R Basics

Andreas Herz, Till Krenz

2016

Contents

1. Calculate and Assigning Values

```
4+5+2.5
```

```
## [1] 11.5
```

```
12*12
```

```
## [1] 144
```

The arrow <- operator assigns a value to a variable.

```
a <- 4 + 5 + 2.5
b <- 12 * 12
```

The variable name returns the value.

```
a
```

```
## [1] 11.5
```

We can use variables in calculations.

```
a + b
```

```
## [1] 155.5
```

2. Working Directory

The working directory is a folder on your computer where all the data you are using in your R session is loaded from and stored to.

setwd() sets the working directory, *getwd()* returns the current working directory.

Excercise: Set your working directory to a folder were you want to save the data of this workshop. Check if the working directory was set correctly.

A setwd() command could look like this

```
setwd(c:\\Users\\Username\\Documents\\workshop)
```

Caution for Windows-Users: In the path to the folder, use one forward slash "/" or two backward slashes "\\"

R-Studio: You can also use the 'File' dialogue in the down right pane to navigate to any folder on your computer, click the 'More' button and select 'Set As Working Directory'.

R-GUI: Click on 'File' - 'Change Dir' in order to open a dialogue for selecting any folder on your computer as your working directory.

3. Read Data

Basic R provides commands for reading common file formats used to store data for statistical analysis. *read.csv()* can be used to read data that is stored as comma seperated values, if the values are seperated by semicolons use *read.csv2()*.

```
df <- read.csv2("01_egos.csv")
df
```

```
##   egoID sex      age netsize X1.to.2 X1.to.3 X1.to.4 X1.to.5 X2.to.3
## 1     1   w  36 - 45       5       1       2       2       0       0
## 2     2   w 66 - 100       5       2       1       1       1       2
## 3     3   w  56 - 65       5       2       1       2       1       2
##   X2.to.4 X2.to.5 X3.to.4 X3.to.5 X4.to.5
## 1       2       2       0       2       0
## 2       0       0       1       1       1
## 3       1       2       2       0       1
```

Data stored in R's native format 'Rda' can be read with *load()*.

```
load("01_stans_network.Rda")
```

R-Studio: An object named 'stans.network' should appear in the 'Evnironment' Tab.

4. Object Classes: vector, dataframe, list, graph

Objects can be of different classes. The most important ones for us are: vector, dataframe, list and graph.

Vector

c() generates vectors:

```r
c(1, 4, 7, 2)
```

```
## [1] 1 4 7 2
```

```r
vec <- c("Hello", "you", "!")
vec
```

```
## [1] "Hello" "you"   "!"
```

All entries in a vector need to be of the same class.

We can access a single value in a vector with squared brackets.

```r
vec[2]
```

```
## [1] "you"
```

Dataframe

The CSV file we loaded before is a dataframe

```r
class(df)
```

```
## [1] "data.frame"
```

A dataframe is R's version of a dataset. It consits of rows and colums, which represent cases and variables.

We can address each 'variable' in a dataframe either by using '$'

```r
df$age
```

```
## [1] 36 - 45  66 - 100 56 - 65
## Levels: 36 - 45 56 - 65 66 - 100
```

or by using squared brackets and quotation marks

```r
df[["netsize"]]
```

```
## [1] 5 5 5
```

The variables in a dataframe are vectors.

```r
is.vector(df$netsize)
```

```
## [1] TRUE
```

All vectors in a dataframe need to be of the same length.

We can replace variables in a dataframe:

```
df$sex <- c("w", "m", "w")
```

Or add new vairables:

```
df$exact.age <- c(39, 67, 60)
```

```
new.case <- c(4,"m","36 - 45", 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 44)
df <- rbind(df, new.case)
df
```

```
##   egoID sex      age netsize X1.to.2 X1.to.3 X1.to.4 X1.to.5 X2.to.3
## 1     1   w  36 - 45       5       1       2       2       0       0
## 2     2   m 66 - 100       5       2       1       1       1       2
## 3     3   w  56 - 65       5       2       1       2       1       2
## 4     4   m  36 - 45       3       0       0       0       0       0
##   X2.to.4 X2.to.5 X3.to.4 X3.to.5 X4.to.5 exact.age
## 1       2       2       0       2       0        39
## 2       0       0       1       1       1        67
## 3       1       2       2       0       1        60
## 4       0       0       0       0       0        44
```

If we work a lot with a certain dataframe, we can enclose our command in a *with()*, so that we don't need to write down the name of the dataframe for each variable.

```
with(df, data.frame(sex, exact.age, age))
```

```
##   sex exact.age      age
## 1   w        39  36 - 45
## 2   m        67 66 - 100
## 3   w        60  56 - 65
## 4   m        44  36 - 45
```

List

Lists are like vectors, but with less limitations. Each entry of list can be of a different class.

```
lst <- list(df, vec)
lst
```

```
## [[1]]
##   egoID sex      age netsize X1.to.2 X1.to.3 X1.to.4 X1.to.5 X2.to.3
## 1     1   w  36 - 45       5       1       2       2       0       0
## 2     2   m 66 - 100       5       2       1       1       1       2
## 3     3   w  56 - 65       5       2       1       2       1       2
## 4     4   m  36 - 45       3       0       0       0       0       0
##   X2.to.4 X2.to.5 X3.to.4 X3.to.5 X4.to.5 exact.age
## 1       2       2       0       2       0        39
## 2       0       0       1       1       1        67
## 3       1       2       2       0       1        60
## 4       0       0       0       0       0        44
## 
## [[2]]
## [1] "Hello" "you"   "!"
```

List entries are also accessed by squared brackets

```
lst[[2]]
```

```
## [1] "Hello" "you"   "!"
```

Caution: Always use double brackets [[]] to access list entries, otherwise you will end up with a single-entry list instead of the actual object.

Excercise: Bundle up all the previouly created objects in a list.

Graph

The igraph package provides support for storing, analysing and visualising network data. The object class used to store network data is 'igraph'.

```
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     decompose, spectrum
```

```
## Das folgende Objekt ist maskiert 'package:base':
##
##     union
```

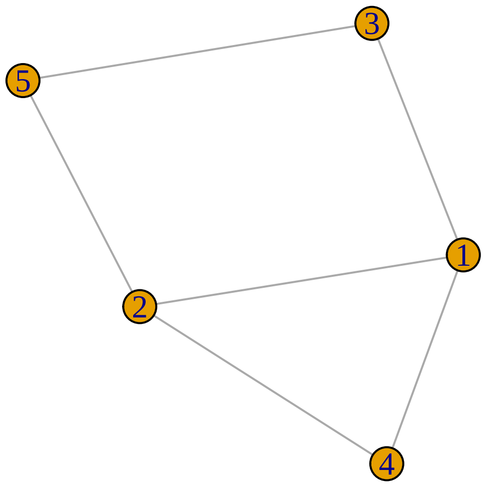
```
class(stans.network)
```

```
## [1] "igraph"
```

```
stans.network
```

```
## IGRAPH UNW- 5 6 --
## + attr: name (v/c), X (v/c), egoID (v/n), alterID.1 (v/c),
## | alter.sex (v/c), alter.age (v/n), weight (e/n)
## + edges (vertex names):
## [1] 1--2 1--3 1--4 2--4 2--5 3--5
```

```
plot(stans.network)
```



5. Base Functions and Packages

R Base provides a wide range of functions for calculating statistical measurments, as *mean()*, *median()*, *sd()*, *anova()*.

There are functions to do basic calculations easily, like *sum()* and *rowSums()*.

The *aggregate()* command is very useful, in order to calculate measurements for subgroups of a dataframe.

```r
aggregate(df$exact.age, by = df["sex"], FUN = mean)
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```

```
##   sex  x
## 1   m NA
## 2   w NA
```

Packages add all kinfs of functions to R. Packages stored on CRAN, the Comprehensive R Archive Network, can be installed with *install.packages()*.

```r
#install.packages("descr")
```

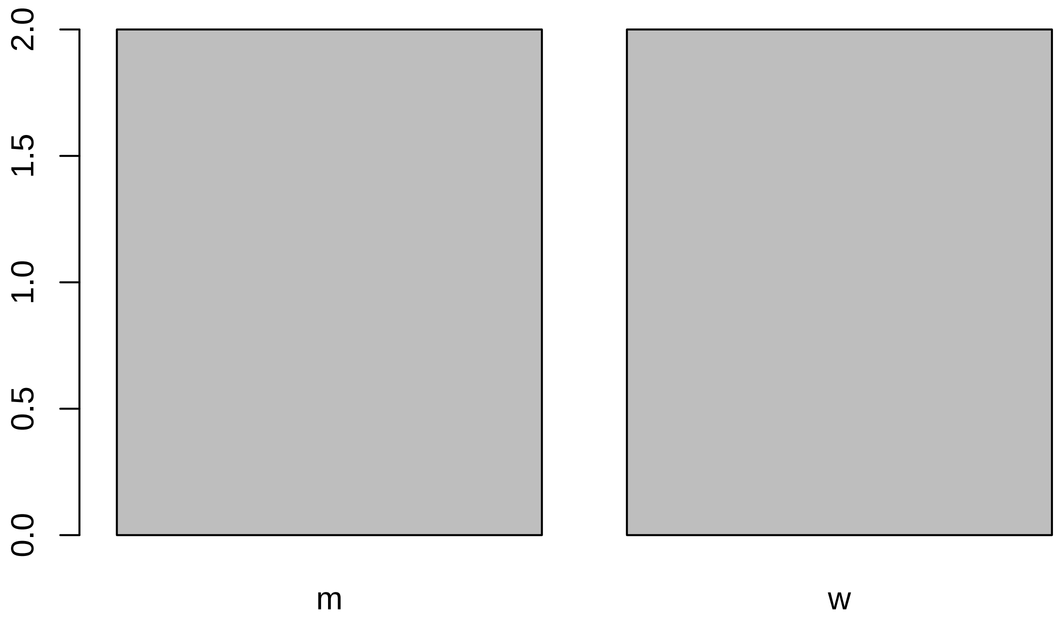
descr is a small package that provides functions for the display of univariat and bivariat frequency distributions.

Once a package is installed it can be loaded with *library()*, to make its functions available in our R session.

```r
library(descr)
```

```
## Warning: package 'descr' was built under R version 3.2.4
```

```r
freq(df$sex)
```



```
## df$sex
##       Frequency Percent
## m             2      50
## w             2      50
## Total         4     100
```

There are 8192 (April 2016) packages on CRAN providing all kinds of functions and procedures. Some packages that might be interesting for people adapting from another statistical software or language, are packages that faciltate the import of data from SPSS, Stata, SAS etc.

- i.e. readr, haven, foreign

There is a set of packages that attempt to make R code more straight forward and better fitted for everyday business of a data scientist, colloquially these are called the 'Hadleyverse' since Hadley Wickham ist the creator of most of these packages

- i.e. plyr, dplyr, margrittr, ggplot2, ggvis, tidyr

6. Save Data

Of course we can save our data on to the harddrive. *write.csv()* saves the data as a CSV file, making it easily available to use in other sofware. *save()* uses the Rda format.

```r
write.csv2(x = df, file = "df2.csv")
save(stans.network, df, lst, file = "saved_objects.Rda")
```