

RStudio tutorial

Experiential Data science for Undergraduate Cross-disciplinary Education

Dr. Kim Dill-McFarland, U. of British Columbia

Contents

RStudio tutorial	1
Learning objectives	1
A Tour of RStudio	1
R Projects	2
R Scripts	3
R packages	4
Help function	4
Key shortcuts	5

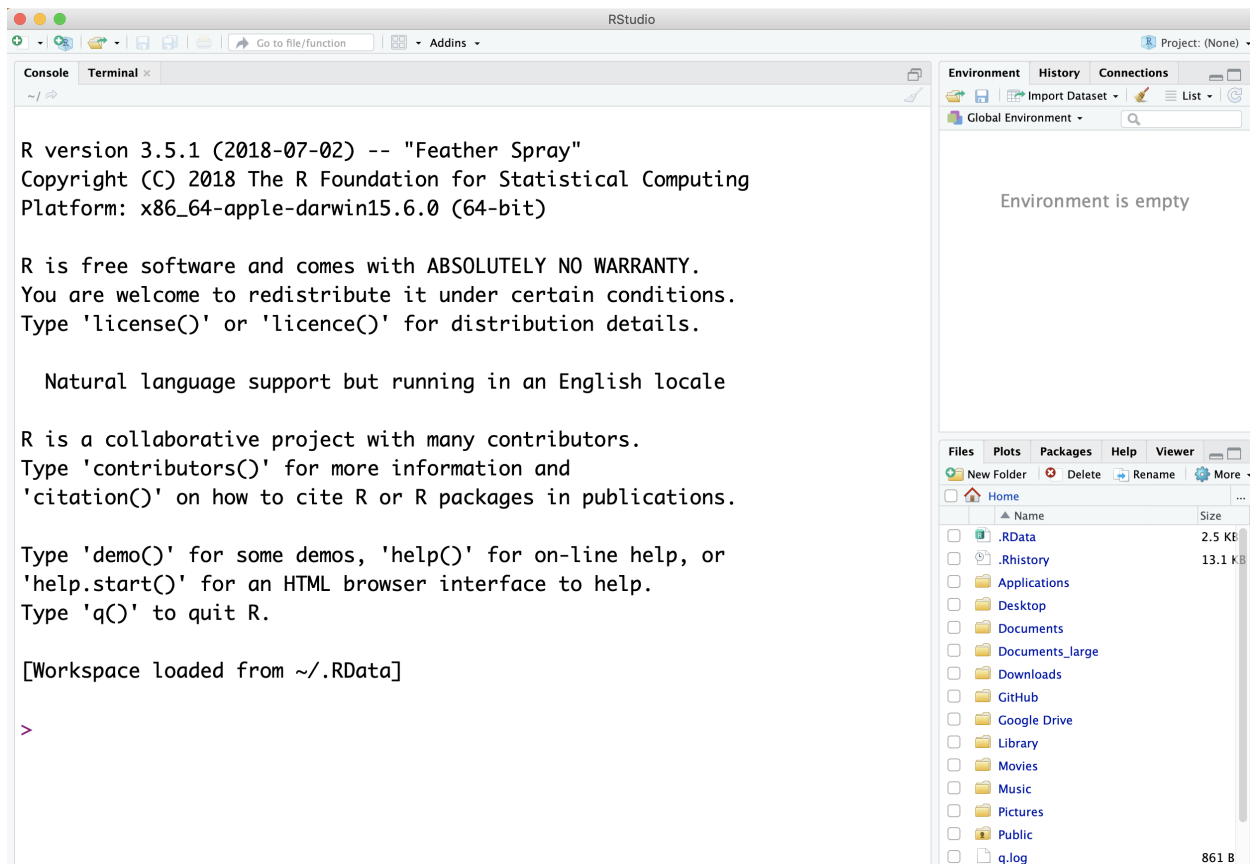
RStudio tutorial

Learning objectives

- Navigate the RStudio software including:
 - Projects
 - Packages
 - Help
 - Key shortcuts

A Tour of RStudio

When you start RStudio, you will see something like the following window appear:



Notice that the window is divided into three “panes”:

- Console (the entire left side): this is your view into the R engine. You can type in R commands here and see the output printed by R. (To make it easier to tell them apart, your input is printed in blue, while the output is black.) There are several editing conveniences available: use up and down arrow keys to go back to previously entered commands, which can then be edited and re-run; TAB for completing the name before the cursor; see more in [online docs](#).
- Environment/History (tabbed in upper right): view current user-defined objects and previously-entered commands, respectively.
- Files/Plots/Packages/Help (tabbed in lower right): as their names suggest, these are used to view the contents of the current directory, graphics created by the user, install packages, and view the built-in help pages.

To change the look of RStudio, you can go to Tools > Global Options > Appearance and select colors, font size, etc. If you plan to be working for longer periods, we suggest choosing a dark background color scheme to save your computer battery and your eyes.

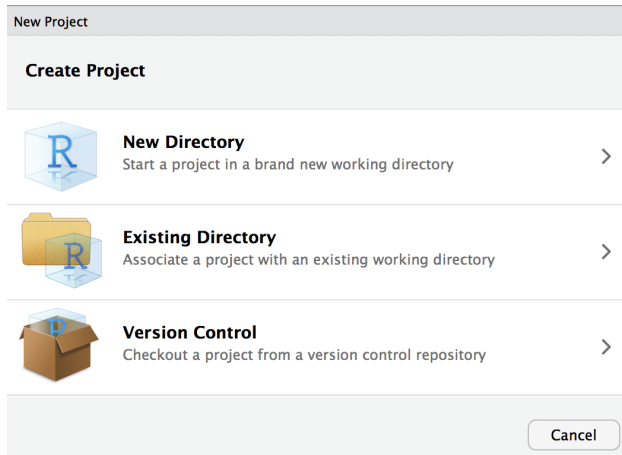
R Projects

Projects are a great feature of RStudio. When you create a project, RStudio creates an **.Rproj** file that links all of your files and outputs to the project directory. When you import data, R automatically looks for the file in the project directory instead of you having to specify a full file path on your computer like `/Users/username/Desktop/`. R also automatically saves any output to the project directory. Finally, projects

allow you to save your R environment in `.RData` so that when you close RStudio and then re-open it, you can start right where you left off without re-importing any data or re-calculating any intermediate steps.

RStudio has a simple interface to create and switch between projects, accessed from the button in the top-right corner of the RStudio window (labelled “Project: (None)”, initially) or under File > New Project.

Using either of these options, you will see the following window.



You can either create a project in an existing directory or make a new directory on your computer - just be sure you know where it is.

After your project is created, you can navigate to its directory using your Finder/File explorer or the integrated Terminal in RStudio. You will see the `.RProj` file has been created. This is the file R uses to link all your project files together in this directory.

To access this project in the future, choose File > Open Project in RStudio or simply double-click the `RProj` file and RStudio will open in that project.

R Scripts

R script files are the primary way in which R facilitates reproducible research. They contain the code that loads your raw data, cleans it, performs the analyses, and creates and saves visualizations. R scripts maintain a record of everything that is done to the raw data to reach the final result. That way, it is very easy to write up and communicate your methods because you have a document listing the precise steps you used to conduct your analyses. This is one of R’s primary advantages compared to traditional tools like Excel, where it may be unclear how to reproduce the results.

Generally, if you are testing an operation (*e.g.* what would my data look like if I applied a log-transformation to it?), you should do it in the console (left pane of RStudio). If you are committing a step to your analysis (*e.g.* I want to apply a log-transformation to my data and then conduct the rest of my analyses on the log-transformed data), you should add it to your R script so that it is saved for future use.

Additionally, you should annotate your R scripts with comments. In each line of code, any text preceded by the `#` symbol will not execute. Comments can be useful to remind yourself and to tell other readers what a specific chunk of code does. For example,

```
# This is a comment with an R script
1+2+3
```

To create a new script, use File > New File > R Script. This will split the left side of RStudio into the script (top) and console (bottom). If you save your script, you will see that you are automatically directed into your project directory!

R packages

R packages are units of shareable code, containing functions that facilitate and enhance analyses.

Install from CRAN

Packages are typically installed from [CRAN](#) (The Comprehensive R Archive Network), which is a database containing R itself as well as many R packages. Any package can be installed from CRAN using the `install.packages()` function run in the console or by clicking ‘Install’ in the Packages tab (lower right quadrant). For example, the [tidyverse](#) would be installed with

```
install.packages("tidyverse")
```

Install from Bioconductor

A number of other packages are stored on [Bioconductor](#) and need to be installed differently than CRAN packages. Bioconductor has its own installation function in a (you guessed it!) package. You must first install the `BiocManager` package like so.

```
install.packages("BiocManager")
```

Then you have access to the `BiocManager` install function. Leaving the package name blank will install the basic Bioconductor packages.

```
BiocManager::install()
```

And finally, you can install other Bioconductor packages like `GenomicFeatures` for example.

```
BiocManager::install("GenomicFeatures")
```

Load packages

After installing a package, and *everytime* you open a new RStudio session, the packages you want to use need to be loaded into the R work-space with the `library` function. This tells R to access the package’s functions and prevents RStudio from lags that would occur if it automatically loaded every downloaded package every time you opened it.

For example, you can load the `tidyverse` package like so.

```
library(tidyverse)
```

Help function

RStudio has integrated help pages *e.g.* it downloads the [CRAN](#) pages to your computer.

If you know the function name, you can access its help page using `?` or `help()`. If you are unsure of the exact function name or want to see all the functions related to a given topic, you can search all the help pages with `??` or `help.search()`.

For example, if you wanted to calculate the mean of a string of numbers, you could look-up the `mean()` function with

```
?mean
```

Or if you were unsure of the function name, you could search for all functions related to means with



Figure 1:

??mean

Importantly, these functions will only search base R and the packages that are downloaded and *loaded* in your current RStudio session.

Key shortcuts

There are a number of shortcuts in RStudio that may help you as your code.

Running code

To execute code in a script, you can:

- Click the ‘Run’ button in the upper right of the script pane
- Use Cmd+Enter (Mac) or Ctrl+Enter (PC)

In either case, you *do not* need to highlight the entire line of code being run. You just need to have your cursor anywhere on that line. In contrast, you can run multiple lines of code all at once by highlighting all of the necessary lines in your script, and executing with ‘Run’ or Cmd/Ctrl+Enter.

Tab-completion

You can tab-complete names including functions, data, and variables. This works in either a script or the R console. For example, if you start typing a function `read.` and then press Tab, RStudio will show you a list of all of its functions that start with `read..`

Code history

In the R console, you can scroll through recently used code with the up arrow. This goes through your History (upper right pane, second tab), which is saved if you are working in an R project. This is useful if you want to re-run a previous function with some minor modifications.