Environment, Packages, Functions, Import/Export

Enrico Toffalini

Install packages… and more

Traditional installing of a package from CRAN:

```
install.packages("effsize")
```

installing of multiple packages from CRAN at once:

```
install.packages( c("effsize","psych","ggplot2") )
```

For development or personal use, you may occasionally install packages from outside CRAN, such as from GitHub:

```
devtools::install_github("FilippoGambarota/filor")
devtools::install_github("EnricoToffalini/toffee")
```

After installing, you need to load the packages using function `library`:

```
library(effsize)
library(ggplot2)
```

Install packages… and more

call functions

After loading a package, its functions are directly callable throughout the R session:

```
library(psych)

fisherz(rho=0.8) # use a function from the "psych" package
```

```
[1] 1.098612
```

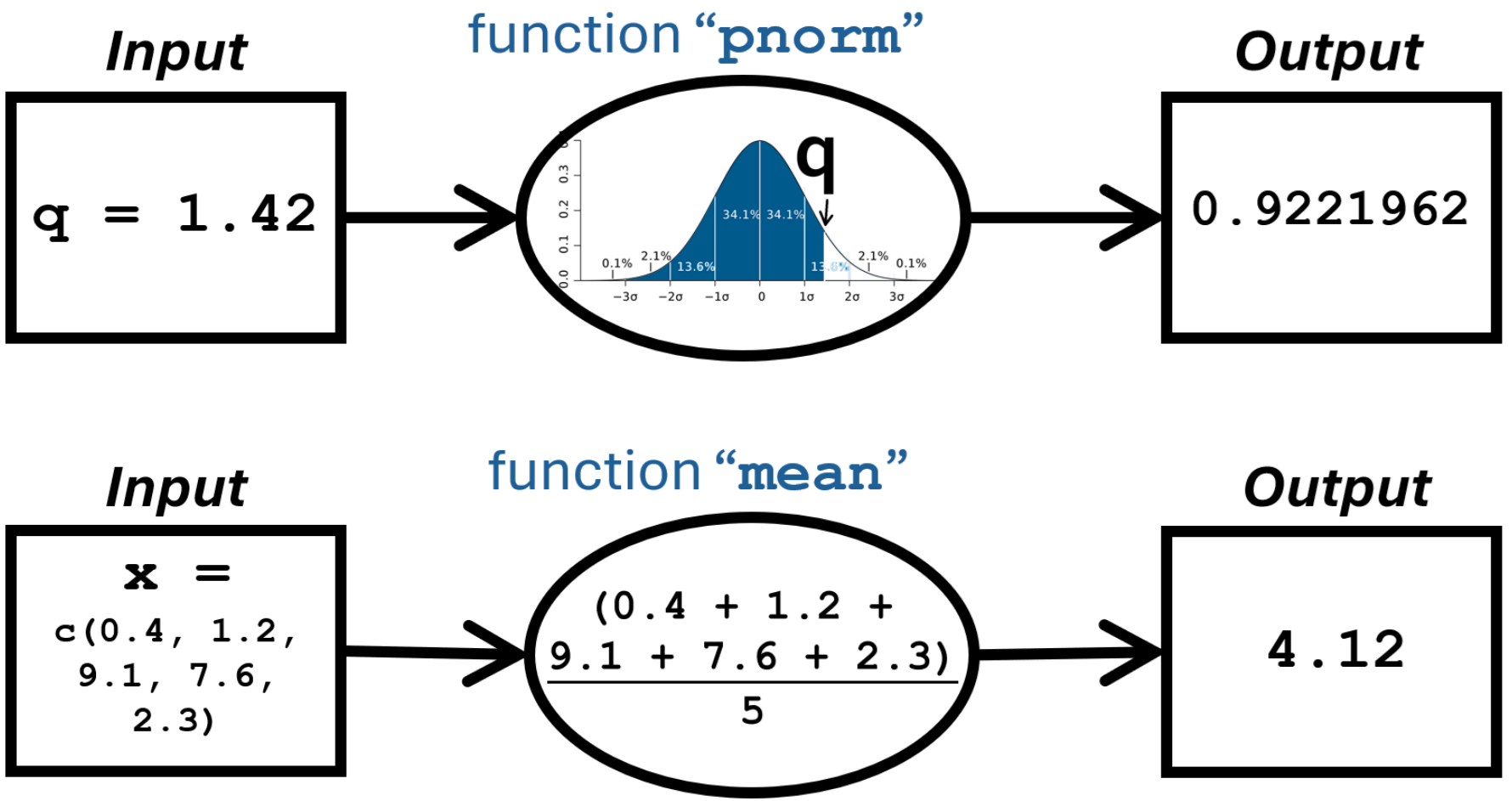
you may directly call any function from any installed package, even without loading it, using "`::`"; this is especially useful when there is a risk of functions with conflicting names, or if you just don't want to load an entire package for using a single function:

```
psych::fisherz(rho=0.8)
```

```
[1] 1.098612
```

Functions and arguments

Functions typically take some ***input*** parameters, known as *arguments*, process that, and yield some ***output***/result(s)



Functions and arguments

arguments

- values or variables you pass to a function as input, or to control its behavior

for example, `seq()` generates a sequence of numbers; "`from`" and "`to`" are arguments: it will provide the integers between these two extremes:

```
seq(from = 3, to = 7)
```

```
[1] 3 4 5 6 7
```

`length.out` controls how many equally spaced numbers must be generated:

```
seq(from = 3, to = 7, length.out = 4)
```

```
[1] 3.000000 4.333333 5.666667 7.000000
```

alternatively, `by` defines the step size between numbers:

```
seq(from = 3, to = 7, by = 0.6)
```

```
[1] 3.0 3.6 4.2 4.8 5.4 6.0 6.6
```

Functions and arguments

arguments

- values or variables you pass to a function as input, or to control its behavior

`rnorm()` will generate "`n`" random numbers from a normal distribution with "`mean`" as the average and "`sd`" as the standard deviation:

```
rnorm(n = 5, mean = 100, sd = 15)
```

```
[1]  78.03473  93.76165  87.50103 105.93306  88.99768
```

Positional matching - know that arguments names may be omitted if placed in the correct order

```
rnorm(5, 100, 15)
```

```
[1] 105.70539 115.89619  98.42592 124.04761  94.47411
```

Functions and arguments

Default arguments - a function *might* still work even if some arguments are omitted, as it can use its own *default values* (in this case "`mean=0, sd=1`")

```
rnorm(n = 5)
```

```
[1]  1.10281492 -1.24614283  0.29678721 -0.50821066 -0.08600416
```

Errors - however, omitting mandatory arguments will result in an *Error*

```
rnorm(mean = 100, sd = 15)
```

```
Error in rnorm(mean = 100, sd = 15): argument "n" is missing, with no default
```

Warnings - Some inputs may cause the function to produce *Warnings* and bad output, but do **not** stop code execution

```
rnorm(n = 5, mean = 100, sd = -15)
```

```
Warning in rnorm(n = 5, mean = 100, sd = -15): NAs produced
```

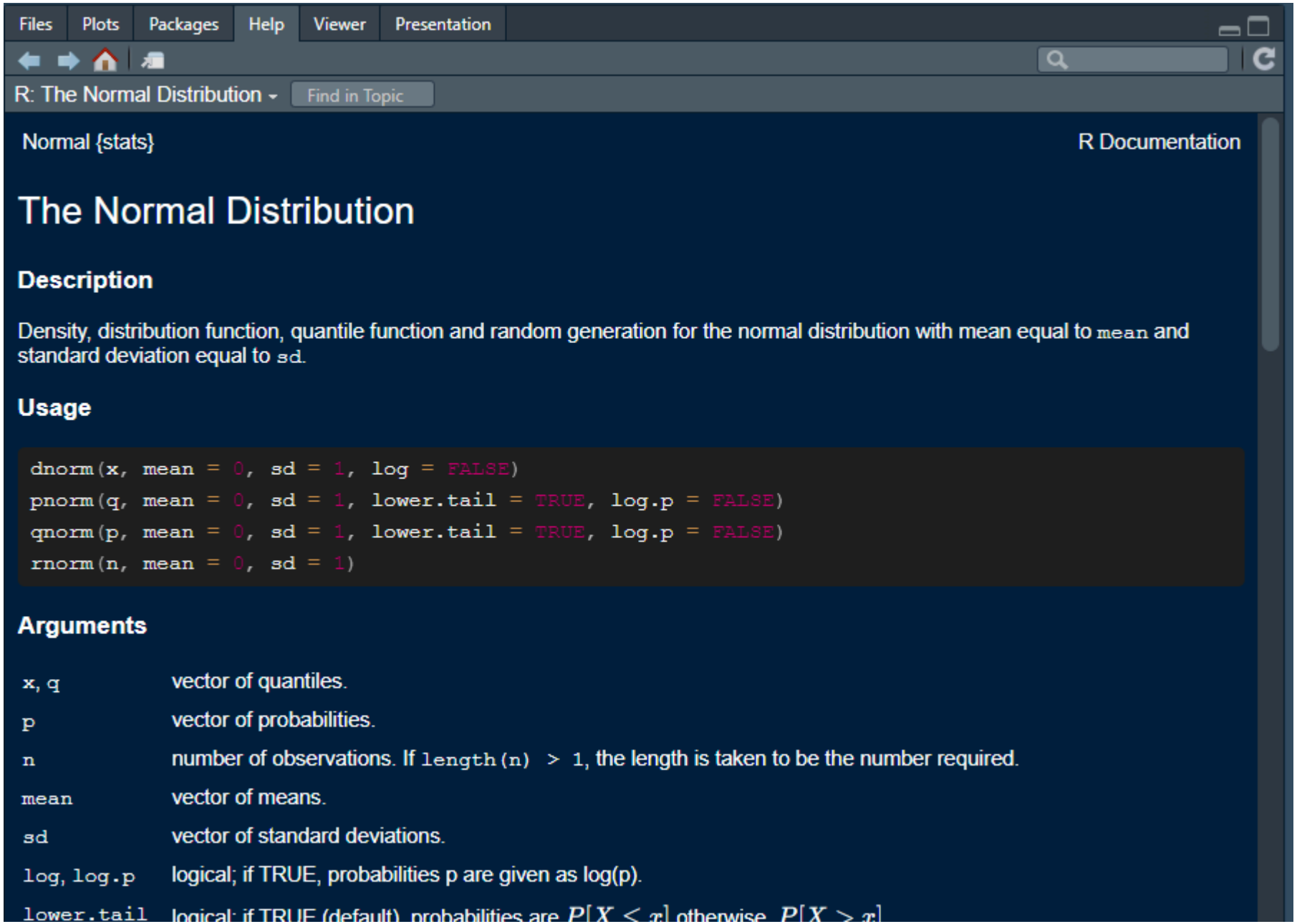
```
[1] NaN NaN NaN NaN NaN
```

Functions and arguments

HELP! see the documentation of a function

There are two ways to access documentation: using "`?`" and using `help()`

```
?rnorm # this will work
help(rnorm) # this does the same
```


Files
Plots
Packages
Help
Viewer
Presentation
R: The Normal Distribution
Find in Topic
Normal {stats}
R Documentation
The Normal Distribution
Description
Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.
Usage
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
Arguments
x, q vector of quantiles.
p vector of probabilities.
n number of observations. If length(n) > 1, the length is taken to be the number required.
mean vector of means.
sd vector of standard deviations.
log, log.p logical; if TRUE, probabilities p are given as log(p).
lower.tail logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise, $P[X > x]$

Set up Working Directory

The **Working Directory** (WD) is the location of the folder in your computer where R reads and saves files by default.

If you import/export anything (*data*, *figures*, *workspaces*, etc.) you need to know your WD!

The `getwd()` function allows you to display the location of your current WD. Let's see my own:

```
getwd()
```

```
[1] "C:/Users/enric/Desktop/Basics R DataScience/Slides"
```

Set up Working Directory

As a general rule:

- When you open R or the RStudio app, the default WD may be the `documents` folder (in Windows) or the `home directory` (e.g., `/home/username`; in Linux or macOS);
- This default may be reset at any time from inside RStudio on `Tools > Global Options... > General`;
- When RStudio is newly open by opening a file (e.g., a `.R` script file), the WD may be set at that file location *(actually my favorite)*;
- However, you can set a new WD at any time from within the R code, using the `setwd()` function, for example:

```
setwd( "C:/Users/enric/" )
```

Set up Working Directory

RStudio Projects may eliminate the need of using `setwd()` within scripts.

- You can create a *new project* with `File > New Project...` choose a specific folder
- Keep all materials of your project in the same folder as the newly created `.Rproj` file
- As you open the `.Rproj`, it will automatically start a new *RStudio* session with the WD set into that folder.

Set up Working Directory

Finally, *not vital for now*, but know the difference between:

- **Absolute path**: `"C:/Users/enric/"` indicates the full directory path from the root
- **Relative paths**: for import/export purposes you may move around the current WD
  - for example `png(filename="figures/Fig1.png")` may save $Fig1.png$ into the $figures$ directory which is **inside** the current WD;
  - differently, `png(filename="../figures/Fig1.png")` may save $Fig1.png$ into the $figures$ directory which is **outside, one level up** the current WD

Import/export

Now let's see how to perform **import/export** operations for:

- **The Workspace**: all objects that exist in your current R session, all results and computations stored so far (see them in the *"Environment" panel* or with `ls()`);
- **Data: SUPER IMPORTANT!** we will focus especially on tabular (Excel-like) data, that we treat as dataframes;
- **Figures**: save your plots for reports and more in `.pdf`, `.png`, and more formats.

Import/export

Workspace

All your R code (script) is generally stored in text files with a `.R` extension. But where do you save your results and objects?!

You can export the entire ***workspace*** (with all your objects) using the **`save.image()`** function:

```
# let's populate the workspace first
myName = "Enrico"
prof = TRUE
coursesTaught = 4L
age = 36

# now let's save it
save.image("myWS.RData")
```

*Specifying `"myWS.RData"` is not mandatory but recommended, otherwise your file will simply be named `".RData"`. (By the way… **where** will it be saved?)*

Import/export

Workspace

Alternatively, you may even save just one or a few workspace objects, rather than all:

```
# let's populate the workspace first
myName = "Enrico"
prof = TRUE
coursesTaught = 4L
age = 36

# now let's save only two objects
save(myName, age, file="myWS.RData")
```

This will save only variables `myName` and `age` into a newly created file named `myWS.RData`

This may be useful when you have an overcrowded workspace and prefer to save only a few objects that store the final results

Import/export

Workspace

Once you open a new R session, you may load the previously stored workspace using the `load()` function, specifying `load("workspace_name.RData")`, like this:

```
# empty the workspace to make sure there's actually nothing!
rm(list=ls())
ls()

# now load the previously saved workspace
load("myWS.RData")
# make sure that the objects have been loaded
ls()
```

Import/export

Data

Arguably a **fundamental skill** for anyone working in data science!

Most people use *MS Excel* or similar software (e.g., *LibreOffice Calc*) for handling data, which produce their own file formats (e.g., `.xlsx`). That's perfectly fine. However… the **most versatile data format is `.csv` (comma-separated values)**, a simple text (no formatting, no licences required) file format for storing tabular data/dataframes.

- **Best practice**: Save data in `.csv` format from your software of choice before importing it in R.

Import/export

Data

Here's an example of using the **`read.csv()`** function for **importing** data:

```
# IMPORT csv data from a "data" subfolder, and store it in an object named "df"
df = read.csv("data/Performance.csv", header=TRUE, sep=",", dec=".")

head(df) # have a look at the first few rows
```

```
  id  name anx acc     time
1  1 nydga  20  15 2.077932
2  2 bwknr  14   9 2.436858
3  3 sauuj  18  12 2.549814
4  4 vnjgi  27  15 4.386718
5  5 oueiy  21  11 5.248933
6  6 neebj  12  13 3.463094
```

Actually, specifying "`header=TRUE, sep=",", dec="."`" is unnecessary and could be omitted because it is the default… but it may be useful to get accustomed with functions arguments; also, in Italian Excel export settings, it is possible that *separator character* (`sep`) be "`;`", and *decimal point character* be "`,`" so… be aware of your settings!

Import/export

Data

If you absolutely want to **import** your data directly from a **MS Excel** document (`.xlsx`), you may use function **`read_excel()`** from the package `readxl`:

```
library(readxl)
df = data.frame( read_excel("data/Performance.xlsx") )
# data.frame() forces it to be a dataframe, otherwise it's a tibble
head(df)
```

```
  id  name anx acc     time
1  1 nydga  20  15 2.077932
2  2 bwknr  14   9 2.436858
3  3 sauuj  18  12 2.549814
4  4 vnjgi  27  15 4.386718
5  5 oueiy  21  11 5.248933
6  6 neebj  12  13 3.463094
```

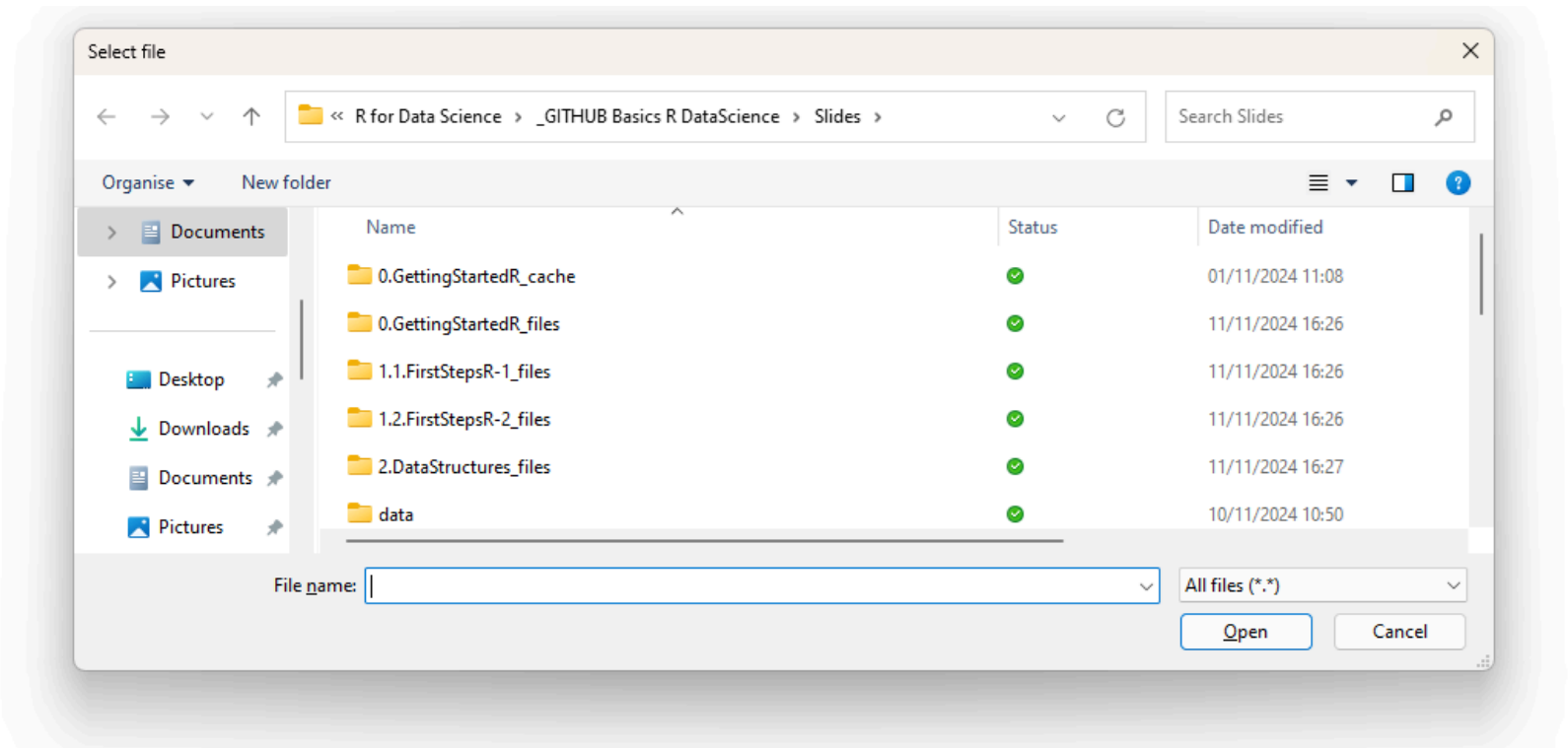
- You may even import data from an **SPSS** document (`.sav`) using the `read.spss()` function from the `foreign` package

Import/export

Data

A good trick if you don’t want to specify any relative or absolute path, and want to manually select data each time, is using the `file.choose()` function:

```
df = read.csv(file.choose(), header=TRUE, sep=",", dec=".")
```



Import/export

Data

Other “tricks” for importing data involve using the functions in the **RStudio menu**, particularly:

- File > Import Dataset > From text (base)…
- File > Import Dataset > From Excel
- File > Import Dataset > From SPSS…

However … using these functions is not best practice, because they are specific to the RStudio IDE. **It’s better to use code for reproducibility**

Import/export

Data

You have processed data with R, now… how to **export** it?

When collaborating with someone also using R, you may choose to exchange data directly by exporting the object or the entire workspace as a `.RData` file, using the `save()` or `save.image()` function respectively.

However, if you need to export your data in a more universally readable tabular format, such as `.csv`, you may use `write.table()`:

```
# specify the dataframe to export (here named "df")
# along with the desired file name, and other arguments

write.table(df, file="myExportedData.csv", sep=",", row.names=F)
```

Import/export

Figures

R has a collection of functions for exporting figures in different formats: `pdf()`, `png()`, `jpeg()`, `bmp()`, `tiff()`, `svg()`.

Here is an example using `png()` :

```
# set up a graphic output file named "MyFigure.png" with some settings
png("MyFigure.png", height=1500, width=2000, units="px", res=300)

# code for creating a simple boxplot
boxplot(iris$Sepal.Width)

# close the graphic output file and actually export the plot
dev.off()
```