Statistical Computing with R

Lecture 3: installing and using R packages; functions (part 2); the working directory; importing, saving and exporting data

Mirko Signorelli
: mirkosignorelli.github.io
: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)

Universiteit Leiden

Recap

Lecture 2:

- R scripts and comments
- matrices (and linear algebra)
- data frames
- functions (part 1)
- consulting help pages

Today:

- installing and loading R packages
- using functions from R packages
- writing your own functions
- the working directory
- importing, saving and exporting data

Installing and using R packages

Writing your own functions

The working directory

Importing data

Saving / exporting data

Functions

3 "groups" of functions:

1. built-in ("base R") functions → available as soon as you install R
2. functions from R packages → not included in base R, they require you to install an R package
3. user-defined functions → you write your own functions!

We covered (1) during lecture 2. Today we will introduce (2) and (3).

Example: how to compute the skewness of a quantitative variable?

- Last week we saw how we can use `median( )`, `mean( )`, `var( )`, ... to compute some simple descriptive statistics in R:

```
set.seed(3001)
x1 = rpois(100, lambda = 10)
median(x1)
```

```
## [1] 9
```

```
mean(x1)
```

```
## [1] 9.63
```

```
var(x1)
```

```
## [1] 12.17485
```

Beyond mean and variance

But...

What if we want to know more about the distribution of `x1`?

Beyond mean and variance

But...
What if we want to know more about the distribution of `x1`?

For example: how can we determine
whether `x1` is symmetric or asymmetric?

Background: skewness

- Skewness measures the extent to which a variable is symmetric / asymmetric. It can be measured using the following index:

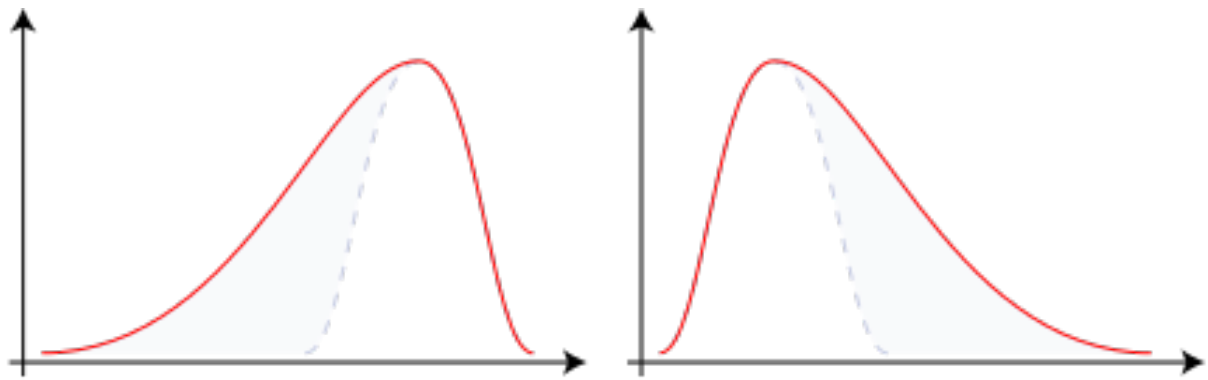
$$\gamma = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{\left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2\right]^{3/2}}, \text{ where } \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Background: skewness

- Skewness measures the extent to which a variable is symmetric / asymmetric. It can be measured using the following index:

$$\gamma = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{\left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2\right]^{3/2}}, \text{ where } \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- $\gamma = 0$: the distribution of X is symmetric
- $\gamma > 0$: the distribution of X is positively skewed (or ''right-skewed")
- $\gamma < 0$: the distribution of X is negatively skewed (or ''left-skewed")



How to compute skewness in R?

- Base R does not include a function to compute the skewness index
- However, chances are that other R programmers already wrote functions to compute it before you...
- ... and someone may have published their function to compute skewness!

R packages

- R is a collaborative project where people from all over the world can contribute code!
- R packages offer a standardized way to share functions, data and documentation with other R users
  1. Standard / official way to share functions with the whole R community: R packages published on official repositories such as CRAN and Bioconductor
  2. Informal ways to share code: scripts / R packages hosted on github / personal or insitutional webpages

⚠ Packages hosted on CRAN and Bioconductor need to comply with an extensive series of formal requirements. So, usually (but not always!) they are more reliable, stable and trustworthy than packages / functions from other sources ⚠

Installing and loading CRAN packages

1. After an online search, you find out that the R package `moments`, published on CRAN, contains a function called `skewness( )` that computes the skewness of a given variable
2. To use this function, you first need to install the `R` package from CRAN:

```
install.packages('moments')
```

3. Alternative way to install a CRAN package in RStudio: Tools → Install packages... → enter the name of the package(s) in the "packages" field
4. After successfully installing the package, you need to load it:

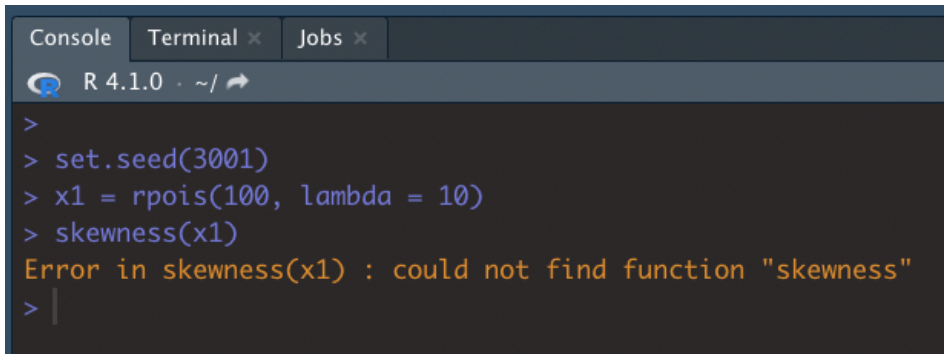
```
library(moments)
```

Installing vs loading R packages

What's the difference between installing and loading a package?

- `install.packages( )` downloads the R package to your computer, and adds its functions, data and documentation to your current R installation. You need to do this only once (until you update your R version)
- `library( )` loads the package into your current R session. You need to do this once for each R session in which you want to use the package (so: every time you restart R)

What happens if you don't load the package?



More about installing and loading R packages

- Still puzzled? Check this out:
  1. video: introduction to R packages
  2. R Packages: A Beginner's Tutorial

Back to our problem...

- Here's how we can compute the skewness of `x1`:

```
# 1: install the package (if you haven't installed it yet)
install.packages('moments')
```

```
# 2: load the package
library(moments)
# 3: use the function skewness( )
?skewness # check out the help page of the function
skewness(x1) # use the function
```

```
## [1] 0.4311006
```

Your turn

Exercises

1. Use your favourite search engine to find an `R` package that contains a function that can generate random numbers from the Tweedie distribution
2. Install that `R` package, and simulate 10 random numbers from a Tweedie distribution with mean parameter $= 4$, dispersion parameter $= 2$, and power parameter $= 1$

Solutions

```r
# Ex 1: I searched "tweedie distribution R" in google,
# and found out that the R package called tweedie
# contains a function called rtweedie :)
# Ex 2
# install the package tweedie
install.packages('tweedie')
```

```r
# load the package and check the help page
library(tweedie)
?rtweedie
# set a random seed and generate the random numbers
set.seed(5067)
rtweedie(n = 10, mu = 4, phi = 2, power = 1)
```

```
##  [1] 4 4 6 4 2 4 2 0 0 4
```

Installing and using R packages

Writing your own functions

The working directory

Importing data

Saving / exporting data

Creating a function

Functions can be created using `function( )` or `\( )`:

```
function.name = function(arg1, arg2) {
  # here you can specify what the function does
  # you can use as many rows as you need
  # and you can of course include comments
  2*arg1 - arg2
}
```

- `function.name = function(arg1, arg2) { }` creates a function named `function.name` that takes two arguments as input: `arg1` and `arg2`
- the code between `{` and `}` is the "body" of the function: it contains all the instructions that need to be executed when calling the function

Possible variations

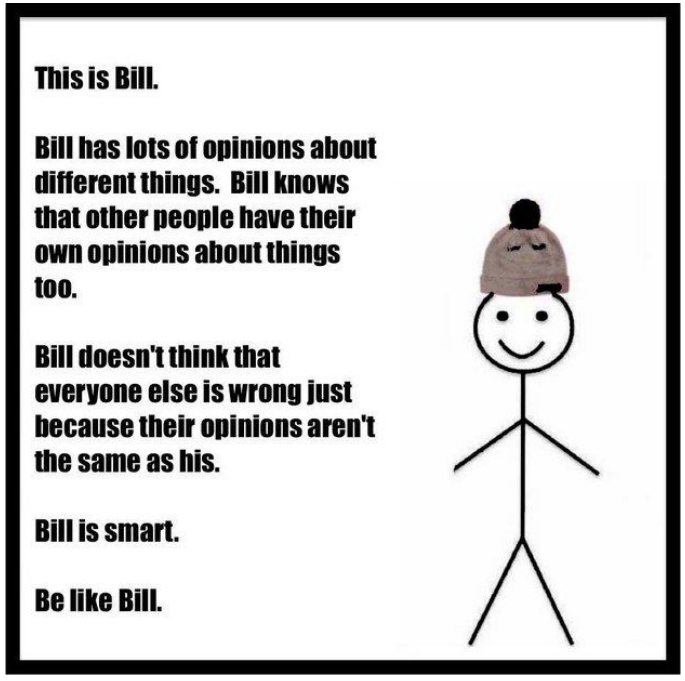
- The following 5 functions are equivalent (they have the same input, and produce the same output):

```
f1 = function(x, y) x^2 + 3*y
f2 = function(x, y) {
  x^2 + 3*y
}
f3 <- function(x, y) x^2 + 3*y
f4 = \(x, y) x^2 + 3*y
f5 <- \(x, y) {
  x^2 + 3*y
}
```

⚠ The shortcut `\( )` was introduced with R 4.0.0. It is not back-compatible, so: use it with care! (= don't use it if you need your code to be compatible with versions $< 4.0.0$!) ⚠

Coding and... Diversity!

- `R` often offers multiple ways to do (almost) the same thing
- Examples so far: `^` vs `**`, `=` vs `<-`, `function( )` vs `\( )`. More to come!
- Choosing one way over another is often a matter of personal preference, and there's no right or wrong choice
- Sometimes, ⚠ programmers can be quite judgemental ⚠ towards those who do things differently from them
- Don't judge people just because they do things differently from you. Be like Bill!



Functions spanning multiple lines

- Previous example: `x^2 + 3*y` is a very simple function body (it fits in one line!)
- Function bodies are usually (much) more complex than this
- Simple example: write a function to compute the sample variance (based on $Var(X) = E(X^2) - E(X)^2$)

$$\hat{s}^2 = \frac{n}{n-1}\left[\frac{1}{n}\sum_i x_i^2 - \left(\frac{1}{n}\sum_i x_i\right)^2\right] \tag{1}$$

- We can break (1) into 3 steps:
  1. Estimate $E(X^2)$
  2. Estimate $E(X)$
  3. Compute $\hat{s}^2$

Function to estimate the sample variance

```r
sample_var = function(x) {
  if (!is.numeric(x)) stop('x should be a vector of
                           type: numeric')
  mean.x2 = mean(x^2, na.rm = T)
  mean.x = mean(x, na.rm = T)
  n = sum(!is.na(x))
  (mean.x2 - mean.x^2)*n/(n-1)
}
```

R function for variance: `var( )` $\Rightarrow$ I named my function `sample_var( )`

⚠ Good coding practice: avoid naming your function with the same name of an existing R function to prevent naming conflicts ⚠

Check: is our function correct?

```
v1 = rnorm(100, sd = 2)
sample_var(v1)
```

```
## [1] 4.210338
```

```
var(v1)
```

```
## [1] 4.210338
```

```
v2 = rpois(100, lambda = 5)
sample_var(v2)
```

```
## [1] 5.825758
```

```
var(v2)
```

```
## [1] 5.825758
```

Using { } and `return( )`

- If the body of a function spans multiple lines:
  1. you cannot omit `{ }`
  2. the output of the function is the last line of code that is executed, e.g. `(mean.x2 - mean.x^2)*n/(n-1)` in `sample_var( )`
- You may use `return(output)` at the end of the function to explicitly specify what the output is:
  1. useful if output requires more than 1 line of code to be created
  2. useful also to make clear and explicit what the output is going to be

Using return( )

```
mean.and.var = function(x) {
  if (!is.numeric(x)) stop('x should be a vector of
                           type: numeric')
  mean.x2 = mean(x^2, na.rm = T)
  mean.x = mean(x, na.rm = T)
  n = length(x)
  s2 = (mean.x2 - mean.x^2)*n/(n-1)
  output = c('sample.mean' = mean.x,
             'sample.var' = s2)
  return(output)
}

mean.and.var(v1)
```

```
##  sample.mean   sample.var
## -0.006842648  4.210337521
```

FAQ: when should I use `return( )`?

FAQ: when should you use `return( )`?

- Default function output is the last line of code that gets executed in `R`
- If you want the output to be different from that, then you can use `return( )` to specify what the function should output
- You **may** also choose to use `return( )` just to make more apparent what the function output is (might be redundant, but it doesn't hurt)

Default values

- You can specify default values for some / all function arguments:

```
f1 = function(vec, mult.fac = 3, na.rm = T) {
  mult.fac*mean(vec, na.rm = na.rm)
}
```

- Only argument that **must** be supplied to `f1( )`: `vec` (no default)

```
f1(c(3, 2))
```

```
## [1] 7.5
```

```
f1(c(3, 2), 2)
```

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

```
f1(c(2, 3, NA))
```

```
## [1] 7.5
```

```
f1(c(2, 3, NA), na.rm = F)
```

```
## [1] NA
```

Order of the inputs

```
f2 = function(x, y, z) 2*x + 3*y - 4*z
```

- When applying a function, you can explicitly name the inputs in any order you want:

```
f2(y = 4, x = -3, z = pi)
```

```
## [1] -6.566371
```

- If you don't specify (some of) the input names, R will assign the unnamed inputs in the order you provided them

```
f2(1, 3, 2) # here: x = 1, y = 3, z = 2
```

```
## [1] 3
```

```
f2(y = 1, 3, 2) # here: x = 3, y = 1, z = 2
```

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

Your turn

Exercises

1. Write a function that takes as input a person's name, and greets them with "Good morning [person name]! How are you doing?". If no name is provided as input, the function should greet you.
2. Write a function that receives as input a numeric vector, and returns as output its mean, median, min and max. Include as argument the `na.rm` option, and set its default to `TRUE`

Solutions

```
# Ex 1
greet = function(name = 'Mirko') {
  paste('Good morning ', name,
        '. How are you doing?', sep = '')
}
greet('Fernanda')
```

```
## [1] "Good morning Fernanda. How are you doing?"
```

```
greet()
```

```
## [1] "Good morning Mirko. How are you doing?"
```

Solutions (cont'd)

```r
# Ex 2
four.summaries = function(v, na.rm = T) {
  x1 = mean(v, na.rm = na.rm)
  x2 = median(v, na.rm = na.rm)
  x3 = min(v, na.rm = na.rm)
  x4 = max(v, na.rm = na.rm)
  return(c('mean' = x1, 'median' = x2,
           'min' = x3, 'max' = x4))
}

# let's check that the function works!
four.summaries(c(3, NA, 15, 7, 2))
```

```
##    mean median     min     max
##    6.75   5.00    2.00   15.00
```

```r
four.summaries(c(3, NA, 15, 7, 2), na.rm = F)
```

```
##    mean median     min     max
##      NA     NA      NA      NA
```

Installing and using R packages

Writing your own functions

The working directory

Importing data

Saving / exporting data

Absolute paths, relative pahts and `getwd( )`

Absolute vs relative paths:

- An absolute path specifies the location of a file / folder from the root directory

```
'C://documents/mydata/apples.csv' # Windows example
'/Users/john/Desktop/mydata' # MacOS example
```

- A relative path does not specify the path from the root

```
'apples.csv'
'Desktop/mydata'
```

When you supply relative paths, `R` imports data from, and saves them to, the current **working directory**

- Use `getwd()` to locate the current directory:

```
getwd()
```

```
## [1] "/Users/ms"
```

Changing the working directory

- The working directory can be changed using `setwd( )`:

```
setwd('C://documents/mydata/')
```

- In RStudio, it can also be changed using: Session -> Set working directory (NB: this change is executed in the Console. Copy it to your script if you want the change to apply each time you rerun the script!)

⚠ The change of working directory is not permanent: it only affects your current `R` session! ⚠

Nice to know

- Use `setwd('..')` to "move up" of one folder:

```
getwd()
```

```
## [1] "/Users/ms/Desktop"
```

```
setwd('..')
getwd()
```

```
## [1] "/Users/ms"
```

- Alternative way to manage working directories: RStudio projects (we will cover this later in the course)

Your turn

Exercises

1. Create a subfolder for today's lecture material. Example absolute path: `/Users/ms/Desktop/SCwR/lecture3` (replace it with yours!)
2. Create an `R` script, and save it in that subfolder
3. At the beginning of the script, set the working directory to today's subfolder with

```
setwd('/Users/ms/Desktop/SCwR/lecture3') # change the path
# as appropriate!
```

Why do we need to do this?

- Why do we need to set the working directory?
- To tell `R`:
  1. where to find a file with data that we want to read
  2. where to save results to an external data file
  3. where to save outputs, for example images (pdf/png/jpg files. . .)

Installing and using R packages

Writing your own functions

The working directory

Importing data

Saving / exporting data

Loading your data in R

The file `polls_Germany_30072021.csv` contains data on political opinion polls performed in Germany up until the end of July 2021:

polls_Germany_30072021

Polling Firm	Commissioners	Fieldwork Start	Fieldwork End	Scope	Sample Size	Sample Size Qualification	Participation	Precision	Union	SPD	Alternative für Deutschland	FDP	DIE LINKE	BÜNDNIS 90/DIE GRÜNEN	CDU	CSU	Pir
GMS		2021-07-21	2021-07-27	National	1003	Provided	Not Available	1%	30%	15%	10%	12%	7%	18%	Not Available	Not Available	No
INSA and YouGov		2021-07-23	2021-07-26	National	2007	Provided	Not Available	1%	Not Available	17.5%	12%	13%	6%	17.5%	22%	5%	No
Forsa		2021-07-20	2021-07-26	National	2501	Provided	75%	1%	26%	15%	10%	13%	7%	21%	Not Available	Not Available	No
INSA and YouGov		2021-07-19	2021-07-23	National	1316	Provided	Not Available	1%	27%	17%	11%	13%	7%	18%	Not Available	Not Available	No
Allensbach		2021-07-03	2021-07-22	National	1243	Provided	Not Available	0.5%	30%	16%	9.5%	12%	7%	19.5%	Not Available	Not Available	No
Infratest dimap		2021-07-20	2021-07-21	National	1188	Provided	Not Available	1%	29%	16%	10%	12%	6%	19%	Not Available	Not Available	No
Kantar		2021-07-14	2021-07-20	National	1421	Provided	Not Available	1%	28%	16%	11%	12%	7%	19%	Not Available	Not Available	No
INSA and YouGov		2021-07-16	2021-07-19	National	2064	Provided	Not Available	0.5%	Not Available	16.5%	11.5%	12%	6%	18%	23.5%	5.5%	No
Forsa		2021-07-13	2021-07-19	National	2503	Provided	78.0%	1%	28%	16%	10%	12%	7%	19%	Not Available	Not Available	No
INSA and YouGov		2021-07-12	2021-07-16	National	1354	Provided	Not Available	1%	28%	17%	11%	12%	7%	18%	Not Available	Not Available	No
Forschungsgruppe Wahlen		2021-07-13	2021-07-15	National	1224	Provided	Not Available	1%	30%	15%	10%	10%	7%	20%	Not Available	Not Available	No
Allensbach		2021-07-03	2021-07-14	National	1028	Provided	Not Available	0.5%	31.5%	16.5%	9.5%	12%	6.5%	18%	Not Available	Not Available	No
Kantar		2021-07-07	2021-07-13	National	1495	Provided	Not Available	1%	28%	15%	11%	11%	8%	20%	Not Available	Not Available	No
INSA and YouGov		2021-07-09	2021-07-12	National	2087	Provided	Not Available	0.5%	Not Available	17%	11%	12.5%	7%	17%	23%	5%	No
Forsa		2021-07-06	2021-07-12	National	2502	Provided	78.0%	1%	30%	15%	9%	12%	7%	19%	Not Available	Not Available	No
INSA and YouGov		2021-07-05	2021-07-09	National	1352	Provided	Not Available	1%	28%	17%	11%	12%	8%	17%	Not Available	Not Available	No
Kantar		2021-06-29	2021-07-06	National	1405	Provided	Not Available	1%	29%	15%	11%	11%	8%	19%	Not Available	Not Available	No

Important features of this file:

- relative path on my laptop: `data/polls_Germany_30072021.csv`
- the first row contains the variable names
- the file is a CSV file, with commas used as separators

Functions for data import

- A selection of functions that you can use to import data in R:

Function	R package	File type
`read.table`	base R	General (.txt, .csv, .dat, ...)
`read.csv`	base R	.csv (Comma Separated Value)
`read_excel`	readxl	.xls, .xslx (Excel files)
`read_ods`	readODS	.ods (Open Document Spreadsheets)
`read.spss`	foreign	.sav (SPSS files)
`read.dta`	foreign	.dta (Stata files)
`read.sas7bdat`	sas7bdat	.sas7bdat files (SAS files)

- Useful tutorial from Karlijn Willems (DataCamp)

read.table( )

How to import polls_Germany_30072021.csv?

- Option 1: read.table( )

```
df1 = read.table('data/polls_Germany_30072021.csv',
                 sep = ',', header = T)
# what type of object is df1?
is(df1)
```

```
## [1] "data.frame" "list"       "oldClass"   "vector"
```

```
# let's have a look at a selection of the data
df1[1:3, c(1, 6, 10, 11)]
```

```
##      Polling.Firm Sample.Size          Union   SPD
## 1             GMS        1003            30%   15%
## 2 INSA and YouGov        2007 Not Available 17.5%
## 3           Forsa        2501            26%   15%
```

read.csv( )

How to import polls_Germany_30072021.csv?

- Option 2: read.csv( )

```
df2 = read.csv('data/polls_Germany_30072021.csv')
# what type of object is df1?
is(df2)
```

```
## [1] "data.frame" "list"       "oldClass"   "vector"
```

```
# let's have a look at a selection of the data
df1[1:2, c(1, 6, 10, 11)]
```

```
##      Polling.Firm Sample.Size          Union   SPD
## 1             GMS        1003            30%   15%
## 2 INSA and YouGov        2007 Not Available 17.5%
```

⚠ Here we don't need to specify header and sep! Why? ⚠

Installing and using R packages

Writing your own functions

The working directory

Importing data

Saving / exporting data

save( ), save.image( ) and load( )

- Typical extension of R data files: .RData
- Use save( ) to save specific objects present in your workspace

```
save(df1, v2, file = 'folder/filename.RData')
```

- Use save.image( ) to save all objects in your workspace

```
save.image('folder/filename.RData')
```

- Use load( ) to load an .Rdata file in R

```
load('folder/filename.RData')
```

Exporting data

- `.Rdata` files useful within `R`, but not suitable for usage with other software
- You may need to save a data frame in a format other than `.RData`. Some useful functions to export data:

Function	R package	File type
`write.table`	base `R`	General (.txt, .csv, .dat, ...)
`write.csv`	base `R`	.csv (Comma Separated Value)
`write.xlsx`	xlsx	.xls, .xslx (Excel files)
`write_ods`	readODS	.ods (Open Document Spreadsheets)