



1.3.1: Introduction to R and R Studio

(Asynchronous-Online)

Session Objectives

1. Get acquainted with R and R Studio
 2. Write simple R code in Console
 3. Create your first R script
 4. Install and load R packages (understanding your R session)
 5. Create your first R Markdown report and produce output files in different formats (HTML, PDF, or DOCX)
-

0. Prework - Before You Begin

i R versus RStudio

Note: **R** is the name of the programming language itself and **RStudio** is an integrated development environment (IDE) which is an enhanced interface for better organization, files management and analysis workflows.

Software and Applications to Download

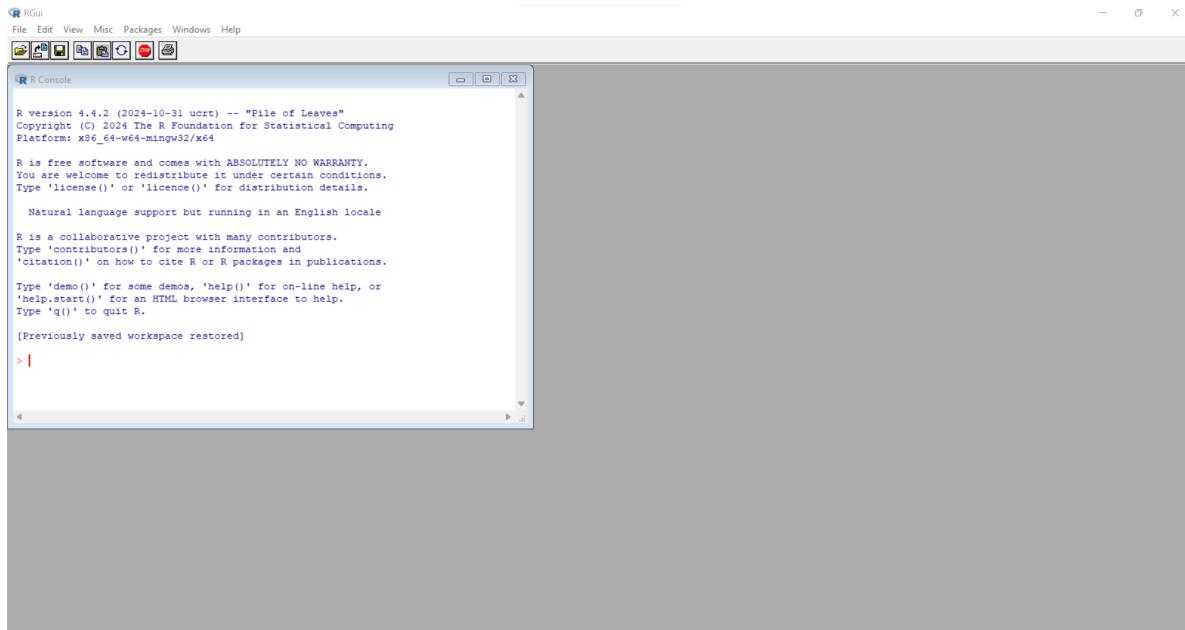
1. FIRST, Download and install R onto your computer from <https://cran.r-project.org/>.
 2. NEXT, After installing R, download and install RStudio Desktop onto your computer from <https://posit.co/download/rstudio-desktop/>.
-



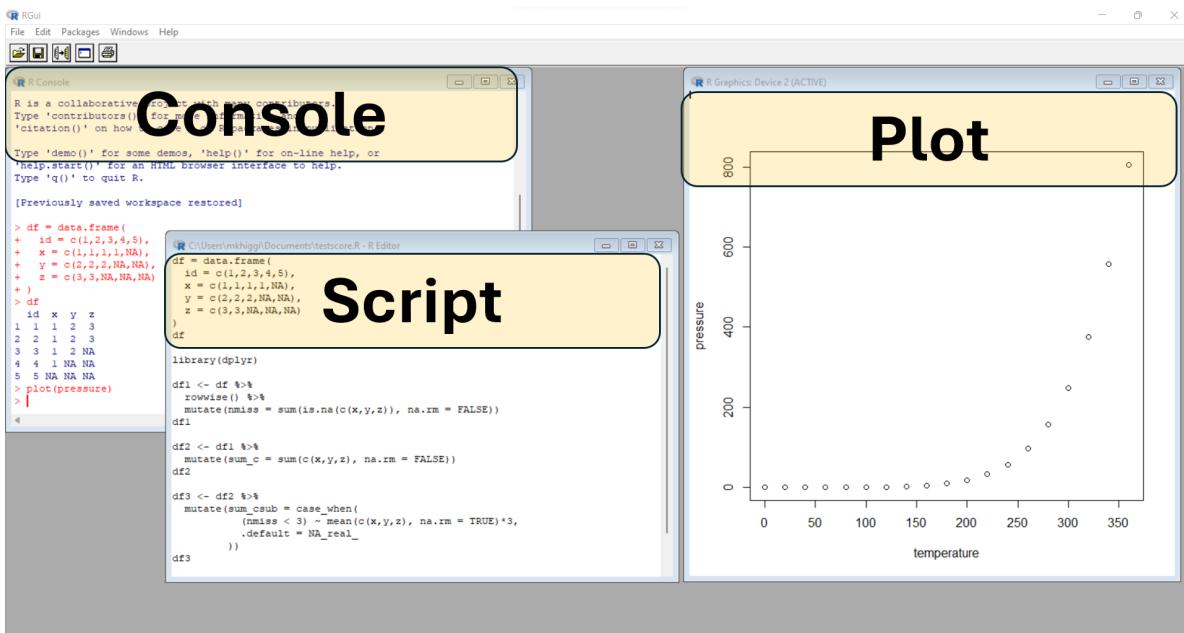
1. Get aquainted with R and R Studio

Basic R

When you download **R** from [CRAN](#) and install it on your computer, there is an R application that you can run. However, it is very bare bones. Here is a screenshot of what it looks like on my computer (Windows 11 operating system).



You can type commands in the console window at the prompt “>” but this is slow and tedious. You can also write and execute scripts from inside this application and see the output back in the console window as well as creating plots. But managing large projects using this interface is not efficient.





RStudio IDE

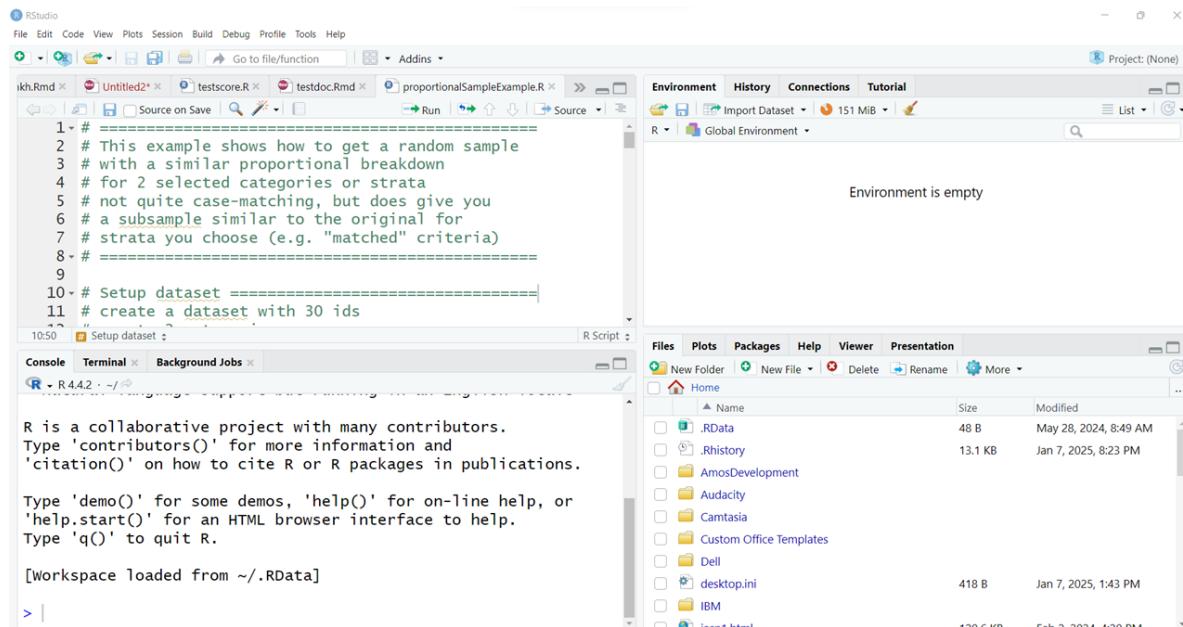
RStudio Desktop Software vs Posit the company

RStudio was founded in 2009 <https://posit.co/about/> when it published the “free and open source” RStudio software. But over time, the RStudio application has expanded beyond just being used for the R programming language. You can now use RStudio for writing and managing projects with Python code, Markdown, LaTeX, Cascading Style Sheets and more.

So, in 2022, RStudio the company became Posit <https://posit.co/blog/rstudio-is-becoming-posit/> to encompass the broader computing community.

The **RStudio Integrated Development Environment (IDE)** application provides much better tools for managing files within a given “project”. This biggest advantage of working in an IDE is everything is contained and managed within a given project, which is linked to a specific folder (container) on your computer (or cloud drive you may have access to).

However, you will still need to write and execute code using scripts and related files. An IDE is NOT a GUI (graphical user interface) which is the “point and click” workflow you may have experience with if you’ve used other analysis software applications such as SPSS, SAS Studio, Excel and similar.



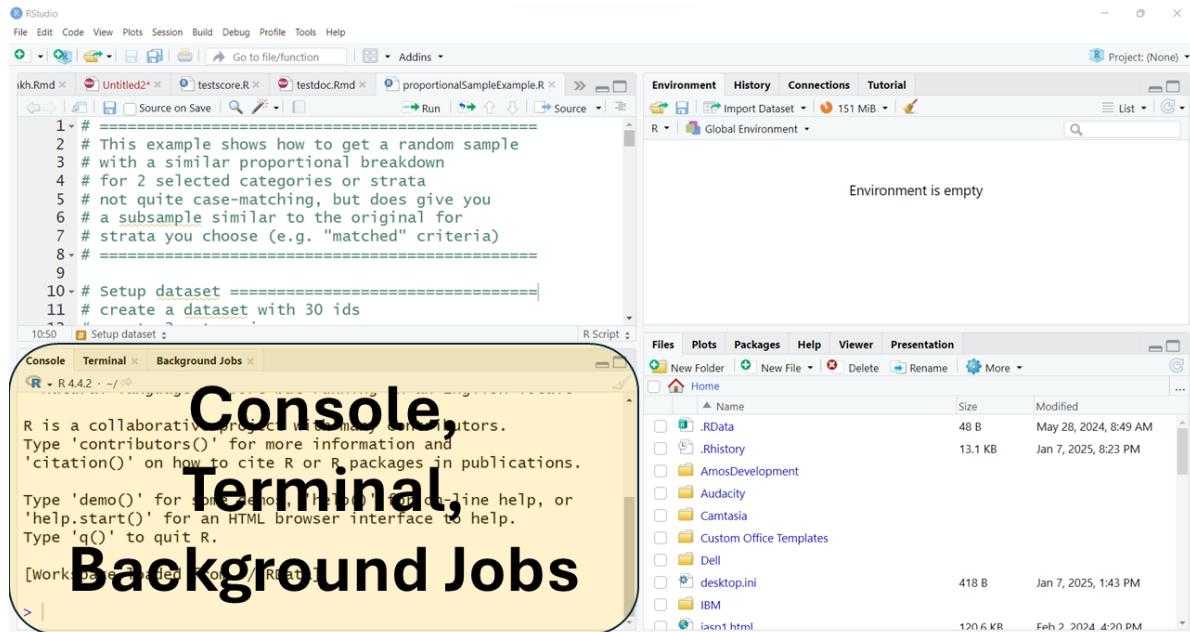
The interface is usually arranged with the following 4 “window panes”:



- Console
- Source
- Environment
- Files



The typical arrangement, usually has the “Console” window pane at the bottom left. This window also usually has TABS for the “Terminal” and any “Background Jobs” that might be running.





The “Source” window pane is usually at the top left. This is where you will do most of your editing of your R program scripts (*.R) or Rmarkdown files (*.Rmd). This is also where the data viewer window will open. You can also open and edit other kinds of files here as well (*.tex, *.css, *.txt, and more).

The screenshot shows the RStudio interface. The Source pane is highlighted with a large black oval. It contains R code for generating a random sample from a dataset. The rest of the interface includes the Environment pane, Files pane, and Console pane.

```
1 # =====
2 # This example shows how to get a random sample
3 # with a similar structure to the original
4 # for 2 selected variables
5 # not quite case-matching, but does give you
6 # a subsample similar to the original for
7 # strata you choose (case "match" matters)
8 # =====
10 # Setup dataset =====
11 # create a dataset with 30 ids
```

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.

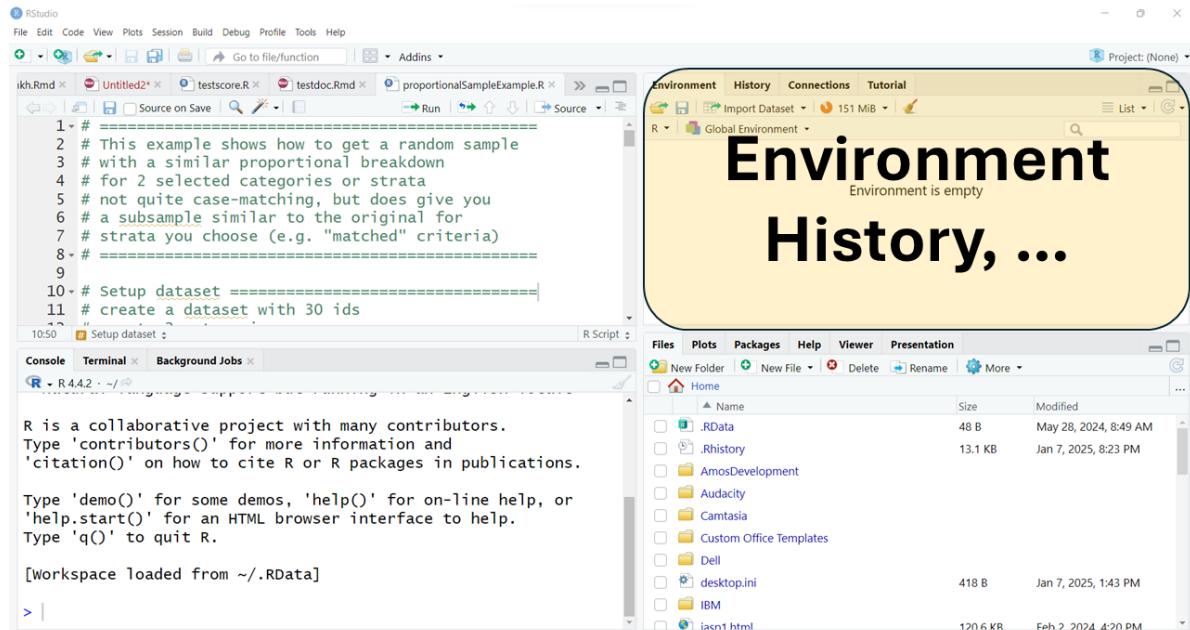
[workspace loaded from ~/.RData]

> |

Name	Size	Modified
.RData	48 B	May 28, 2024, 8:49 AM
.Rhistory	13.1 KB	Jan 7, 2025, 8:23 PM
AmosDevelopment		
Audacity		
Camtasia		
Custom Office Templates		
Dell		
desktop.ini	418 B	Jan 7, 2025, 1:43 PM
IBM		
jasn1.html	120.6 KB	Feb 2, 2024, 4:20 PM



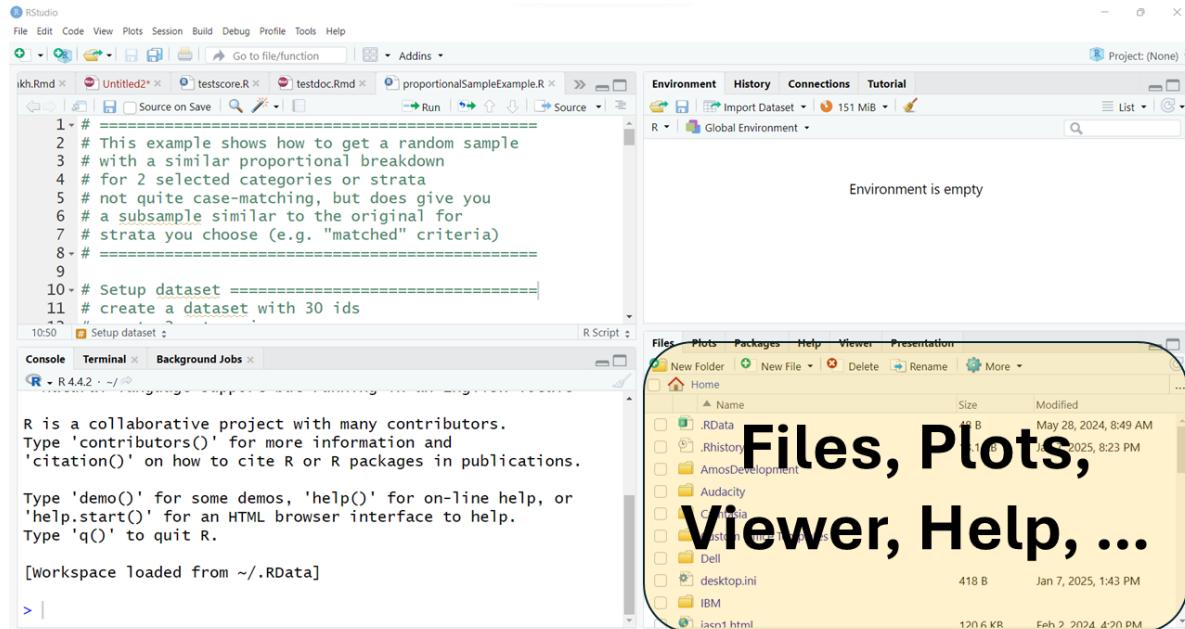
The top right window pane should always have your “Environment”, “History” and “Tutorial” TABs but may also have TABs for “Build” and “Git” and others depending on your project type and options selected.





The bottom right window pane has TABS for your:

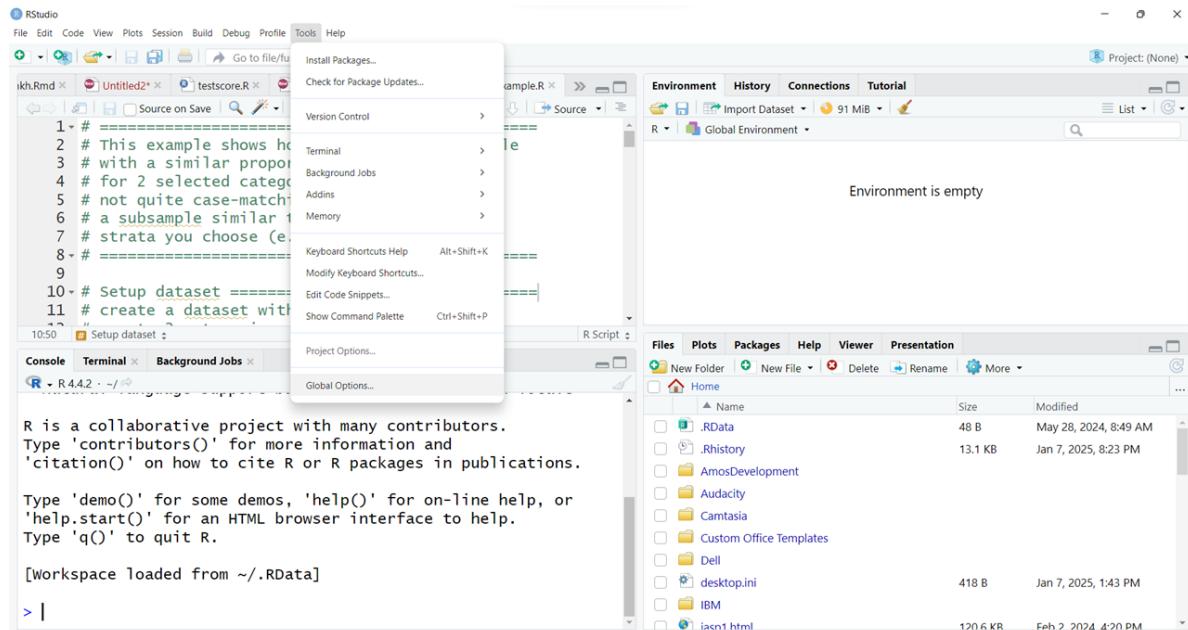
- “Files” directory
- “Plots” window for graphical output
- “Packages” - which lists all add-on R packages installed on your computer
- “Help” window
- as well as other TABs for “Viewer” and “Presentation” for viewing other kinds of output.





Customizing your RStudio interface

You also have the option to rearrange your window panes as well as change the look and feel of your programming interface and much more. To explore all of your options, click on the menu at the top for “Tools/Global Options”:



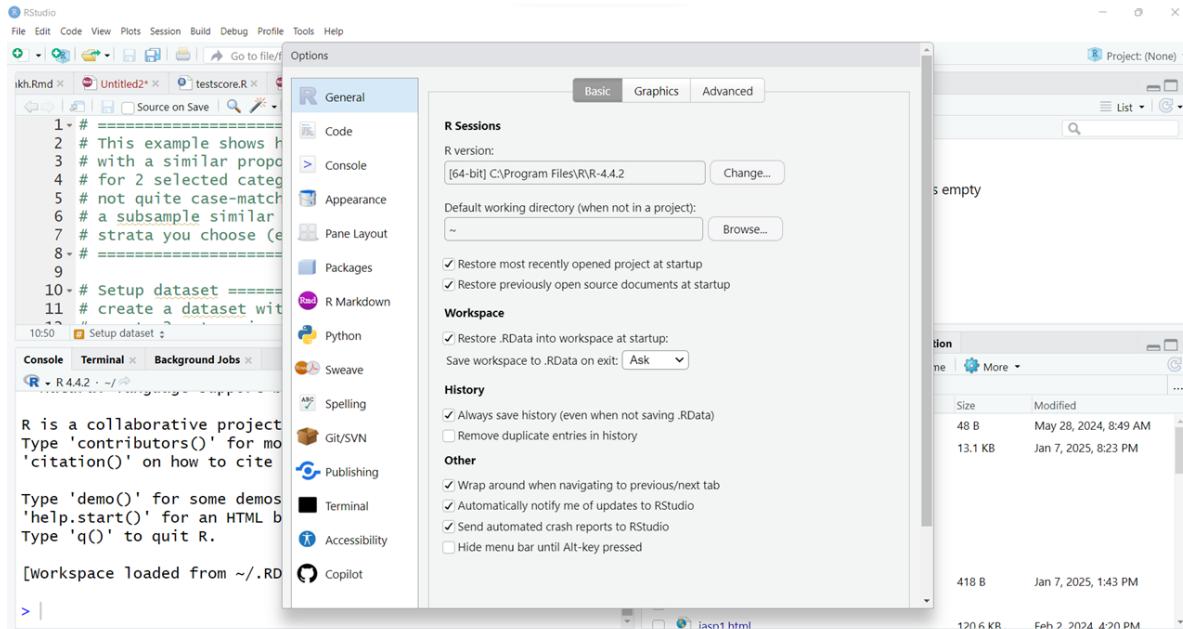


Take a look at the left side for the list of all of the options. Some of the most useful options to be aware of are:

- General
- Appearance, and
- Pane Layout



In the “General” TAB is where you can see and confirm that R is installed and where the R programming language executable is installed on your computer.



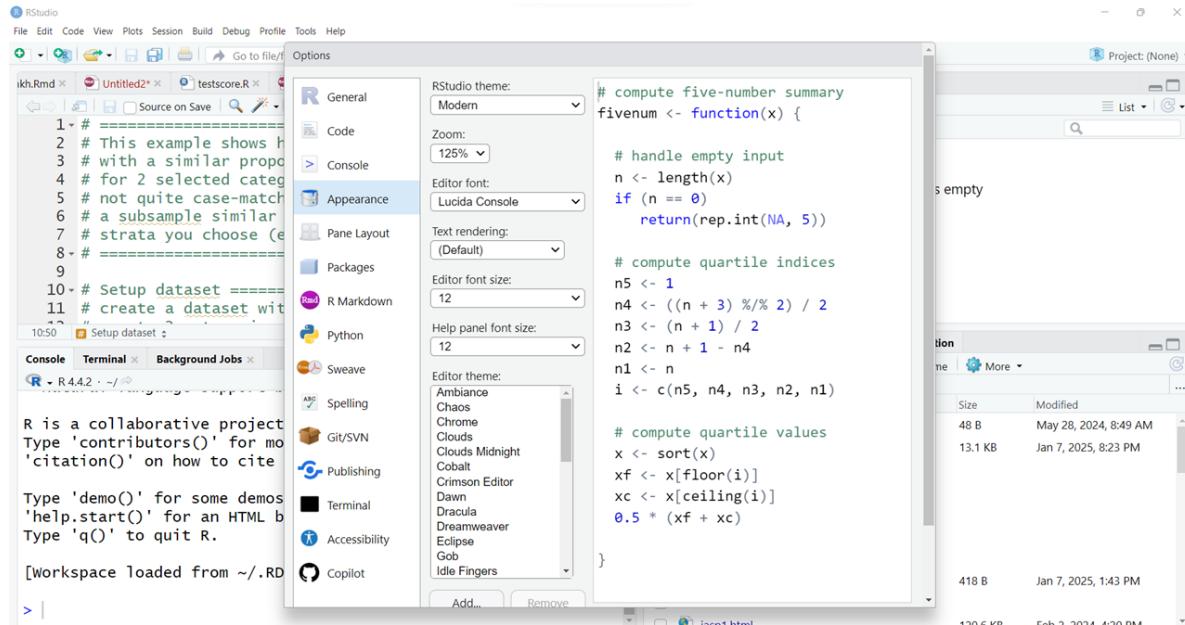


You will probably want to explore fine-tuning these parameters to customize the appearance of your RStudio preferences. For example, you can change the ZOOM level to improve readability. You may also want to change the FONT sizes for the Editor and Help windows as needed.

💡 ZOOM + FONT

When making changes to your RStudio interface appearance, be aware that ZOOM and FONT size settings work together, so you may need to play around with the settings that work best for your monitor or device you are using.

I also encourage you to try out different “Editor Themes” which will change the colors of the R code as well as background colors (light or dark).





The default “Editor Theme” is textmate.

The screenshot shows the RStudio Options dialog with the Appearance tab selected. The 'Editor theme:' dropdown is set to 'Textmate (default)'. The main pane displays R code for computing a five-number summary:

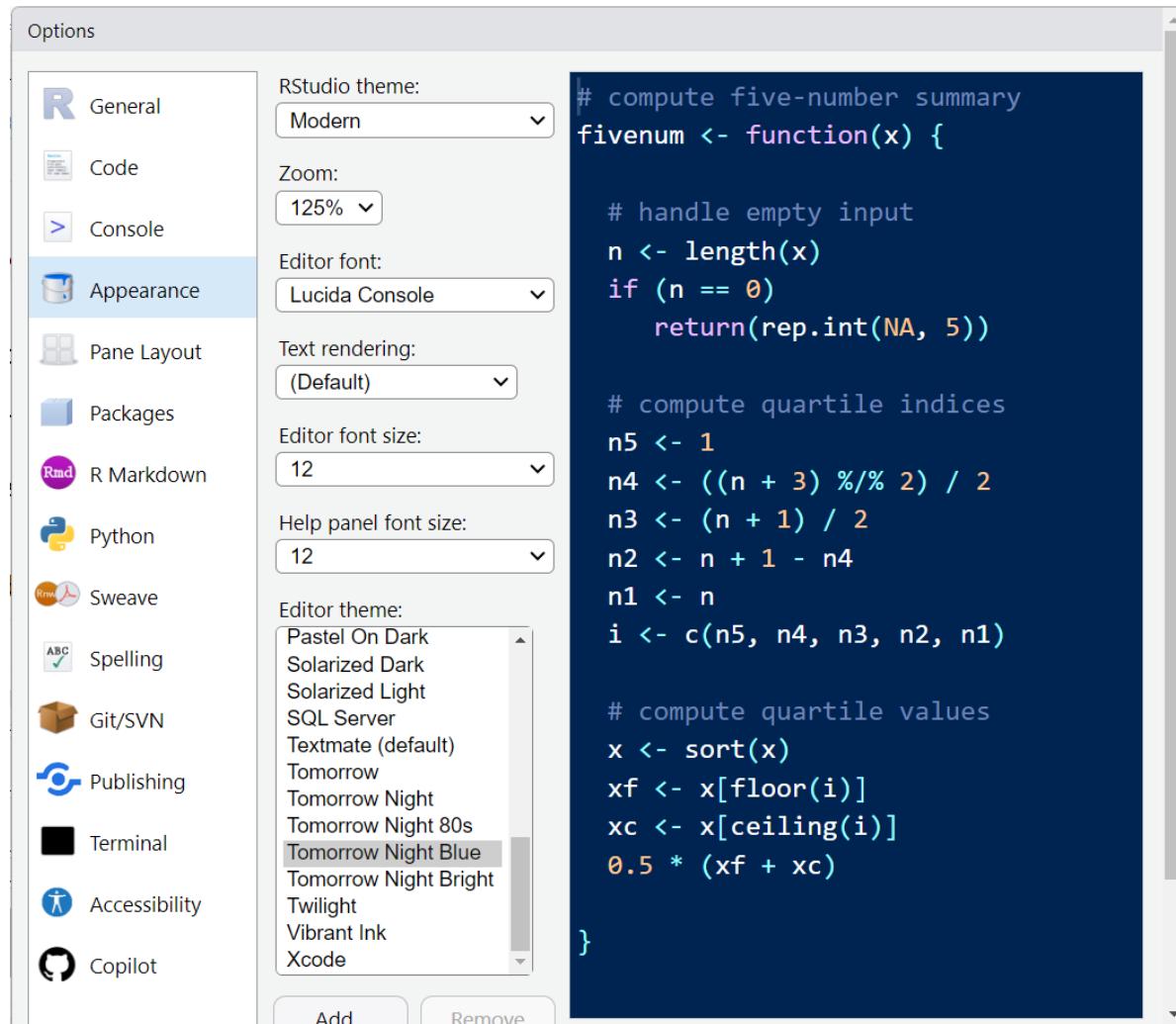
```
# compute five-number summary
fivenum <- function(x) {
  # handle empty input
  n <- length(x)
  if (n == 0)
    return(rep.int(NA, 5))

  # compute quartile indices
  n5 <- 1
  n4 <- ((n + 3) %/% 2) / 2
  n3 <- (n + 1) / 2
  n2 <- n + 1 - n4
  n1 <- n
  i <- c(n5, n4, n3, n2, n1)

  # compute quartile values
  x <- sort(x)
  xf <- x[floor(i)]
  xc <- x[ceiling(i)]
  0.5 * (xf + xc)
}
```

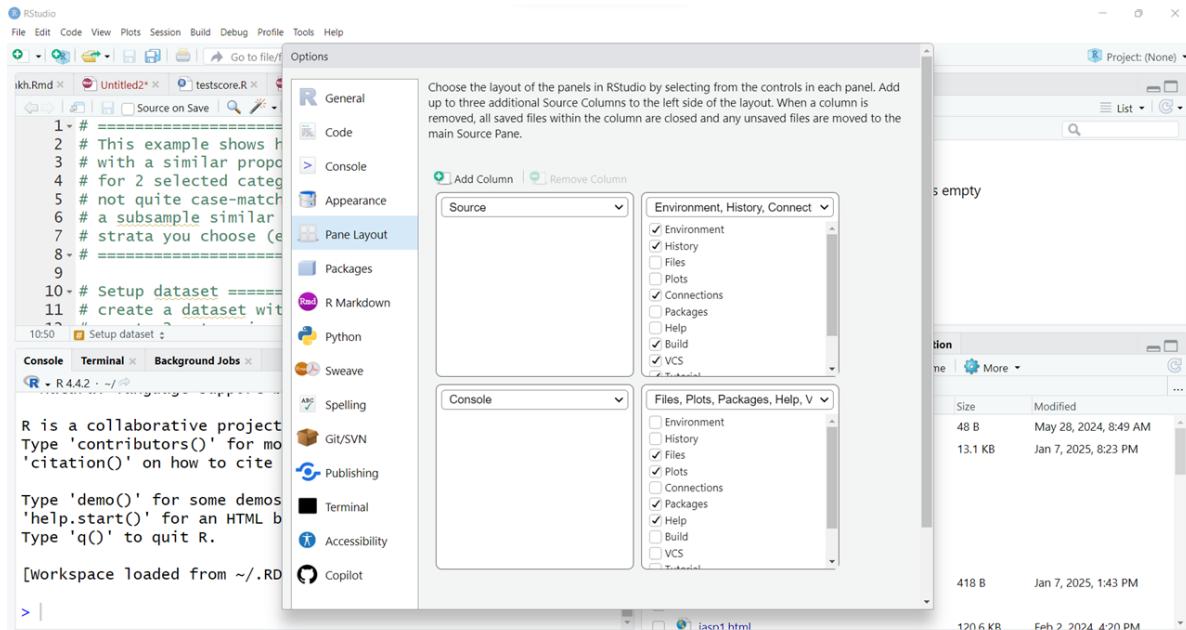


But here is an example of changing the theme to “Tomorrow Night Blue”.





I would also suggest NOT changing the layout of the window panes until you are very familiar with the default settings. But in “Pane Layout” is where you can see what the default layout settings are and what other options are available to you.





2. Write simple R code in Console

Simple math

So, let's start with some simple R code using the Console window and typing commands at the > prompt (which is the greater than symbol).

You can write simple math expressions like 5 + 5.

```
5 + 5
```

```
[1] 10
```

Notice that the output shows the number 1 enclosed in square brackets [] followed by the answer (or output) of 10.

This is because R performed the addition operation using the + operator and then “saved” the output in temporary memory as a scalar object with 1 element, which is the number 10.

You can actually see this temporary object by typing `.Last.value` - which is only temporary and will be overwritten after the execution of the next R command.

```
.Last.value  
[1] 10
```



However, if we look at our current computing environment (see “Global Environment” upper right window pane), it is still showing as empty.

The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help, and Addins. The title bar says "emory_tidal_Rlectures - main - RStudio". The left sidebar has tabs for Source, Console (which is active and shows R startup messages), Terminal, and Background Jobs. The main area shows a command-line session starting with "R is free software and comes with ABSOLUTELY NO WARRANTY...". The bottom right corner of the main area has a red callout pointing to the "Environment" tab in the top navigation bar, which is highlighted. The "Environment" tab is part of a larger tab bar that also includes History, Connections, Build, Git, Tutorial, and others. Below the tabs, the "Global Environment" pane is visible and contains the message "Environment is empty". The bottom of the screen shows the RStudio footer with tabs for Files, Plots, Packages, Help, Viewer, Presentation, and a few icons.

This is because we have not yet “saved” the output into an object that we created. Let’s save the output from $5 + 5$ into an object called `ten`.

To do this we need to do 2 things:

1. Create the object called `ten` by
2. Using the “assign” operator `<-` to take the result of $5 + 5$ and move it (save it or pipe it) into the object `ten`.

```
ten <- 5 + 5
```



TL;DR What is the Assign Operator <-?

The “R” language is actually a derivative of the [original “S” language](#) which stood for the “language of statistics” - it was written by statisticians for statisticians (and now data scientists). *The original S language was written in the mid-1970’s by programmers/statisticians at Bell Labs/AT&T.*

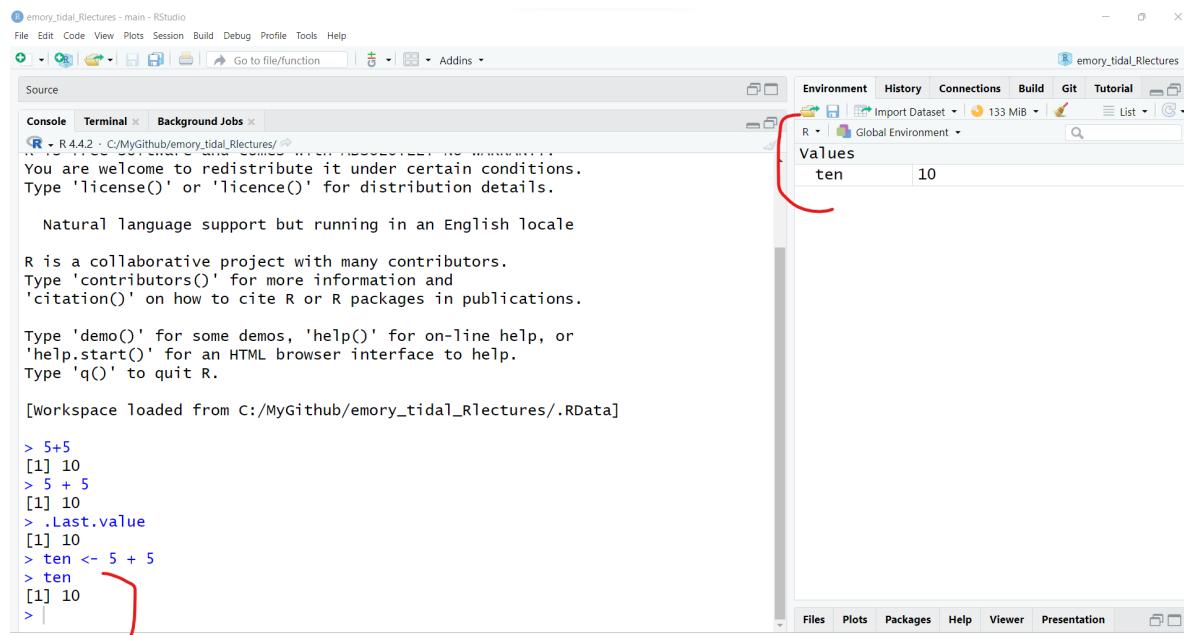
The <- actually comes from the physical key on their [“APL” keyboards](#), for the APL programming language they were using at Bell Labs.

[A Nice Blog Post on the History of <-](#)

To “see” the output of this object called `ten` - you can either see it now in your Global Environment or type the object name in the Console to view it.

```
ten
```

```
[1] 10
```



```
emory_tidal_Rlectures - main - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Source Environment History Connections Build Git Tutorial
Console Terminal Background Jobs
R 4.4.2 - C:/MyGithub/emory_tidal_Rlectures/
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.

[Workspace loaded from C:/MyGithub/emory_tidal_Rlectures/.RData]

> 5+5
[1] 10
> 5 + 5
[1] 10
> .Last.value
[1] 10
> ten <- 5 + 5
> ten
[1] 10
> |
```

It is important to remember that R is an “object-oriented” programming language - everything in R is an object.



Built in constants

There are several built in “constants” in R. Try typing these in at the Console to see the results.

```
pi
```

```
[1] 3.141593
```

```
letters
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"  
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
month.name
```

```
[1] "January"    "February"    "March"        "April"        "May"         "June"  
[7] "July"       "August"       "September"   "October"     "November"    "December"
```

! R is Case Sensitive!

A pro and con of the R language is that it is case sensitive. Lower case x and uppercase X are different objects. As seen above, the lowercase `letters` object is a vector of the 26 lowercase letters, whereas `LETTERS` is a different object vector of the 26 uppercase letters. Be on the lookout for case sensitive spelling and formatting of object, package and function names in R.

For the constants like `letters` you get a list of the 26 lower case letters in the alphabet. Notice that the number in [square brackets] updates for each new line printed out. This allows you to keep track of the number of elements in the output object. `letters` is an “character” array (or vector) with 26 elements.

To confirm these details, we can use the `class()` function to determine that the `letters` object has all “character” elements. The `length()` function will let you know that there are 26 elements.



```
class(letters)
```

```
[1] "character"
```

```
length(letters)
```

```
[1] 26
```



Getting help

If you would like to learn more about these built-in “constants”, you can get help in one of two ways. You can either type `help(pi)` in the “Console” (lower left) or type `pi` in the “Help” window (lower right).

```
help(pi)
```

The screenshot shows the RStudio interface. In the top-left corner, there's a small logo of a person's head with blue and purple hair. The main title bar says "emory_tidal_Rlectures - main - RStudio". The menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. Below the menu is a toolbar with various icons. The left pane contains tabs for Source, Console, Terminal, and Background Jobs. The Console tab is active, showing the command `> help(pi)`. The right pane is the Help window, which has tabs for Environment, History, Connections, Build, Git, and Tutorial. It currently displays the "R: Built-in Constants" section. A search bar at the top of the Help window says "pi". The main content area shows the "Built-in Constants" section, which includes a "Description" (Constants built into R.) and a "Usage" section listing "LETTERS", "letters", "month.abb", "month.name", and "pi". There's also a "Details" section stating "R has a small number of built-in constants." and a note about available constants.

The `help()` function defaults to searching for a built-in object, function or dataset by default in the `base` R package. But some functions may exist in multiple packages, so it is always a good idea to add the package when running the `help()` function if possible.

Since `pi` is in the `base` R package, it would be better to run:

```
help(pi, package = "base")
```

If you have no idea what package a function may be in, you can use the `??` search operator. For example, many packages include a plotting related function. If you want to see how many R packages currently installed on your computer have a `plot` related function, type the following:

```
??plot
```



Try out a built-in R function

The majority of the R programming language is driven by functions. Technically the `+` operator is actually a function that performs a sum.

You can even get help on these operators, by typing `help("+")`. We have to add the quotes `" "` so that R knows we are looking for this operator and not trying to perform an addition operation inside the function call.

```
help("+")
```

But let's try a function to create a sequence of numbers - for example, let's use the `seq()` function to create a sequence of numbers from 1 to 10.

```
seq(10)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

And let's look at the help page for the `seq()` function.



The screenshot shows the R documentation for the `seq` function. The title bar says "R: Sequence Generation". The main content is titled "Sequence Generation". It includes sections for "Description", "Usage", "Arguments", and "Details". The "Usage" section contains code examples for the default S3 method and its primitives. The "Arguments" section lists parameters like `from`, `to`, `by`, `length.out`, and `along.with`. The "Details" section notes that numerical inputs should be finite and that unnamed arguments are not standard.

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

Arguments

<code>...</code>	arguments passed to or from methods.
<code>from, to</code>	the starting and (maximal) end values of the sequence. Of length 1 unless just <code>from</code> is supplied as an unnamed argument.
<code>by</code>	number: increment of the sequence.
<code>length.out</code>	desired length of the sequence. A non-negative number, which for <code>seq</code> and <code>seq.int</code> will be rounded up if fractional.
<code>along.with</code>	take the length from the length of this argument.

Details

Numerical inputs should all be `finite` (that is, not infinite, `NaN` or `NA`).

The interpretation of the unnamed arguments of `seq` and `seq.int` is *not* standard, and it is recommended always to name the

R allows for what is called “lazy” coding. This basically means you can provide very minimal input and R will try to figure out what you want using the default settings for a given function. In the case of `seq()` the function begins by default at 1 and creates the output in steps of 1 up to the value of 10.

While “lazy” coding practices are common with R, it would actually be better to explicitly define each **argument** to make sure you get the exact output you want. To do this, inside the parentheses () we should assign a value to each argument.

Notice in the “Help” page for `seq()` shown above that the first 3 arguments are: `from`, `to` and `by`. Each of these can be defined inside the () by using the syntax of the name of the



argument, an equals sign =, and then the value (or object) you want to assign:

$$\text{argument} = \text{value}$$

For example, the explicit function call should be:

```
seq(from = 1,  
     to = 10,  
     by = 1)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

You could easily change these values to get a sequence from 0 to 5 in increments of 0.1 as follows:

```
seq(from = 0,  
     to = 5,  
     by = 0.1)
```

```
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8  
[20] 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7  
[39] 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0
```

Notice the incremental counter [#] on the left to help you keep track of how many elements are in the resulting numeric vector that was the “result” or “output” from the `seq()` function.



3. Create your first R script

Save your code in a new script

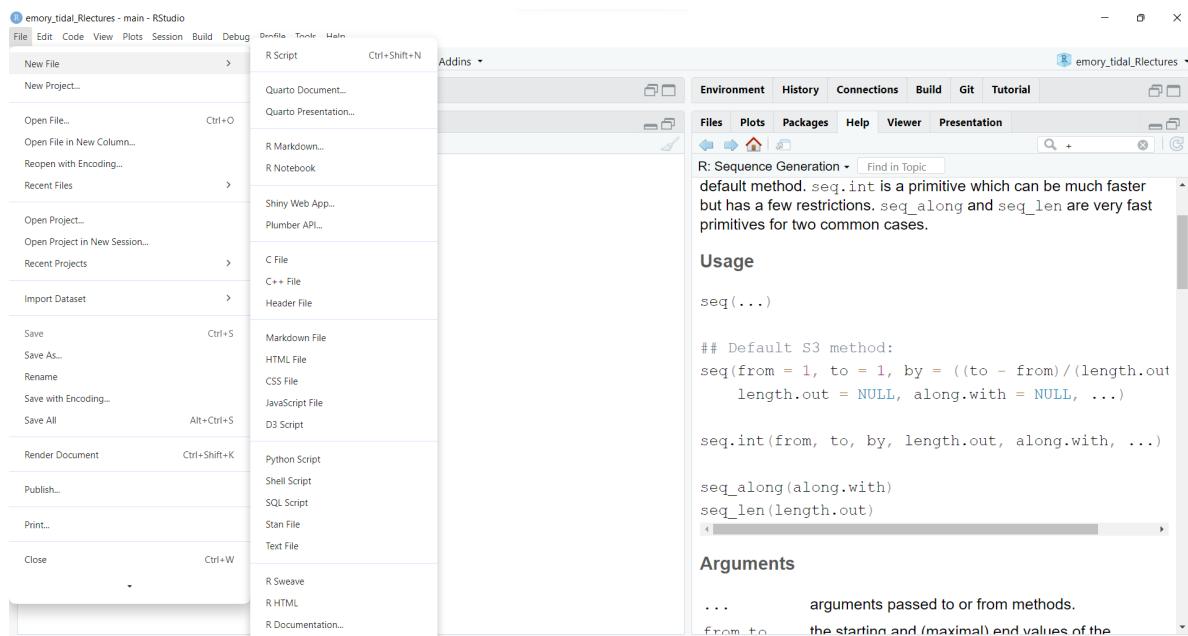
So, as you can tell, the R Console is useful but slow and tedious. Let's create an R script to save all of these commands in a file so that we can easily access everything we've done so far and re-run these commands as needed.

💡 Good Reproducible Research Coding Practice

It is a good coding practice to create R code (saved in `*.R` script files or `*.Rmd` Rmarkdown files) for every step in your data preparation and analysis so that:

- you have a record of everything you've done and why
- other people on your team (including yourself in a few weeks) will know what you did and why
- you can share your code with others so they will understand what you did and why (*and to publish your code and data with your research articles - YES you can get a DOI citation to add to your CV for data and code as well as for the article!*)

In RStudio go to the top menu “File/New File/R Script”:



Once the R Script file is created, type in some of the commands we did above in the Console and put one command on each line.

Just select each line and click “Run”.



The screenshot shows the RStudio interface with a script editor window open. The code in the editor is:

```
1 4 + 4
2 sqrt(25)
3 pi
4 seq(from=1, to=10, by=0.5)
5
```

A red arrow points from the word "sqrt" in the second line to the "Run" button in the toolbar above the editor.

Then you can save the file on your computer as “myscript.R”, for example.

You can also select all of the rows and click run to execute all of the code in sequence and see the output in the “Console” Window.

The screenshot shows the RStudio interface with a script editor window open. The code in the editor is:

```
1 4 + 4
2 sqrt(25)
3 pi
4 seq(from=1, to=10, by=0.5)
5
```

A red bracket highlights the entire code block. A red arrow points from this bracket to the "Run" button in the toolbar above the editor. Below the editor is a "Console" window showing the execution of the code:

```
> 4 + 4
[1] 8
> sqrt(25)
[1] 5
> pi
[1] 3.141593
> seq(from=1, to=10, by=0.5)
[1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0
[16]  8.5  9.0  9.5 10.0
>
```

Here is the code and output:



```
4 + 4
```

```
[1] 8
```

```
sqrt(25)
```

```
[1] 5
```

```
pi
```

```
[1] 3.141593
```

```
seq(from=1, to=10, by=0.5)
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0  
[16] 8.5 9.0 9.5 10.0
```



Create R objects and Use Them

Let's try out some more built-in R functions, save the output in objects in your "Global Environment" and then use them in other functions and subsequent analysis steps.

Create a sequence of numbers and save them as an object called `x`. I also added a comment in the R code block below. Everything after the `#` hashtag is a comment which R will ignore. It is a good idea to add comments in your code to make sure that you and others understand what each part of your code does (*including yourself in a few weeks when you've forgotten why you wrote that code step*).

```
# save sequence of numbers
# from 1 to 10 in steps of 0.5
# in an object named x
x <- seq(from=1, to=10, by=0.5)

# Type x to view the contents
x
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
[16] 8.5 9.0 9.5 10.0
```

Also take a look at the "Global Environment" to see the new object `x`.

The screenshot shows the RStudio interface with the "Global Environment" tab selected in the top navigation bar. The "Values" section displays two objects: "ten" with value 10 and "x" with value [1:19] 1 1.5 2 2.5 3... A red arrow points from the "x" entry in the environment pane back to the "x" assignment in the R code block above. Another red bracket highlights the "x" assignment and its resulting output in the code block.

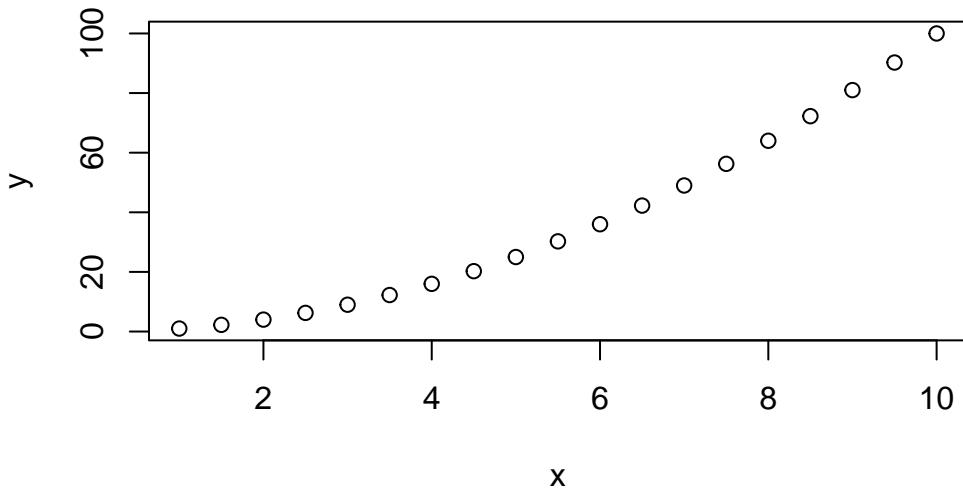


```
# use x to create new object y  
y <- x*x
```

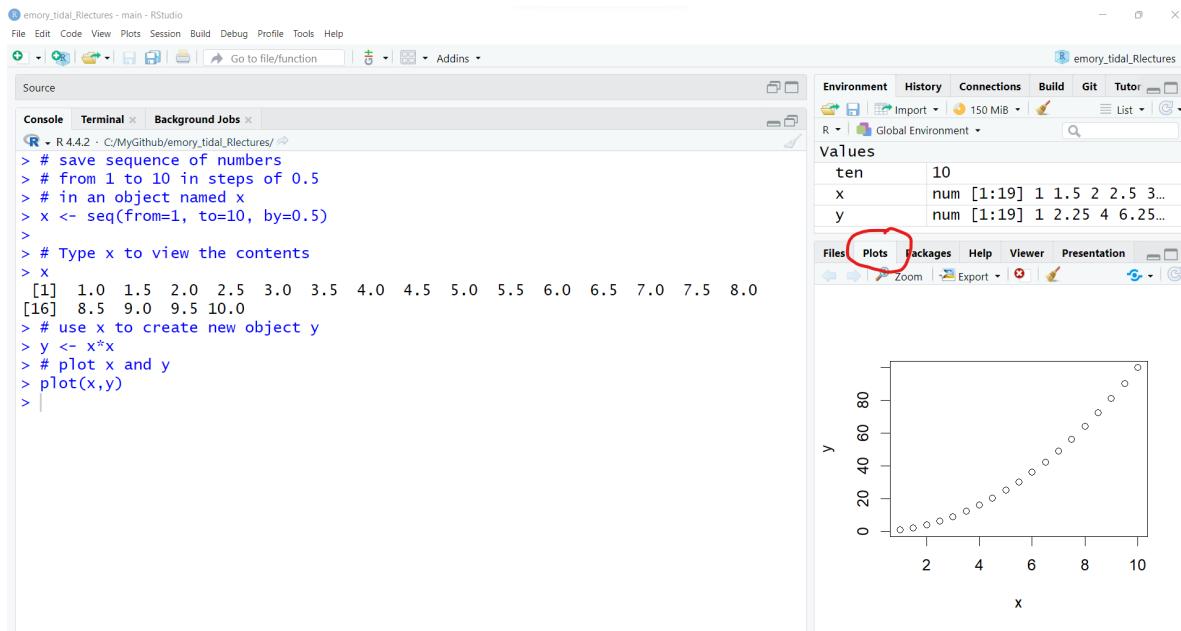


Once the object `y` is created, we can make a simple 2-dimensional scatterplot using the built-in `plot()` base R function.

```
# plot x and y  
plot(x,y)
```



The plot is shown above, but if you are running this code interactively in the RStudio desktop, check the “Plots” window pane (lower right).



On your own

Download [Rscript_01.R](#) (*right click the linked file and “Save As” the file on your computer*), then open it in your RStudio and run through the code. Try out new variations on your own.



4. Install and load R packages (*understanding your R session*)

Status of your current computing R session with `sessionInfo()`

While the base installation of R is pretty powerful on its own, the beauty of R and the R programming community is that there are literally hundreds of thousands if not millions of people programming in R and creating new functions everyday.

In order to use these new functions, the developers put them together in packages that we can install to extend the functionality of R.

But first, let's take a look at the packages that are part of the basic installation of R. One way to see which packages are currently installed and loaded into your current R computing session, is by running the command `sessionInfo()`.

Watch spelling - R is case sensitive!

Notice: This function name is all lowercase except for the capital "I" in the middle. Be sure you are typing `sessionInfo()` and not `sessioninfo()`.

You will also notice that the `sessionInfo()` command also lists the version of R I'm currently running (4.4.2), my operating system (Windows 11) and my locale (USA, East Coast). These details can sometimes be helpful for collaborating with others who may be working with different system settings and for debugging errors.

```
sessionInfo()
```



```
Console Terminal x Background Jobs x
R -> sessionInfo()
R version 4.4.2 (2024-10-31 ucrt)
Platform: x86_64-w64-mingw32/x64
Running under: Windows 11 x64 (build 22000)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] stats      graphics   grDevices utils      datasets   methods    base

loaded via a namespace (and not attached):
[1] compiler_4.4.2    tools_4.4.2       rstudioapi_0.15.0
> |
```

7 Base R Packages

The basic installation of R includes 7 packages:

- stats
- graphics
- grDevices
- utils
- datasets
- methods
- base

To learn more click on the “Packages” TAB in the lower right window pane to see the list of packages installed on your computer. I have a lot of packages on my computer, but here is a screenshot of the base R packages.

See the packages listed under “System Library” which are the ones that were installed with base R. You’ll notice that only some of these have checkmarks next to them. The checkmark means those are also loaded into your R session. Only some of them are loaded into memory by default to minimize the use of your computer’s memory.



The screenshot shows the RStudio interface with the 'Packages' tab selected in the top navigation bar. The left pane displays the R console output, which includes session information, locale settings, time zone, attached base packages, and loaded via namespaces. The right pane shows the 'System Library' where the 'base' package is listed with its version (4.4.2) and a brief description. A red arrow points from the 'base' entry in the table to the 'base' package entry in the console output.

Name	Description	Version
base	The R Base Package	4.4.2
boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-31
class	Functions for Classification	7.3-22
cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.1.6
codetools	Code Analysis Tools for R	0.2-20
compiler	The R Compiler Package	4.4.2
datasets	The R Datasets Package	4.4.2
foreign	Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase'	0.8-87
graphics	The R Graphics Package	4.4.2
grDevices	The R Graphics Devices and Support for Colours and Fonts	4.4.2
grid	The Grid Graphics Package	4.4.2
KernSmooth	Functions for Kernel Smoothing Sunilnrao Wand & Jones (1995)	2.23-24



Install a Package and Load it into R session memory

Let's install a "new" R package, like `ggplot2`.

Go to the RStudio menu "Tools/Install" Packages

The screenshot shows the RStudio interface. On the left, the console window displays R session information and command history. On the right, the environment pane shows various packages installed. A context menu is open over the 'Tools' menu bar, with 'Install Packages...' highlighted.

Console output:

```
R - R 4.4.2 · C:/MyGitHub/emory_tidal_Rlectures/
> sessionInfo()
R version 4.4.2 (2024-10-31 ucrt)
Platform: x86_64-w64-mingw32/
Running under: Windows 11 x64

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] stats      graphics   grDevices utils      datasets   methods    base

loaded via a namespace (and not attached):
[1] compiler_4.4.2    tools_4.4.2      rstudioapi_0.15.0
> |
```

Packages installed (selected in the 'Install' tab):

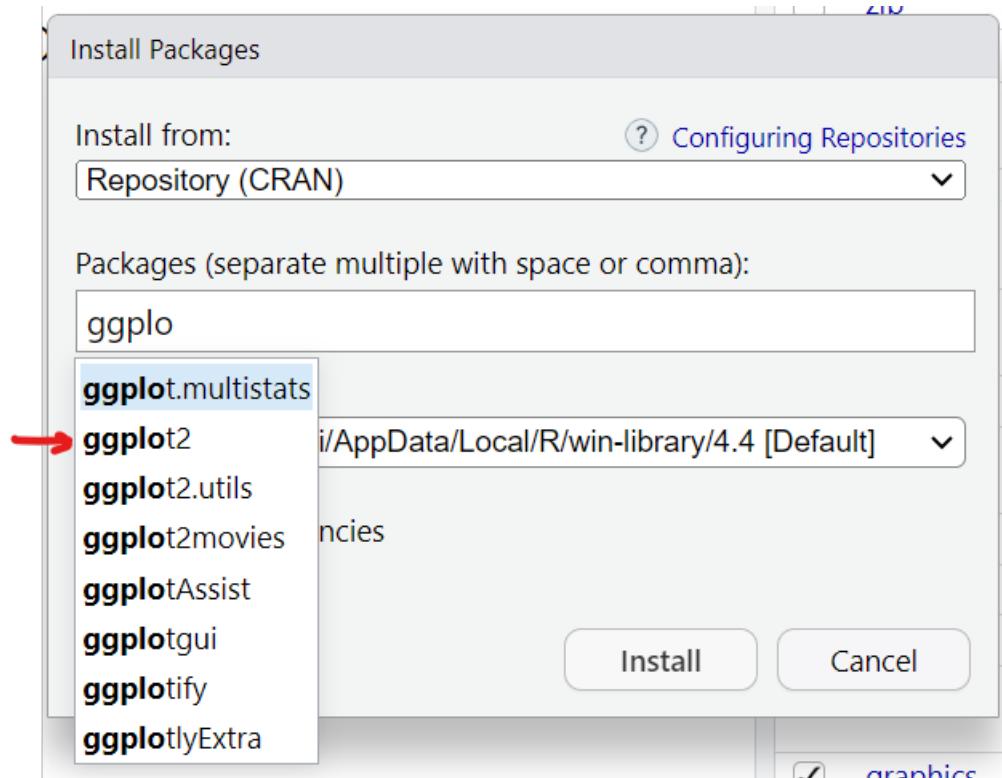
Name	Description	Version
zip	Cross-Platform 'zip' Compression	2.3.1
zlibbioc	An R Packaged zlib-1.2.5	1.48.0
zoo	S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)	1.8-12

System Library (selected in the 'Update' tab):

Name	Description	Version
base	The R Base Package	4.4.2
boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-31
class	Functions for Classification	7.3-22
cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.1.6
codetools	Code Analysis Tools for R	0.2-20
compiler	The R Compiler Package	4.4.2
datasets	The R Datasets Package	4.4.2
foreign	Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...	0.8-87
graphics	The R Graphics Package	4.4.2
grDevices	The R Graphics Devices and Support for Colours and Fonts	4.4.2
grid	The Grid Graphics Package	4.4.2
KernSmooth	Functions for Kernel Smoothing Supporting Wand & Jones (1995)	2.23-24
lattice	Trellis Graphics for R	0.22-6
MASS	Support Functions and Datasets for Venables	7.3-61



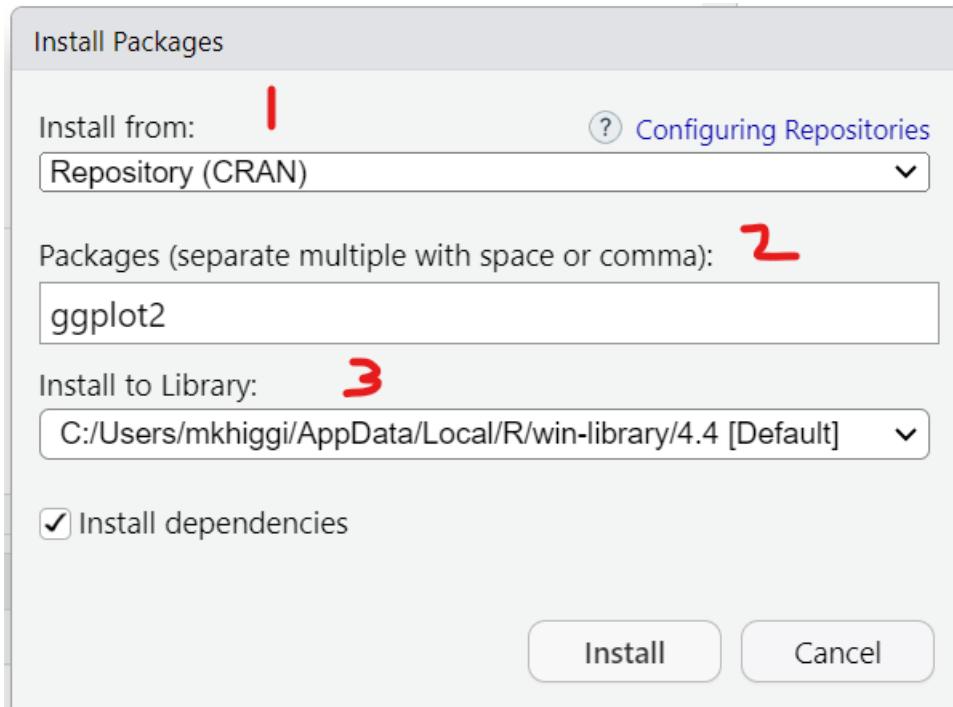
This will then open up a window where you can type in the name of the package you want. As soon as we start typing `ggplot2` the menu begins listing all packages with that partial spelling...





You'll notice that there are 3 parts to the installation:

1. Where you want to get the package from (*i.e., which repository - more on repositories below*).
2. The name of the package. You can actually type more than one package name at a time separated by commas if you want to install several packages at once.
3. The file location on your computer where the new package is installed - your file location may be different than mine. But this is useful to know in case something goes wrong. **I would suggest keeping the default settings.**





Where to get packages - CRAN Repository

Using the “Tools/Install” Packages menu from within RStudio automatically links to [CRAN](#), which is the “The Comprehensive R Archive Network”. You’ve already been here once to download and install the R programming language application.

Here is a screenshot of the [CRAN](#) homepage.



Next click on “Packages” at the left to see the full list of packages currently available. As of right now (01/10/2025 at 5:12pm EST) there are 21,872 packages. This number increases every day as people create, validate and publish their packages on CRAN. You can get a list of all of the packages or if you have no idea what package you need, you can also look at the “Task Views” to see groupings of packages.

The screenshot shows the CRAN homepage with a sidebar on the left containing links for CRAN, About R, Software, Documentation, and Donations. A red arrow points from the 'Task Views' link in the sidebar to the 'CRAN Task Views' section in the main content area. Another red arrow points from the 'Available Packages' link to the 'Table of available packages, sorted by date of publication'. The main content area displays the 'Contributed Packages' section, which includes a summary of 21872 available packages, tables for sorting by publication date and name, and information about CRAN Task Views.



Here is what the list of Packages looks like sorted by name:

The screenshot shows a web browser window with the title "Available CRAN Packages By Name". The URL is "cran.r-project.org/web/packages/available_packages_by_name.html". The page is sorted by package name, showing packages starting with 'A'. Each package name is a link to its detailed description.

Package	Description
A3	Accurate, Adaptable, and Accessible Error Metrics for Predictive Models
AalenJohansen	Conditional Aalen-Johansen Estimation
AATools	Reliability and Scoring Routines for the Approach-Avoidance Task
ABACUS	Apps Based Activities for Communicating and Understanding Statistics
absequence	Coding 'ABA' Patterns for Sequence Data
abbreviate	Readable String Abbreviation
abc	Tools for Approximate Bayesian Computation (ABC)
abc_data	Data Only: Tools for Approximate Bayesian Computation (ABC)
ABC.RAP	Array Based CpG Region Analysis Pipeline
ABCAnalysis	Computed ABC Analysis
abclass	Angle-Based Large-Margin Classifiers
ABCoptim	Implementation of Artificial Bee Colony (ABC) Optimization
ABCp2	Approximate Bayesian Computational Model for Estimating P2
abcrf	Approximate Bayesian Computation via Random Forests
abcrlda	Asymptotically Bias-Corrected Regularized Linear Discriminant Analysis
abctools	Tools for ABC Analyses
abd	The Analysis of Biological Data
abdiv	Alpha and Beta Diversity Measures
abe	Augmented Backward Elimination
aberrance	Detect Aberrant Behavior in Test Data
abess	Fast Best Subset Selection



However, you can also browse Packages by “Task View”:

The screenshot shows a web browser window with the title "CRAN Task Views". The URL in the address bar is "cran.r-project.org/web/views/". The page content describes CRAN task views as providing guidance on relevant packages for specific tasks. It includes code snippets for installing the "ctv" package and using its functions like "install.views" and "update.views". It also provides commands for querying task views from R. A note states that the views are not meant to endorse the "best" packages. Below this, a section titled "Topics" lists various task views with their descriptions:

ActuarialScience	Actuarial Science
Agriculture	Agricultural Science
Bayesian	Bayesian Inference
CausalInference	Causal Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
Databases	Databases with R
DifferentialEquations	Differential Equations



For example, suppose you are interested in survival analysis, here is a screenshot of the [Survival Task View](#).

As you can see each Task View has a person(s) listed who help to maintain these collections. As you scroll through the webpage, you'll see links to packages they recommend along with a description of what the packages do. For example, see the links below to the **survival** and **rms** packages.

CRAN Task View: Survival Analysis

Maintainer: Arthur Allignol, Aurelien Latouche
Contact: arthur.allignol@gmail.com
Version: 2023-09-10
URL: <https://CRAN.R-project.org/view=Survival>
Source: <https://github.com/cran-task-views/Survival/>

Contributions: Suggestions and improvements for this task view are very welcome and can be made through issues or pull requests on GitHub or via e-mail to the maintainer address. For further details see the [Contributing guide](#).

Citation: Arthur Allignol, Aurelien Latouche (2023). CRAN Task View: Survival Analysis. Version 2023-09-10. URL <https://CRAN.R-project.org/view=Survival>.

Installation: The packages from this task view can be installed automatically using the [ctv](#) package. For example, `ctv::install.views("survival", coreOnly = TRUE)` installs all the core packages or `ctv::update.views("survival")` installs all packages that are not yet installed and up-to-date. See the [CRAN Task View Initiative](#) for more details.

Survival analysis, also called event history analysis in social science, or reliability analysis in engineering, deals with time until occurrence of an event of interest. However, this failure time may not be observed within the relevant time period, producing so-called censored observations.

This task view aims at presenting the useful R packages for the analysis of time to event data.

Please let the maintainers know if something is inaccurate or missing, either via e-mail or by submitting an issue or pull request in the GitHub repository linked above.

Standard Survival Analysis

Estimation of the Survival Distribution

- **Kaplan-Meier:** The `survfit` function from the `survival` package computes the Kaplan-Meier estimator for truncated and/or censored data. `rms` (replacement of the `Design` package) proposes a modified version of the `survfit` function. The `prodlim` package implements a fast algorithm and some features not included in `survival`. Various confidence intervals and confidence bands for the Kaplan-Meier estimator are implemented in the `km.ci` package. `plot.Surv` of package `eha` plots the Kaplan-Meier estimator. The `NADA` package includes a function to compute the Kaplan-Meier estimator for left-censored data. `svykm` in `survey` provides a weighted Kaplan-Meier estimator. The `kaplan-meier` function in `spatstat` computes the Kaplan-Meier estimator from histogram data. The `km` function in package `thompson` plots the survival function using a variant of the Kaplan-Meier estimator in a



Where to get packages - Bioconductor Repository

While the list of R packages on CRAN is impressive, if you plan to do analyses of biological data, there is a good chance you will need a package from [Bioconductor.org](https://www.bioconductor.org/).

As of right now (01/10/2025 at 6:45pm EST) there are 2289 packages on Bioconductor. Similar to CRAN, Bioconductor requires each package to meet certain validation criteria and code testing requirements but these criteria are even more stringent on Bioconductor than on CRAN. You'll notice that you can search for packages under the `biocViews` (left side column) or you can sort them alphabetically or search for individual packages in the section on the right side.

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "Bioconductor - BiocViews". The page content is as follows:

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

Home > BiocViews

Packages found under Software:

Rank based on number of downloads: lower numbers are more frequently downloaded.

Show All entries Search table:

Package	Maintainer	Title	Rank
BiocVersion	Bioconductor Package Maintainer	Set the appropriate version of Bioconductor packages	1
BiocGenerics	Hervé Pagès	S4 generic functions used in Bioconductor	2
GenomeInfoDb	Hervé Pagès	Utilities for manipulating chromosome names, including modifying them to follow a particular naming style	3
S4Vectors	Hervé Pagès	Foundation of vector-like and list-like containers in Bioconductor	4

The one disadvantage of R packages from Bioconductor is that you cannot install them directly using the RStudio menu for “Tools/Install” Packages because you cannot “see” the Bioconductor repository from inside RStudio. Instead you have to install Bioconductor packages using R code.



For example, here is what you need to do to install the `phyloseq` package which “... provides a set of classes and tools to facilitate the import, storage, analysis, and graphical display of microbiome census data”.

To install `phyloseq` you need to (*see the black box of code in the screenshot shown below*):

1. Install `BiocManager` from CRAN - this package you can install from the RStudio menu for “Tools/Install Packages” - or you can run the code shown below for `install.packages()`.

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

2. Then go to the Console or open an R script and run:

```
BiocManager::install("phyloseq")
```

The screenshot shows a web browser window with the Bioconductor website loaded. The URL in the address bar is bioconductor.org/packages/release/bioc/html/phyloseq.html. The page content includes the package's author (Paul J. McMurdie), maintainer (Paul J. McMurdie), and citation information. Below this, there is a section titled "Installation" with the following R code:

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("phyloseq")
```

At the bottom of the page, there is a note about older versions of R and a "Documentation" section with the command `browseVignettes("phyloseq")`.



Where to get packages - Github, friends, teammates, ...

In addition to the CRAN and Bioconductor repositories, you can also get packages from Github (and other cloud-based repositories), friends, teammates or write your own.

To get an idea of how many packages may be currently on [Github](#), we can “search” for “R package” at <https://github.com/search?q=R+package&type=repositories> and as you can see this is well over 118,000+ packages.

The screenshot shows a browser window with multiple tabs open, including "melindahiggins2", "1.3.1: Introductio...", "CRAN: Package", "Bioconductor", "Repository search", "melindahiggins2", and "Module 02: Get". The main content area displays the GitHub search results for "R package".
The search bar at the top contains "R package".
The results summary is "118k results (294 ms)".
The results list includes:

- hadley/r-pkgs**: Building R packages. Updated 16 hours ago. Stars: 889.
- qinwf/awesome-R**: A curated list of awesome R packages, frameworks and software. Updated on Dec 5, 2024. Stars: 6.1k.
- r-lib/http**: http: a friendly http package for R. Updated on Oct 11, 2024. Stars: 985.

A sidebar on the right provides information about the R language:

- R**: R is a free programming language and software environment for statistical computing and graphics.
- [r-project.org](#)
- [Wikipedia](#)
- [View topic](#)



While you can find packages on Github that have not (yet) been published on CRAN or Bioconductor, the developers of packages currently on CRAN and Bioconductor also often publish their development version (*think of these as in “beta” and still undergoing testing*) on Github.

For example, the current published version of the data wrangling R package [dplyr](#) on CRAN was last updated on 11/17/2023 (*over a year ago*).

dplyr: A Grammar of Data Manipulation

A fast, consistent tool for working with data frame like objects, both in memory and out of memory.

Version: 1.1.4
Depends: R (≥ 3.5.0)
Imports: [cli](#) (≥ 3.4.0), [generics](#), [glue](#) (≥ 1.3.2), [lifecycle](#) (≥ 1.0.3), [magrittr](#) (≥ 1.5), methods, [pillar](#) (≥ 1.9.0), [R6](#), [rlang](#) (≥ 1.1.0), [tibble](#) (≥ 3.2.0), [tidyselect](#) (≥ 1.2.0), utils, [vctrs](#) (≥ 0.6.4)
Suggests: [bench](#), [broom](#), [callr](#), [covr](#), [DBI](#), [dplyr](#) (≥ 2.2.1), [ggplot2](#), [knitr](#), [Lahman](#), [lobstr](#), [microbenchmark](#), [nycflights13](#), [purrr](#), [markdown](#), [RMySQL](#), [RPostgreSQL](#), [RSQlite](#), [stringi](#) (≥ 1.7.6), [testthat](#) (≥ 3.1.5), [tidyv](#) (≥ 1.3.0), [withr](#)
Published: 2023-11-17
DOI: [10.32614/CRAN/package.dplyr](#)
Author: Hadley Wickham [aut, cre], Romain François [aut], Lionel Henry [aut], Kirill Müller [aut], Davis Vaughan [aut], Posit Software, PBC [cph, fnd]
Maintainer: Hadley Wickham <hadley at posit.co>
BugReports: <https://github.com/tidyverse/dplyr/issues>
License: [MIT + file LICENSE](#)
URL: <https://dplyr.tidyverse.org>, <https://github.com/tidyverse/dplyr>
NeedsCompilation: yes
Materials: [README NEWS](#)
In views: [Databases](#), [ModelDeployment](#)
CRAN checks: [dplyr results](#)

Documentation:

Reference manual: [dplyr.pdf](#)
Vignettes: [dplyr <-> base R](#)
[Column-wise operations](#)



However, the development version of [dplyr on Github](#) was last updated 5 months ago in August 2024. So, there is probably a new version of [dplyr](#) coming soon for CRAN.

The screenshot shows the GitHub repository page for `dplyr`. The repository is part of the `tidyverse` organization. It has 38 branches and 58 tags. The main branch is the latest. Recent commits include moves to `tidyverse`, updates to `GHA workflows`, and changes to `compilation-database infrastructure`. The repository has 7,781 commits, 243 watchers, 2.1k forks, and 4.8k stars. The `About` section describes `dplyr` as a grammar of data manipulation and links to `dplyr.tidyverse.org`. It also lists repository statistics such as 4.8k stars, 243 watching, and 2.1k forks. The `Releases` section shows the latest release is 1.1.4.

While the developers haven't published this "Github" version of `dplyr` yet on CRAN, if you want to test out new `dplyr` functions and updates under development, you can go to the R Console or write an R script to install the development version using these commands (see below) which is explained on the [dplyr on Github](#) website.

```
# install.packages("pak")
pak::pak("tidyverse/dplyr")
```



Finding and vetting R packages

So, as you have seen there are numerous ways to find R packages and there are hundreds of thousands of them out there. Your company or team may also have their own custom R package tailored for your specific research areas and data analysis workflows.

Finding R packages is similar to finding new questionnaires, surveys or instruments for your research. For example, if you want to measure someone's depression levels, you should use a validated instrument like the [Center for Epidemiological Studies-Depression \(CESD\)](#) or the [Beck Depression Index \(BDI\)](#). These measurement instruments have both been well published and are well established for depression research.

Finding R packages is similar - do your research! Make sure that the R package has been published and is well established to do the analysis you want. In terms of reliability, getting packages from CRAN or Bioconductor are the best followed by Github or other individuals. The best suggestion is look to see which R packages are being used by other people in your field.

No oversight company or agency

While it may seem worrisome that there is no governing company or organization that verifies and validates and certifies all R packages, the good news is that the R community is a vast Global community. The development of R is not controlled by a limited number of people hired within a single company - instead there are literally millions of R programmers across the Globe testing and providing feedback on a 24/7 basis. If there is a problem with a package or function, there will be people posting about these issues - see [Additional Resources](#).

This is the power of Open Source computing!!



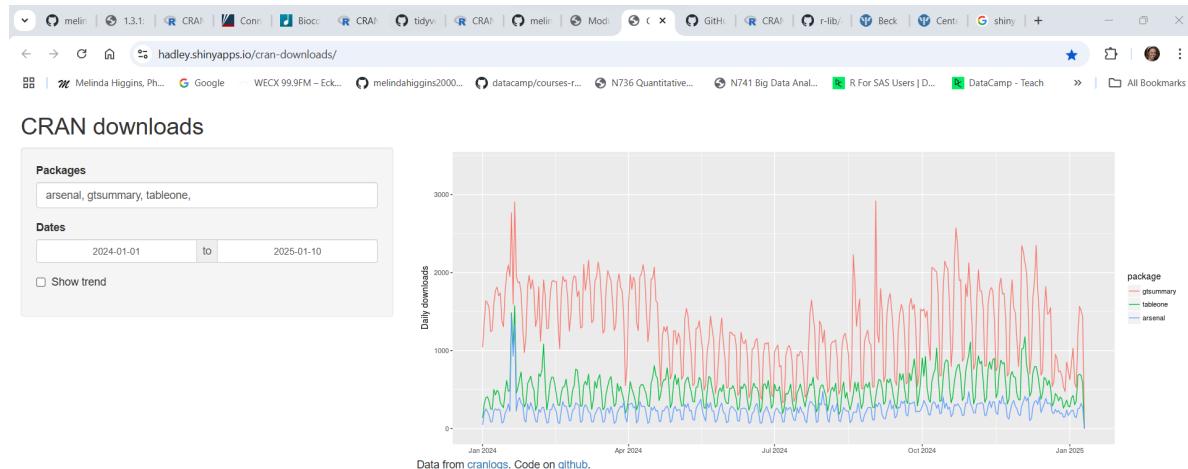
Popularity of R Packages

To get an idea of how long a package has been in use and if it is still being actively supported and how it relates to other similar packages, check out this interactive Shiny app website for package downloads from CRAN <https://hadley.shinyapps.io/cran-downloads/>. Type in the packages you want (separated by commas) to compare and put in the date range of interest.

Here is an example comparing the `arsenal`, `gtsummary`, and `tableone` packages all of which are useful for making tables of summary statistics (aka, “Table 1”) - showing the number of downloads since the beginning of Jan 1, 2024.

As you can see the most downloaded of these 3 packages is `gtsummary` followed by `tableone` and then `arsenal` having the fewest downloads. This does NOT necessarily imply quality, but it does give you some insight into the popularity of these packages. I actually prefer the `arsenal` table package but `tableone` has been around longer and `gtsummary` is written by members of the RStudio/Posit development community and is more well known and popular. All 3 of these packages can be found in use in current research literature.

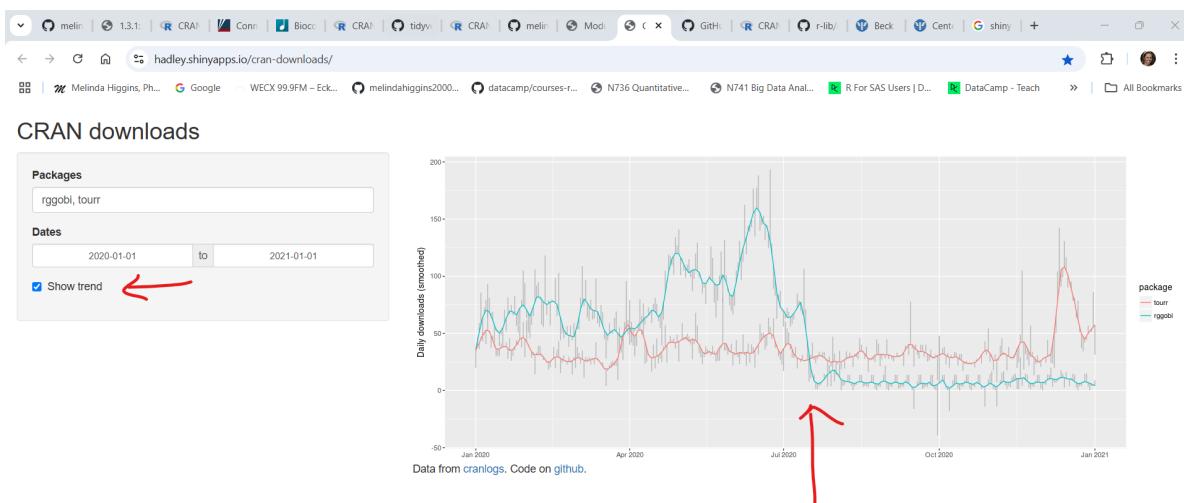
You will see examples of all 3 of these table-making packages in [Module lesson 1.3.2](#)





Here is an example of two specific packages I like. The `rggobi` package which was great for visualizing multiple dimensions of data simultaneously but which is no longer supported. But there is now a newer `tourrr` package which was written by the same developers to replace the `rggobi` package. You can see that in the middle of 2020, the number of downloads for `rggobi` dropped almost to 0 and the `tourrr` package downloads started to rise - this is about when `rggobi` was archived on CRAN and they switched over to maintaining the newer `tourrr` package.

- [rggobi on CRAN](#) moved to archived status in July 2020, but
- [tourrr on CRAN](#) was last updated in April 2024.



In summary:

- do your homework,
- check to see when the package was last updated,
- research who maintains it and
- review how good their documentation is for the package and what it does, and
- see if the package has been used by others in your research area.



Load the new R package into your R session

After you've decided what package you want and have installed it onto your computer, you **must load it into memory for EVERY new R session** for which you want those functions available.

Packages - install once, (re)-load every R session

Unless you upgrade R or change computers, you only need to install a given R package once. But you do need to (re)-load the package into your current R session every time you (re)-start R (or RStudio).

For example, suppose I want to make a plot using the `ggplot2` package. Before I can use the `ggplot()` function, I have to load that package into my computing session.

Here is my current R session status **BEFORE** I load the `ggplot2` package.

```
# show current sessionInfo  
sessionInfo()
```

```
R version 4.4.2 (2024-10-31 ucrt)  
Platform: x86_64-w64-mingw32/x64  
Running under: Windows 11 x64 (build 22000)
```

```
Matrix products: default
```

```
locale:  
[1] LC_COLLATE=English_United States.utf8  
[2] LC_CTYPE=English_United States.utf8  
[3] LC_MONETARY=English_United States.utf8  
[4] LC_NUMERIC=C  
[5] LC_TIME=English_United States.utf8
```

```
time zone: America/New_York  
tzcode source: internal
```

```
attached base packages:  
[1] stats      graphics   grDevices utils      datasets  methods    base
```

```
loaded via a namespace (and not attached):  
[1] compiler_4.4.2    fastmap_1.1.1     cli_3.6.3        tools_4.4.2  
[5] htmltools_0.5.8.1 rstudioapi_0.15.0  yaml_2.3.8       rmarkdown_2.26
```



```
[9] knitr_1.49      jsonlite_1.8.8    xfun_0.49       digest_0.6.35
[13] rlang_1.1.4     evaluate_0.23
```

Since I have not yet loaded the `ggplot2` package into the session, I will get an error.

```
# I have not yet loaded ggplot2 into the session
# try the ggplot() function with the
# built-in pressure dataset to see error
ggplot(pressure, aes(temperature, pressure)) +
  geom_point()
```

```
Error in ggplot(pressure, aes(temperature, pressure)): could not find function "ggplot"
```

The code above generates an error since these functions are not yet available in our session.



To fix this error, we need to use the `library()` function to load the `ggplot2` functions into current working memory.

```
# load ggplot2 package
library(ggplot2)

# look at sessionInfo again
sessionInfo()
```

```
R version 4.4.2 (2024-10-31 ucrt)
Platform: x86_64-w64-mingw32/x64
Running under: Windows 11 x64 (build 22000)
```

```
Matrix products: default
```

```
locale:
```

```
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8
```

```
time zone: America/New_York
tzcode source: internal
```

```
attached base packages:
```

```
[1] stats      graphics   grDevices  utils      datasets  methods    base
```

```
other attached packages:
```

```
[1] ggplot2_3.5.1
```

```
loaded via a namespace (and not attached):
```

```
[1] vctrs_0.6.5        cli_3.6.3        knitr_1.49       rlang_1.1.4
[5] xfun_0.49          generics_0.1.3   jsonlite_1.8.8  glue_1.8.0
[9] colorspace_2.1-0   htmltools_0.5.8.1 scales_1.3.0     fansi_1.0.6
[13] rmarkdown_2.26     grid_4.4.2       evaluate_0.23   munsell_0.5.0
[17] tibble_3.2.1       fastmap_1.1.1    yaml_2.3.8      lifecycle_1.0.4
[21] compiler_4.4.2    dplyr_1.1.4     pkgconfig_2.0.3  rstudioapi_0.15.0
[25] digest_0.6.35     R6_2.5.1        tidyselect_1.2.1 utf8_1.2.4
[29] pillar_1.9.0      magrittr_2.0.3   withr_3.0.2     tools_4.4.2
[33] gtable_0.3.6
```

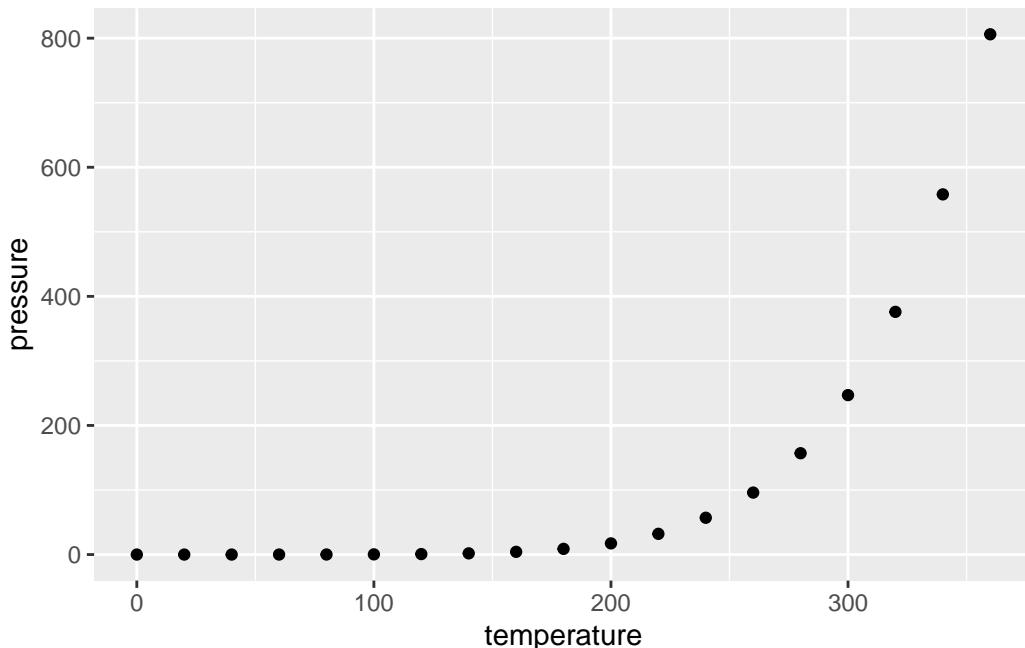


Notice that under `other attached packages` we can now see `ggplot2_3.5.1` indicating that yes `ggplot2` is installed and in memory and that version 3.5.1 is the version I am currently using.



Let's try the plot again with the `ggplot()` function from the `ggplot2` package.

```
# try the plot again  
ggplot(pressure, aes(temperature, pressure)) +  
  geom_point()
```



Reload packages for every new R session

Everything you close out your R/RStudio computing session (or restart your R session) you will need to load all of your package again. I know this seems like a HUGE pain, but there is a rationale for this.

1. You may not need the same packages for every new computing session - so R begins with the minimum loaded to save computing memory.
2. The GOOD NEWS is you do not have to re-install the packages - these are already saved on your computer. You only have to re-load them into memory using the `library()` function.
3. This workflow forces you to document (in your code) which packages you need for your computing sessions and why you are using them.

BUT ... If you do have a core set of packages that you would like to make sure get loaded into memory every time you start R/RStudio, see these helpful posts on customizing your startup:



- <https://www.datacamp.com/doc/r/customizing>
 - <https://www.r-bloggers.com/2014/09/fun-with-rprofile-and-customizing-r-startup/>
-

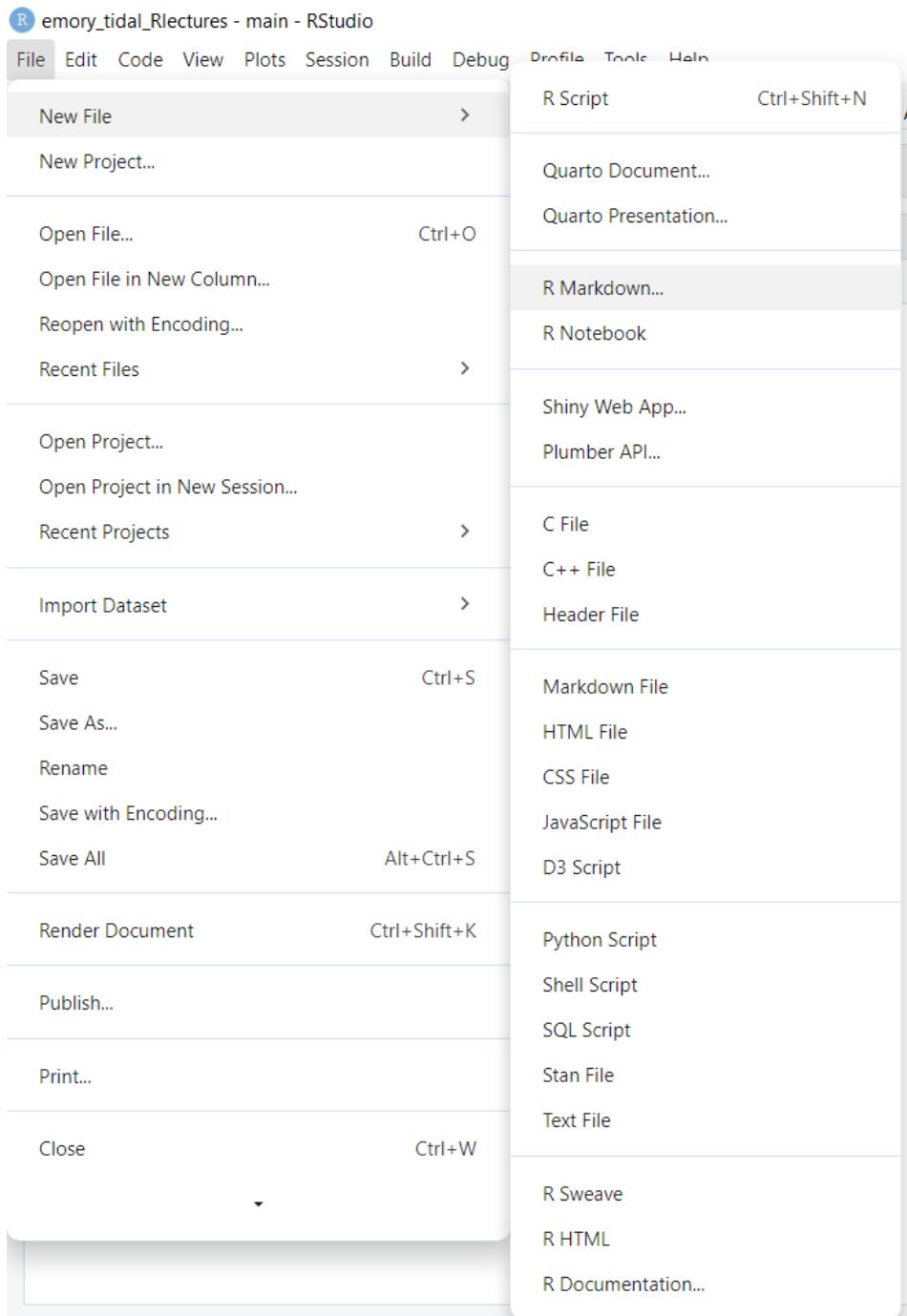


5. Create your first R Markdown report and produce output files in different formats (HTML, PDF, or DOCX)

Create a new Rmarkdown File

We will do more in the later lesson [1.3.6: Putting reproducible research principles into practice](#), but let's take a look at an Rmarkdown file and how we can use it to create a report that combines together `data + code + documentation` to produce a seamless report.

Go to the RStudio menu and click “File/New File/R Markdown”:





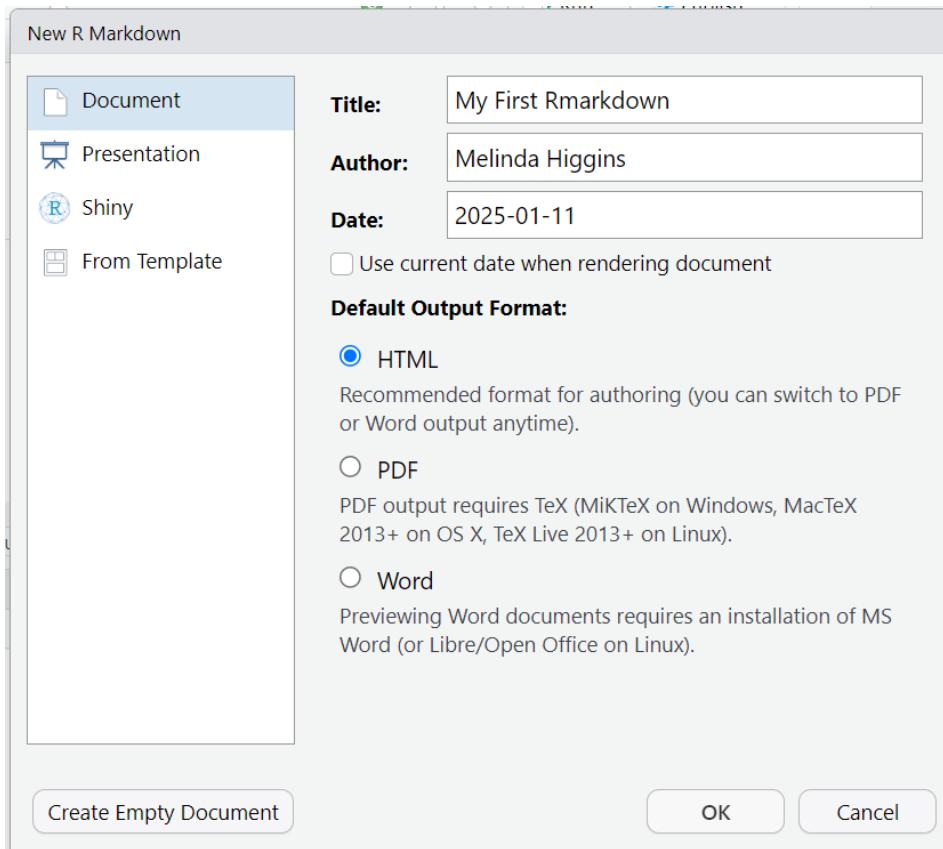
Type in a title, your name, the date and choose the format you'd like to create. For your first document I encourage you to try HTML. But you can create WORD (DOC) documents and even PDFs. In addition to documents, you can also create slide deck presentations, Shiny apps and other custom products like R packages, websites, books, dashboards and many more.

 Rmarkdown ideas and inspiration

- [Rmarkdown Gallery](#)
- [Rmarkdown Formats](#)
- [Rmarkdown Cookbook](#)

To get started, use the built-in template:

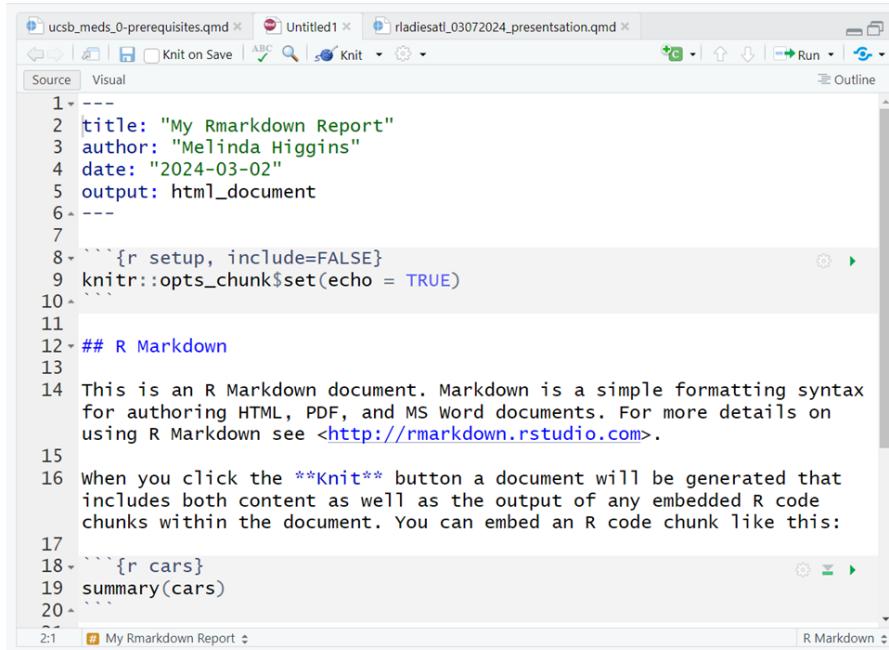
- Type in a title
- Type in your name as author
- Choose and output document format
 - HTML is always a good place to start - **only need a browser to read the output *.html file.**
 - DOC usually works OK - **but you need MS Word or Open Office installed on your computer.**
 - PDF **NOTE: You need a TEX compiler on your computer** - Learn about installing the `tinytex` <https://yihui.org/tinytex/> R package to create PDFs.





Rmarkdown sections

Here is the **Example RMarkdown Template** provided by RStudio to help you get started with your first Rmarkdown document.



The screenshot shows the RStudio interface with the "Source" tab selected. The code editor displays an R Markdown template. The code includes YAML front matter, setup chunks, and a summary code chunk. The RStudio toolbar at the top shows multiple tabs and various icons for file operations, search, and knit.

```
1 ---  
2 title: "My Rmarkdown Report"  
3 author: "Melinda Higgins"  
4 date: "2024-03-02"  
5 output: html_document  
6 ---  
7  
8 ````{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10 ````  
11  
12 ## R Markdown  
13  
14 This is an R Markdown document. Markdown is a simple formatting syntax  
for authoring HTML, PDF, and MS Word documents. For more details on  
using R Markdown see <http://rmarkdown.rstudio.com>.  
15  
16 When you click the **Knit** button a document will be generated that  
includes both content as well as the output of any embedded R code  
chunks within the document. You can embed an R code chunk like this:  
17  
18 ````{r cars}  
19 summary(cars)  
20 ````
```



This document consists of the following 3 key sections:

1. YAML (yet another markup language) - this is essentially the metadata for your document and defines elements like the title, author, date and type of output document to be created (HTML in this example).

YAML

The screenshot shows the RStudio interface with three tabs open: 'ucsb_meds_0-prerequisites.qmd', 'Untitled1', and 'rladiesatl_03072024_presentation.qmd'. The 'Source' tab is selected for the main document. A large curly brace on the left indicates the scope of the YAML configuration. The code content is as follows:

```
1 ---  
2 title: "My Rmarkdown Report"  
3 author: "Melinda Higgins"  
4 date: "2024-03-02"  
5 output: html_document  
6 ---  
7  
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10  
11  
12 ## R Markdown  
13  
14 This is an R Markdown document. Markdown is a simple formatting syntax  
for authoring HTML, PDF, and MS Word documents. For more details on  
using R Markdown see <http://rmarkdown.rstudio.com>.  
15  
16 When you click the **Knit** button a document will be generated that  
includes both content as well as the output of any embedded R code  
chunks within the document. You can embed an R code chunk like this:  
17  
18 ```{r cars}  
19 summary(cars)  
20  
21
```



2. R code blocks - the goal is to “interweave” code and documentation so these 2 elements live together. That way the analysis output and any associated tables or figures are updated automatically without having to cut-and-paste from other applications into your document - which is time consuming and prone to human errors.

Notice that the code block starts and ends with 3 backticks ` ` ` and includes the {r} Rlanguage designation inside the curly braces.

i Rmarkdown

Rmarkdown can be used for many different programming languages including python, sas, and more, see [rmarkdown - language-engines](#).

R Code
Chunks

The screenshot shows an RStudio interface with three tabs at the top: "ucsb_meds_0-prerequisites.qmd", "Untitled1", and "rladiesatl_03072024_presentation.qmd". The "Source" tab is selected. The code editor contains the following R Markdown code:

```
1 ---  
2 title: "My Rmarkdown Report"  
3 author: "Melinda Higgins"  
4 date: "2024-03-02"  
5 output: html_document  
6 ---  
7  
8 ````{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10 ...  
11  
12 ## R Markdown  
13  
14 This is an R Markdown document. Markdown is a simple formatting syntax  
for authoring HTML, PDF, and MS Word documents. For more details on  
using R Markdown see <http://rmarkdown.rstudio.com>.  
15  
16 When you click the **Knit** button a document will be generated that  
includes both content as well as the output of any embedded R code  
chunks within the document. You can embed an R code chunk like this:  
17  
18 ````{r cars}  
19 summary(cars)  
20 ...  
21
```

A red curly brace on the left side of the code editor highlights the two code chunks, labeled "R Code Chunks".



- Along with the R code blocks, we can also create our document with “marked up (or marked down)” text. **Rmarkdown** is a version of “**markdown**” which is a simplified set of tags that tell the computer how you want a piece of text formatted.

For example putting 2 asterisks ** before and after a word will make it **bold**, putting one _ underscore before and after a word will make the word *italics*; one or more hashtags # indicate a header at certain levels, e.g. 2 hashtags ## indicate a header level 2.

💡 Rmarkdown Tutorial

I encourage you to go through the step by step tutorial at <https://rmarkdown.rstudio.com/lesson-1.html>.

Marked
up text

The screenshot shows the RStudio interface with the Source tab selected. The code editor displays an R Markdown document. A green callout box highlights the text starting from line 12, which is enclosed in a green rounded rectangle. The text within the box is:

```
12## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax
15 for authoring HTML, PDF, and MS Word documents. For more details on
16 using R Markdown see <http://rmarkdown.rstudio.com>.
17
18```{r cars}
19 summary(cars)
20```
21
```

A green curly brace on the left side of the image groups the text from line 12 to the end of the highlighted section.



Here are all 3 sections outlined.

```
1+ ---  
2+ title: "My Rmarkdown Report"  
3+ author: "Melinda Higgins"  
4+ date: "2024-03-02"  
5+ output: html_document  
6+ ---  
7+  
8+ ```{r setup, include=FALSE}  
9+ knitr::opts_chunk$set(echo = TRUE)  
10+  
11+  
12+ ## R Markdown  
13+  
14+ This is an R Markdown document. Markdown is a simple formatting syntax  
15+ for authoring HTML, PDF, and MS Word documents. For more details on  
16+ using R Markdown see <http://rmarkdown.rstudio.com>.  
17+  
18+ ```{r cars}  
19+ summary(cars)  
20+
```



At the top of the page you'll notice a little blue button that says "knit" - this will "knit" (or combine) the output from the R code chunks and format the text as "marked up" and produce this HTML file (*which will open in a browser window*):

The screenshot shows a web browser window with the title "My First Rmarkdown". The page content includes:

- A header section with the title "My First Rmarkdown", author "Melinda Higgins", and date "2025-01-11".
- A section titled "R Markdown" with a note explaining what R Markdown is and how to use it.
- An R code chunk for "summary(cars)" which outputs the following summary statistics:

```
##      speed      dist
## Min.   :4.0   Min.   : 2.00
## 1st Qu.:12.0  1st Qu.:26.00
## Median :15.0  Median :36.00
## Mean   :15.4  Mean   :42.98
## 3rd Qu.:19.0  3rd Qu.:56.00
## Max.   :25.0  Max.   :120.00
```
- A section titled "Including Plots" with a note about embedding plots.
- A scatter plot with "pressure" on the y-axis (ranging from 0 to 800) and "dist" on the x-axis (ranging from 0 to 100). The plot shows a positive correlation between pressure and distance.



References

- R Core Team. 2024. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.
- Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, Dewey Dunnington, and Teun van den Brand. 2024. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. <https://ggplot2.tidyverse.org>.

Other Helpful Resources

Other Helpful Resources