```
## [1] 32
```

3 Learning about new functions in R

R offers functions for many different tasks. However, often one might not know whether a specific function exists and under which name it is available. And if it exists, one first needs to learn how it is applied. Such information can be found in the internal help pages or by browsing the internet.

a) Let's look again at the `mpg` variable of the `cardata` data. We want to show the `mpg` vector with all elements sorted by their values in increasing order. Try to find out whether there is a function for that purpose available and if yes, use it.

```r
sort(cardata$mpg)
```

```
##  [1] 10.4 10.4 13.3 14.3 14.7 15.0 15.2 15.2 15.5 15.8 16.4 17.3 17.8 18.1 18.7
## [16] 19.2 19.2 19.7 21.0 21.0 21.4 21.4 21.5 22.8 22.8 24.4 26.0 27.3 30.4 30.4
## [31] 32.4 33.9
```

b) **Extra:** Look at the help page of that function. Can you find a way to sort the values of `mpg` in descending order instead?

```r
sort(cardata$mpg, decreasing = TRUE)
```

```
##  [1] 33.9 32.4 30.4 30.4 27.3 26.0 24.4 22.8 22.8 21.5 21.4 21.4 21.0 21.0 19.7
## [16] 19.2 19.2 18.7 18.1 17.8 17.3 16.4 15.8 15.5 15.2 15.2 15.0 14.7 14.3 13.3
## [31] 10.4 10.4
```

c) Now we want to calculate the Pearson correlation coefficient between the two columns `mpg` and `disp`. Again, try to find the corresponding R function and apply it.

```r
cor(cardata$mpg, cardata$disp)
```

```
## [1] -0.8475514
```

d) **Extra:** Can you find a way to calculate the Spearman correlation instead of the Pearson correlation?

```r
cor(cardata$mpg, cardata$disp, method = "spearman")
```

```
## [1] -0.9088824
```

4 Writing your own function

a) Write a function which takes two input arguments (both are expected to be numbers) and calculates the sum of the two.

```r
mysum <- function(x, y){
  rval <- x + y
  return(rval)
}

### Try out the function:
mysum(10, 8)
```

```
## [1] 18
```

b) There is already a function in R to calculate the mean value of a vector. Try to write your own function which takes a (numeric) vector as input and returns its mean value (without using the `mean` function).

```r
mymean <- function(x){
  rval <- sum(x)/length(x)
  return(rval)
}

### Try out the function:
a <- c(1, 3, 80, 100, 55)
mymean(a)
```

```
## [1] 47.8
```

```r
### Compare with available R function:
mean(a)
```

```
## [1] 47.8
```

c) **Extra:** Try to write a function which takes a (numeric) vector as an input and scales it to the 0-1 range (linear transformation so that the smallest element is zero and the largest element is 1). To do this, we must first subtract the smallest element from all elements and then divide the elements by the largest element.

```r
range01 <- function(x){
  y <- x - min(x)
  rval <- y/max(y)
  return(rval)
}

### Try out the function:
range01(x = c(10, 30, 15.2, 24, 110, 88))
```

```
## [1] 0.000 0.200 0.052 0.140 1.000 0.780
```

d) **Extra:** Extend the function from the previous exercise so that it not only returns the scaled vector but also the unscaled original vector.

```
range01 <- function(x){
  y <- x - min(x)
  vscl <- y/max(y)
  rval <- list(scaledVector = vscl,
               originalVector = x)
  return(rval)
}

### Try out the function:
range01(x = c(10, 30, 15.2, 24, 110, 88))

## $scaledVector
## [1] 0.000 0.200 0.052 0.140 1.000 0.780
##
## $originalVector
## [1]  10.0  30.0  15.2  24.0 110.0  88.0
```

5 `sample` function in R

There are many functions in R which make use of a random number generator. The `sample` function for example can be used to take a random sample from a vector. Try to guess what the following command does and run it multiple times:

```
sample(1:10, size = 5)
```

The above call takes a random sample of the vector `1:10`. Becaue it is a *random* sample the result can differ everytime we run the command. In order to make the results of R code including random processes reproducible, we can "fix" the random number generator by setting a "random seed". For example, run the two commands below multiple times after each other and observe the result:

```
set.seed(4984928)
sample(1:10, size = 5)
```

You should observe that the (random) result stays the same. The number which we put into the `set.seed` function can be chosen arbitrarily, each number fixes the random number generator at a different state.

a) Try to write a function which takes two input arguments: A data frame and a number. The function should then pick a random sample from the data frame (random selection of rows) of the size specified by the second input argument and return this new data frame containing the subsample.

```
subsamp <- function(x, size){
  ind <- sample(1:nrow(x), size = size)
  rval <- x[ind,]
  return(rval)
}

### Try out function:
set.seed(27273)
subsamp(cardata, size = 4)
```

```
##              car  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 10      Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 21 Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 12    Merc 450SE 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 30  Ferrari Dino 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

6 Exporting data

In the previous exercise we have created a function which takes a random subsample from a data frame. Apply the function to our `cardata` data frame to take a random subsample of it. Let's assume that we now want to export this subsample to a file for further storage. We can for example use the `write.csv` function to write a data frame into a .csv file. Use the command below to create a .csv file from our data frame (replace `NAME_OF_DATAFRAME` with the name of your data frame). The file will be saved in the working directory. Look in your working directory if the file was indeed created.

```
write.csv(NAME_OF_DATAFRAME, file = "data_prep.csv", row.names = FALSE)
```