

Getting Started with R and Rstudio

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

11/13/2023

Why R?

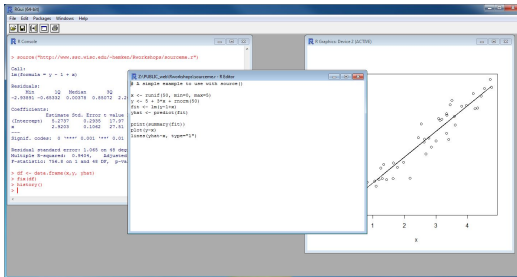
R is not a programming language like C or Java. It was not created by software engineers for software development. Instead, it was developed by statisticians as an interactive environment for data analysis. A history of R is summarized: A Brief History of S¹.



¹<https://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.131.1428>

Why R?

The **interactivity** of R (more later), is an indispensable feature in data science because, as you will soon learn, the ability to quickly explore data is a necessity for success in this field.



However, like in other programming languages, you can save your work as scripts that can be easily executed at any moment. These scripts serve as a record of the analysis you performed, a key feature that facilitates reproducible work.

Why R?

If you are an expert programmer, you should not expect R to follow the conventions you are used to since you will be disappointed. If you are patient, you will come to appreciate the unequal power of R when it comes to data analysis and, specifically, data visualization.

Exploratory Data Analysis by using Rstudio



Why R?

Other attractive features of R are:

- ① R is free and open source: <https://opensource.org/history>.
- ② It runs on all major platforms: Windows, Mac Os, UNIX/Linux.
- ③ Scripts and data objects can be shared seamlessly across platforms.
- ④ There is a large, growing, and active community of R users and, as a result, there are numerous resources for learning and asking questions^{2 3 4}.
- ⑤ It is easy for others to contribute add-ons which enables developers to share software implementations of new data science methodologies.

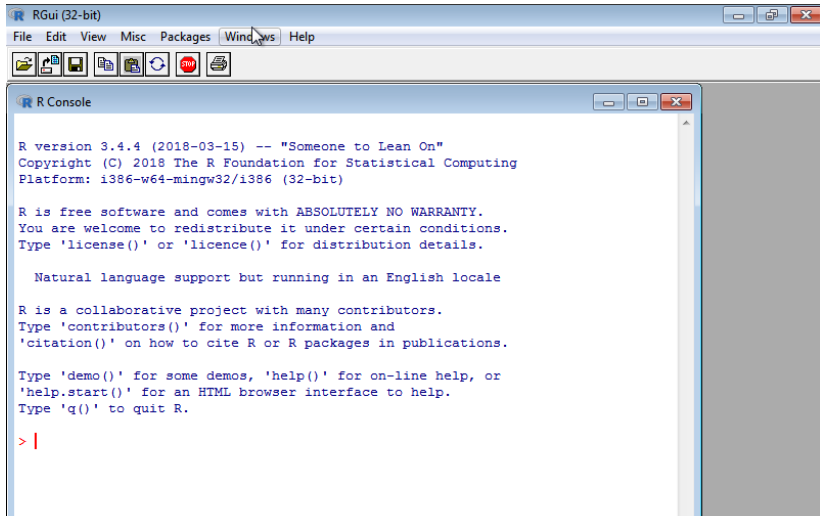
²<https://stats.stackexchange.com/questions/138/free-resources-for-learning-r>

³<https://www.r-project.org/help.html>

⁴<https://stackoverflow.com/documentation/r/topics>

The R console

Interactive data analysis usually occurs on the *R console* that executes commands as you type them. It looks like this:



```
RGui (32-bit)
File Edit View Misc Packages Windows Help

R Console

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-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.

> |
```

The R console

As a quick example, try using the console to calculate a 15% tip on a meal that cost \$19.71:

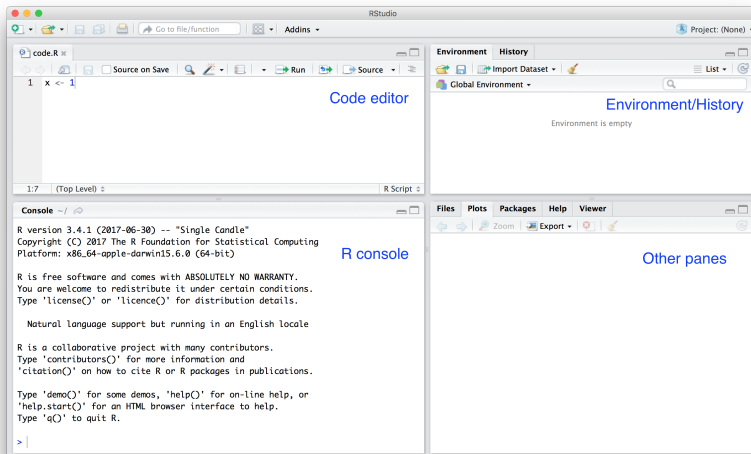
```
0.15 * 19.71
```

```
## [1] 2.9565
```

Pro Tip: Grey boxes are used to show R code typed into the R console. The symbol `##` is used to denote what the R console outputs.

Scripts

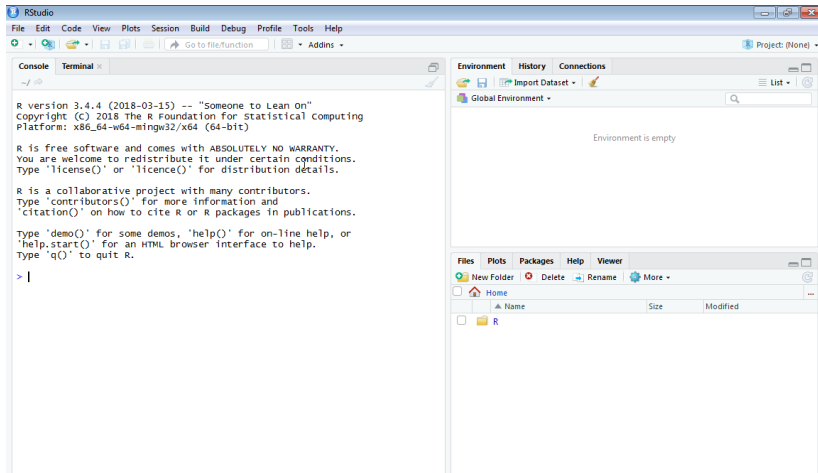
One of the great advantages of R over point-and-click analysis software is that you can save your work as scripts. You can edit and save these scripts using a text editor. We will use the interactive *integrated development environment* (IDE) RStudio.



RStudio will be our launching pad for data science projects. It not only provides an editor for us to create and edit our scripts but also provides many other useful tools. In this section, we go over some of the basics.

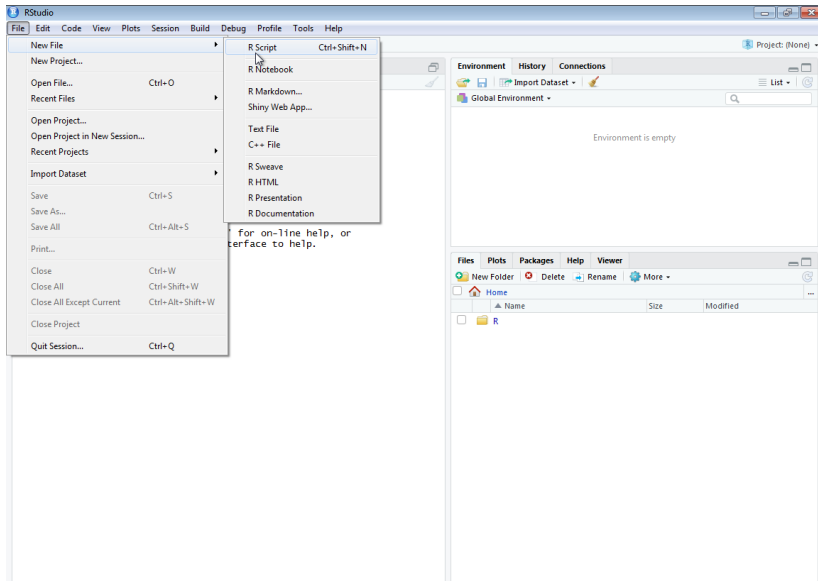
RStudio Panes

When you start RStudio for the first time, you will see three panes. The left pane shows the R console, the right top pane includes tabs for the *Environment* and *History*, while the bottom right shows five tabs: *File*, *Plots*, *Packages*, *Help*, and *Viewer*.



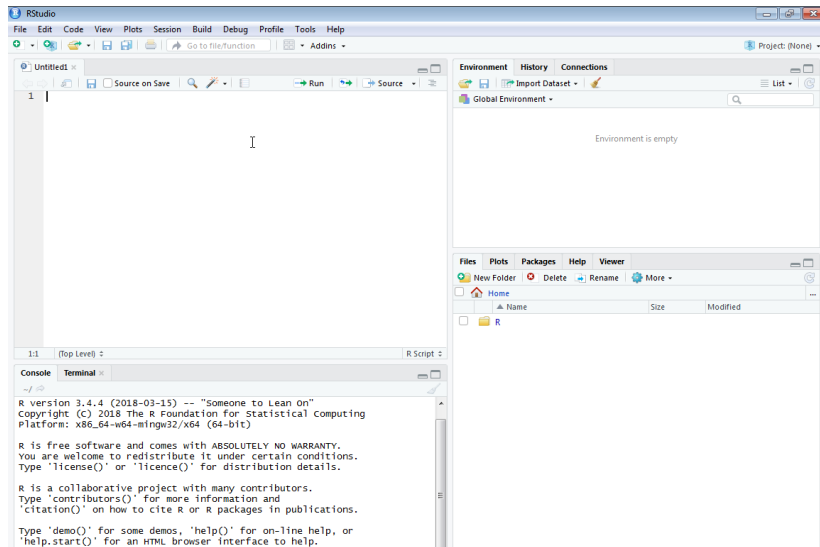
RStudio Scripts

For a new script, you can click on File, then New File, then R Script.



RStudio Scripts

This starts a new pane on the left and it is here where you can start writing your script.

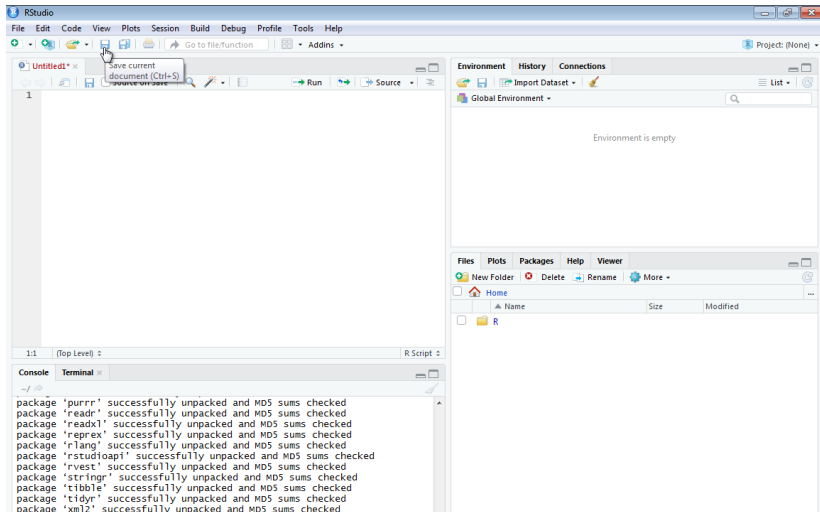


Running commands while editing scripts

There are many editors specifically made for coding. These are useful because color and indentation are automatically added to make code more readable. RStudio is one of these editors, and it was specifically developed for R. One of the main advantages provided by RStudio over other editors is that we can test our code easily as we edit our scripts. Here we show an example.

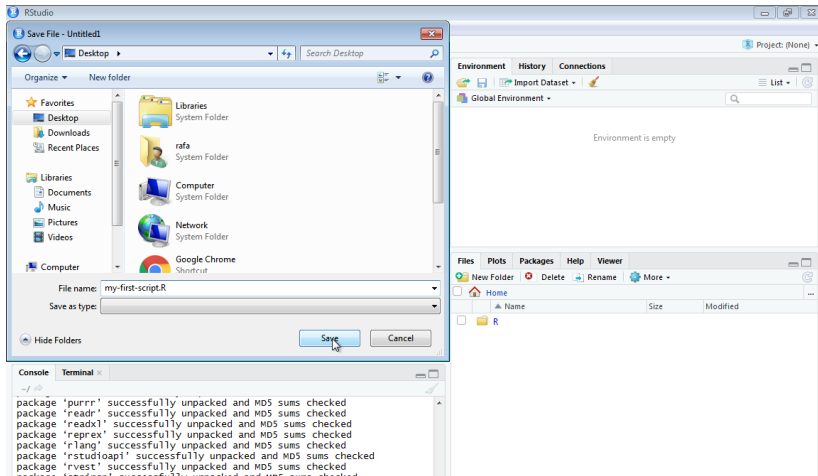
Running commands while editing scripts

Let's start by opening a new script and giving the script a name. We can do this through the editor by saving the current unnamed script.



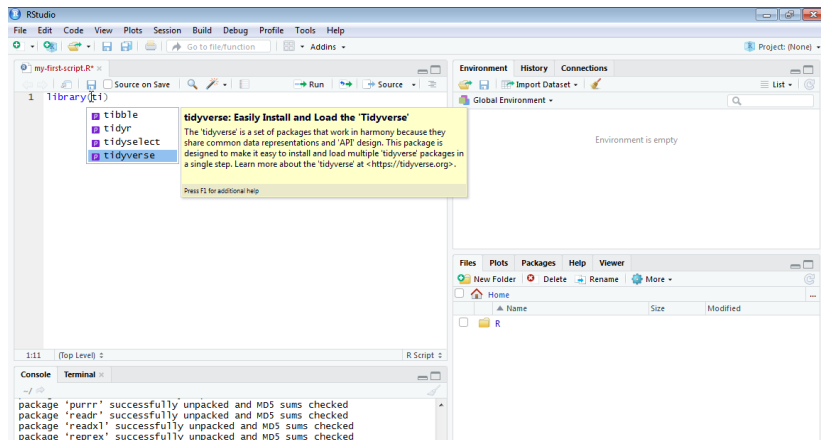
Running commands while editing scripts

When saving a script, a good convention is to use a descriptive name, with lower case letters, no spaces, only hyphens to separate words, and then followed by the suffix `.R`. We will call this script *my-first-script.R*.



Running commands while editing scripts

Now we are ready to start editing our first script. The first lines of code in an R script are dedicated to loading the libraries we will use. Another useful RStudio feature is that once we type `library()` it starts auto-completing with libraries that we have installed. Note what happens when we type `library(t`):



Running commands while editing scripts

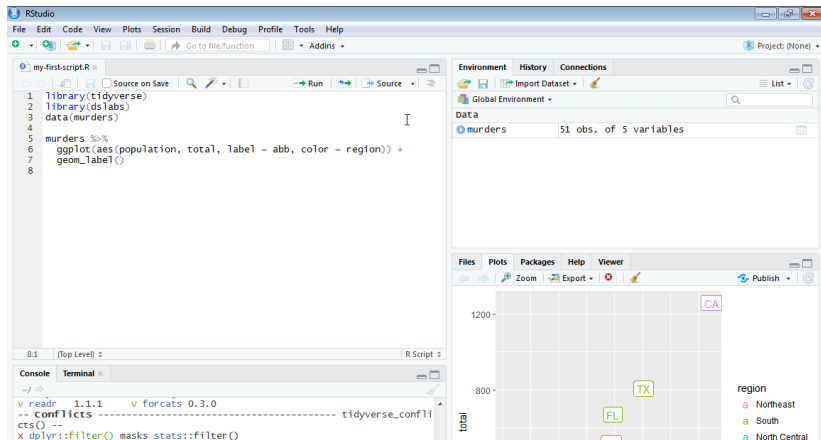
Another feature you may have noticed is that when you type `library(` the second parenthesis is automatically added. This will help you avoid one of the most common errors in coding: forgetting to close a parenthesis.

Running commands while editing scripts

Now we can continue to write code. As an example, we will make a graph showing murder totals versus population totals by state. Once you are done writing the code needed to make this plot, you can try it out by *executing* the code. To do this, click on the *Run* button on the upper right side of the editing pane. You can also use the key binding: Ctrl+Shift+Enter on Windows or command+shift+return on the Mac.

Running commands while editing scripts

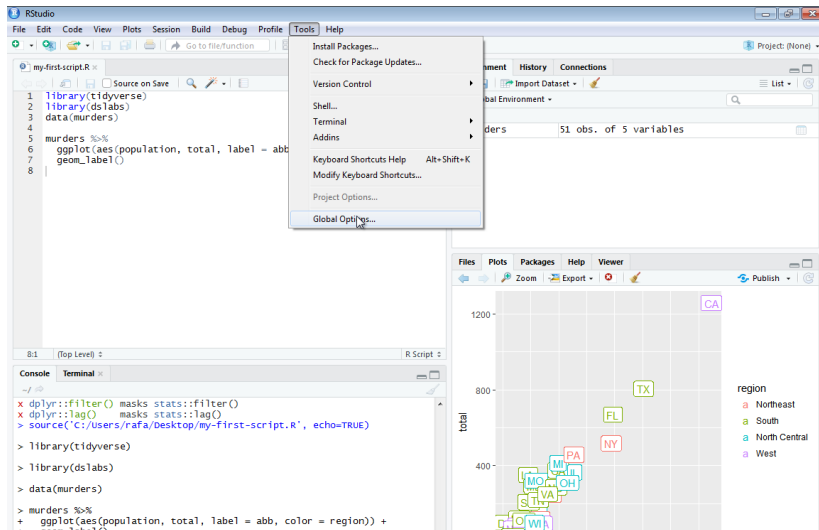
Once you run the code, you will see it appear in the R console and, in this case, the generated plot appears in the plots console. Note that the plot console has a useful interface that permits you to click back and forward across different plots, zoom in to the plot, or save the plots as files.



Changing global options

You can change the look and functionality of RStudio quite a bit.

To change global options you click on *Tools* then *Global Options*:



The screenshot shows the RStudio interface. The 'Tools' menu is open, and 'Global Options...' is highlighted. The background shows a script editor with R code, a console with output, and a plot of murder rates by region.

Script Editor Code:

```
1 library(tidyverse)
2 library(ds.labs)
3 data(murders)
4
5 murders %>%
6   ggplot(aes(population, total, label = abb)) +
7   geom_label()
```

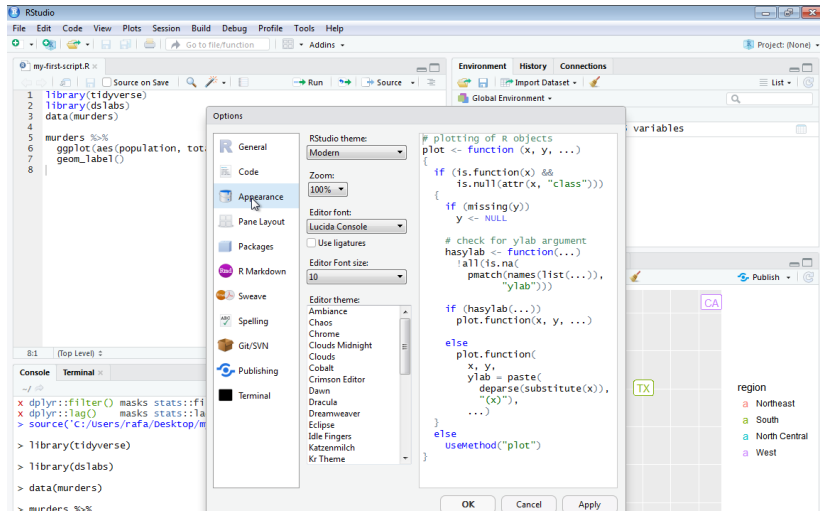
Console Output:

```
x dplyr::filter() masks stats::filter()
x dplyr::lag() masks stats::lag()
> source('C:/Users/rafa/Desktop/my-first-script.R', echo=TRUE)
> library(tidyverse)
> library(ds.labs)
> data(murders)
> murders %>%
+   ggplot(aes(population, total, label = abb, color = region)) +
+   geom_label()
```

Plot: A scatter plot showing the relationship between population and total murder rate, colored by region. The legend indicates four regions: Northeast (red), South (green), North Central (blue), and West (purple).

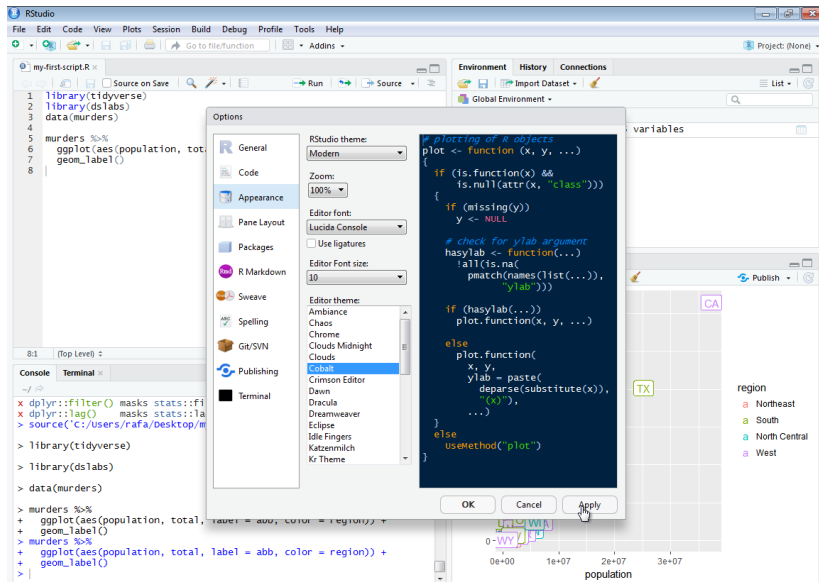
Changing global options

As an example, we show how to change the appearance of the editor. To do this click on *Appearance* and then notice the *Editor theme* options.



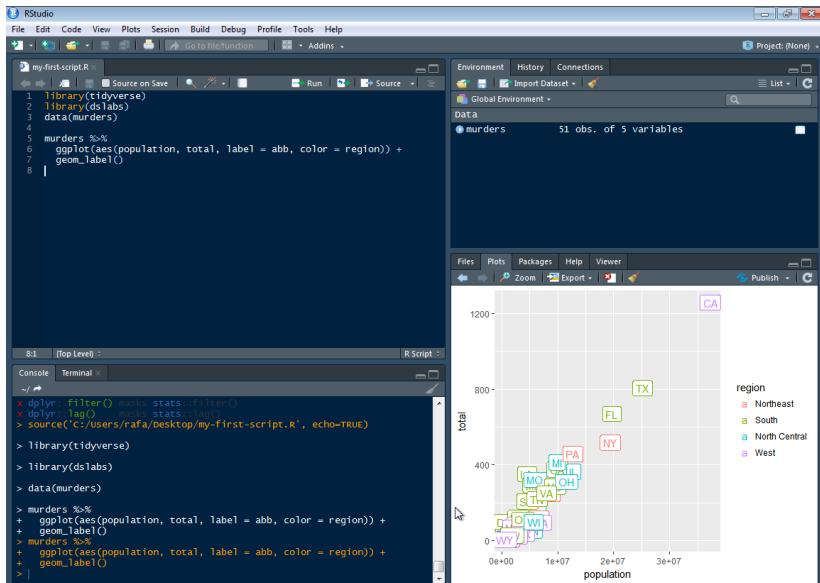
Changing global options

You can click on these and see examples of how your editor will look.



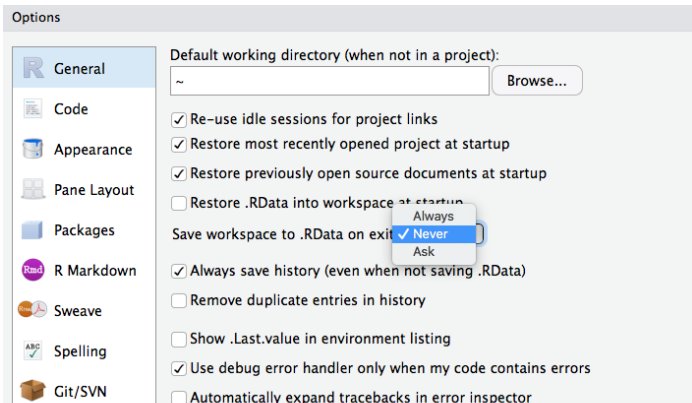
Changing global options

I personally like the *Cobalt* option:



Changing global options

As a second example, we show how to make a change that we **highly recommend**. This is to change the *Save workspace to .RData on exit* to *Never* and uncheck the *Restore .RData into workspace at start*. By default, when you exit R saves all the objects you have created into a file called *.RData*. To change these options, make your *General* settings look like this:



Installing R packages

The functionality provided by a fresh install of R is only a small fraction of what is possible. In fact, we refer to what you get after your first install as **base R**. The extra functionality comes from add-ons available from developers.

There are currently hundreds of these available from CRAN and many others shared via other repositories such as GitHub. However, because not everybody needs all available functionality, R instead makes different components available via **packages**.

Installing R packages

R makes it very easy to install packages from within R. For example, to install the **dslabs** package, which we use to share datasets and code related to this book, you would type:

```
install.packages("dslabs")
```

Installing R packages

In RStudio, you can navigate to the **Tools** tab and select install packages. We can then load the package into our R sessions using the `library` function:

```
library(dslabs)
```

We can install more than one package at once by feeding a character vector to this function:

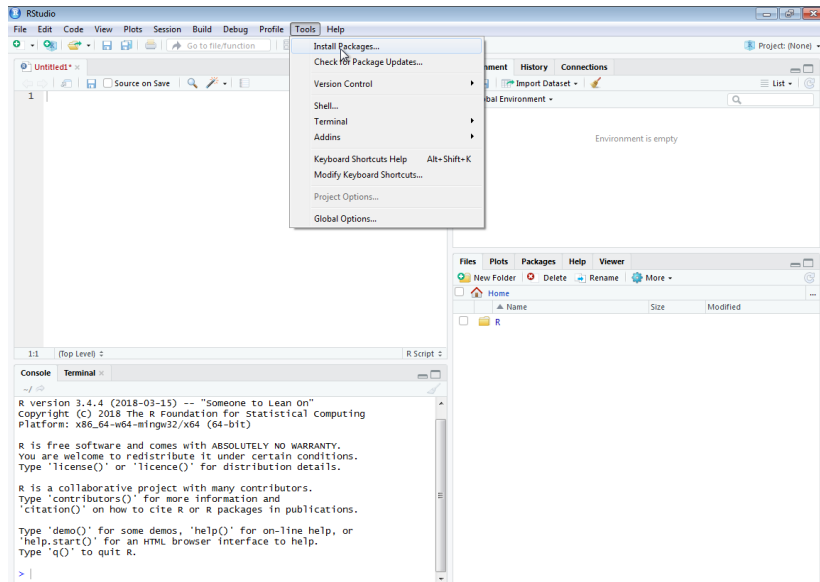
```
install.packages(c("tidyverse", "dslabs"))
```

Installing R packages

Note that installing **tidyverse** actually installs several packages. This commonly occurs when a package has **dependencies**, or uses functions from other packages. When you load a package using `library`, you also load its dependencies.

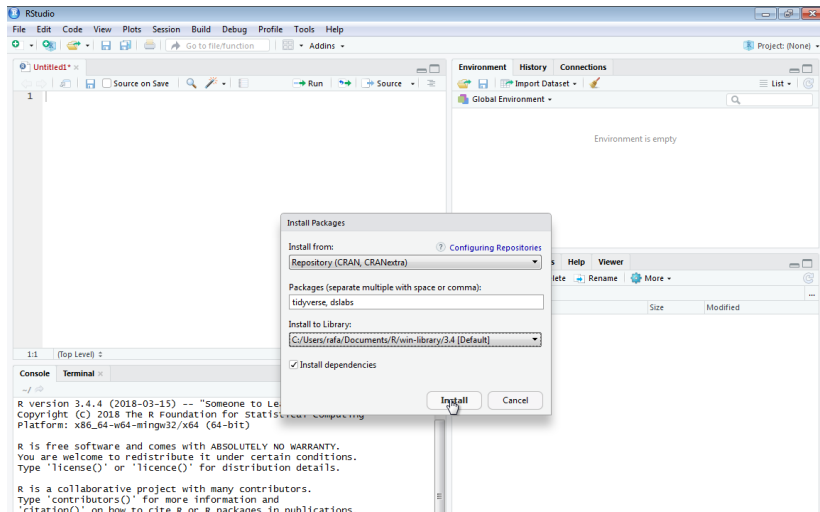
Installing R packages

You can also install packages using RStudio in the following way:



Installing R packages

One advantage of using RStudio is that it auto-completes package names once you start typing, which is helpful when you do not remember the exact spelling of the package:



Installing R packages

Once packages are installed, you can load them into R and you do not need to install them again, unless you install a fresh version of R. Remember packages are installed in R not RStudio.

As you go through this tutorial, you will see that we load packages without installing them. This is because once you install a package, it remains installed and only needs to be loaded with `library`. The package remains loaded until we quit the R session. If you try to load a package and get an error, it probably means you need to install it first. `## Installing R packages` You can see all the packages you have installed using the following function:

```
installed.packages()
```

Installing R packages

It is helpful to keep a list of all the packages you need for your work in a script because if you need to perform a fresh install of R, you can re-install all your packages by simply running a script.

Session Info

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.5.1
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/lib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/lib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods    base
##
## other attached packages:
## [1] dslabs_0.7.6
##
## loaded via a namespace (and not attached):
## [1] compiler_4.3.1    fastmap_1.1.1     cli_3.6.1        tools_4.3.1
## [5] htmltools_0.5.6.1 rstudioapi_0.15.0 yaml_2.3.7        rmarkdown_2.25
## [9] knitr_1.44        xfun_0.40         digest_0.6.33     rlang_1.1.1
```