SCCDS Professional Development Handbook

Isaac Quintanilla Salinas

Table of contents

Teaching Data Science

> **Warning**
>
> This is a current work in progress. There will be several revisions before the final product is produced.

SCCDS

The Southern California Consortium For Data Science is a collaboration of 8 institutions from the CCC, CSU, and UC systems.

Acknowledgements

Parts of this webbook have been adapted from the Statistical Computing book from Isaac Quintanilla Salinas.

Preface

Part I

R Programming

R is an open-source programming language used to conduct statistical analysis.

Installing R

You can freely download and install R here.

Installing on Windows

Installing on Mac

Installing RStudio

RStudio is an Integrated Development Environment (IDE) used for data science. It contains several tools needed to extend your programming and project management skills.

You can download and install the open-source (free) version of RStudio here.

Installing R Packages

R Packages extends the functionality from the base functions in R. R packages contain extra functions to conduct uncommon statistical models.

As of right now, the tidyverse is a set of comprehensive packages to prepare and analyze data. To install tidyverse, use the following line in the console:

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

Posit Cloud

Posit Cloud is the online platform for RStudio. The free account provides users with limited computing resources that allow them to learn R and RStudio. One can access Posit Cloud here.

Resources

Website	Description
R 4 Data Science Book	A highly recommended book to learn more about R for data science with a tidyverse implementation.
Advanced R	A book that provides an in depth tutorial of using more advanced R concepts such as control flow concepts and functional programming.
Efficient R	A book that provides tips to write more performative R code such as vectorization or compiling code.
R 4 Data Science Community	This is an online learning community of R users that provides events and data for individuals to learn about data science and R.
Deep R	A book that provides advance programming concepts in terms of R.
Big Book of R	A website containing a list of resources for individuals to learn R.

1 RStudio

RStudio is an integrated development environment (IDE) that allows users to write code in a script and execute code to the R console in one application. Furthermore, RStudio has rich set of features that make data science projects easier to execute.

While RStudio has originally been used to program in R, it has been extended to program in python with the use of the reticulate package.

1.1 Installing

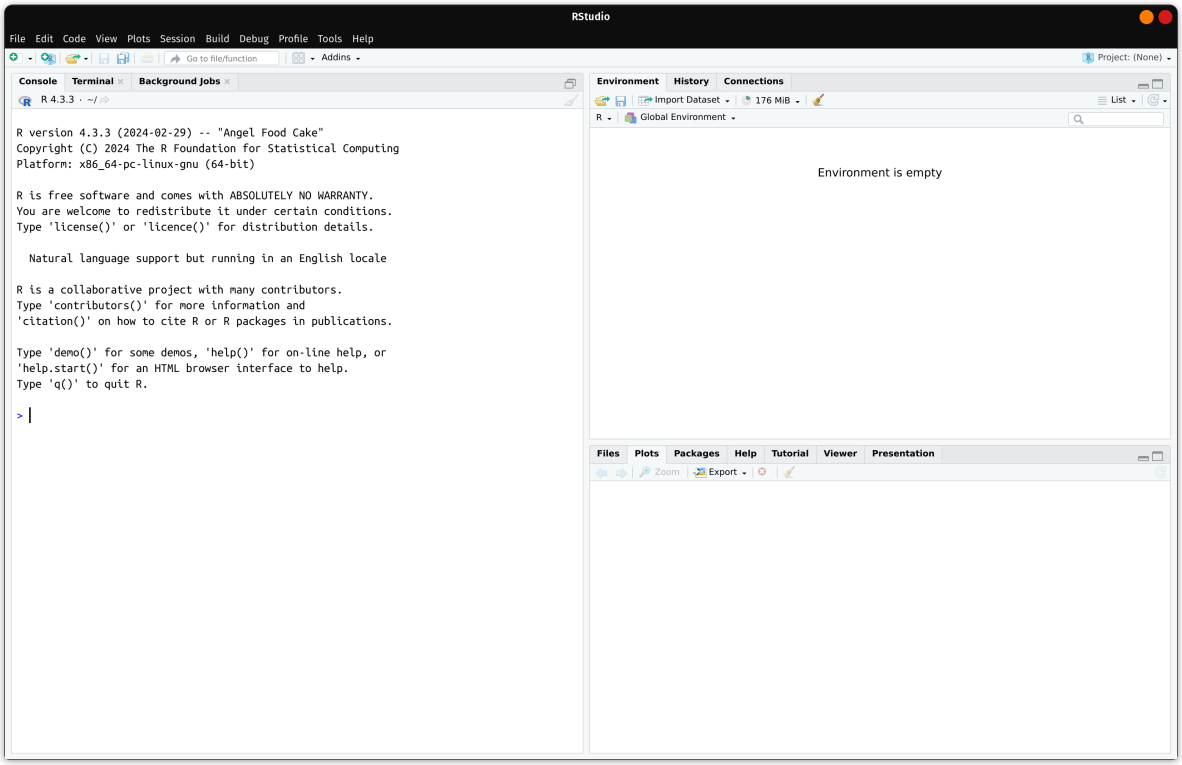
You can download and install the open-source (free) version of RStudio here.

1.1.1 Installing on Windows

1.1.2 Installing on Mac

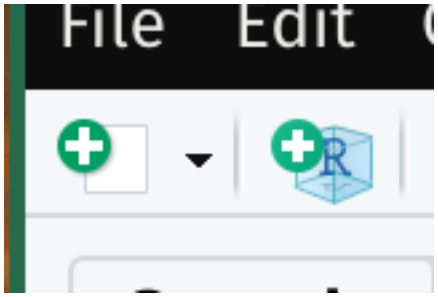
1.2 Start-up

On start-up, RStudio will look like very similar to the image below:

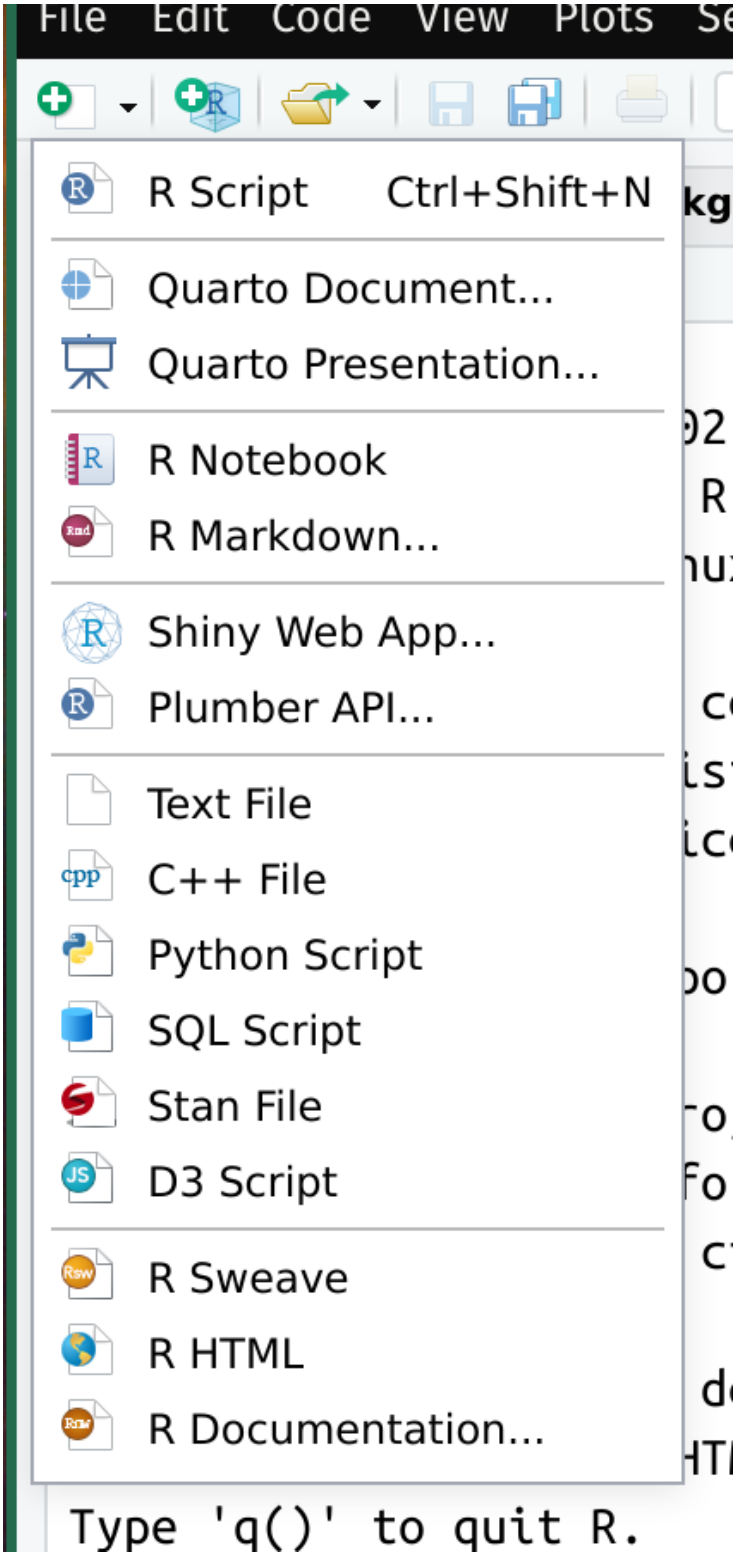


You can see that there are 3 parts in RStudio, these are known as panes.

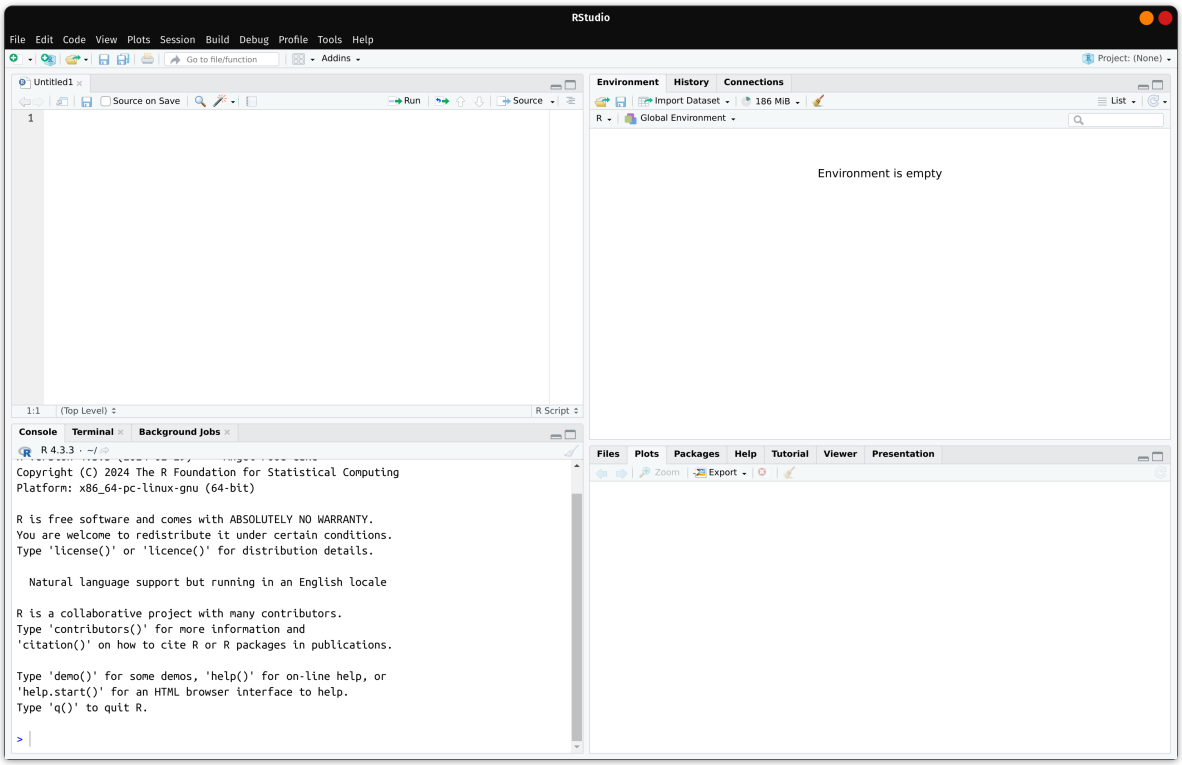
Additionally, we can add a fourth pane to RStudio for writing code in a text file. Choosing the white plus sign with a green border followed by a white document on the upper-right hand side:



This will open up a menu of text files that a user can choose to code in:


File Edit Code View Plots
R Script Ctrl+Shift+N
Quarto Document...
Quarto Presentation...
R Notebook
R Markdown...
Shiny Web App...
Plumber API...
Text File
C++ File
Python Script
SQL Script
Stan File
D3 Script
R Sweave
R HTML
R Documentation...
Type 'q()' to quit R.

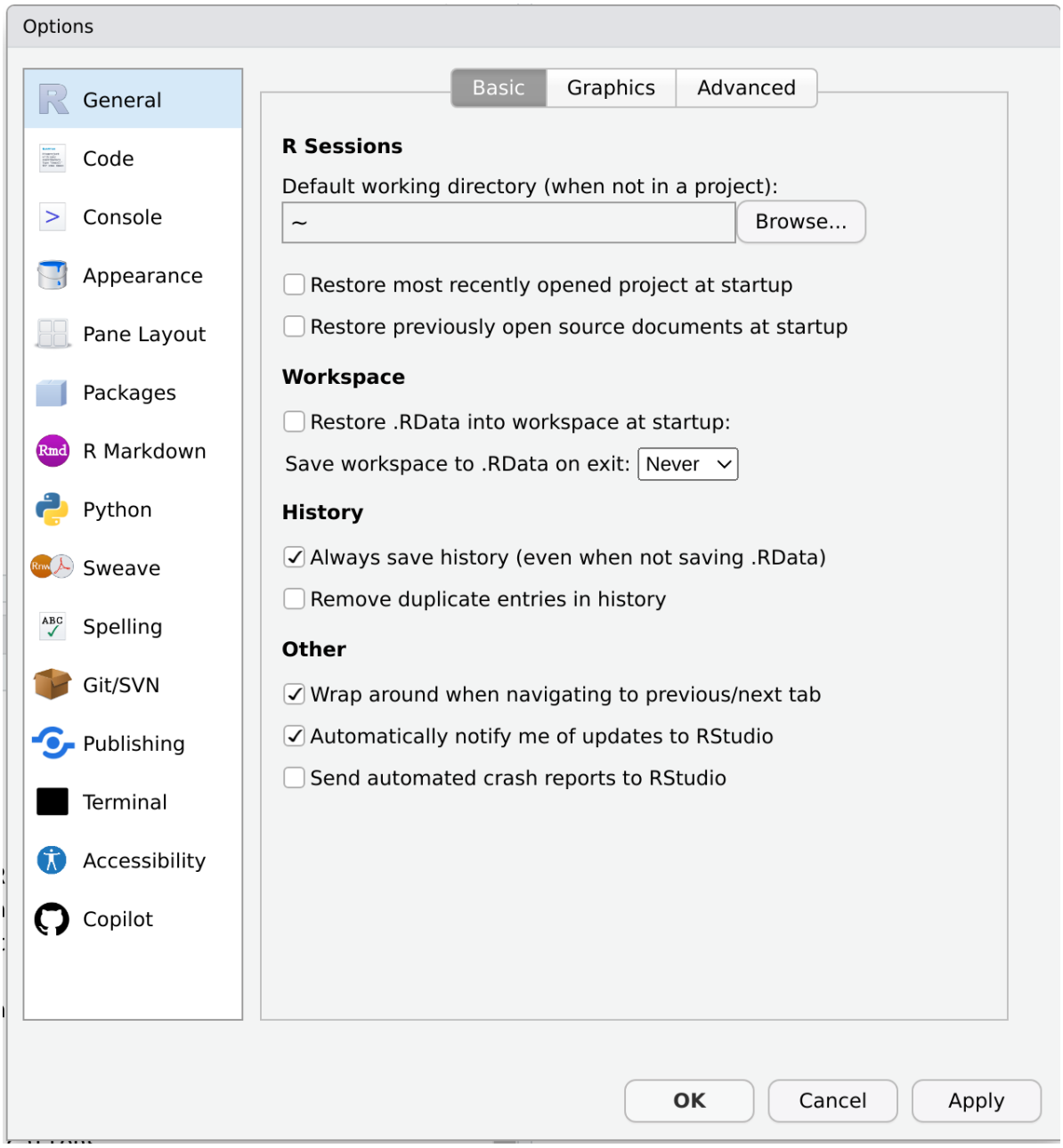
The “R Script” Button will open a standard R text file with the extension as “.R”. This is the text file that most R programmers used to save and execute code. This will make RStudio to look like this:



Notice a new pane is created on the top-left that allows you to write R code in a script. This script is also connected to the R console below which will allow you to send lines of code from the script to the console to be executed (also known as REPL).

1.3 Global Options

In this section, here are some recommended “Global Options” for users to set in RStudio. To begin, click on *Tools* *Global Options* from the top-menu. The following window should open:

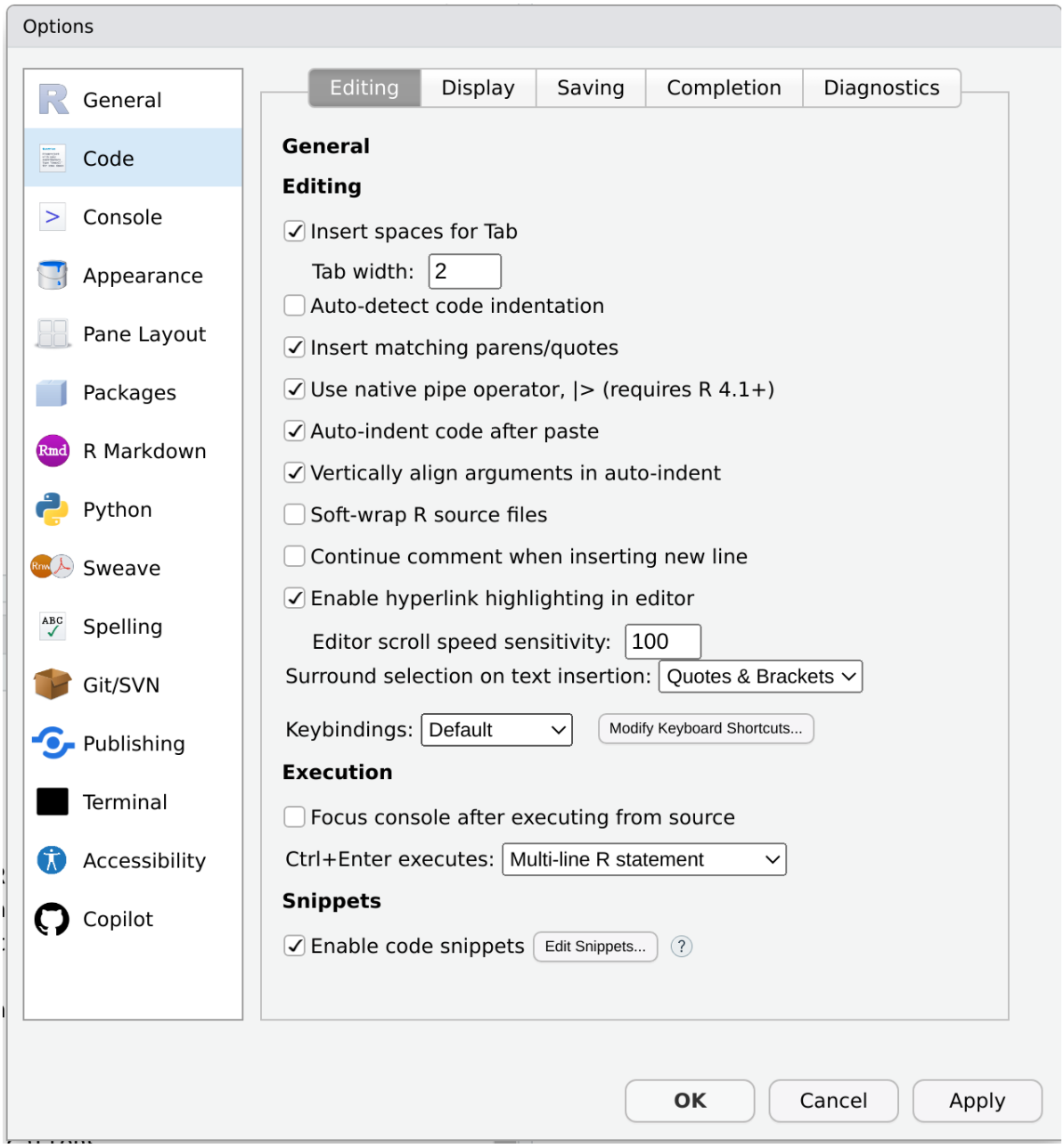


The window allows you to make several changes in RStudio that will make your experience better. Here is a list of items that are recommended for users to change:

1. *R General*
    1. Make sure "Restore .RData into workspace at startup:" is unchecked (**Highly**

Recommended[1])

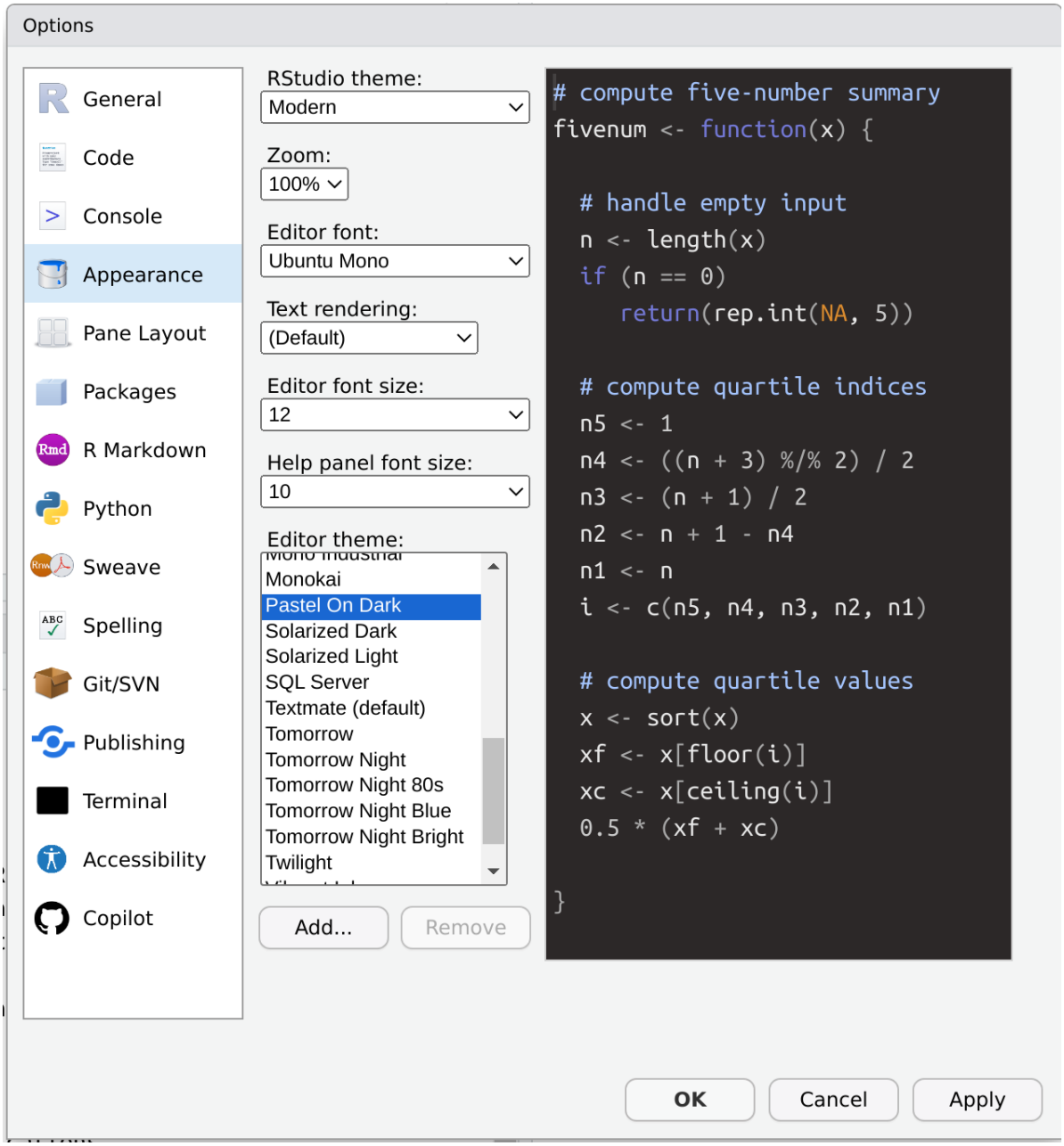
2. Set "Save workspce to .RData on exit:" to "Never" (**Highly** Recommended)



[1]This will ensure that your environment is always recreated from the code you write and not from anything else. It increases reproducibility.

2. *Code*

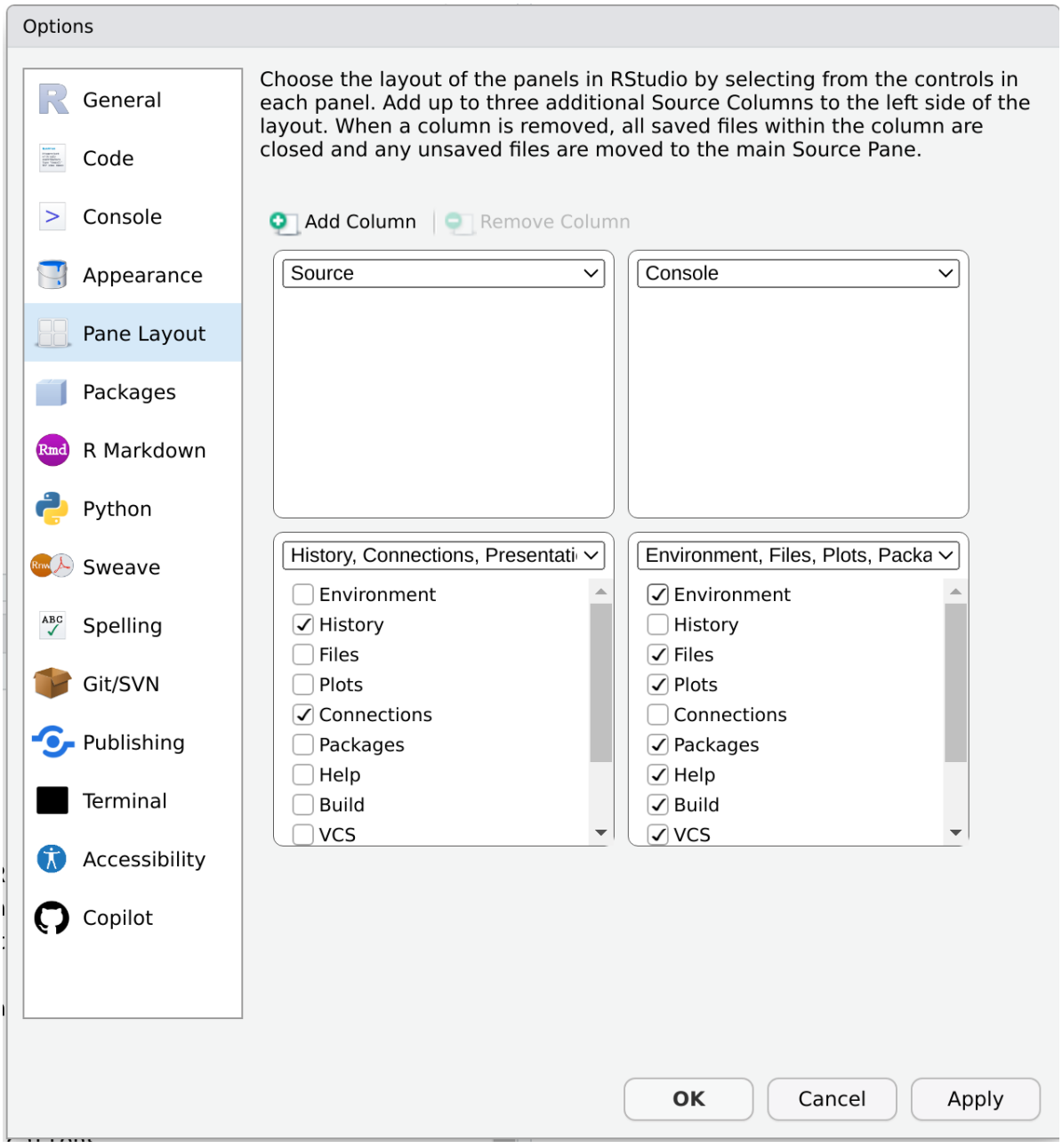
    1. "Use native pipe operator `|>`" is recommended[2] (Optional)



[2]The native pipe does not require to have any packages installed. Additionally, it executes code slightly faster than `%>%`.

3. *Appearance*

    1. In the "Editor theme:" box, choose a setting that you will prefer to work in (Optional)

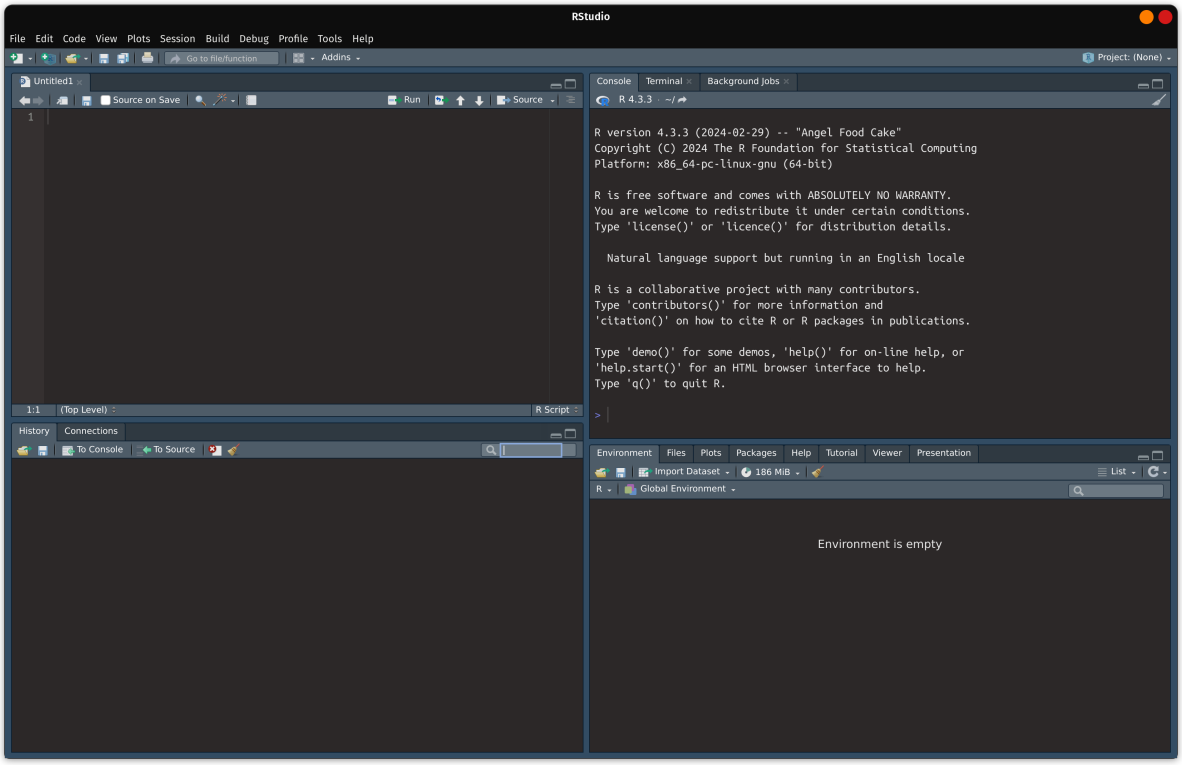


4. *Pane Layout* (Optional)

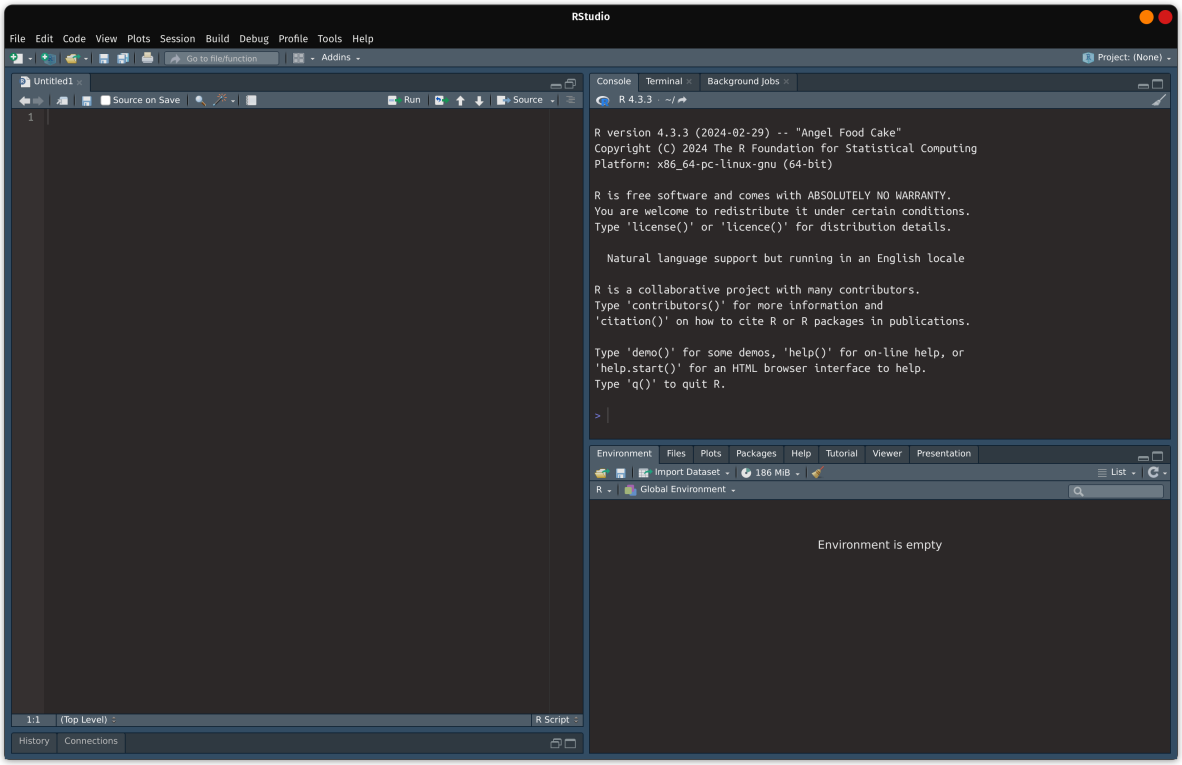
1. Change the pane layout to have the “Console” on the top-right corner
2. Add all components (checkmark) to the lower-right corner except for “History” and “Connections”

This will allow for you to expand the “Source” (script) to be expanded for the entire left hand side. It will allow you to view more code at one time.

RStudio will look more like this:

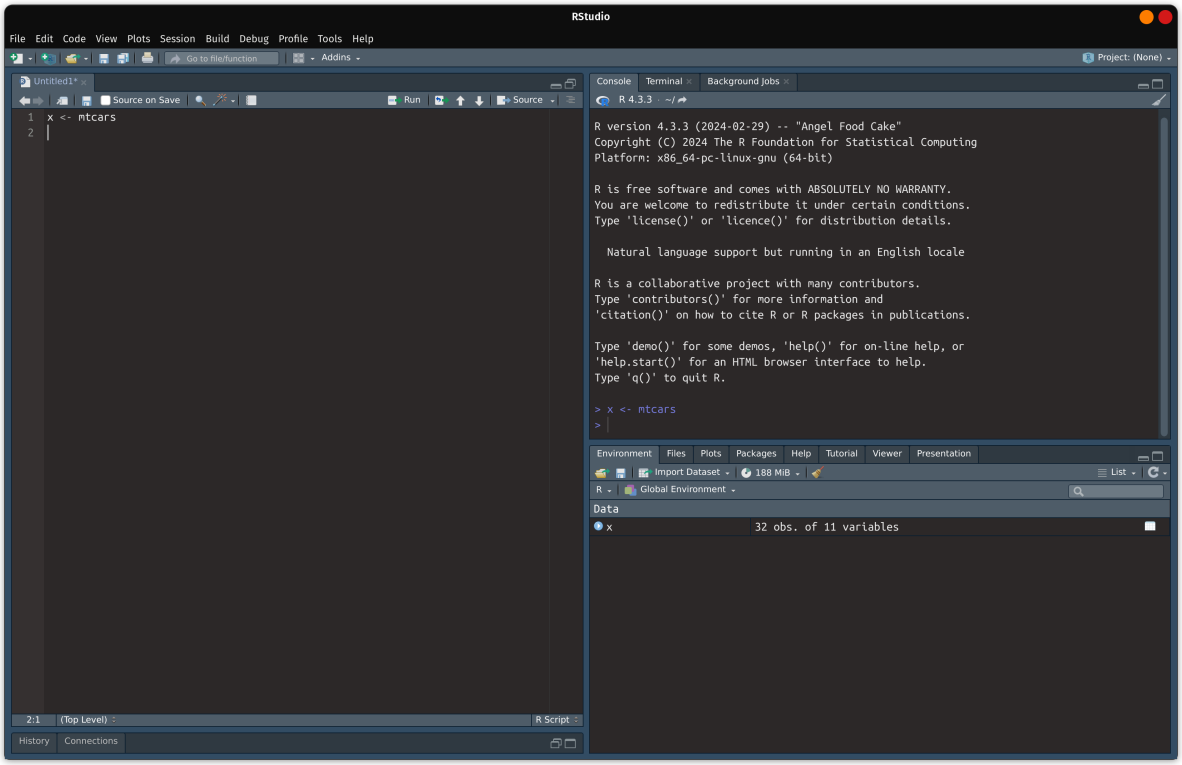


With the expanded script:

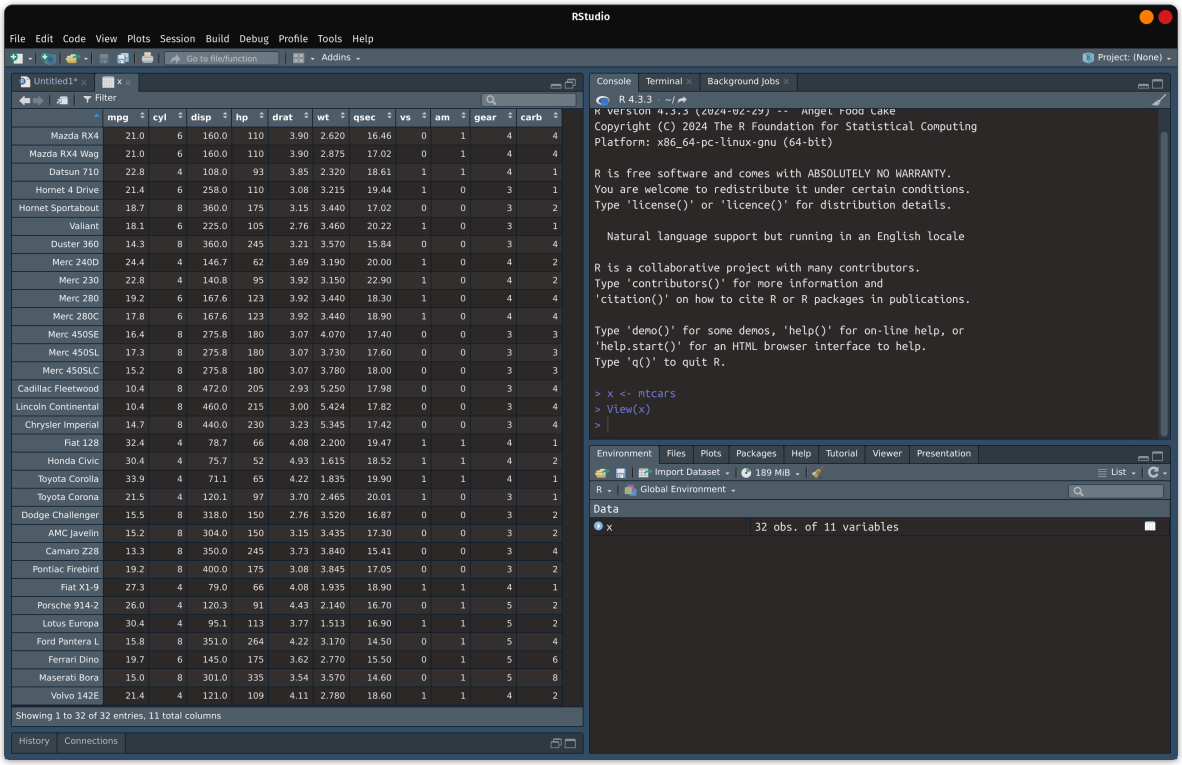


1.4 Source, Console and Plots

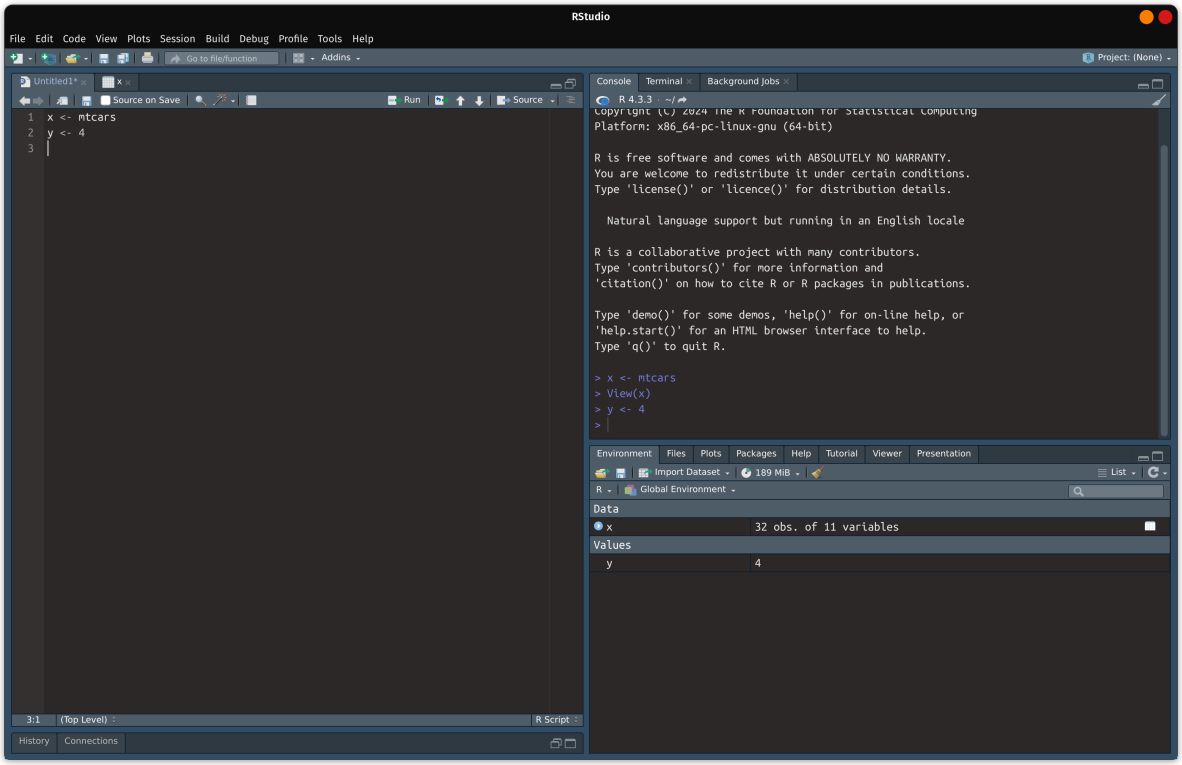
The source pane allows you to write an R script for analysis. Below `x <- mtcars` is written (top-left) and executed to R (top-right). Afterwards the "Environment" Tab in the lower right pane now how `x`. The "Environment" tab displays which R objects were created and available to use for further analysis.



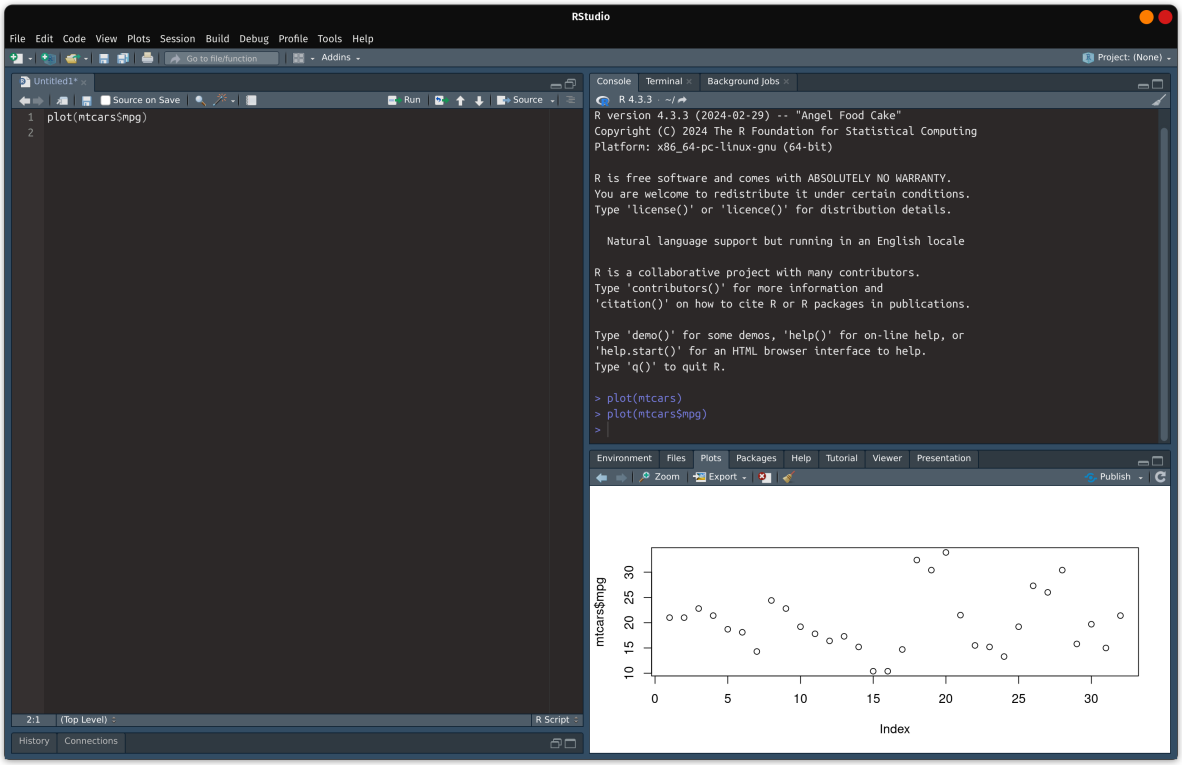
Since `x` is a data frame, clicking on `x` from the "Environment" tab will open a new tab in the Source pane containing the data set:



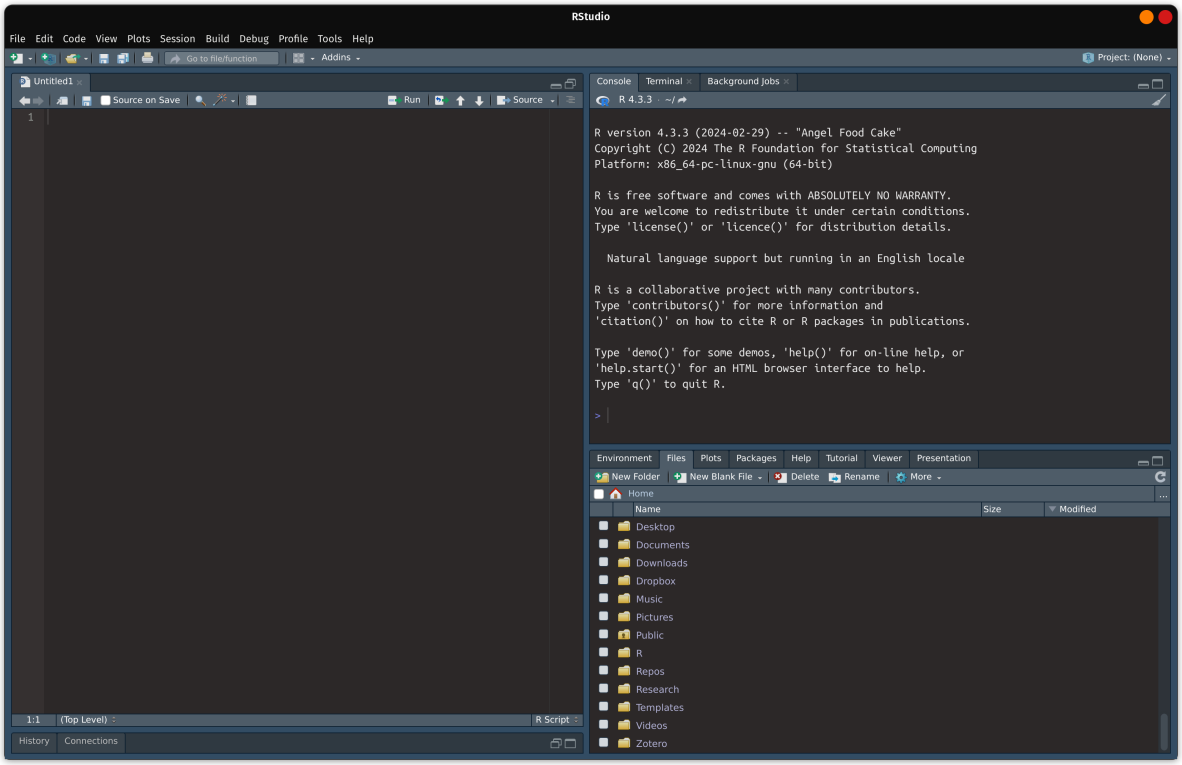
If we create an object that is a vector (`y <- 4` as pictured below), the "Environment" tab now shows a new object as a value.



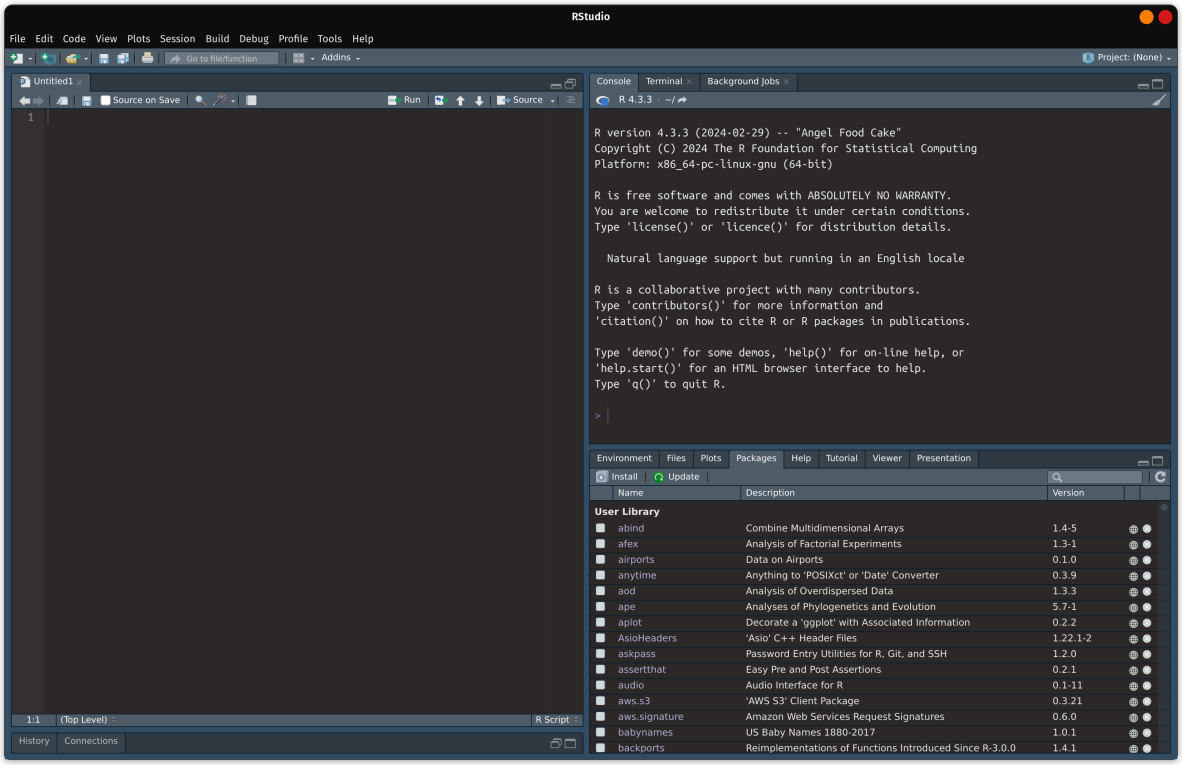
If a plot is created (`plot(mtcars$mpg)`), a plot will be displayed in the "Plots" tab in the lower-right pane.



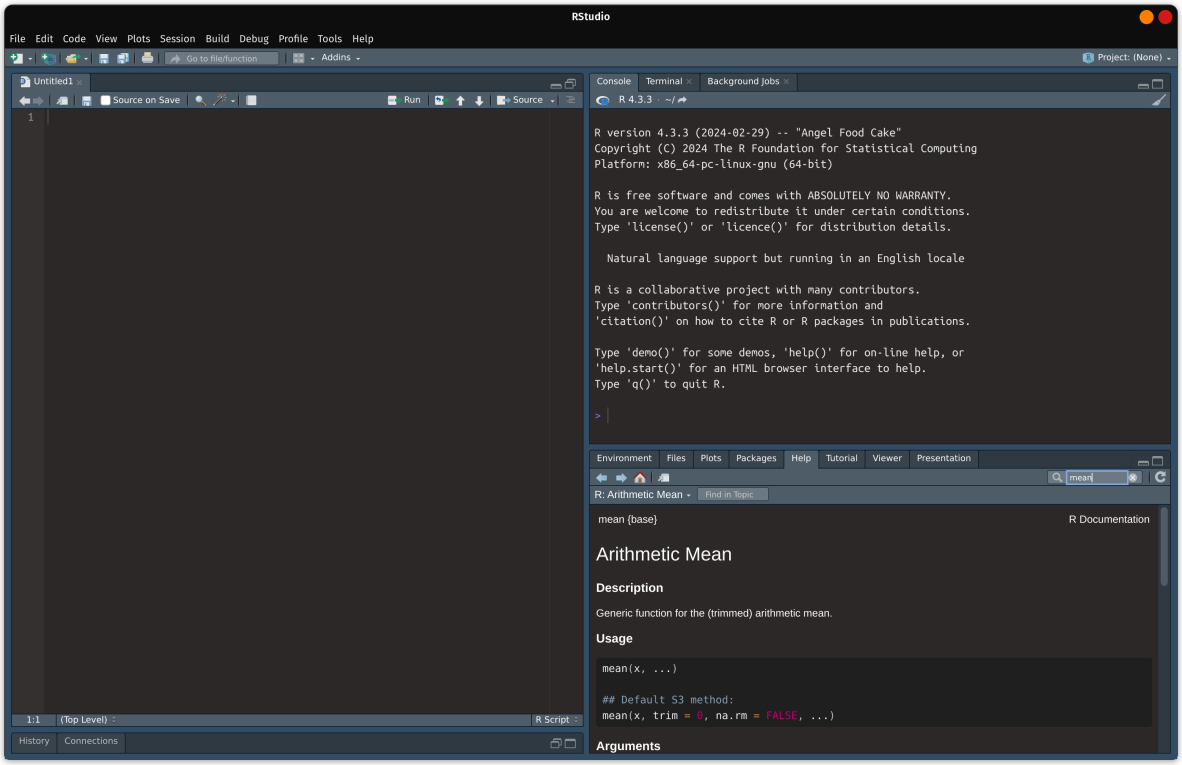
The lower right-pane also contains other useful features such as access to your computer's file directory:



Access to installed packages:



And access to help documentation:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
History
Connections
Console
Terminal
Background Jobs
R 4.3.3 · ~/
R version 4.3.3 (2024-02-29) -- "Angel Food Cake"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment
Files
Plots
Packages
Help
Tutorial
Viewer
Presentation
mean
R: Arithmetic Mean
Find in Topic
mean {base}
R Documentation
Arithmetic Mean
Description
Generic function for the (trimmed) arithmetic mean.
Usage
mean(x, ...)
Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
Arguments

2 Basic R Programming

This chapter focuses on the basics of R programming. While most of your statistical analysis will be done with R functions, it is important to have an idea of what is going on. Additionally, we will cover other topics that you may or may not need to know. The topics we will cover are:

1. Basic calculations in R
2. Types of Data
3. R Objects
4. R Functions
5. R Packages

2.1 Basic Calculations

This section focuses on the basic calculation that can be done in R. This is done by using different operators in R. The table below provides some of the basic operators R can use:

Operator	Description
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Divides
`^` or `**`	Exponent
`?`	Help Documentation

2.1.1 Calculator

2.1.1.1 Addition

To add numbers in R, all you need to use the `+` operator. For example $2 + 2 = 4$. When you type it in R you have:

```
2 + 2
```

```
[1] 4
```

When you ask R to perform a task, it prints out the result of the task. As we can see above, R prints out the number 4.

To add more than 2 numbers, you can simply just type it in.

```
2 + 2 + 2
```

```
[1] 6
```

This provides the number 6.

2.1.1.2 Subtraction

To subtract numbers, you need to use the `-` operator. Try `4 - 2`:

```
4 - 2
```

```
[1] 2
```

Try `4 - 6 - 4`

```
4 - 6 - 4
```

```
[1] -6
```

Notice that you get a negative number.

Now try `4 + 4 - 2 + 8`:

```
4 + 4 - 2 + 8
```

```
[1] 14
```

2.1.1.3 Multiplication

To multiply numbers, you will need to use the `*` operator. Try `4 * 4`:

```
4 * 4
```

```
[1] 16
```

2.1.1.4 Division

To divide numbers, you can use the `/` operator. Try `9 / 3`:

```
9 / 3
```

```
[1] 3
```

2.1.1.5 Exponents

To exponentiate a number to the power of another number, you can use the `^` operator. Try `2^5`:

```
2^5
```

```
[1] 32
```

If you want to find e^2, you will use the `exp()` function. Try `exp(2)`:

```
exp(2)
```

```
[1] 7.389056
```

2.1.1.6 Roots

To take the n-th root of a value, use the `^` operator with the `/` operator to take the n-th root. For example, to take $\sqrt[5]{35}$, type `32^(1/5)`:

```
32^(1/5)
```

```
[1] 2
```

2.1.1.7 Logarithms

To take the natural logarithm of a value, you will use the `log()` function. Try `log(5)`:

```
log(5)
```

```
[1] 1.609438
```

If you want to take the logarithm of a different base, you will use the `log()` function with `base` argument. We will discuss this more in Section 2.3.

2.1.2 Comparing Numbers

Another important part of R is comparing numbers. When you compare two numbers, R will tell if the statement is `TRUE` or `FALSE`. Below are the different comparisons you can make:

Operator	Description
`>`	Greater Than
`<`	Less Than
`>=`	Greater than or equal
`<=`	Less than or equal
`==`	Equals
`!=`	Not Equals

2.1.2.1 Less than/Greater than

To check if one number is less than or greater than another number, you will use the `>` or `<` operators. Try `5 > 4`:

```
5 > 4
```

```
[1] TRUE
```

Notice that R states it's true. It evaluates the expression and tells you if it's true or not. Try `5 < 4`:

```
5 < 4
```

```
[1] FALSE
```

Notice that R tells you it is false.

2.1.2.2 Less than or equal to/Greater than or equal to

To check if one number is less than or equal to/greater than or equal to another number, you will use the `>=` or `<=` operators. Try `5 >= 5`:

```
5 >= 5
```

```
[1] TRUE
```

Try `5 >= 4`:

```
5 >= 4
```

```
[1] TRUE
```

Try `5 <= 4`

```
5 <= 4
```

```
[1] FALSE
```

2.1.2.3 Equals and Not Equals

To check if 2 numbers are equal to each other, you can use the `==` operator. Try `3 == 3`:

```
3 == 3
```

```
[1] TRUE
```

Try `4 == 3`

```
3 == 4
```

```
[1] FALSE
```

Another way to see if 2 numbers are not equal to each other, you can use the `!=`. Try `3 != 4`:

```
3 != 4
```

```
[1] TRUE
```

Try `3 != 3`:

```
3 != 3
```

```
[1] FALSE
```

You may be asking why use `!=` instead of `==`. They both provides similar results. Well the reason is that you may need the `TRUE` output for analysis. One is only true when they are equal, while the other is true when they are not equal.

In general, the `!` operator means not or opposite. It can be used to change an `TRUE` to a `FALSE` and vice-versa.

2.1.3 Help

The last operator we will discuss is the help operator `?`. If you want to know more about anything we talked about you can type `?` in front of a function and a help page will pop-up in your browser or in RStudio's 'Help' tab. For example you can type `?Arithmetic` or `?Comparison`, to review what we talked about. For other operators we didn't talk about use `?assignOps` and `?Logic`.

2.2 Types of Data

In R, the type of data, also known as class, we are using dictates how the programming works. For the most part, users will use *numeric*, *logical*, *POSIX* and *character* data types. Other types of data you may encounter are *complex* and *raw*. To obtain more information on them, use the `?` operator.

2.2.1 Numeric

The *numeric* class is the data that are numbers. Almost every analysis that you use will be based on the numeric class. To check if you have a numeric class, you just need to use the `is.numeric()` function. For example, try `is.numeric(5)`:

```
is.numeric(5)
```

```
[1] TRUE
```

Numeric classes are essentially *double* and *integer* types of data. For example a *double* data is essentially a number with decimal value. An *integer* data are whole numbers. Try `is.numeric(5.63)`, `is.double(5.63)` and `is.integer(5.63)`:

```
is.numeric(5.63)
```

```
[1] TRUE
```

```
is.double(5.63)
```

```
[1] TRUE
```

```
is.integer(5.63)
```

```
[1] FALSE
```

Notice how the value 5.63 is a *numeric* and *double* but not *integer*. Now let's try `is.numeric(7)`, `is.double(7)` and `is.integer(7)`:

```
is.numeric(7)
```

```
[1] TRUE
```

```
is.double(7)
```

```
[1] TRUE
```

```
is.integer(7)
```

```
[1] FALSE
```

Notice how the value 7 is also considered a *numeric* and *double* but not *integer*. This is because typing a whole number will be stored as a *double*. However, if we need to store an *integer*, we will need to type the letter "L" after the number. Try `is.numeric(7L)`, `is.double(7L)`, and `is.integer(7L)`:

```
is.numeric(7L)
```

```
[1] TRUE
```

```
is.double(7L)
```

```
[1] FALSE
```

```
is.integer(7L)
```

```
[1] TRUE
```

2.2.2 Logical

A *logical* class are data where the only value is `TRUE` or `FALSE`. Sometimes the data is coded as `1` for `TRUE` and `0` for `FALSE`. The data may also be coded as `T` or `F`. To check if data belongs in the *logical* class, you will need the `is.logical()` function. Try `is.logical(3 < 4)`:

```
is.logical(3 < 4)
```

```
[1] TRUE
```

This is same comparison from Section 2.1.2. The output was `TRUE`. Now R is checking whether the output is of a *logical* class. Since it it, R returns `TRUE`. Now try `is.logical(3 > 4)`:

```
is.logical(3 > 4)
```

```
[1] TRUE
```

The output is `TRUE` as well even though the condition `3 > 4` is `FALSE`. Since the output is a *logical* data type, it is a *logical* variable.

2.2.3 POSIX

The *POSIX* class are date-time data. Where the data value is a time component. The *POSIX* class can be very complex in how it is formatted. IF you would like to learn more try `?POSIXct` or `?POSIClt`. First, lets run `Sys.time()` to check what is today's data and time:

```
Sys.time()
```

```
[1] "2024-03-19 23:38:51 PDT"
```

Now lets check if its of POSIX class, you can use the `class()` function to figure out which class is it. Try `class(Sys.time())`:

```
class(Sys.time())
```

```
[1] "POSIXct" "POSIXt"
```

2.2.4 Character

A *character* value is where the data values follow a *string* format. Examples of *character* values are letters, words and even numbers. A *character* value is any value surrounded by quotation marks. For example, the phrase "Hello World!" is considered as one *character* value. Another example is if your data is coded with the actual words "yes" or "no". To check if you have *character* data, use the `is.character()` function. Try `is.character("Hello World!")`:

```
is.character("Hello World!")
```

```
[1] TRUE
```

Notice that the output says `TRUE`. *Character* values can be created with single quotations. Try `is.character('Hello World!')`:

```
is.character('Hello World!')
```

```
[1] TRUE
```

2.2.5 Complex Numbers

Complex numbers are data values where there is a real component and an imaginary component. The imaginary component is a number multiplied by $i = \sqrt{-1}$. To create a *complex* number, use the `complex()` function. To check if a number is complex, use the `is.complex()` function. Try the following to create a complex number `complex(1, 4, 5)`:

```
complex(1, 4, 5)
```

```
[1] 4+5i
```

Now try `is.complex(complex(1, 4, 5))`:

```
is.complex(complex(1, 4, 5))
```

```
[1] TRUE
```

2.2.6 Raw

You will probably never use raw data. I have never used raw data in R. To create a raw value, use the `raw()` or `charToRaw()` functions. Try `charToRaw('Hello World!')`:

```
charToRaw('Hello World!')
```

```
 [1] 48 65 6c 6c 6f 20 57 6f 72 6c 64 21
```

To check if you have raw data, use the `is.raw()` function. Try `is.raw(charToRaw('Hello World!'))`:

```
is.raw(charToRaw('Hello World!'))
```

```
[1] TRUE
```

2.2.7 Missing

The last data class in R is missing data. The table below provides a brief introduction of the different types of missing data

Value	Description	Functions
`NULL`	These are values indicating an object is empty. Often used for functions with values that are undefined.	`is.null()`
`NA`	Stands for "Not Available", used to indicate that the value is missing in the data.	`is.na()`

Value	Description	Functions
`NaN`	Stands for "Not an Number". Used to indicate a missing number.	`is.nan()`
`Inf` and `-Inf`	Indicating an extremely large value or a value divided by 0.	`is.infinite()`

2.3 R Functions

An R function is the procedure that R will execute to certain data. For example, the `log(x)` is an R function. It takes the value `x` and provides you the natural logarithm. Here `x` is known as an argument which needs to be specified to us the `log()` function. Find the `log(x = 5)`

```
log(x = 5)
```

```
[1] 1.609438
```

Another argument for the `log()` function is the `base` argument. With the previous code, we did not specify the `base` argument, so R makes the `base` argument equal to the number e. If you want to use the common log with base 10, you will need to set the `base` argument equal to 10.

Try `log(x = 5, base = 10)`

```
log(x = 5, base = 10)
```

```
[1] 0.69897
```

Now try `log(5,10)`

```
log(5,10)
```

```
[1] 0.69897
```

Notice that it provides the same value. This is because R can set arguments based on the values position in the function, regardless if the arguments are specified. For `log(5,10)`, R thinks that 5 corresponds to the first argument `x` and 10 is the second argument `base`.

To learn more about a functions, use the `?` operator on the function: `?log`.

2.4 R Objects

R objects are where most of your data will be stored. An R object can be thought of as a container of data. Each object will share some sort of characteristics that will make the unique for different types of analysis.

2.4.1 Assigning objects

To create an R object, all we need to do is assign data to a variable. The variable is the name of the R object. it can be called anything, but you can only use alphanumeric values, underscore, and periods. To assign a value to a variable, use the `<-` operator. This is known a left assignment. Kinda like an arrow pointing left. Try assigning 9 to 'x' (`x <- 9`):

```
x <- 9
```

To see if `x` contains 9, type `x` in the console:

```
x
```

```
[1] 9
```

Now `x` can be treated as data and we can perform data analysis on it. For example, try squaring it:

```
x^2
```

```
[1] 81
```

You can use any mathematical operation from the previous sections. Try some other operations and see what happens.

The output R prints out can be stored in a variable using the asign operator, `<-`. Try storing `x^3` in a variable called `x_cubed`:

```
x_cubed <- x^3
```

To see what is stored in `x_cubed` you can either type `x_cubed` in the console or use the `print()` function with `x_cubed` inside the parenthesis.

```
x_cubed
```

```
[1] 729
```

```
print(x_cubed)
```

```
[1] 729
```

2.4.2 Vectors

A vector is a set data values of a certain length. The R object `x` is considered as a numerical vector (because it contains a number) with the length 1. To check, try `is.numeric(x)` and `is.vector(x)`:

```
is.numeric(x)
```

```
[1] TRUE
```

```
is.vector(x)
```

```
[1] TRUE
```

Now let's create a logical vector that contains 4 elements (have it follow this sequence: T, F, T, F) and assign it to `y`. To create a vector use the `c()`[1] function and type all the values and separating them with columns. Type `y <- c(T, F, T, F)`:

```
y <- c(T, F, T, F)
```

Now, lets see how `y` looks like. Type `y`:

```
y
```

```
[1]  TRUE FALSE  TRUE FALSE
```

Now lets see if it's a logical vector:

[1] The `c()` function allows you to put any data type and as many values as you wish. The only condition of a vector is that it must be the same data type.

```
is.logical(y)
```

```
[1] TRUE
```

```
is.vector(y)
```

```
[1] TRUE
```

Fortunately, this vector is really small to count how many elements it has, but what if the vector is really large? To find out how many elements a vector has, use the `length()` function. Try `length(y)`:

```
length(y)
```

```
[1] 4
```

2.4.3 Matrices

A matrix can be thought as a square or rectangular grid of data values. This grid can be constructed can be any size. Similar to vectors they must contain the same data type. The size of a matrix is usually denoted as $n \times k$, where n represents the number of rows and k represents the number of columns. To get a rough idea of how a matrix may look like, type `matrix(rep(1,12), nrow = 4, ncol = 3)`[2]:

```
matrix(rep(1, 12), nrow = 4, ncol = 3)
```

```
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
[4,]    1    1    1
```

Notice that this is a 4×3 matrix. Each element in the matrix has the value 1. Now try this `matrix(rbinom(12,1.5), nrow = 4, ncol = 3)`[3]:

[2]The function `rep()` creates a vector by repeating a value for a certain length. `rep(1,12)` creates a vector of length 12 with each element being 1. We use the `nrow` and `ncol` arguments in the function to specify the number of rows and columns, respectfully.

[3]The `rbinom()` function generates binomial random variables and stores them in a vector. `rbinom(12,1,5)` This creates 12 random binomial numbers with parameter $n = 1$ and $p = 0.5$.

```
matrix(rbinom(12, 1, .5), nrow = 4, ncol = 3)
```

```
     [,1] [,2] [,3]
[1,]    1    0    1
[2,]    0    1    0
[3,]    1    0    0
[4,]    0    1    0
```

Your matrix may look different, but that is to be expected. Notice that some elements in a matrix are 0's and some are 1's. Each element in a matrix can hold any value.

An alternate approach to creating matrices is with the use of `rbind()` and `cbind()` functions. Using 2 vectors, and matrices, of the same length, the `rbind()` will append the vectors together by each row. Similarly, the `cbind()` function will append vectors, and matrices, of the same length by columns.

```
x <- 1:4
y <- 5:8
z <- 9:12
cbind(x, y, z)
```

```
     x y  z
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
```

```
rbind(x, y, z)
```

```
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y    5    6    7    8
z    9   10   11   12
```

If you want to create a matrix of a specific size without any data, you can use the `matrix()` function and only specify the `nrow` and `ncol` arguments. Here we are creating a 5×11 empty matrix:

```
matrix(nrow = 5, ncol = 11)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
[1,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
[2,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
[3,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
[4,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
[5,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA    NA
```

Lastly, if you need to find out the dimensions of a matrix, you can use `dim()` function on a matrix:

```
dim(matrix(nrow = 5, ncol = 11))
```

```
[1]  5 11
```

This will return a vector of length 2 with the first element being the number of rows and the second element being the number of columns.

2.4.4 Arrays

Matrices can be considered as a 2-dimensional block of numbers. An array is an n-dimensional block of numbers. While you may never need to use an array for data analysis. It may come in handy when programming by hand. To create an array, use the `array()` function. Below is an example of a $3 \times 3 \times 3$ with the numbers 1, 2, and 3 representing the 3rd dimension stored in an R object called `first_array`[4].

```
(first_array <- array(c(rep(1, 9), rep(2, 9), rep(3, 9)),
                      dim=c(3,3,3)))
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
```

[4]Notice the code is surrounded by parenthesis. This tells R to store the array and print out the results. You can surround code with parenthesis every time you create an object to also print what is stored.

```
, , 2

     [,1] [,2] [,3]
[1,]    2    2    2
[2,]    2    2    2
[3,]    2    2    2

, , 3

     [,1] [,2] [,3]
[1,]    3    3    3
[2,]    3    3    3
[3,]    3    3    3
```

2.4.5 Data Frames

Data frames are similar to data set that you may encounter in an excel file. However, there are a couple of differences. First, each row represents an observation, and each column represents a characteristic of the observation. Additionally, each column in a data frame will be the same data type. To get an idea of what a data frame looks like, try `head(iris)` [5]:

```
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

In the data frame, the rows indicate a specific observation and the columns are the values of a variable. In terms of the `iris` data set, we can see that row 1 is a specific flower that has a sepal length of 5.1. We can also see that flower 1 has other characteristics such as sepal width and petal length. Lastly, there are results for the other flowers.

Now try `tail(iris)`:

```
tail(iris)
```

[5] The `head()` function just tells R to only print the top few components of the data frame.

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

The `tail()` function provides the last 6 rows of the data frame.

Lastly, if you are interested in viewing a specific variable (column) from a data frame, you can use the `$` operator to specify which variable from a specific data frame. For example, if we are interested in observing the `Sepal.Length` variable from the `iris` data frame, we will type `iris$Sepal.Length`:

```
iris$Sepal.Length
```

```
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
 [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
 [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
 [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
 [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
 [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
[109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
[127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
```

2.4.6 Lists

To me a list is just a container that you can store practically anything. It is compiled of elements, where each element contains an R object. For example, the first element of a list may contain a data frame, the second element may contain a vector, and the third element may contain another list. It is just a way to store things.

To create a list, use the `list()` function. Create a list compiled of first element with the mtcars data set, second element with a vector of zeros of size 4, and a matrix 3×3 identity matrix[6]. Store the list in an object called `list_one`:

```
list_one <- list(mtcars, rep(0, 4),
                 diag(rep(1, 3)))
```

[6] An identity matrix is a matrix where the diagonal elements are 1 and the non-diagonal elements are 0

Type `list_one` to see what pops out:

```
list_one
```

```
[[1]]
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2

[[2]]
[1] 0 0 0 0
```

```
[[3]]
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Each element in the list is labeled as a number. It is more useful to have the elements named. An element is named by typing the name in quotes followed by the = symbol before your object in the `list()` function (`mtcars=mtcars`).

```
list_one <- list(mtcars = mtcars,
                 vector = rep(0, 4),
                 identity = diag(rep(1, 3)))
```

Here I am creating an object called `list_one`, where the first element is `mtcars` labeled `mtcars`, the second element is a vector of zeros labeled `vector` and the last element is the identity matrix labeled `identity`.'

Now create a new list called `list_two` and store `list_one` labeled as `list_one` and `first_array` labeled as `array`.

```
(list_two <- list(list_one = list_one,
                  array = first_array))
```

```
$list_one
$list_one$mtcars
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98  0  0    3    4
```

```
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2

$list_one$vector
[1] 0 0 0 0

$list_one$identity
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1


$array
, , 1

     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1

, , 2

     [,1] [,2] [,3]
[1,]    2    2    2
[2,]    2    2    2
[3,]    2    2    2
```

```
, , 3

     [,1] [,2] [,3]
[1,]    3    3    3
[2,]    3    3    3
[3,]    3    3    3
```

2.5 R Packages

As stated before, R's functionality can be extended to do more things by installing R packages. An R package can be thought as extra software. This allows you to do more with R. To install an R package, you will need to use the `install.packages("NAME_OF_PACKAGE")` function. Once you install it, you do not need to install it again. To use the R package, use `library("NAME_OF_PACKAGE")`. This allows you to load the package in R. You will need to load the package every time you start R. For more information, please watch the video:

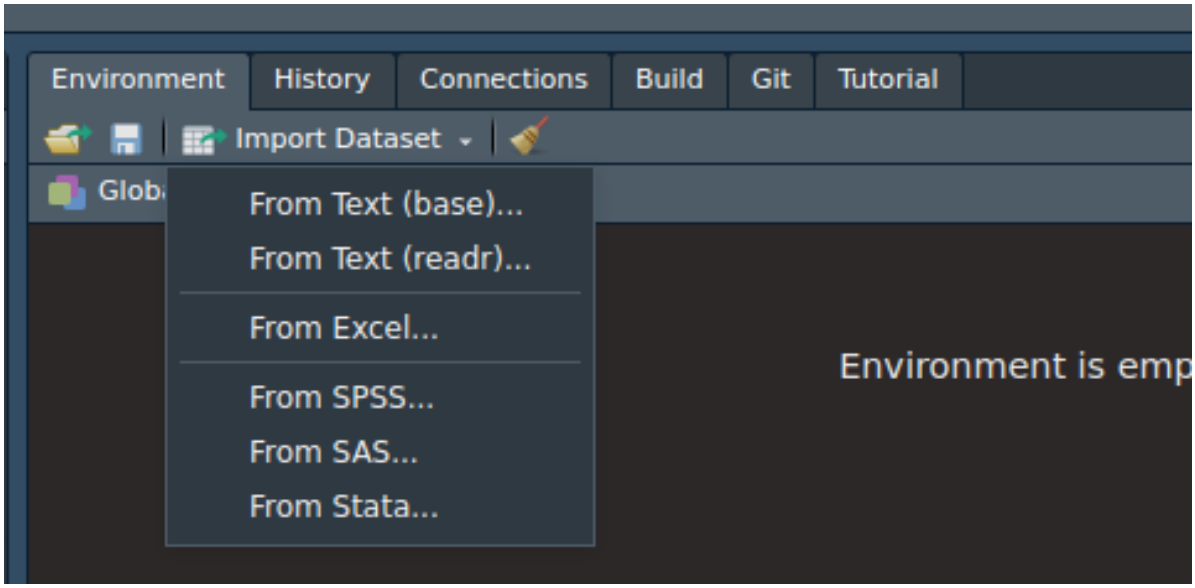
2.6 Load Data

In order to analyze data in R, we must load it into the R environment. This can be done in 2 ways, using the "Import Dataset" button in the "Environment" tab in RStudio or use R code.

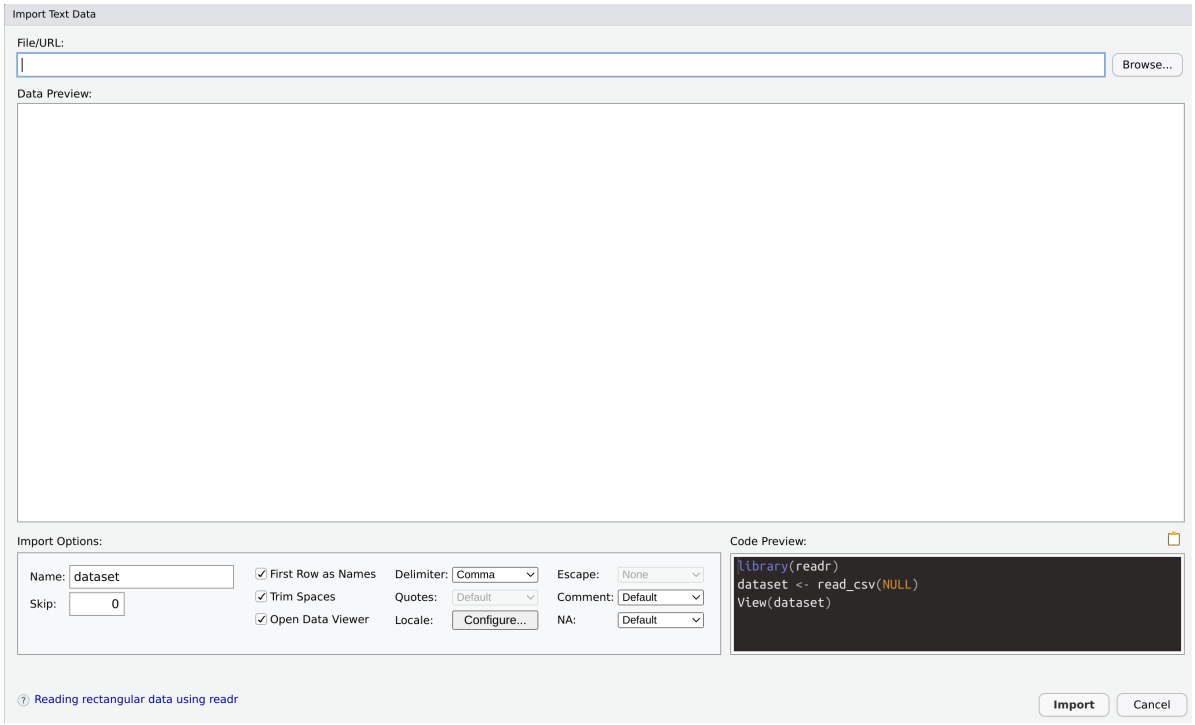
2.6.1 Importing Data Via RStudio

This is the most recommended way to import data in RStudio because it can provide R code that you can copy and paste in an R Script.

To begin choose the "Import Dataset" from the "Environment" tab in RStudio:



Afterwards, select the type of file that you may need to import. If you select the "From Text (readr)..." option, a popup window will appear:



You can now navigate to the file that you may want to import with the "Browse…" button and modify it the process as needed with the options. Afterwards, you can copy the code in the lower-right hand corner and save it in an R script

3 Base R

4 Tidyverse

Tidyverse is a set of R packages that are commonly used to transform data for analysis. These packages share the same philosophy and structure when fixing data.

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

4.1 tidyr

4.2 dplyr

4.3 ggplot2

5 Teaching R

Part II

Python Programming

6 Python Basics

7 Python and Data Science

8 Teaching Python

Part III

Online Computing

9 Notebooks

10 Google Colab

11 JupyterHub

12 Integrating in Learning Management System

Part IV

Version Control

How many times when you are working finalizing a document and you save it as `final_15.docx` or something like `true_final_5.docx` or `FINAL_FINAL_3.docx` or `ONLY_LOOK_AT_THIS_ONE_FINAL_2.docx`. This is a common trait all of do to ensure that we save our updates so we can look at the differences between versions. This is where version control can help organize and track your changes between different versions of a file.

In this part of the book, we will demonstrate the basics of version control with the use of git and GitHub.

If you want a more in depth version of using git and GitHub, take a look at the following resource table:

Website	Description
Happy Git	Provide an overview of git and GitHub while using RStudio.
Pro Git	A highly recommended book for those who want to gain a deep understanding of git.
Oh S***, Git!?!	Provides troubleshooting techniques when the inevitable mistakes occur.
Git in Simple Words	Provides git basics in simplified words.

13 GitHub

14 Git Basics

Part V

Scientific Documentation and Communication

15 Markdown Basics

Markdown files contain basic formatting capabilities. The use of the `#` followed by text creates a heading. Using two or more `#` symbols will create subheadings based on the number of `#`. A text is *italicized* by surrounding the text with one asterisk (`*italicized*`). A text is **boldfaced** by surrounding it with 2 asterisk (`**boldfaced**`). A text is ~~strikethrough~~ by surrounding the text with 2 tildes (`~~strikethrough~~`). A text with$^{\text{superscript}}$ is created by surrounding a word with caret (`^superscript^`). A text with$_{\text{subscript}}$ is created by surrounding a word with 1 tilde (`~subscript~`).

To create an unordered list, use the `-` symbol at the beginning of each line. To create a sub-item, press the tab button, then the `-` symbol. Repeat this method for further sub-items.

Syntax	Output

`- First Item`
`- Second Item`
`  - First Sub-Item`
`  - First Sub-Sub Item`
`  - First Sub-Sub-Sub Item`
	• First Item
	• Second Item
	– First Sub-Item
	∗ First Sub-Sub Item
	· First Sub-Sub-Sub Item

To created an ordered list, type the number followed by a period for each line. To create sub-lists, press the tab button twice and order them appropriately.

Syntax	Output

`1. First Item`
`2. Second Item`
`  a. First Sub-Item`
`  i. First Sub-Sub-Item`
`  ii. Second Sub-Sub-Item`
	1. First Item
	2. Second Item
	a. First Sub-Item
	i. First Sub-Sub Item
	ii. Second Sub-Sub Item

A block quote is created with the `>` symbol at the beginning of a line.

Markdown files allows a table to be constructed in 2 ways, manually or using a package such as the `gt` package. A table is manually created by using `|`, `:`, and `-`. The first line contains

| and the column names in between. The second line contains |:-|:-| which indicates how the table is aligned. The location of : symbol just tells RStudio about the alignment. Below is the example code of Table 15.4:

```
| Letter     |   Lowercase   |      Code     |  Uppercase |    Code    |
|:-----------|:-------------:|:-------------:|:----------:|:----------:|
| alpha      |    $\alpha$   |    `\alpha`   |     --     |     --     |
| beta       |    $\beta$    |    `\beta`    |     --     |     --     |
| gamma      |    $\gamma$   |    `\gamma`   |  $\Gamma$  |  `\Gamma`  |
| delta      |    $\delta$   |    `\delta`   |  $\Delta$  |  `\Delta`  |
| epsilon    |   $\epsilon$  |   `\epsilon`  |     --     |     --     |
| zeta       |    $\zeta$    |    `\zeta`    |     --     |     --     |
| eta        |     $\eta$    |     `\eta`    |     --     |     --     |
| theta      |    $\theta$   |    `\theta`   |  $\Theta$  |  `\Theta`  |
| iota       |    $\iota$    |    `\iota`    |     --     |     --     |
| kappa      |    $\kappa$   |    `\kappa`   |     --     |     --     |
| lambda     |   $\lambda$   |   `\lambda`   |  $\Lambda$ |  `\Lambda` |
| mu         |     $\mu$     |     `\mu`     |     --     |     --     |
| nu         |     $\nu$     |     `\nu`     |     --     |     --     |
| xi         |     $\xi$     |     `\xi`     |    $\Xi$   |    `\Xi`   |
| pi         |     $\pi$     |     `\pi`     |    $\Pi$   |    `\pi`   |
| rho        |     $\rho$    |     `\rho`    |     --     |     --     |
| sigma      |    $\sigma$   |    `\sigma`   |  $\Sigma$  |  `\Sigma`  |
| tau        |     $\tau$    |     `\tau`    |     --     |     --     |
| upsilon    |   $\upsilon$  |   `\upsilon`  | $\Upsilon$ | `\Upsilon` |
| phi        |     $\phi$    |     `\phi`    |   $\Phi$   |   `\Phi`   |
| chi        |     $\chi$    |     `\chi`    |     --     |     --     |
| psi        |     $\psi$    |     `\psi`    |   $\Psi$   |   `\Psi`   |
| omega      |    $\omega$   |    `\omega`   |  $\Omega$  |  `\Omega`  |
| varepsilon | $\varepsilon$ | `\varepsilon` |     --     |     --     |
```

15.1 Math

Quarto is capable of writing mathematical formulas using LaTeX code. A mathematical symbol can be written inline using single $ signs. For example, `$\alpha$` is viewed as α in a document. To write mathematical formulas on its own line use $$. For example, `$$Y=mX+b$$` is viewed as

$$Y = mX + b.$$

15.1.1 Mathematical Notation

Table 15.3: LaTeX syntax for common examples.

Notation	code
$x = y$	`$x=y$`
$x > y$	`$x>y$`
$x < y$	`$x<y$`
$x \geq y$	`$x\geq y$`
$x \leq y$	`$x\leq y$`
x^y	`$x^{y}$`
x_y	`$x_{y}$`
$\bar{x}$	`$\bar x$`
$\hat{x}$	`$\hat x$`
$\tilde{x}$	`$\tilde x$`
$\frac{x}{y}$	`$\frac{x}{y}$`
$\frac{\partial x}{\partial y}$	`$\frac{\partial x}{\partial y}$`
$x \in A$	`$x\in A$`
$x \subset A$	`$x\subset A$`
$x \subseteq A$	`$x\subseteq A$`
$x \cup A$	`$x\cup A$`
$x \cap A$	`$x\cap A$`
$\{1, 2, 3\}$	`$\{1,2,3\}$`
$\int_a^b f(x)dx$	`$\int_a^bf(x)dx$`
$\left\{\int_a^b f(x)dx\right\}$	`$\left\{\int_a^bf(x)dx\right\}$`
$\sum_{i=1}^{n} x_i$	`$\sum^n_{i=1}x_i$`
$\prod_{i=1}^{n} x_i$	`$\prod^n_{i=1}x_i$`
$\lim_{x \to 0} f(x)$	`$\lim_{x\to0}f(x)$`
$X \sim \Gamma(\alpha, \beta)$	`$X\sim \Gamma(\alpha,\beta)$`

15.1.2 Greek Letters

Table 15.4: LaTeX syntax for greek letters.

Letter	Lowercase	Code	Uppercase	Code
alpha	α	`\alpha`	–	–
beta	β	`\beta`	–	–
gamma	γ	`\gamma`	Γ	`\Gamma`
delta	δ	`\delta`	Δ	`\Delta`
epsilon	ϵ	`\epsilon`	–	–
zeta	ζ	`\zeta`	–	–

Letter	Lowercase	Code	Uppercase	Code
eta	η	`\eta`	–	–
theta	θ	`\theta`	Θ	`\Theta`
iota	ι	`\iota`	–	–
kappa	κ	`\kappa`	–	–
lambda	λ	`\lambda`	Λ	`\Lambda`
mu	μ	`\mu`	–	–
nu	ν	`\nu`	–	–
xi	ξ	`\xi`	Ξ	`\Xi`
pi	π	`\pi`	Π	`\pi`
rho	ρ	`\rho`	–	–
sigma	σ	`\sigma`	Σ	`\Sigma`
tau	τ	`\tau`	–	–
upsilon	υ	`\upsilon`	Υ	`\Upsilon`
phi	ϕ	`\phi`	Φ	`\Phi`
chi	χ	`\chi`	–	–
psi	ψ	`\psi`	Ψ	`\Psi`
omega	ω	`\omega`	Ω	`\Omega`
varepsilon	ε	`\varepsilon`	–	–

16 Quarto Documents with R

16.1 Introduction

Quarto is a file type used to create technical reports while including both R code, or other programming languages, and output in a document. A qmd[1] file is a fancy R Script containing extra capabilities. Additionally, qmd files allow for citations, footnotes, mathematical expressions, links, and much more. Once the document is finished, it can be rendered to a word file, pdf, html file, and much more. Quarto is the considered the next generation of RMarkdown.

16.2 Anatomy of a Quarto Document

There are three main components in an qmd file: the YAML header, R code, and basic text.

The YAML header contains information on how to render the document. It is located at the beginning of the document surrounded by 3 dashes (`---`) above and below it. For starters, the YAML header will contain a 'title', 'author', 'date', and 'output' line.

The R code is located in a block known as chunks. A chunk tells RStudio to read the next lines as code. A chunk begins with three back ticks followed by `{r}` and ends with three back ticks. Everything in between the back ticks will be executed by R. In RStudio, a chunk can be inserted using the keyboard shortcut `ctrl+alt+I` or `cmd+option+I`.

An example of an R chunk is shown below:

````
```{r}
mean(mtcars$mpg)
```
````

The R chunk will be rendered as below:

```r
mean(mtcars$mpg)
```

[1] QMD is the file extension to use the Quarto engine. For this document Quarto and QMD are used interchangeably.

Notice the chunk includes the code in a block followed by the output from the console.

The last component of an qmd document is the text. Write anywhere in the document, and it will be rendered as is.

16.3 Chunk Options

R chunks have options that will alter how the code or the output is rendered. The chunk options can be set either globally to affect the entire qmd document or locally to affect only an individual chunk. For more information about chunk options, visit https://yihui.org/knitr/options/

16.3.1 Global Chunk Options

To set global chunks options, add the two lines to YAML header:

```
#| echo: true
---
knitr:
  opts_chunk:
---
```

Followed by the chunks and R options you want to set:

```
#| echo: true
---
knitr:
  opts_chunk:
    eval: false
    tidy: styler
    R.options:
      digits: 2
---
```

A couple of recommended chunk options set globally are `eval: false`, and `tidy: styler`. These options make rendering the document easier.

One chunk option is `tidy: styler`. This tells Quarto to prevent code from printing in a long line, possibly off the page. For example, look at the output of this chunk:

```
## This comment is designed to show what happens when all your code is in 1 line. This is fi
```

Notice the comment being printed off the page. Using the options `tidy: styler`, the chunk is rendered as

```
## This comment is designed to show what happens when all
## your code is in 1 line. This is fine when you are
## coding, but when you are putting it in a report, it will
## run off the page.
```

The last 2 lines control how R will compute and print output. `R.Options:` tells Quarto that the options R will be changed, and each line after alters options. `digits: 2` indicates R to use 2 significant digits.

16.3.2 Local Chunk Options

Local chunk options can be used to control an individual chunk will behave. To control a specific chunk, place the option below the `{r}` identifier and use the `#|` chunk option indicator. An example is povided below:

````
```{r}
#| eval: false
#| tidy: false

mean(mtcars$mpg)
```
````

The chunk option `eval` set to `false` tells Quarto to not evaluate the code within the chunk. Notice how the output was not printed the R chunk above. When we set `eval` to `true`, the output is printed:

````
```{r}
#| eval: true
mean(mtcars$mpg)
```
````

```
[1] 20.1
```

The `echo` option will control if the code within the chunk should be printed in the document. This next chunk contains `#| echo: true`:

```r
mean(mtcars$mpg)
```

```
[1] 20.1
```

Now the chunk contains `#| echo: false`:

```
[1] 20.1
```

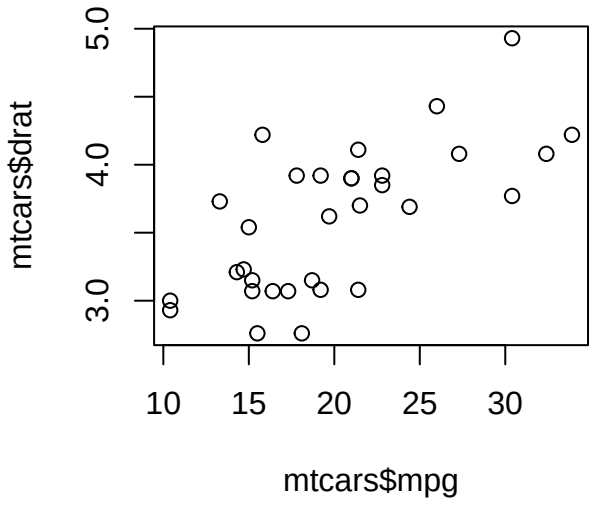
The R Code disappears.

There are chunk options for figures as well. A few options are `fig-height`, `fig-width`, `fig-align`, and `fig-cap`.

This chunk contains `fig-height: 3.5; fig-width: 3.5; fig-align: left`.

````
```{r}
#| eval: true
#| fig-height: 3.5
#| fig-width: 3.5
#| fig-align: left

plot(mtcars$mpg, mtcars$drat)
```
````



The chunk options tells RStudio to create an image that is 3.5 inches in height and width, and align the image to the left.

The following chunk contains `fig-height: 3.5; fig-width: 3.5; fig-align: left; fig-cap: "This is a scatter plot of MTCARS' MPG and DRAT"; label: fig-mtcars`.

````
```{r}
#| eval: true
#| fig-height: 3.5
#| fig-width: 3.5
#| fig-align: center
#| fig-cap: "This is a scatter plot of MTCARS' MPG and DRAT"
#| label: fig-mtcars

plot(mtcars$mpg, mtcars$drat)
```
````

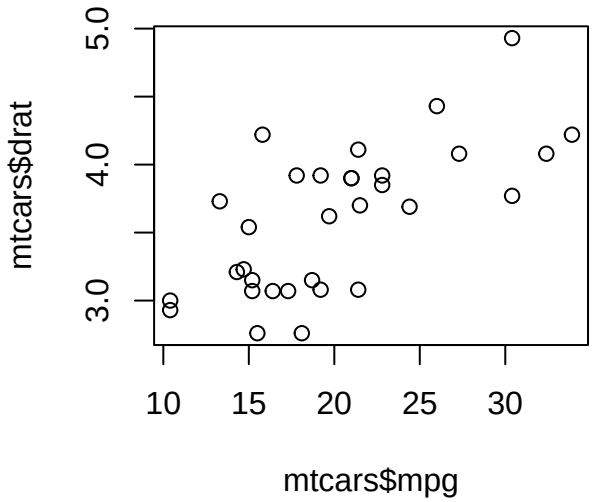


Figure 16.1: This is a scatter plot of MTCARS' MPG and DRAT

The chunk adds a caption with `fig-cap` and reference label with `label`. The label of the plot can be referenced later in the document. Figure 16.1 can be referenced with `@fig-mtcars`.

16.3.3 Inline Code

Instead of evaluating code in a chunk, code can evaluated in the text instead. For example, if we want to write the mean mpg in `mtcars` is 20.090625, one can type 20.090625, surrounded by back ticks (`), instead of writing the entire number and risk of miscopying the results.

16.4 Formatting

Qmd files allows a table to be constructed in 2 ways, manually or using a package such as the `gt` package. Below is the example code of Table 15.4:

```
| Letter     |   Lowercase   |      Code     |  Uppercase |    Code    |
|:-----------|:-------------:|:-------------:|:----------:|:----------:|
| alpha      |    $\alpha$   |    `\alpha`   |     --     |     --     |
| beta       |    $\beta$    |    `\beta`    |     --     |     --     |
| gamma      |    $\gamma$   |    `\gamma`   |  $\Gamma$  |  `\Gamma`  |
| delta      |    $\delta$   |    `\delta`   |  $\Delta$  |  `\Delta`  |
| epsilon    |   $\epsilon$  |   `\epsilon`  |     --     |     --     |
| zeta       |    $\zeta$    |    `\zeta`    |     --     |     --     |
| eta        |     $\eta$    |     `\eta`    |     --     |     --     |
| theta      |    $\theta$   |    `\theta`   |  $\Theta$  |  `\Theta`  |
| iota       |    $\iota$    |    `\iota`    |     --     |     --     |
| kappa      |    $\kappa$   |    `\kappa`   |     --     |     --     |
| lambda     |   $\lambda$   |   `\lambda`   |  $\Lambda$ |  `\Lambda` |
| mu         |     $\mu$     |     `\mu`     |     --     |     --     |
| nu         |     $\nu$     |     `\nu`     |     --     |     --     |
| xi         |     $\xi$     |     `\xi`     |    $\Xi$   |    `\Xi`   |
| pi         |     $\pi$     |     `\pi`     |    $\Pi$   |    `\pi`   |
| rho        |     $\rho$    |     `\rho`    |     --     |     --     |
| sigma      |    $\sigma$   |    `\sigma`   |  $\Sigma$  |  `\Sigma`  |
| tau        |     $\tau$    |     `\tau`    |     --     |     --     |
| upsilon    |   $\upsilon$  |   `\upsilon`  | $\Upsilon$ | `\Upsilon` |
| phi        |     $\phi$    |     `\phi`    |   $\Phi$   |   `\Phi`   |
| chi        |     $\chi$    |     `\chi`    |     --     |     --     |
| psi        |     $\psi$    |     `\psi`    |   $\Psi$   |   `\Psi`   |
| omega      |    $\omega$   |    `\omega`   |  $\Omega$  |  `\Omega`  |
| varepsilon | $\varepsilon$ | `\varepsilon` |     --     |     --     |

: LaTeX syntax for greek letters. {#tbl-greektable}
```

The last line will adds a caption to the table and `{#tbl-greektable}` creates a label to reference the table in the text using `@tbl-greektable`.

The `gt` function from the `gt` package creates a table from a data frame or R object. Here is an example code to create a table from the first 6 rows of the `mtcars` dataset:

```
```{r}
#| label: tbl-mtcarsdata
```
```

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.0	6	160	110	3.90	2.62	16.5	0	1	4	4
21.0	6	160	110	3.90	2.88	17.0	0	1	4	4
22.8	4	108	93	3.85	2.32	18.6	1	1	4	1
21.4	6	258	110	3.08	3.21	19.4	1	0	3	1
18.7	8	360	175	3.15	3.44	17.0	0	0	3	2
18.1	6	225	105	2.76	3.46	20.2	1	0	3	1

Table 16.1

```
#| eval: true

mtcars |>
    head() |>
    gt::gt() |>
    gt::tab_caption("The MTCARS Dataset")
```

Notice that Table 16.1 is easily produced using the `gt()` function with a caption

Table Table 16.1 is referenced by using the label created in the chunk and the `@tbl-mtcarsdata`. To install `gt` run the following line in your console:

```
install.packages("gt")
```

16.5 Citations and Referneces

Qmd documents contains capabilities to add citations and a bibliography. For example, to cite this textbook [@mendenhallSecondCourseStatistics2012], use the `@` symbol followed by a citation identifier from the `.bib` file surrounded by square brackets, `[@mendenhallSecondCourseStatistics2012]`. To cite your textbook again [-@mendenhallSecondCourseStatistics2012] without the authors names, use a `-` sign in front of the `@` symbol, `[-@mendenhallSecondCourseStatistics2012]`. To cite multiple books [@casellaStatisticalInference1990; @rohatgiIntroductionProbabilityStatistics2015; @resnickProbabilityPath2014; @lehmannTheoryPointEstimation1998; @lehmannTestingStatisticalHypotheses2005], add each citation inside the square brackets with the `@` symbol and separate them with semicolons, `[@casellaStatisticalInference1990; @rohatgiIntroductionProbabilityStatistics2015; @resnickProbabilityPath2014; @lehmannTheoryPointEstimation1998; @lehmannTestingStatisticalHypotheses2005]`.

The references will be added automatically at the end of the document.

In order to use citations and references, the qmd file needs needs a `.bib` file containing all the information of the references. First, save the `.bib` file in the same folder (directory) as your `qmd` file. Then add the line `bibliography: NAME.bib` to the YAML header. Make any changes appropriately to the line, such as the name of the `.bib` file.

16.5.1 `.bib` File

The `.bib` file is an ordinary text file containing "bib" entries with information about each reference. Below is an example bib entry about R:

```
@Manual{,
  title = {R: A Language and Environment for Statistical Computing},
  author = {{R Core Team}},
  organization = {R Foundation for Statistical Computing},
  address = {Vienna, Austria},
  year = {2024},
  url = {https://www.R-project.org/},
}
```

Creating a `.bib` file is tedious; however, there are reference managers that can help. I recommend using Zotero, an open-source reference manager designed to import and manage citations. Once a citation is in Zotero, you can export your library as a `.bib` file. Make sure to check your references in Zotero for any mistakes.

16.6 Rendering a Document

A qmd file can be rendered into either an html file, pdf document or word document. Rendering the qmd file to an html file or word document can be easily done using the knit button above. However, rendering the qmd file to a pdf document requires LaTeX to be installed. There are two methods to install LaTeX: from the LaTeX website or from R. I recommend installing the full LaTeX distribution from the https://www.latex-project.org/get/. This provides you with everything you may need. You can also install it from R:

```
install.packages("tinytex")
tinytex::install_tinytex()
```

You will only need to run these lines of code once and then you can render pdf documents easily.

HTML

```
#| echo: true
---
format: html
---
```

PDF

```
#| echo: true
---
format: pdf
---
```

Word Document

```
#| echo: true
---
format: docx
---
```

16.7 Resources and Tips

Quarto

- Quarto
- Markdown Basics
- Formats

RMarkdown

- RStudio
- Bookdown

YAML

- UCLA Resource
- Reproducible Research
- RMarkdown Crash Course

Tips

- Render your document often so it easier to identify problems with rendering
- The Visual Mode in RStudio eases the process of creating a document and makes it more bearable.
- YAML is tricky with spacing. Make sure that spaces when indenting options. Also make sure that there are not extra spaces at the end of each line.

16.8 References

17 Quarto Presentations with R

18 Creating a Quarto Website

Below are instructions to create a website using Quarto and GitHub Pages. Here is an example website: Isaac Quintanilla Salinas

18.1 Creating a GitHub Account (Required)

GitHub can be considered as the google drive of code. It has great features to track code, implement changes, and host websites. You can create a Github account here: https://github.com/.

Once you have your account create a new repository with the following naming scheme: USERNAME.github.io. Then, make sure that repository is **public,** and click to create a README document.

18.2 Building a Website with Quarto

I highly recommend building your website using quarto, it is very simple platform where your text documents become webpages.

18.2.1 Using Posit Cloud

Posit Cloud (posit.cloud) is an excellent resource to create website. It has all the software that you will need to create a website. Before we begin, make sure to create repository as stated by in the Creating a GitHub Account (Required) section.

18.2.1.1 Creating an Account

When you load the website, create a free account using an email account or github account. Once you create an account, the dashboard should load with your workspace.

18.2.1.2 Creating a Project

On the top-right corner, click on the "New Project" blue button, and select "New RStudio Project". This will create a new project called "Untitled Project". Now change the name "Untitled Project" to "Website". This project will create the material you need to develop a website.

18.2.1.3 Creating and Connecting an SSH Key

In you website project, you will need a way for the project to connect with GitHub. You can do this by generating an SSH key in RStudio Cloud and transferring it to GitHub.

The SSH key can be thought of as a key that will unlock your computer to transfer data. You will generate the lock and key in RStudio, and then give the key to GitHub. Afterwards, transferring files will occur securely.

To generate an SSH key, go to the menu bar and select "Tools", then "Global Options". A menu will pop-up, select "Git/SVN" from the side menu. Click on the button "Create SSH key ...". A window will pop-up, then enter a password. Then click "Create". Another window will pop-up showing your lock and key, you can close the window immediately.

In a different tab, go to your github account. On the top-right corner, click on your profile and select "Settings". This will redirect you to a different page. In the left menu, click on "SSH and GPG Keys". In the "SSH keys" section, click on the "New SSH key" green button. A new page will pop-up that will allow you to add a key. In the "Title" section, add any title[1].

Back at RStudio Cloud, click on the "View public key" link. Copy the highlighted text. Go back to Github and paste it in the "Key" section. Lastly, click on the "Add SSH Key" green button.

Now you can transfer file easily between RStudio and GitHub.

18.2.1.4 Cloning a Repository

Cloning a repository is the process of downloading a repository from a remote server, in this case GitHub account. This will allow you to re-download your repository if it is ever deleted.

In GitHub, navigate to your repository. On the top-right hand corner, click on the green button labeled "Code". Make sure the "SSH" tab is selected and copy, or click on the double squares button, the text they provide. You should copy something that look like this:

[1] I usually give different names of galaxies for a title.

```
git@github.com:USERNAME/USERNAME.github.io.git
```

In your RStudio Cloud Project, select the terminal tab in your console pane, usually on the left-side of the IDE. Paste the following text in the terminal tab[2]:

```
git clone git@github.com:USERNAME/USERNAME.github.io.git
```

It will prompt you to accept the SSH connection, type "yes". Afterwards, it will ask you for your SSH password. Type your password. Then, it will download your repository as a folder. In the "Files" tab, you should see the newly created folder.

18.2.1.5 Making a Quarto Website

For this section, we will primarily be working in the "Terminal" Tab in RStudio. Click on the "Terminal" Tab in the "Console" Pane. The you will type the following:

```
cd USERNAME.github.io
```

Make sure to replace USERNAME with your user name for GitHub. The above command will change the working folder to "USERNAME.github.io". Afterwards, you want to fill the folder with the necessary contents for making a Quarto Website. Type the following command in the terminal:

```
quarto create-project . --type website
```

This will generate new files for your website. Next, you will need to create a "nojekyll" file that tells GitHub, not to use Jekyll. Type the following in the terminal:

```
touch .nojekyll
```

Now, go to the "Files" Tab and open "_quarto.yml". At the top of the document, you will need to add `output-dir: docs` under the `project:`, you should have something like this:

```
project:
  type: website
  output-dir: docs
```

Now you are set to Render your website!

[2]You may need to use a special keyboard shortcut to paste text. For me, it is ctrl-shift-v. Right-clicking should work as well.

18.2.1.6 Rendering and Publishing a Website

Whenever you are finished updating your website files, you will need render your website so it will update all the new content. To begin, make sure you are working in the directory (folder) containing your website files:

```
cd USERNAME.github.io
```

Now, you can render your website using the following command:

```
quarto render
```

Your website will be rendered in the `docs` folder. Then you will need to commit the changes using git. First type

```
git add .
```

The commit your files using the following commands:

```
git commit -m "Updates"
```

Lastly, push your updates to GitHub:

```
git push
```

It will prompt your for your password related towards you SSH key. Then it will push all the updates. GitHub will then publish your website in a few minutes. Lastly, type "USERNAME.github.io" in any browser and your website should work.

18.2.2 Using RStudio on Mac/Linux

Coming Soon ...

18.2.3 Using RStudio on Windows

Coming Soon ...

18.3 Creating Webpages

To create a webpage, you will simply open a new quarto document, edit the page, and save it in your website's folder. To link the webpage in your menu bar, add the qmd file to the "_quarto.yml" file:

```
website:
  title: "inqs909.github.io"
  navbar:
    left:
      - href: index.qmd
        text: Home
      - about.qmd
      - NEWFILE.qmd
```

The render your website and push it to GitHub.