Getting Started

Alexia Cardona

What is R?

R is a statistical programming language. It is very popular in the Data Science field, including Bioinformatics. The term “ `R` ” is used to refer to both the programming language and the software that interprets the scripts written using it.

Why learn R?

R does not involve lots of pointing and clicking

The learning curve might be steeper than with other software, but with R, the results of your analysis do not rely on remembering a succession of pointing and clicking, but instead on a series of written commands, and that’s a good thing! So, if you want to redo your analysis because you collected more data, you don’t have to remember which button you clicked in which order to obtain your results; you just have to run your script again.

Working with scripts makes the steps you used in your analysis clear, and the code you write can be inspected by someone else who can give you feedback and spot mistakes.

Working with scripts forces you to have a deeper understanding of what you are doing, and facilitates your learning and comprehension of the methods you use.

R code is great for reproducibility

Reproducibility is when someone else (including your future self) can obtain the same results from the same dataset when using the same analysis.

R integrates with other tools to generate reports from your code. If you collect more data, or fix a mistake in your dataset, the figures and the statistical tests in your manuscript are updated automatically after running the code again.

An increasing number of journals and funding agencies expect analyses to be reproducible, so knowing R will give you an edge with these requirements.

R is interdisciplinary and extensibe

With 10,000+ packages that can be installed to extend its capabilities, R provides a framework that allows you to combine statistical approaches from many scientific disciplines to best suit the analytical framework you need to analyze your data. For instance, R has packages for image analysis, GIS, time series, population genetics, and a lot more.

R works on data of different sizes

The skills you learn with R scale easily with the size of your dataset. Whether your dataset has hundreds or millions of lines, it won’t make much difference to you.

R is designed for data analysis. It comes with special data structures and data types that make handling of missing data and statistical factors convenient.

R can connect to spreadsheets, databases, and many other data formats, on your computer or on the web.

R produces high-quality graphics

The plotting functionalities in R are endless, and allow you to adjust any aspect of your graph to convey most effectively the message from your data.

R has a large and welcoming community

Thousands of people use R daily. Many of them are willing to help you through mailing lists and websites such as Stack Overflow (https://stackoverflow.com/), or on the RStudio community (https://community.rstudio.com/).

Not only is R free, but it is also open-source and cross-platform

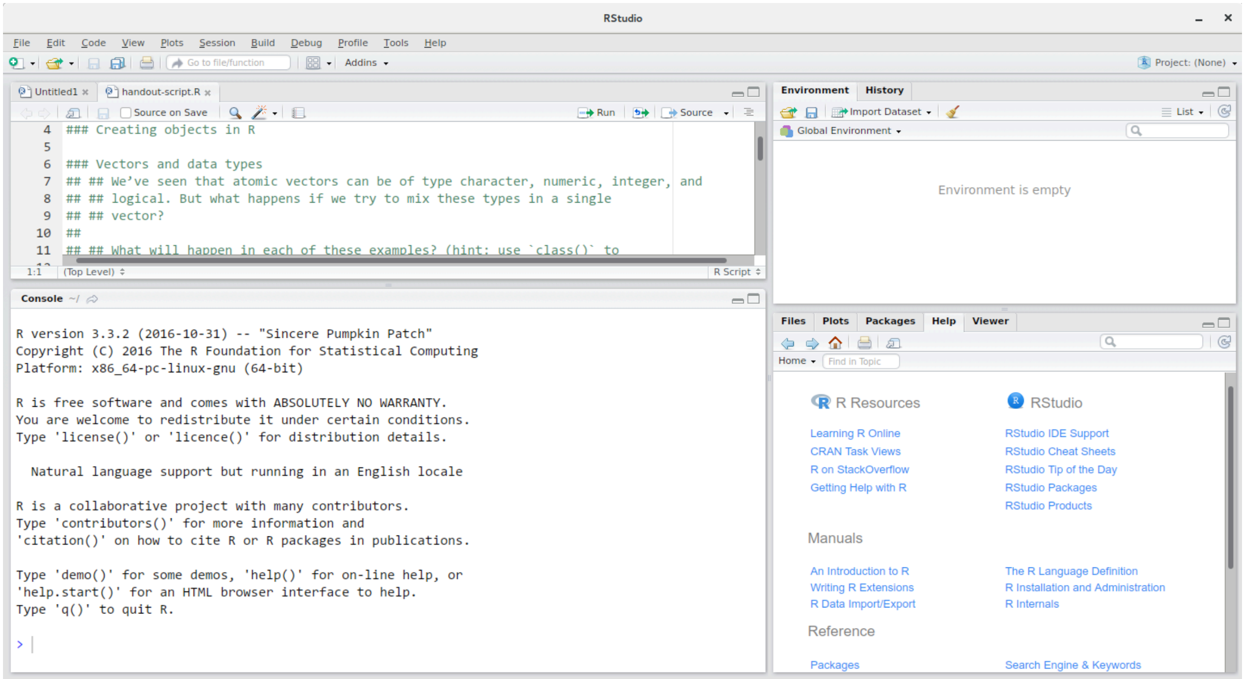
Anyone can inspect the source code to see how R works. Because of this transparency, there is less chance for mistakes, and if you (or someone else) find some, you can report and fix bugs.

What is RStudio?

RStudio (https://rstudio.com) is currently a very popular Integrated Development Environment (IDE) for working with R. An IDE is an application used by software developers that faciliates programming by offering source code editing, building and debugging tools all integarated into one application. To function correctly, RStudio needs R and therefore both need to be installed on your computer.

The RStudio Desktop open-source product is free under the Affero General Public License (AGPL) v3 (https://www.gnu.org/licenses/agpl-3.0.en.html). Other versions of RStudio (https://www.rstudio.com/products/rstudio/) are also available.

We will use RStudio IDE to write code, navigate the files on our computer, inspect the variables we are going to create, and visualize the plots we will generate. RStudio can also be used for other things (*e.g.,* version control, developing packages, writing Shiny apps) that we will not cover during the workshop.



```
### Creating objects in R

### Vectors and data types
## ## We’ve seen that atomic vectors can be of type character, numeric, integer, and
## ## logical. But what happens if we try to mix these types in a single
## ## vector?
##
## ## What will happen in each of these examples? (hint: use `class()` to
```

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

RStudio interface screenshot. Clockwise from top left: Source, Environment/History, Files/Plots/Packages/Help/Viewer, Console.

RStudio is divided into 4 “Panes”: the **Source** for your scripts and documents (top-left, in the default layout), your **Environment/History** (top-right), your **Files/Plots/Packages/Help/Viewer** (bottom-right), and the R **Console** (bottom-left). The placement of these panes and their content can be customized (see menu, Tools -> Global Options -> Pane Layout).

One of the advantages of using RStudio is that all the information you need to write code is available in a single window. Additionally, with many shortcuts, autocompletion, and highlighting for the major file types you use while developing in R, RStudio will make typing easier and less error-prone.

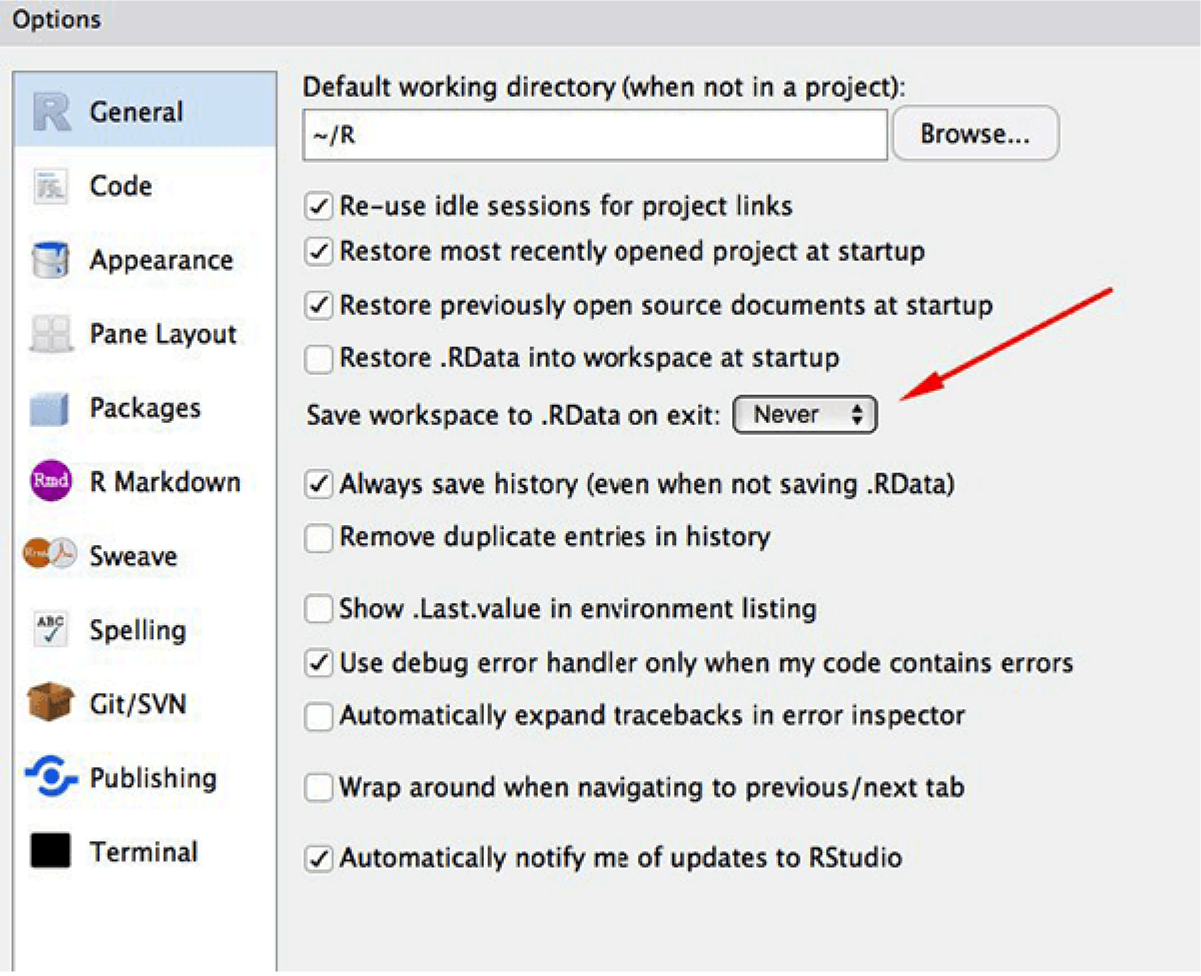
Create an R Project

Before starting to write code in RStudio, we need to create an R Project. The idea behind an R project is to have a workspace where you can keep all the files and settings associated with the project together. In that way, next time you open the R Project it would be easier to resume work.

To create an “R Project”:

1. Start RStudio.
2. Under the `File` menu, click on `New Project`. Choose `New Directory`, then `New Project`.
3. Enter a name for this new folder (or “directory”), and choose a convenient location for it. This will be your **working directory** for the rest of the day (*e.g.,* `~/my-first-project`).
4. Click on `Create Project`.
5. (Optional) Set Preferences to ‘Never’ save workspace in RStudio.

RStudio’s default preferences generally work well, but saving a workspace to `.RData` can be cumbersome, especially if you are working with larger datasets as this would save all the data that is loaded into R into the `.RData` file.
To turn that off, go to `Tools -> Global Options` and select the ‘Never’ option for `Save workspace to .RData' on exit.`



Default working directory (when not in a project): ~/R

☑ Re-use idle sessions for project links
☑ Restore most recently opened project at startup
☑ Restore previously open source documents at startup
□ Restore .RData into workspace at startup
Save workspace to .RData on exit: Never
☑ Always save history (even when not saving .RData)
□ Remove duplicate entries in history
□ Show .Last.value in environment listing
☑ Use debug error handler only when my code contains errors
□ Automatically expand tracebacks in error inspector
□ Wrap around when navigating to previous/next tab
☑ Automatically notify me of updates to RStudio

Set ‘Save workspace to .RData on exit’ to ‘Never’

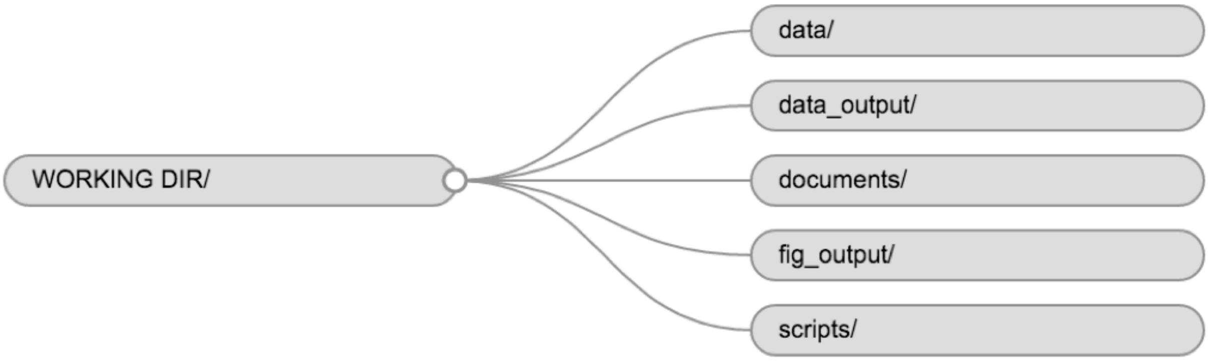
Working directory

Whenever we are working on a project, it is good practice to keep a set of related data, analyses, and text self-contained in a single folder, called the **working directory**.

Using a consistent folder structure across your projects will help keep things organized, and will also make it easy to find things in the future. This can be especially helpful when you have multiple projects. In general, you may create directories (folders) for **scripts**, **data**, and **documents**.

- **`data/`** Use this folder to store your raw data. For the sake of transparency and provenance (https://en.wikipedia.org/wiki/Provenance), you should *always* keep a copy of your raw data accessible and do as much of your data cleanup and preprocessing programmatically (*i.e.,* with scripts, rather than manually). Separating raw data from processed data is also a good idea. For example, you could have files `data/raw/survey.plot1.txt` and `data/raw/survey.plot2.txt` kept separate from a `data_output/survey.csv` file generated by the `scripts/01.preprocess.survey.R` script.
- **`documents/`** This would be a place to keep documentation and other text documents
- **`scripts/`** This would be the location to keep your R scripts for different analyses or plotting.

You may want additional directories or subdirectories depending on your project needs, but these should form the backbone of your working directory.



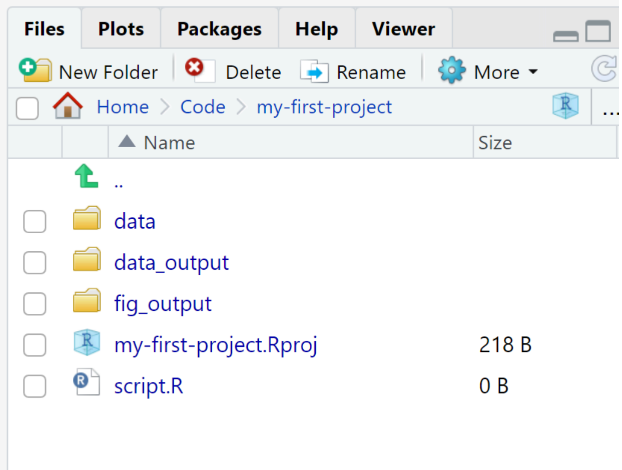
Example of a working directory structure.

For this workshop, we will need a `data/` folder to store our raw data, and we will use `data_output/` for when we learn how to export data as CSV files, and `fig_output/` folder for the figures that we will save.

- Under the `Files` tab on the right of the screen, click on `New Folder` and create a folder named `data` within your newly created working directory (e.g., `~/my-first-project/data`). Repeat these operations to create a `data_output/` and a `fig_output` folders.

We are going to keep the script in the root of our working directory because we are only going to use one file and it will make things easier.

Your working directory should now look like this:



How it should look like at the beginning of this lesson

The working directory is an important concept to understand. It is the place from where R will be looking for and saving the files. When you write code for your project, it should refer to files in relation to the root of your working directory and only need files within this structure.

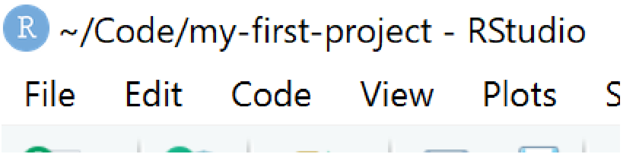
All of the scripts within this folder can then use **relative paths** to files in the working directory that indicate where inside the project a file is located (as opposed to **absolute paths**, which point to where a file is on a specific computer). Working this way makes it a lot easier to move your project around on your computer and share it with others without worrying about whether or not the underlying scripts will still work.

Absolute vs Relative paths examples

Relative path: data/dataset1.txt

Absolute path: C:/Users/User1/Documents/R/my-first-project/**data/dataset1.txt**

Using RStudio projects makes it easy to organise your files in the project and ensures that your working directory is set properly. RStudio shows your current working directory at the top of your window:



Another way to check your working directory is by typing `getwd()` in the console pane. If for some reason your working directory is not what it should be, you can change it in the RStudio interface by navigating in the file browser where your working directory should be, and clicking on the blue gear icon “More”, and select “Set As Working Directory”. Alternatively you can use `setwd("/path/to/working/directory")` to reset your working directory (not recommended). However, your scripts should not include this line because it will fail on someone else’s computer.

Interacting with R

The basis of programming is that we write down instructions for the computer to follow, and then we tell the computer to follow those instructions. We write, or *code*, instructions in R because it is a common language that both the computer and we can understand. We call the instructions *commands* and we tell the computer to follow the instructions by *executing* (also called *running*) those commands.

There are two main ways of interacting with R: by using the console or by using script files (plain text files that contain your code). The console pane (in RStudio, the bottom left panel) is the place where commands written in the R language can be typed and executed immediately by the computer. It is also where the results will be shown for commands that have been executed. You can type commands directly into the console and press `Enter` to execute those commands, but they will be forgotten when you close the session.

Because we want our code and workflow to be reproducible, it is better to type the commands we want in the script editor, and save the script. This way, there is a complete record of what we did, and anyone (including our future selves!) can easily replicate the results on their computer.

RStudio allows you to execute commands directly from the script editor by using the `Ctrl + Enter` shortcut (on Macs, `Cmd + Return` will work, too). The command on the current line in the script (indicated by the cursor) or all of the commands in the currently selected text will be sent to the console and executed when you press `Ctrl + Enter`. You can find other keyboard shortcuts in this RStudio cheatsheet about the RStudio IDE (https://github.com/rstudio/cheatsheets/raw/master/rstudio-ide.pdf).

At some point in your analysis you may want to check the content of a variable or the structure of an object, without necessarily keeping a record of it in your script. You can type these commands and execute them directly in the console. RStudio provides the `Ctrl + 1` and `Ctrl + 2` shortcuts allow you to jump between the script and the console panes.

If R is ready to accept commands, the R console shows a `>` prompt. If it receives a command (by typing, copy-pasting or sent from the script editor using `Ctrl + Enter`), R will try to execute it, and when ready, will show the results and come back with a new `>` prompt to wait for new commands.

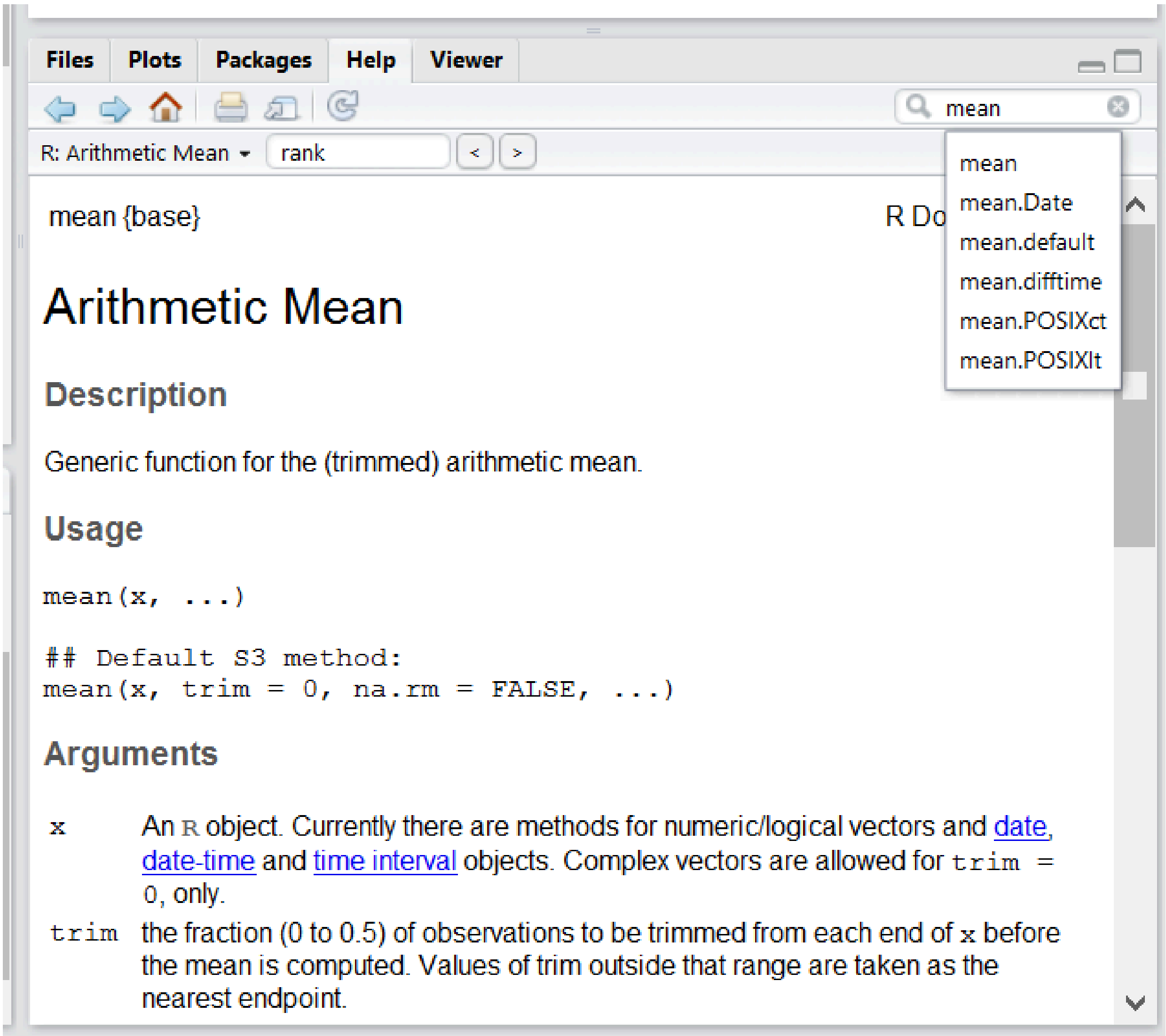
If R is still waiting for you to enter more data because it isn't complete yet, the console will show a `+` prompt. It means that you haven't finished entering a complete command. This is because you have not 'closed' a parenthesis or quotation, i.e. you don't have the same number of left-parentheses as right-parentheses, or the same number of opening and closing quotation marks. When this happens, and you thought you finished typing your command, click inside the console window and press `Esc` ; this will cancel the incomplete command and return you to the `>` prompt.

How to learn more after the workshop?

The material we cover during this workshop will give you an initial taste of how you can use R to analyze data for your own research. However, you will need to learn more to do advanced operations such as cleaning your dataset, using statistical methods, or creating beautiful graphics. The best way to become proficient and efficient at R, as with any other tool, is to use it to address your actual research questions. As a beginner, it can feel daunting to have to write a script from scratch, and given that many people make their code available online, modifying existing code to suit your purpose might make it easier for you to get started.

Seeking help

Use the built-in RStudio help interface to search for more information on R functions



mean {base} R Do

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x` An `R` object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.

`trim` the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.

RStudio help interface.

One of the fastest ways to get help, is to use the RStudio help interface. This panel by default can be found at the lower right hand panel of RStudio. As seen in the screenshot, by typing the word “Mean”, RStudio tries to also give a number of suggestions that you might be interested in. The description is then shown in the display window.

I know the name of the function I want to use, but I’m not sure how to use it

If you need help with a specific function, let’s say `barplot()` , you can type:

```
?barplot
```

If you just need to remind yourself of the names of the arguments, you can use:

```
args(lm)
```

I want to use a function that does X, there must be a function for it but I don’t know which one…

If you are looking for a function to do a particular task, you can use the `help.search()` function, which is called by the double question mark `??`. However, this only looks through the installed packages for help pages with a match to your search request

```
??kruskal
```

If you can't find what you are looking for, you can use the rdocumentation.org (http://www.rdocumentation.org) website that searches through the help files across all packages available.

Finally, a generic Google or internet search "R <task>" will often either send you to the appropriate package documentation or a helpful forum where someone else has already asked your question.

I am stuck… I get an error message that I don't understand

Start by googling the error message. However, this doesn't always work very well because often, package developers rely on the error catching provided by R. You end up with general error messages that might not be very helpful to diagnose a problem (e.g. "subscript out of bounds"). If the message is very generic, you might also include the name of the function or package you're using in your query.

However, you should check Stack Overflow. Search using the `[r]` tag. Most questions have already been answered, but the challenge is to use the right words in the search to find the answers: http://stackoverflow.com/questions/tagged/r (http://stackoverflow.com/questions/tagged/r)

The Introduction to R (http://cran.r-project.org/doc/manuals/R-intro.pdf) can also be dense for people with little programming experience but it is a good place to understand the underpinnings of the R language.

The R FAQ (http://cran.r-project.org/doc/FAQ/R-FAQ.html) is dense and technical but it is full of useful information.

Asking for help

The key to receiving help from someone is for them to rapidly grasp your problem. You should make it as easy as possible to pinpoint where the issue might be.

Try to use the correct words to describe your problem. For instance, a package is not the same thing as a library. Most people will understand what you meant, but others have really strong feelings about the difference in meaning. The key point is that it can make things confusing for people trying to help you. Be as precise as possible when describing your problem.

If possible, try to reduce what doesn't work to a simple *reproducible example*. If you can reproduce the problem using a very small data frame instead of your 50,000 rows and 10,000 columns one, provide the small one with the description of your problem. When appropriate, try to generalize what you are doing so even people who are not in your field can understand the question. For instance instead of using a subset of your real dataset, create a small (3 columns, 5 rows) generic one. For more information on how to write a reproducible example see this article by Hadley Wickham (http://adv-r.had.co.nz/Reproducibility.html).

To share an object with someone else, if it's relatively small, you can use the function `dput()`. It will output R code that can be used to recreate the exact same object as the one in memory:

```
dput(head(iris)) # iris is an example data frame that comes with R
        and head() is a function that returns the first part of t
        he data frame
```

```
## structure(list(Sepal.Length = c(5.1, 4.9, 4.7, 4.6, 5, 5.4),
##     Sepal.Width = c(3.5, 3, 3.2, 3.1, 3.6, 3.9), Petal.Length =
c(1.4,
##     1.4, 1.3, 1.5, 1.4, 1.7), Petal.Width = c(0.2, 0.2, 0.2,
##     0.2, 0.2, 0.4), Species = structure(c(1L, 1L, 1L, 1L, 1L,
##     1L), .Label = c("setosa", "versicolor", "virginica"), class
= "factor")), row.names = c(NA,
## 6L), class = "data.frame")
```

If the object is larger, provide either the raw file (i.e., your CSV file) with your script up to the point of the error (and after removing everything that is not relevant to your issue). Alternatively, in particular if your question is not related to a data frame, you can save any R object to a file:

```
saveRDS(iris, file="/tmp/iris.rds")
```

The content of this file is however not human readable and cannot be posted directly on Stack Overflow. Instead, it can be sent to someone by email who can read it with the `readRDS()` command (here it is assumed that the downloaded file is in a `Downloads` folder in the user's home directory):

```
some_data <- readRDS(file="~/Downloads/iris.rds")
```

Last, but certainly not least, **always include the output of `sessionInfo()`** as it provides critical information about your platform, the versions of R and the packages that you are using, and other information that can be very helpful to understand your problem.

```
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19041)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252  LC_CTYPE=English_Un
ited Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252 LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods
base
##
## other attached packages:
## [1] forcats_0.5.0   stringr_1.4.0   dplyr_1.0.2     purrr_0.3.4
readr_1.4.0     tidyr_1.1.2     tibble_3.0.4
## [8] ggplot2_3.3.2   tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.0 xfun_0.18        haven_2.3.1      colorsp
ace_1.4-1 vctrs_0.3.4      generics_0.0.2
##  [7] htmltools_0.5.0  yaml_2.2.1       blob_1.2.1       rlang_
0.4.8      pillar_1.4.6     glue_1.4.2
## [13] withr_2.3.0      DBI_1.1.0        dbplyr_1.4.4     modelr_
0.1.8     readxl_1.3.1     lifecycle_0.2.0
## [19] munsell_0.5.0    gtable_0.3.0     cellranger_1.1.0 rvest_
0.3.6      evaluate_0.14    labeling_0.4.2
## [25] knitr_1.30       fansi_0.4.1      highr_0.8        broom_
0.7.2      Rcpp_1.0.5       scales_1.1.1
## [31] backports_1.1.10 jsonlite_1.7.1   farver_2.0.3     fs_1.5.
0         hms_0.5.3        digest_0.6.27
## [37] stringi_1.5.3    grid_4.0.3       cli_2.1.0        tools_
4.0.3      magrittr_1.5     crayon_1.3.4
## [43] pkgconfig_2.0.3  ellipsis_0.3.1   xml2_1.3.2       reprex_
0.3.0     lubridate_1.7.9  assertthat_0.2.1
## [49] rmarkdown_2.5    httr_1.4.2       rstudioapi_0.11  R6_2.4.
1         compiler_4.0.3
```

Where to ask for help?

- The person sitting next to you during the workshop. Don’t hesitate to talk to your neighbor during the workshop, compare your answers, and ask for help. You might also be interested in

organizing regular meetings following the workshop to keep learning from each other.
- Your friendly colleagues: if you know someone with more experience than you, they might be able and willing to help you.
- Stack Overflow (http://stackoverflow.com/questions/tagged/r): if your question hasn't been answered before and is well crafted, chances are you will get an answer in less than 5 min. Remember to follow their guidelines on how to ask a good question (http://stackoverflow.com/help/how-to-ask).
- The R-help mailing list (https://stat.ethz.ch/mailman/listinfo/r-help): it is read by a lot of people (including most of the R core team), a lot of people post to it, but the tone can be pretty dry, and it is not always very welcoming to new users. If your question is valid, you are likely to get an answer very fast but don't expect that it will come with smiley faces. Also, here more than anywhere else, be sure to use correct vocabulary (otherwise you might get an answer pointing to the misuse of your words rather than answering your question). You will also have more success if your question is about a base function rather than a specific package.
- If your question is about a specific package, see if there is a mailing list for it. Usually it's included in the DESCRIPTION file of the package that can be accessed using `packageDescription("name-of-package")` . You may also want to try to email the author of the package directly, or open an issue on the code repository (e.g., GitHub).
- There are also some topic-specific mailing lists (GIS, phylogenetics, etc…), the complete list is here (http://www.r-project.org/mail.html).

More resources

- The Posting Guide (http://www.r-project.org/posting-guide.html) for the R mailing lists.
- How to ask for R help (http://blog.revolutionanalytics.com/2014/01/how-to-ask-for-r-help.html) useful guidelines
- This blog post by Jon Skeet (http://codeblog.jonskeet.uk/2010/08/29/writing-the-perfect-question/) has quite comprehensive advice on how to ask programming questions.
- The reprex (https://cran.rstudio.com/web/packages/reprex/) package is very helpful to create reproducible examples when asking for help. The [rOpenSci community call "How to ask questions so they get answered"], rOpenSci Blog (https://ropensci.org/commcalls/2017-03-07/) and video recording (https://vimeo.com/208749032) includes a presentation of the reprex package and of its philosophy.