Lecture 2: 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

4/15/2024

R is not a programming language for software development like C or Java. It was created by statisticians as an environment for data analysis. A history of R is summarized here: A Brief History of S.

The **interactivity** of R (more later), is an indispensable feature in data science because, as you will learn, the ability to quickly explore data is a necessity for success in this field.
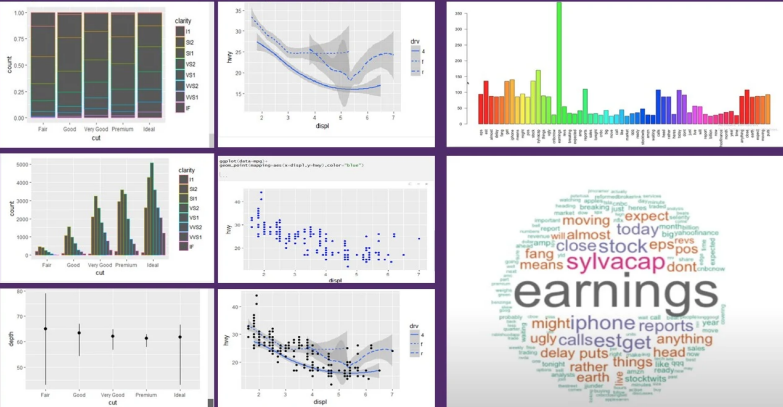
Why R?
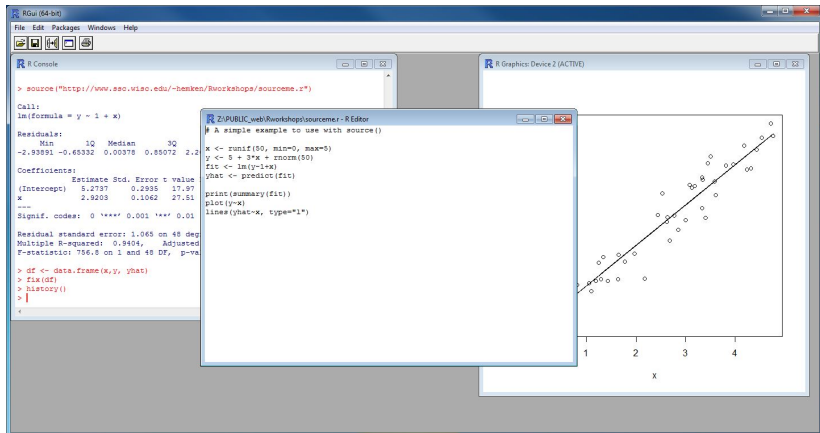
However, like in other programming languages, you can save your work as scripts that can be easily executed at any moment. These scripts serve as a record of the analysis you performed, a key feature that facilitates reproducible work.

Why R?

R does not follow conventions of more "traditional" programming languages. However, you will come to appreciate the unequal power of R when it comes to data analysis and data visualization.

Exploratory Data Analysis by using Rstudio

Why R?

Other attractive features of R are:

1. R is free and open source.
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[1][2][3].
5. It is easy for others to contribute add-ons which enables developers to share software implementations of new data science methodologies.
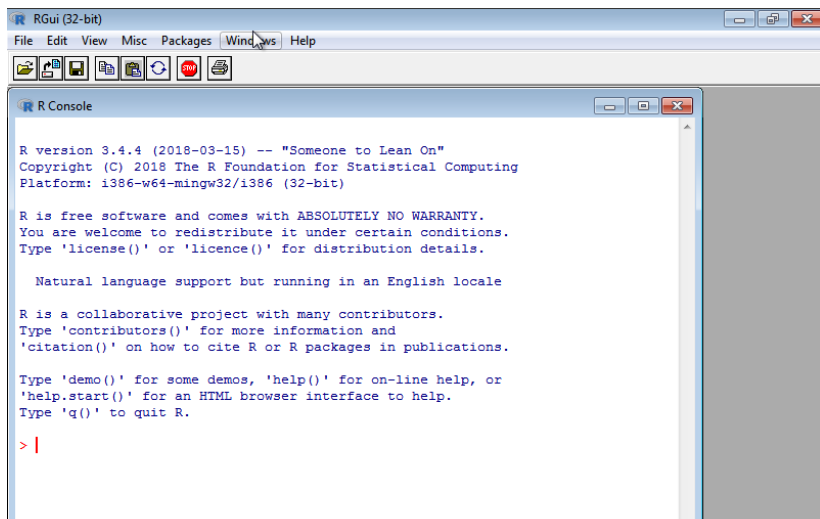
---

[1]https://www.r-project.org/help.html
[2]https://stackoverflow.com/documentation/r/topics
[3]https://stats.stackexchange.com/questions/138/free-resources-for-learning-r

The R console

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

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.
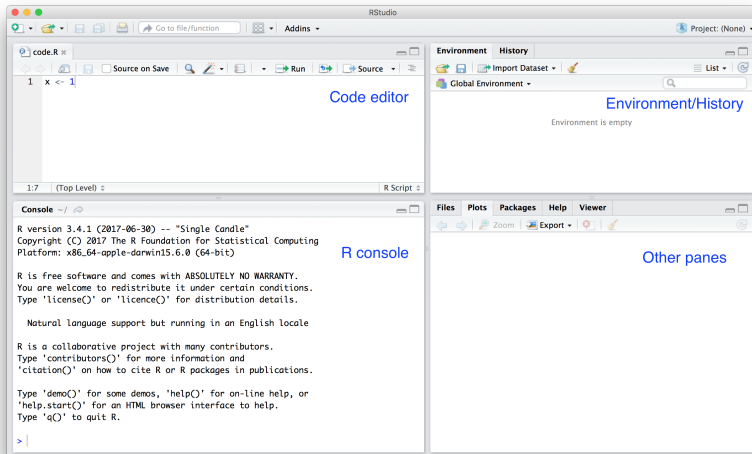
RStudio will be our launching pad for data science projects. It provides an editor for our scripts and provides many other useful tools. We go over some of the basics.
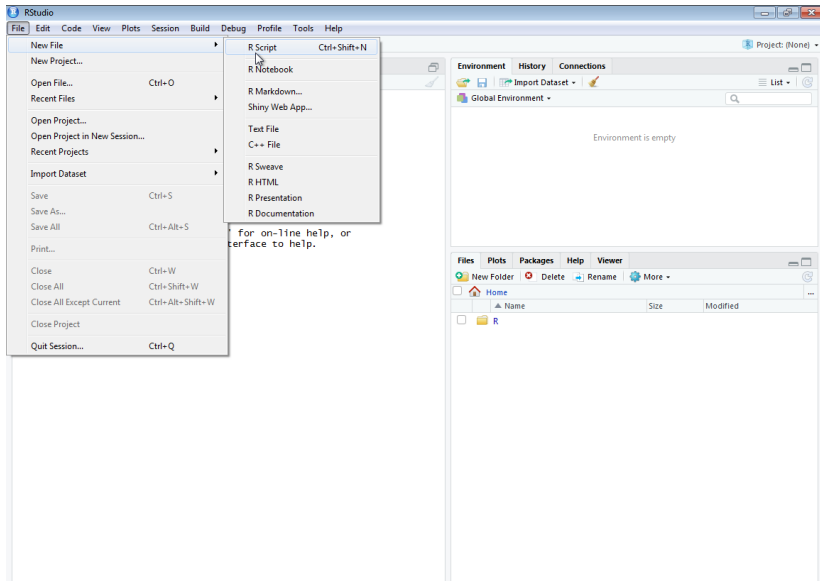
RStudio Panes

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*.
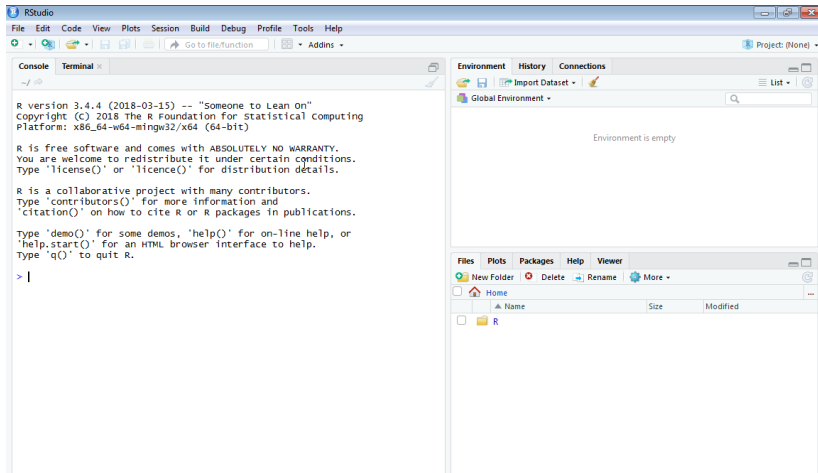
RStudio Scripts

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.
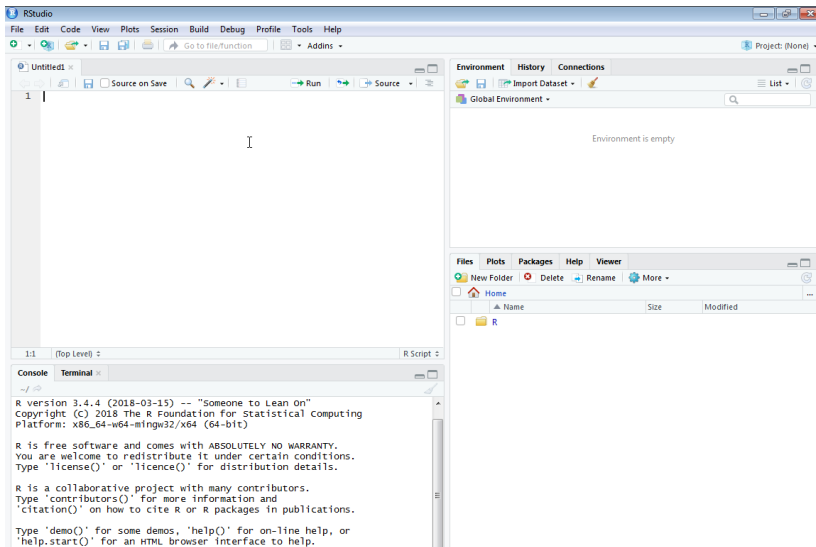
RStudio is an *Integrated Development Environment* (IDE) specifically made for R coding. One of the main advantages provided by RStudio over other editors is that we can test our code easily as we edit our scripts. Here we show an example.

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 new unnamed script.
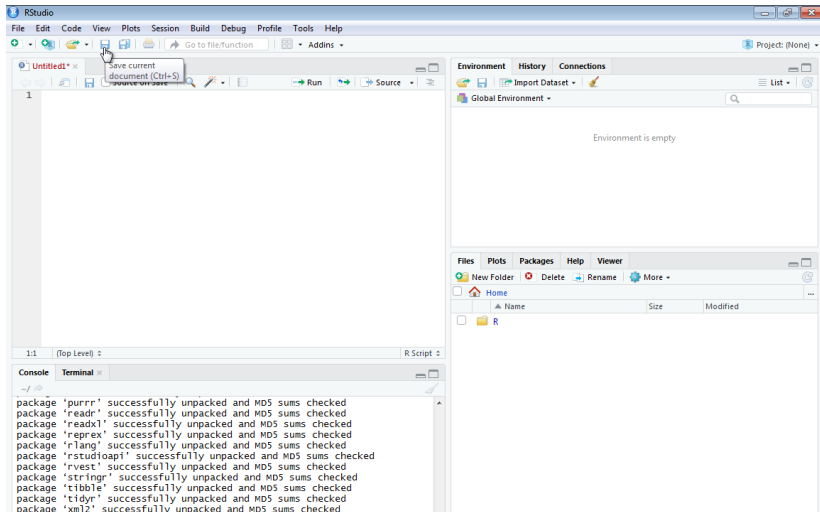
Running commands while editing scripts

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*.
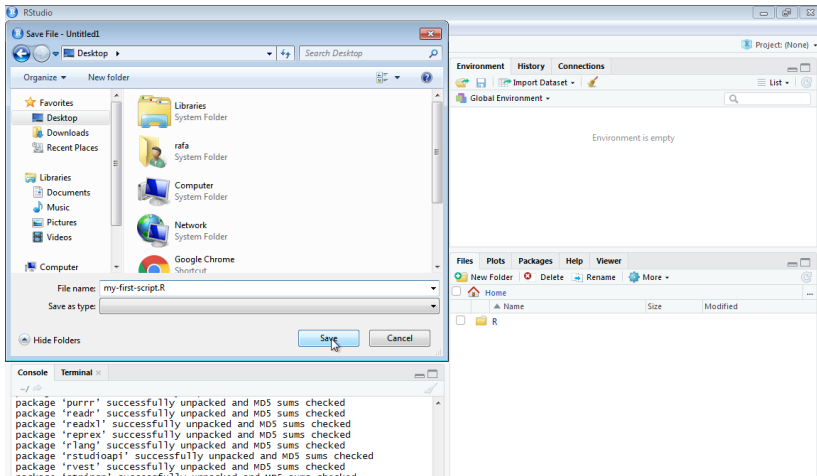
Now we are ready to start editing our first script. The first lines of code in an R script are dedicated to loading the libraries we will use.

Another useful RStudio feature is that once we type `library()` it starts auto-completing: note what happens when we type `library(ti)`:
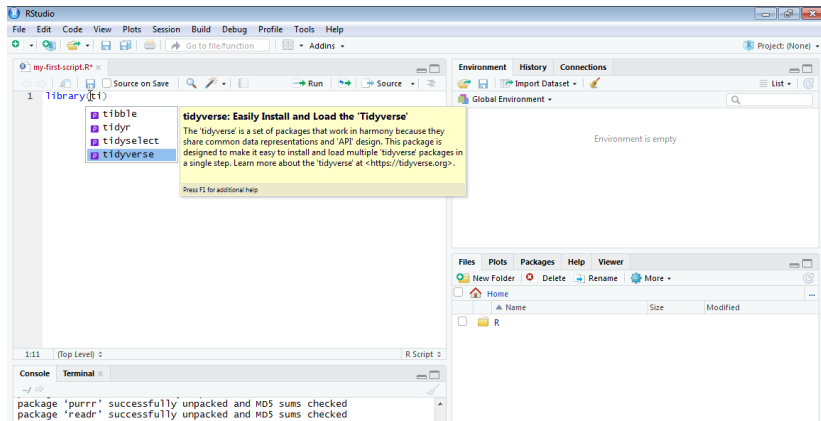
You may have noticed is that when you type `library(` the second parenthesis is automatically added. This will help you avoid one of the most common errors in coding: forgetting to close a parenthesis.

Running commands while editing scripts

Now as an example, we will make a graph showing murder totals versus population totals by state. 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.
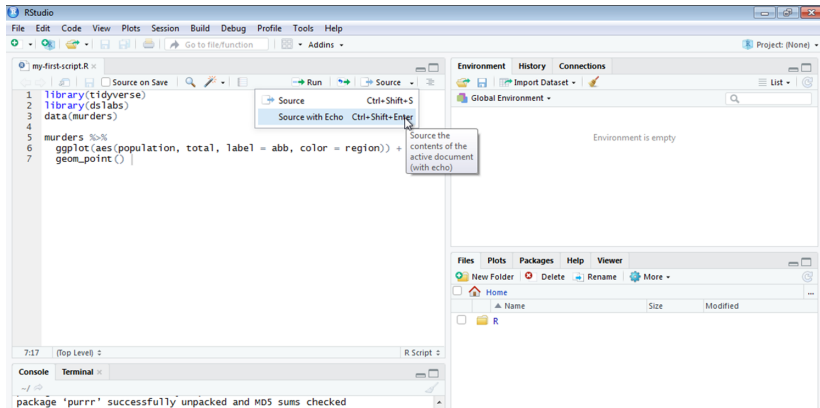
Running commands while editing scripts

You will see your code appear in the R console and, in this case, the generated plot appears. 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 plot as a file.

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*:
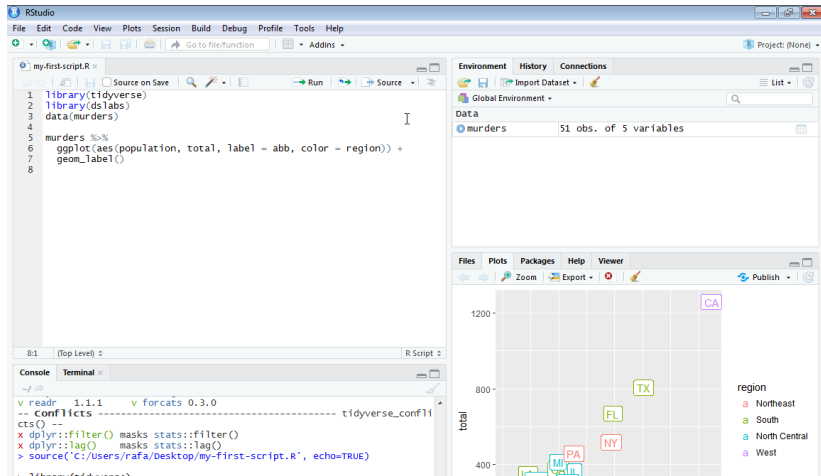
Changing global options

As an example, click on *Appearance* and then notice the *Editor theme* options.
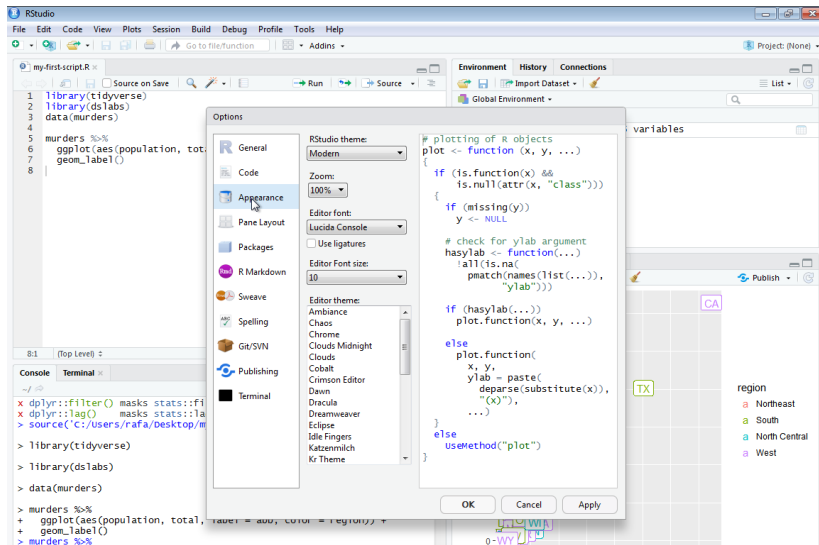
Changing global options

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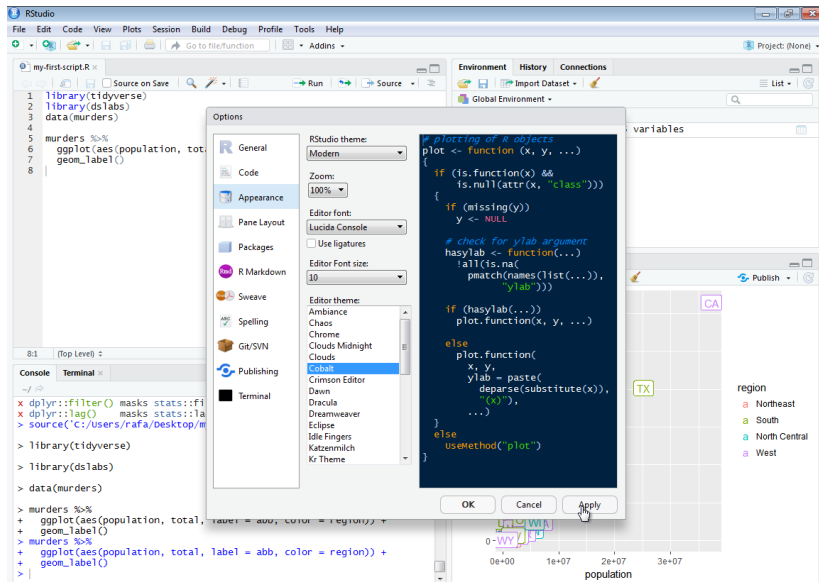
Changing global options

I personally like the *Cobalt* option:
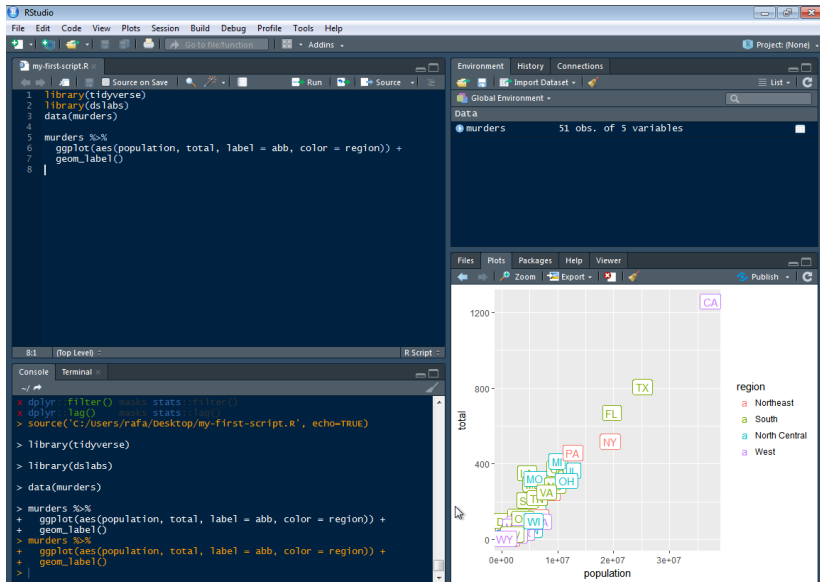
Changing global options

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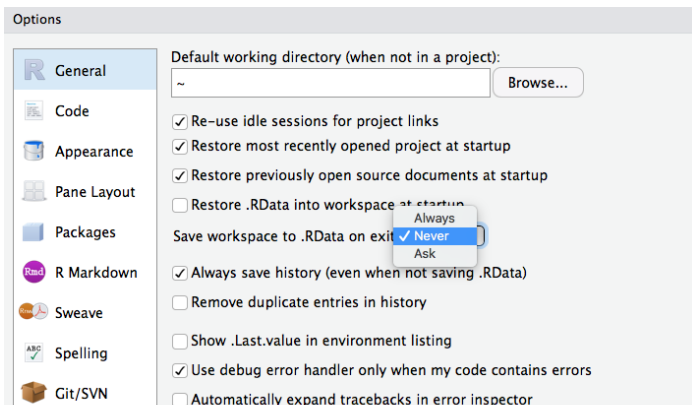
The functionality provided by a fresh install of R is only a small fraction of what is possible. In fact, we refer to what you get after your first install as **base R**. The extra functionality comes from add-ons available from developers.

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:

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

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

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

Installing R packages (Windows)

If you are using Windows, you may get a warning that says:

`WARNING: Rtools is required to build R packages but is not currently installed.`

Rtools is a toolchain bundle used for building R packages from source (those that need compilation of C/C++ or Fortran code) and to build R itself from source. We won't cover this here, but for a detailed step-by-step instruction tutorial can be found by clicking the following link:

https://jtleek.com/modules/01_DataScientistToolbox/02_10_rtools/

Installing R packages

In RStudio, you can navigate to the **Tools** tab and select install packages in the following way:
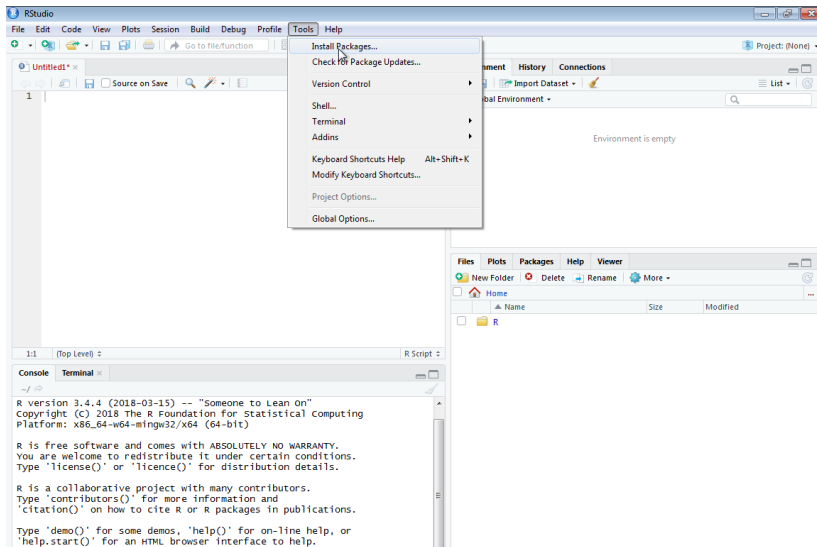
Installing R packages

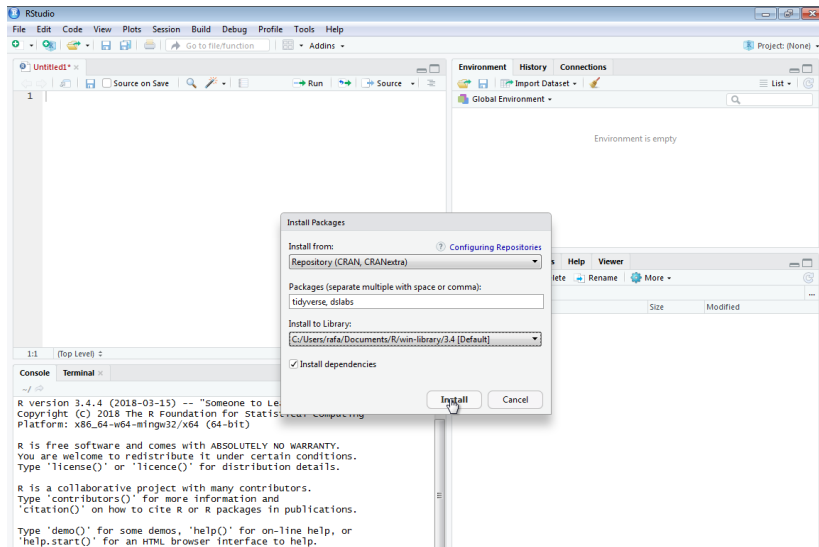RStudio will auto-complete package names, which is helpful when you do not remember the exact spelling of the package:

Installing R packages

Remember packages are installed in R not RStudio.

Also 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

Once packages are installed, you can load them into R using the `library` function:

```
library(dslabs)
```

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.

Pro Tip: 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.

You can see all the packages you have installed using the following function:

```
installed.packages()
```

Session Info

The sessionInfo() function reports information about your R session, including the OS and loaded packages. Its always good to end your analyses and reports with this function for reproducibility.

```
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Sonoma 14.2.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK vers
##
## 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_4.3.2    fastmap_1.1.1     cli_3.6.2         tools_4.3.2
##  [5] htmltools_0.5.8   rstudioapi_0.16.0 yaml_2.3.8        rmarkdown_2.26
##  [9] knitr_1.45        xfun_0.43         digest_0.6.35     rlang_1.1.3
## [13] evaluate_0.23
```