R Programming Basics: A User-Friendly Introduction

Instructor: Justin Barker

Maps, Data and Government Information Center
madgichelp@trentu.ca

Workshop content

All content for this workshop can be found:
E:/RBasics/

- Slides
- Cheat sheet
- Script

All workshop materials are also available on [MaDGIC's Github]

Maps, Data and Government Information Centre

Collections Support

Data Visualization Lab

Advanced computers, spatial, statistical and visualization software

Bata Library 402

GIS, Statistical, and Data Visualization Support

ESRI / ArcGIS

Research Data Management and Archiving

Research Collaborations

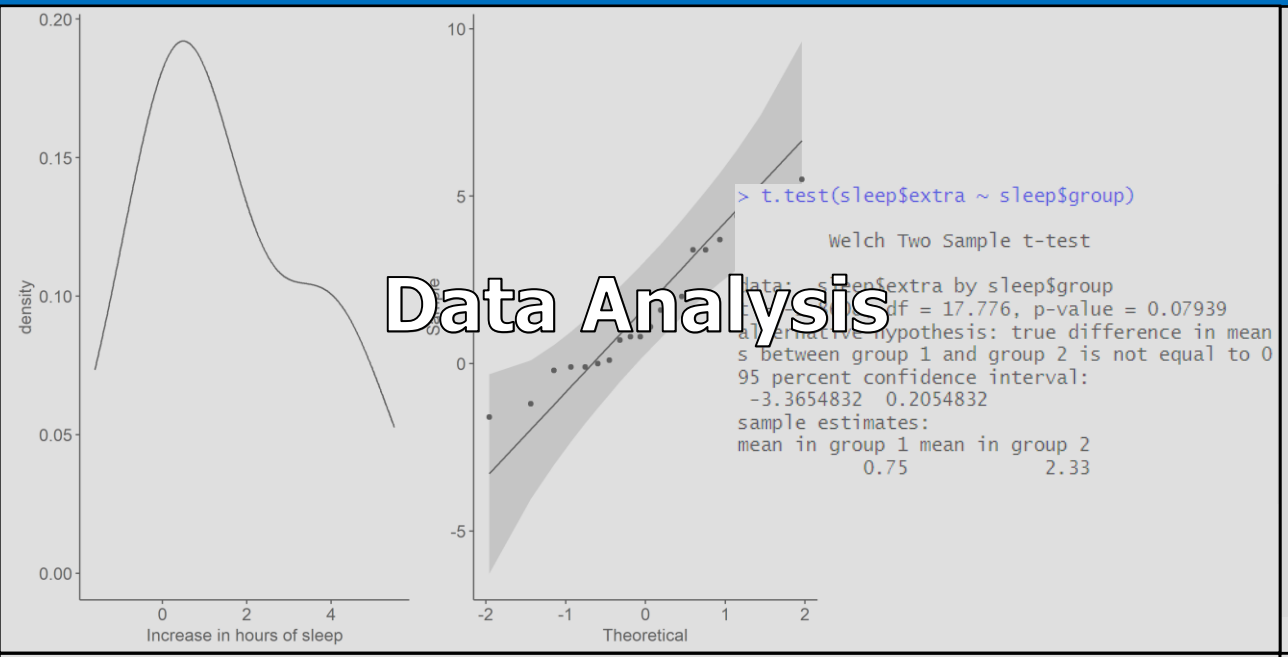
Regional Environmental History Atlas Project

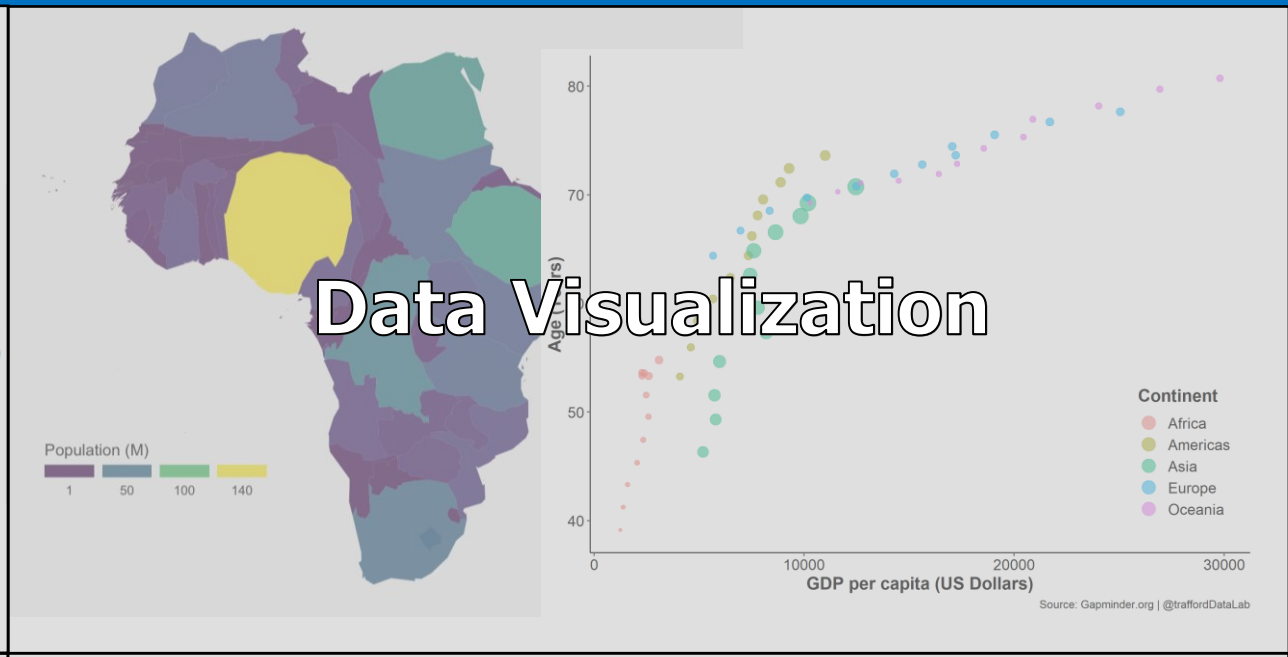
Workshops and teaching support

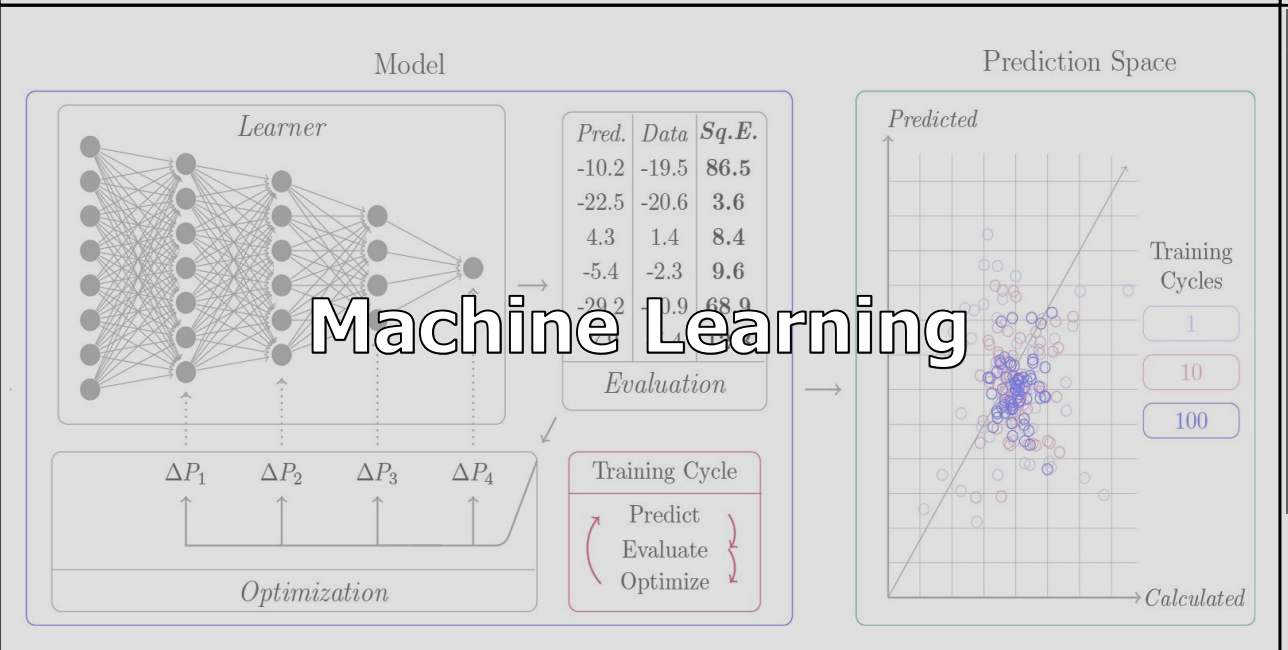
Bata Library 415
8:30 am to 4:30 pm, Monday to Friday

madgichelp@trentu.ca
https://www.trentu.ca/library/madgic

What is R used for?

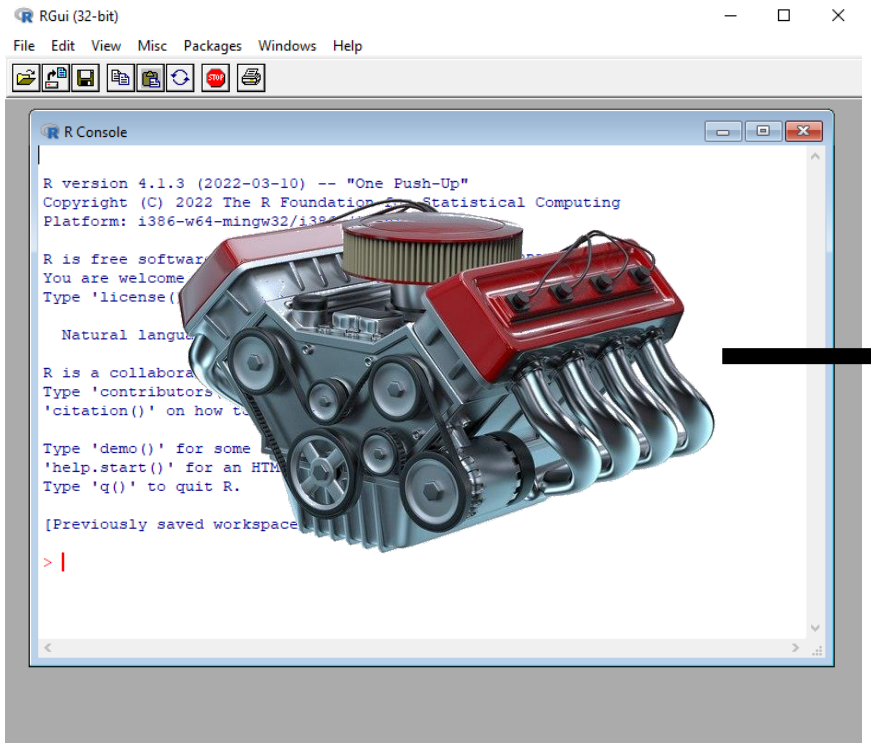






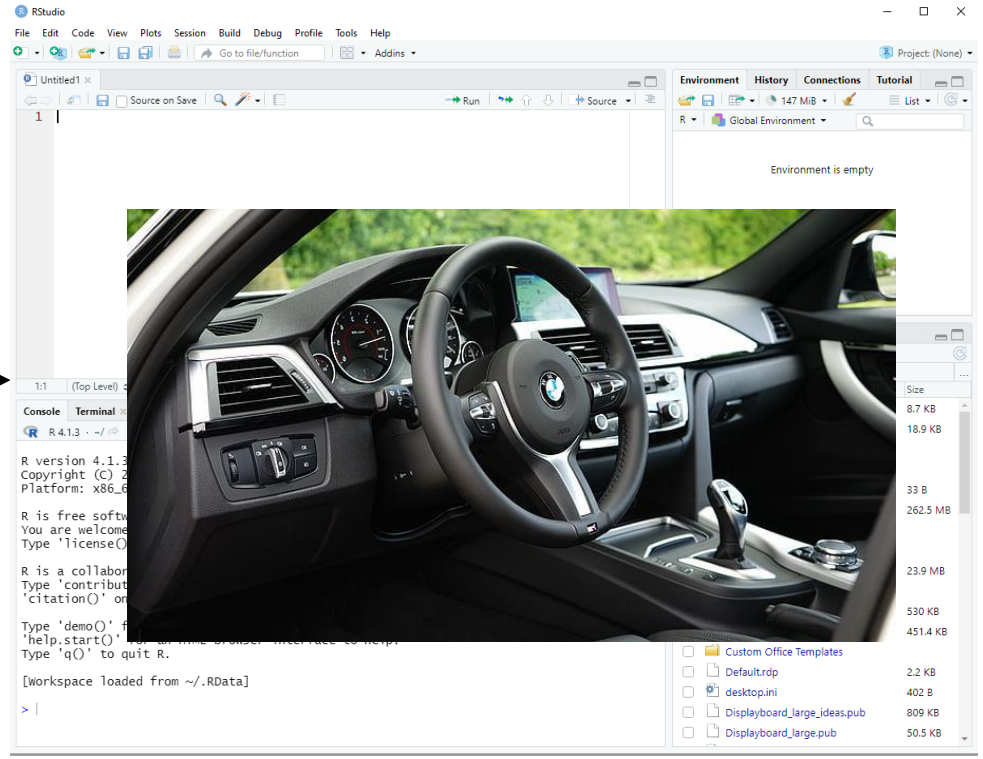
R and RStudio

- An open-source language and statistical program



R is the engine for RStudio

- An integrated development environment for R



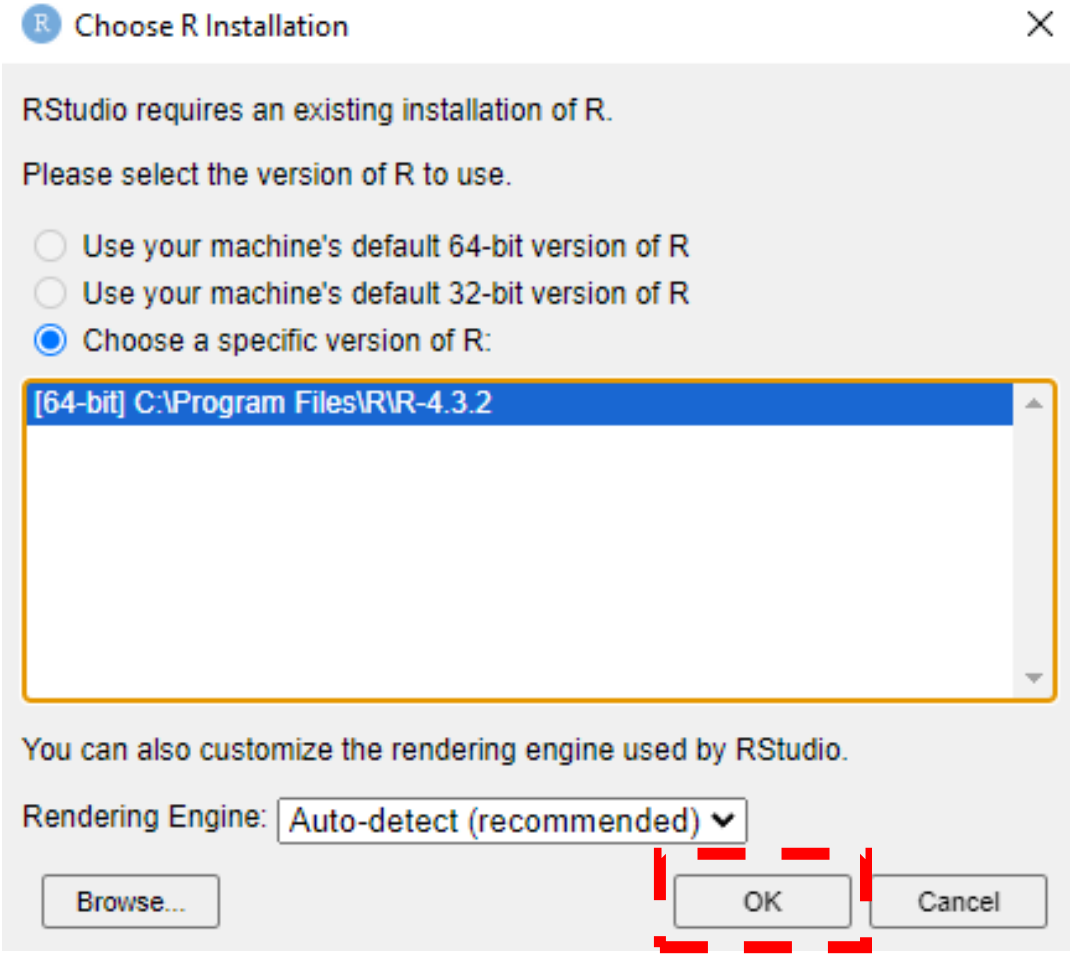
RStudio provides a user-friendly dashboard to use R

This workshop will focus on RStudio®

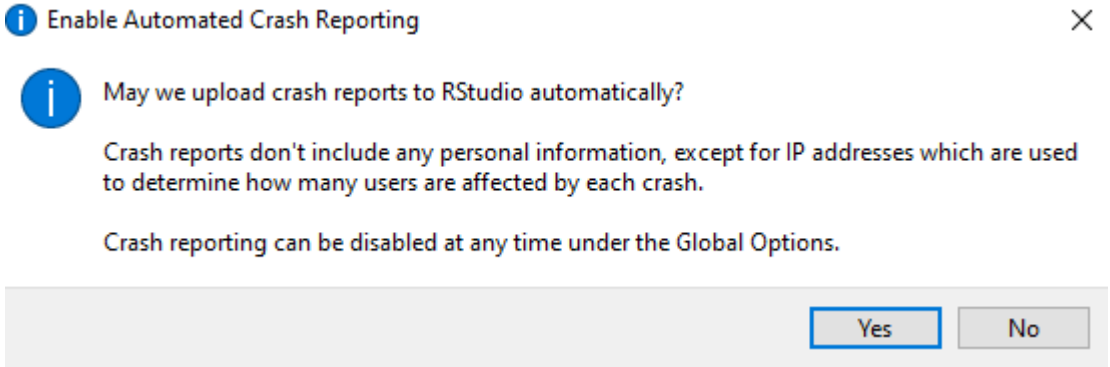
All code presented can be completed in R

Please open RStudio®

Upon opening RStudio for the first time there will be two popups.



Choose a specific version of R



Select Yes or No for R to upload crash reports

Learning objectives

- Understand the `RStudio` layout
- Use functions
- Install packages
- R help pages
- Create objects
- Understand the different data types and structures

Workshop structure

Learn
Short presentation of concepts

Practice
Guided walkthrough of R code

```
# A multiplication
______

# Exponentiation
____

# More complex calculations: BEDMAS
_____ / _____
```

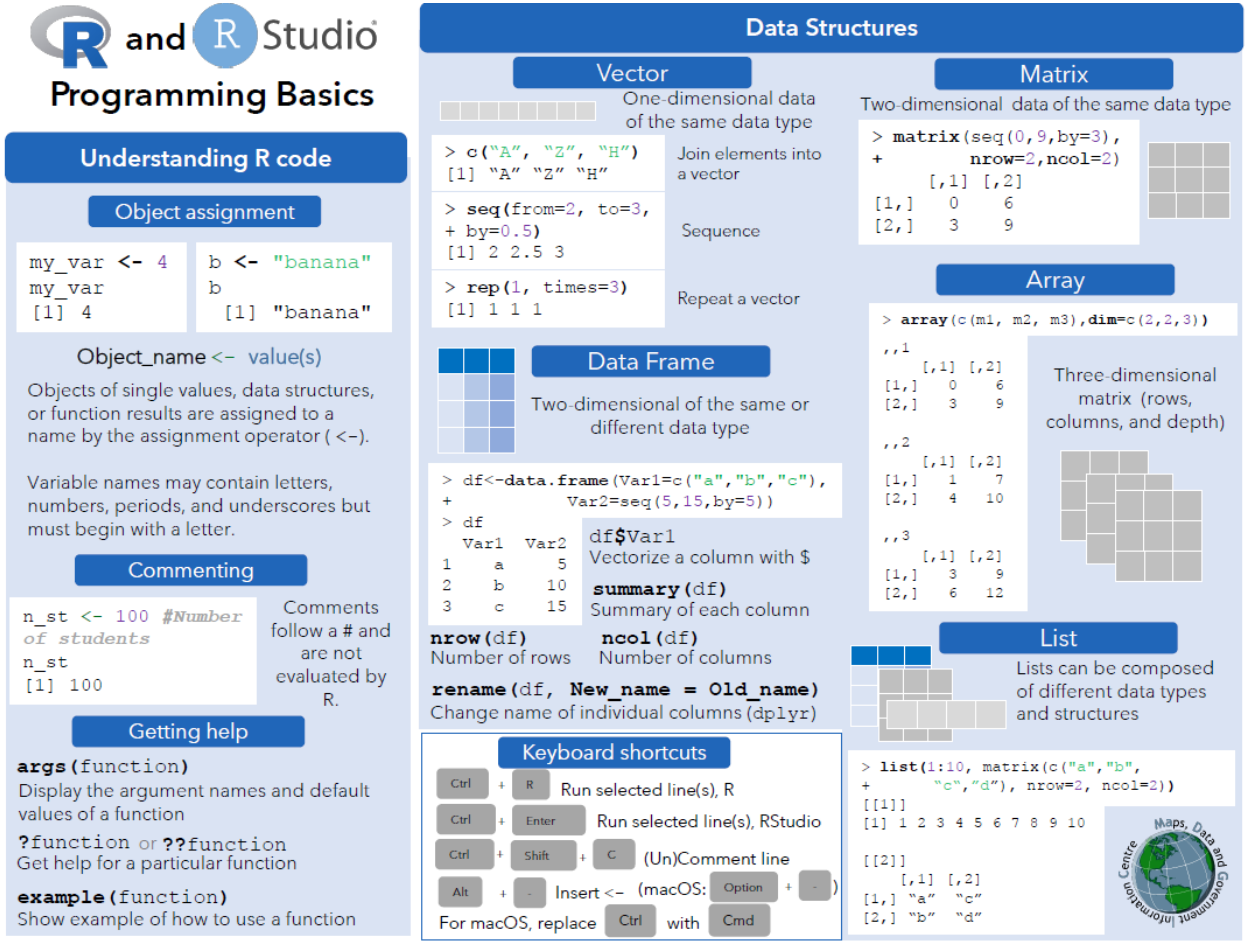
Follow along to fill in the blanks in the script

Apply
Test your skills with a short challenge

R Cheat sheets

The main concepts and commands for this workshop are summarized in the provided handout.

Please feel free to refer to the handout at any time.



R and RStudio

Programming Basics

Understanding R code

Object assignment

```
my_var <- 4
my_var
[1] 4
```

```
b <- "banana"
b
 [1] "banana"
```

Object_name <- value(s)

Objects of single values, data structures, or function results are assigned to a name by the assignment operator (<-).

Variable names may contain letters, numbers, periods, and underscores but must begin with a letter.

Commenting

```
n_st <- 100 #Number of students
n_st
[1] 100
```

Comments follow a # and are not evaluated by R.

Getting help

args(function)
Display the argument names and default values of a function

?function or **??function**
Get help for a particular function

example(function)
Show example of how to use a function

Data Structures

Vector

One-dimensional data of the same data type

```
> c("A", "Z", "H")
[1] "A" "Z" "H"
```

Join elements into a vector

```
> seq(from=2, to=3,
+ by=0.5)
[1] 2 2.5 3
```

Sequence

```
> rep(1, times=3)
[1] 1 1 1
```

Repeat a vector

Matrix

Two-dimensional data of the same data type

```
> matrix(seq(0,9,by=3),
+        nrow=2,ncol=2)
     [,1] [,2]
[1,]    0    6
[2,]    3    9
```

Array

```
> array(c(m1, m2, m3),dim=c(2,2,3))
,,1
     [,1] [,2]
[1,]    0    6
[2,]    3    9

,,2
     [,1] [,2]
[1,]    1    7
[2,]    4   10

,,3
     [,1] [,2]
[1,]    3    9
[2,]    6   12
```

Three-dimensional matrix (rows, columns, and depth)

Data Frame

Two-dimensional of the same or different data type

```
> df<-data.frame(Var1=c("a","b","c"),
+                Var2=seq(5,15,by=5))
> df
  Var1  Var2
1    a     5
2    b    10
3    c    15
```

df$Var1
Vectorize a column with $

summary(df)
Summary of each column

nrow(df)
Number of rows

ncol(df)
Number of columns

rename(df, New_name = Old_name)
Change name of individual columns (dplyr)

List

Lists can be composed of different data types and structures

```
> list(1:10, matrix(c("a","b",
+        "c","d"), nrow=2, ncol=2))
[[1]]
[1] 1 2 3 4 5 6 7 8 9 10

[[2]]
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
```

Keyboard shortcuts

Ctrl + R — Run selected line(s), R

Ctrl + Enter — Run selected line(s), RStudio

Ctrl + Shift + C — (Un)Comment line

Alt + - — Insert <- (macOS: Option + -)

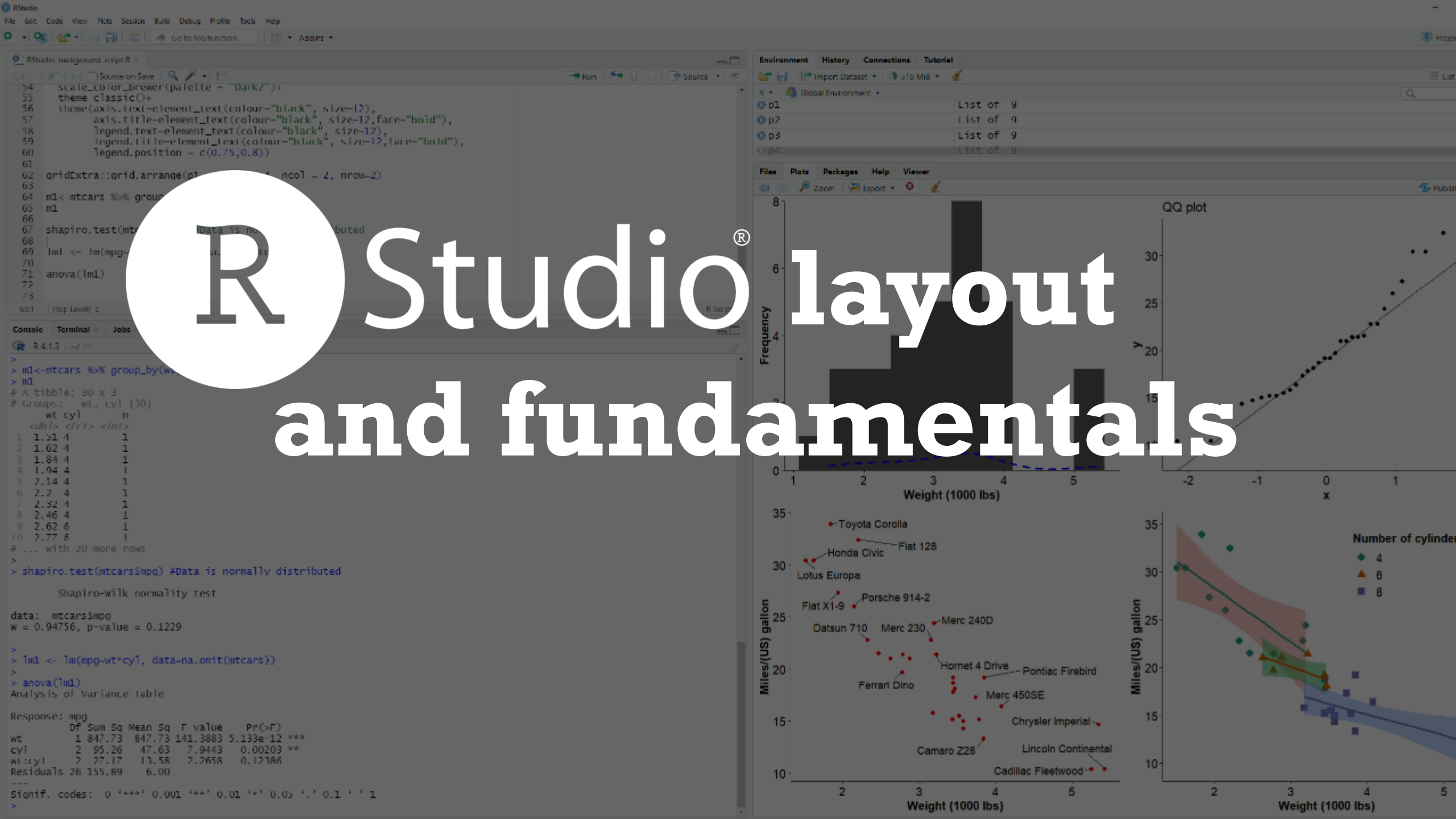
For macOS, replace Ctrl with Cmd

Digital copy is available:
E:/RBasics/Cheatsheet_RBasics.pdf

More available from:

posit™

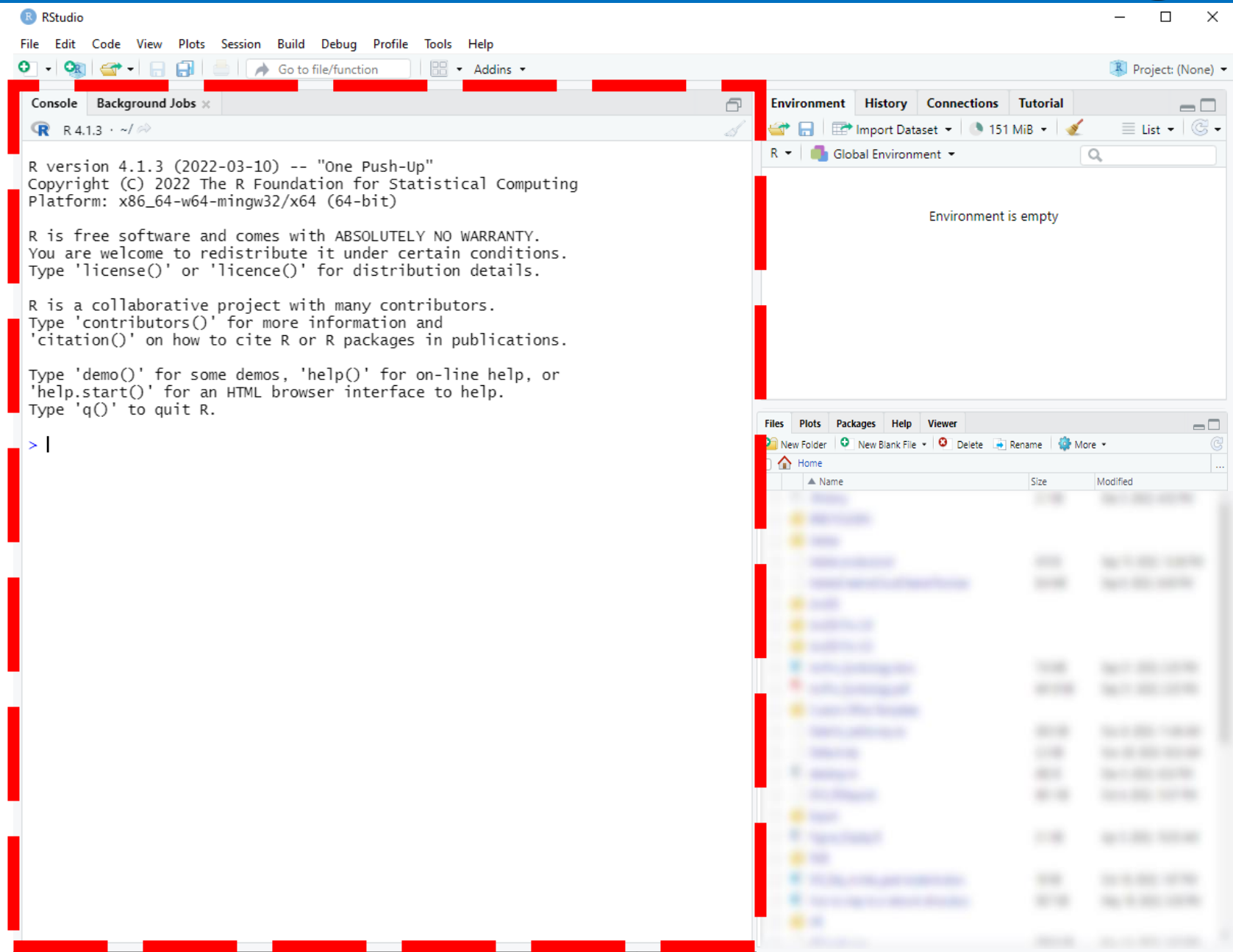
RStudio® layout and fundamentals



RStudio panes

Console

Where code is run, and results shown



RStudio panes

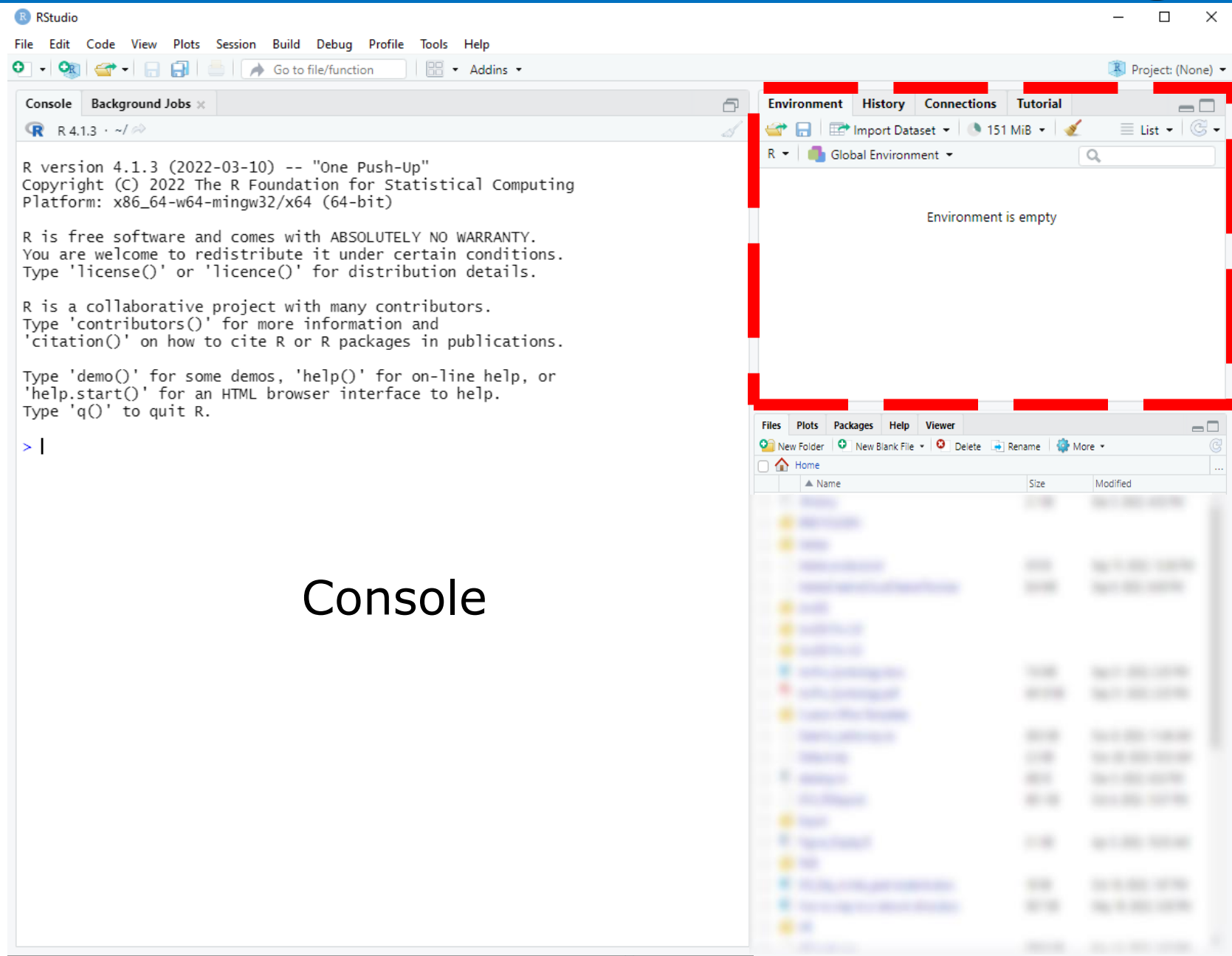
Environment/History/Connections/Tutorial

Environment: summary of data loaded into R

History: code previously run

Connections: helps the user connect to existing data sources (e.g. Access, SQL)

Tutorial: available tutorials within RStudio from `learnr`



Console

RStudio panes

Files/Plots/Packages/Help/Viewer

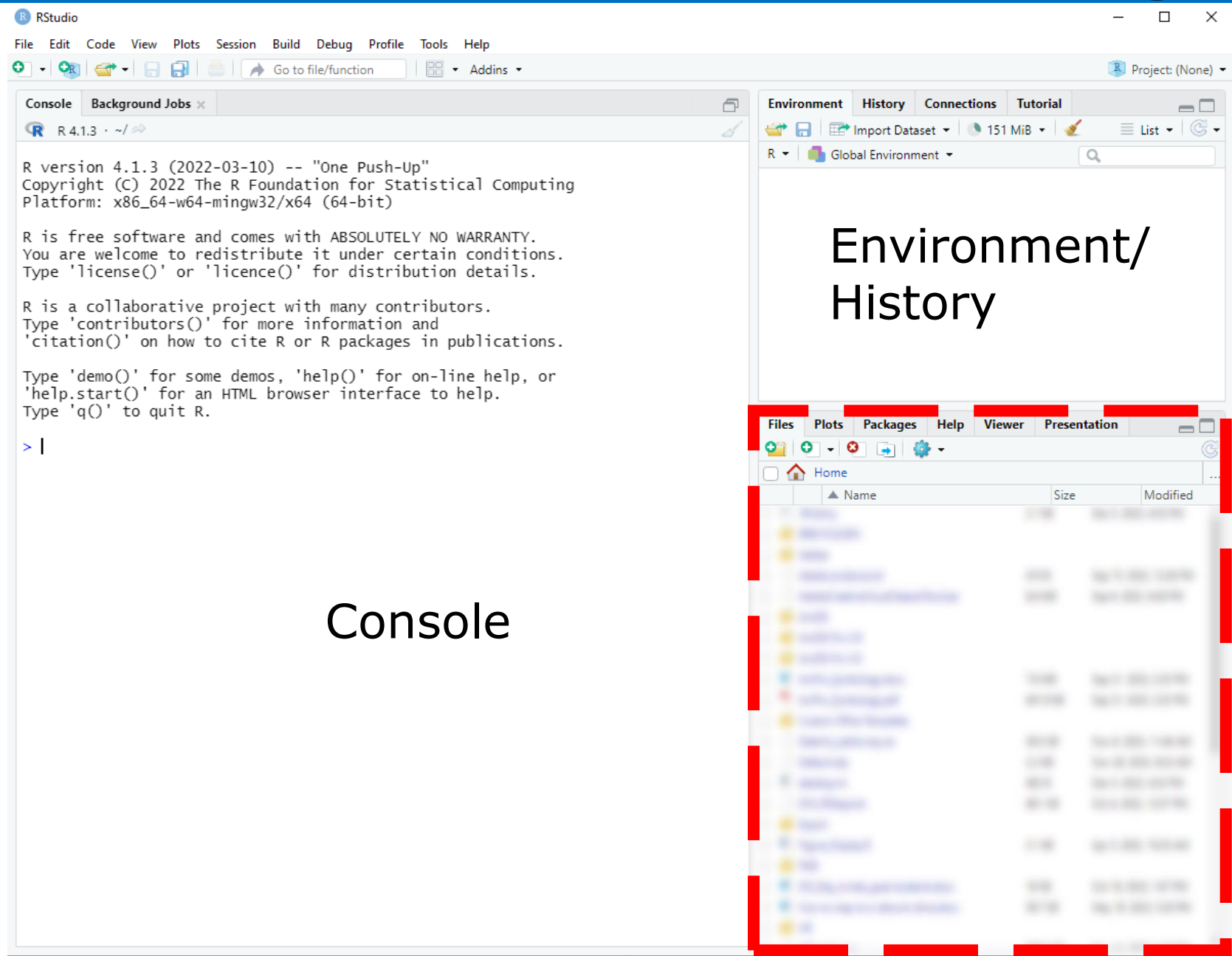
Files: Simplified file explorer

Plots: display generated figures

Packages: install/load packages

Help: R help documentation viewer

Viewer: displays local web content generated by user



RStudio panes

Source

A syntax editor for writing, editing, and saving code.

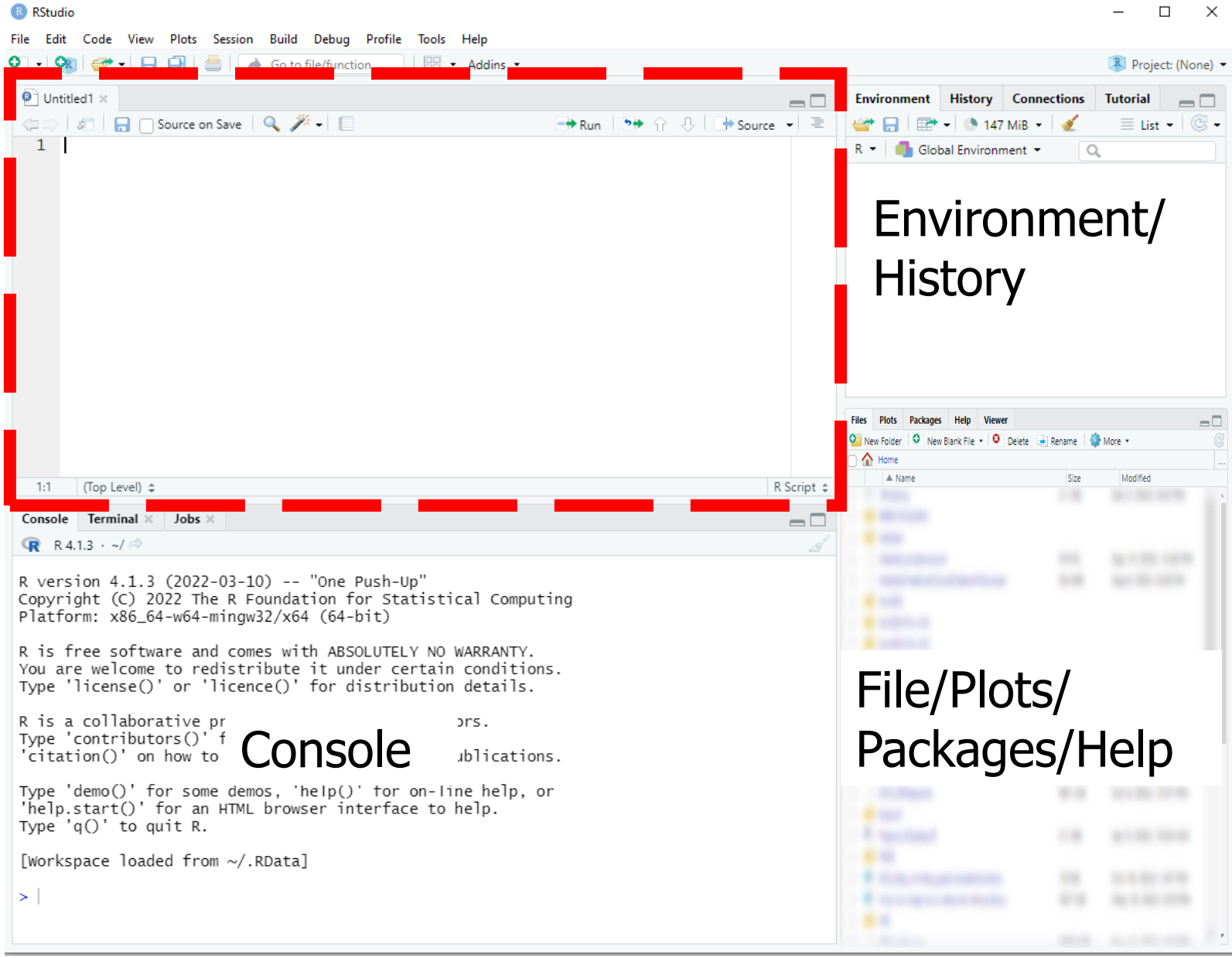
This pane will not appear by default.

Open a new script with

(`Ctrl` + `Shift` + `N` (`Cmd` + `Shift` + `N` on Mac))

Open an existing script with

(`Ctrl` + `O` (`Cmd` + `O` on Mac))



Open the provided workshop script

E:/RBasics/RBasics.R

Within this file, please **fill in the blanks** to complete the R script.

```
# A multiplication
______

# Exponentiation
____

# More complex calculations: BEDMAS
_____ / _____
```

Follow along to fill in the blanks in the script

E:/RBasics/RBasics_filled.R

Within this file, the blanks are filled for you

```
# A multiplication
3*5

# Exponentiation
2^9

# More complex calculations: BEDMAS
(100*5)/(4^16)
```

Commenting

The `#` symbol indicates a comment. Comments tell `R` to ignore anything written to the right within the same line. Use comments to:

- Improve code readability
  - Annotate someone else's script
  - Help collaborators understand
- Prevent execution of code
- Include resources

<u>How to quickly add comments:</u>

Use `Ctrl` + `Shift` + `C` to comment or uncomment an entire line

- `Cmd` + `Shift` + `C` on Mac

```
30 + 4 # This is a comment
5 + 1
# This is a comment
```

Commenting: Section headers

Comments may be used to create sections to move quickly between sections or hide sections.

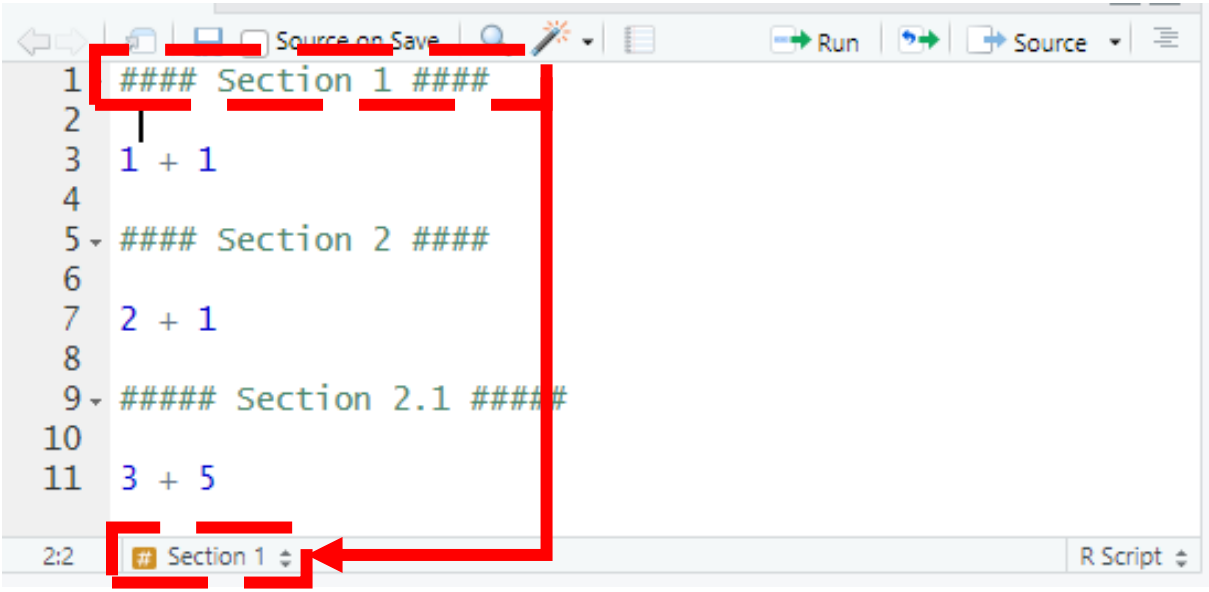
Header (4 #s)

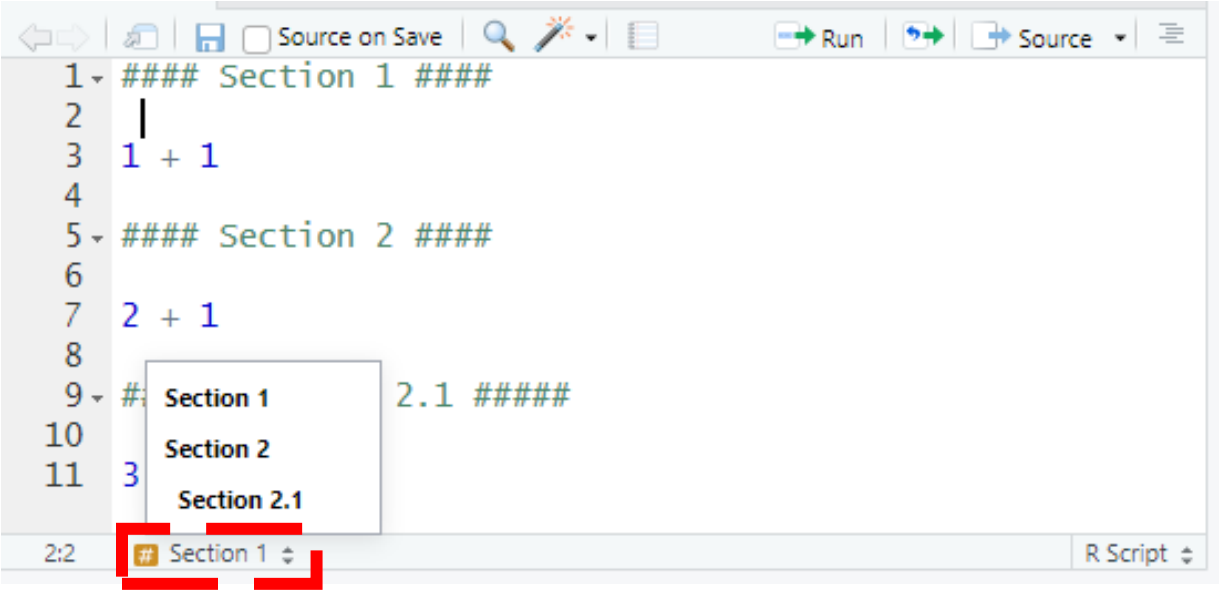
```
#### Header 1 ####
```

Sub-header (5 #s)

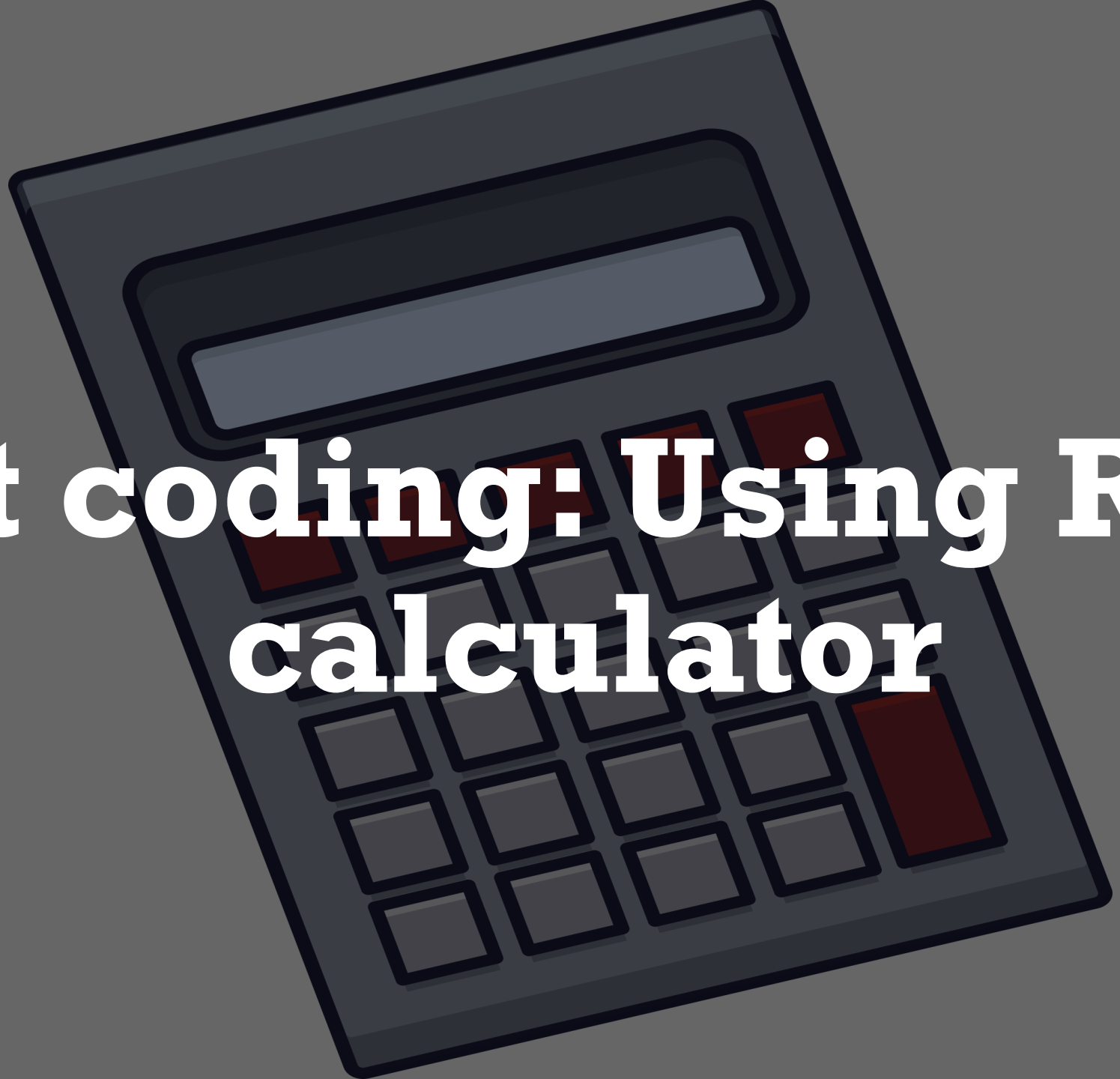
```
##### Header 2 #####
```

Sections have been created on the provided script for your convenience.





Start coding: Using R as a calculator



Code basics: R as a calculator

Addition

```
> 5 + 4
[1] 9
```

Subtraction

```
> 10 - 9
[1] 1
```

Multiplication

```
> 3 * 2
[1] 6
```

Division

```
> 20 / 4
[1] 5
```

Exponent

```
> 2^3
[1] 8
```

Regular sequence

```
> 1:3
[1] 1 2 3
```

Generates a sequence in steps of 1

Combine operators together for more complex calculations following the order of operation (BEDMAS)

```
> 2 + 16 * (24 - 56)
[1] -510
```

Reading the console

Command prompt → >

Command prompt indicates the next available line for a new command or the beginning of a completed command. It is not part of the actual command.

Reading the console

Command prompt → `> 5 + 4` ← Command

Command prompt indicates the next available line for a new command or the beginning of a completed command. It is not part of the actual command.

Command is the code to be executed.

Reading the console

Command prompt → `>` `5 + 4` ← Command

Output → `[1] 9`

Command prompt indicates the next available line for a new command or the beginning of a completed command. It is not part of the actual command.

Command is the code to be executed.

The **Output** can be read as "the 1st value of the command result is 9"

```
> 11:45
 [1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
[20] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
```

"The 20th value of the command result is 30"

R will not execute unfinished commands

```
> 8 *
+
+
+ 2
[1] 16
```

Continued command prompt, not part of the command

If a command is unfinished, the command prompt will change from > to + until the command is finished.

Alternatively, press Esc to cancel the execution

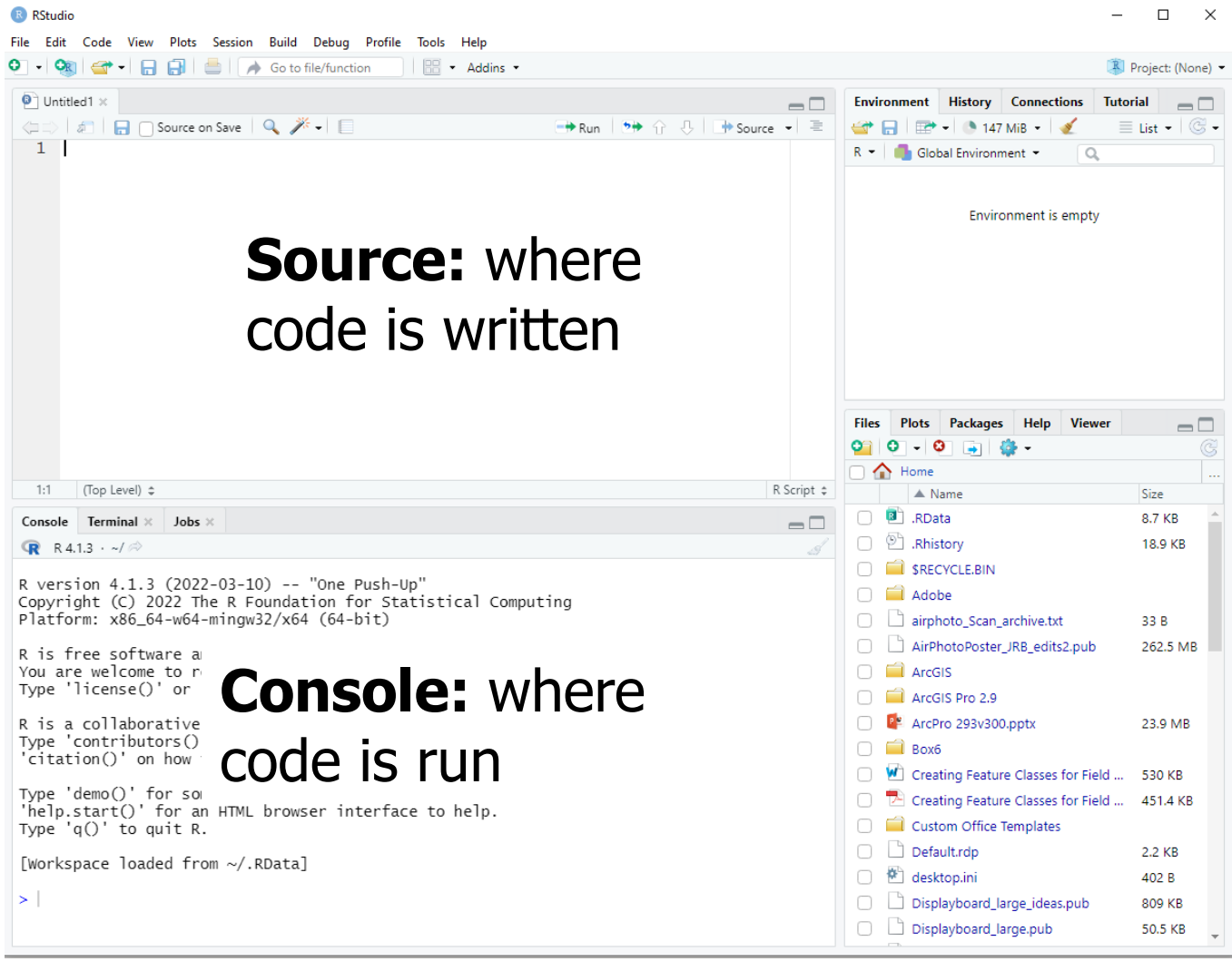
Command read as:
8 * 2

<u>Not</u>:
8 * + + + 2

Running code

- **Source pane:** use for projects or re-occurring analyses
  - Run code by pressing Ctrl + Enter or Alt + Enter on Windows
    - Cmd + Enter on Mac
  - Or press

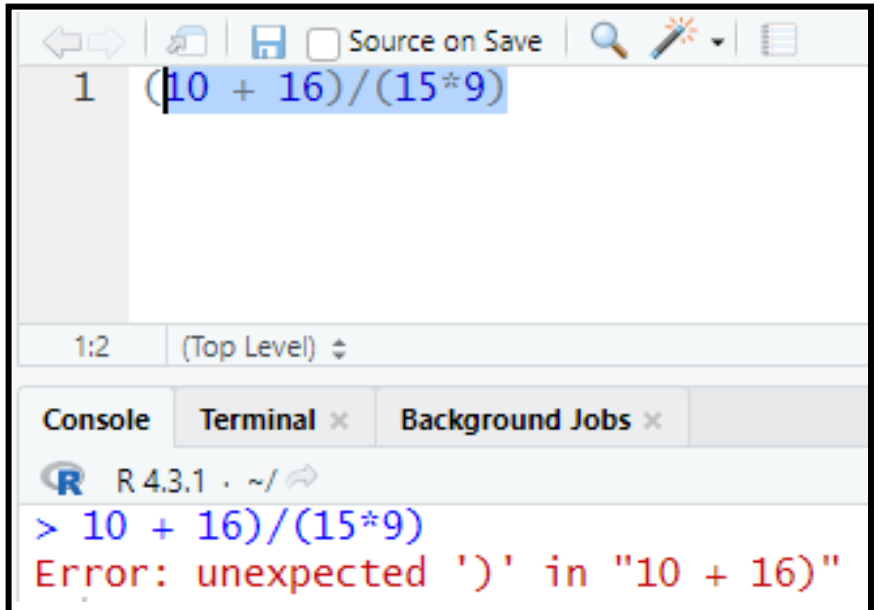
- **Console pane** use for quick calculation
  - Run code by pressing Enter



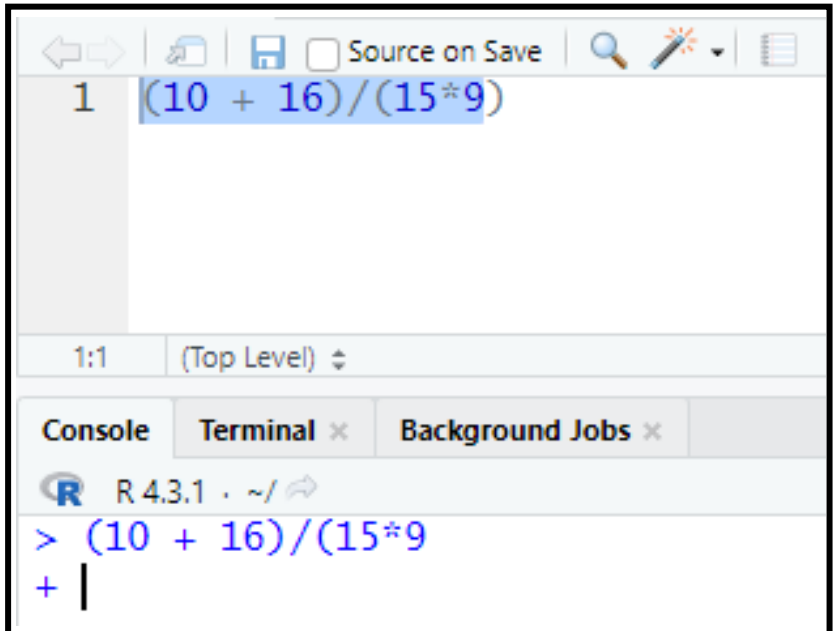
Caution: Highlighting to run

You can highlight part of a line, one line, or multiple lines.

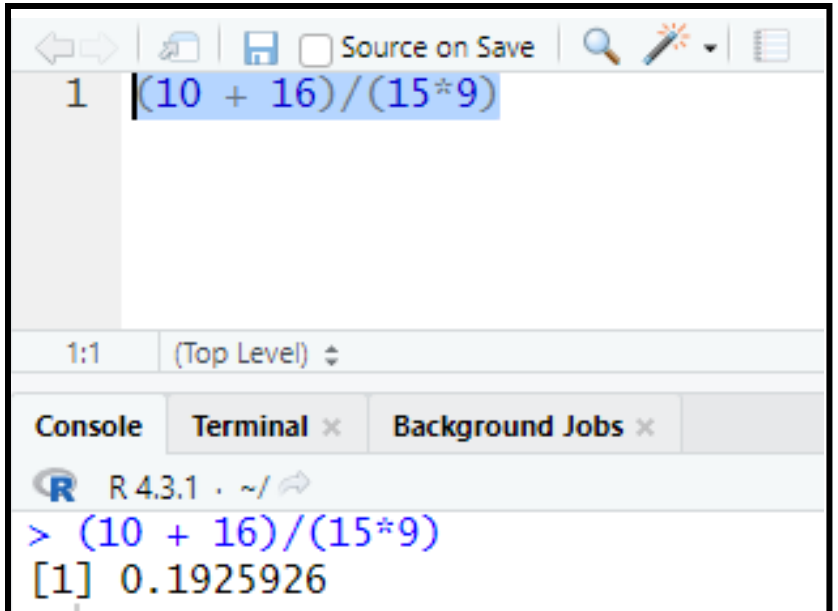
Be careful about what you are highlighting. Highlighting incomplete code may return an error or cause further issues.



The first bracket was missed, returning an Error



The last bracket was missed, R is waiting for more code. Running anything other than) will cause issues



The entire line was highlighted, ensuring proper execution

Source on Save Run Source

1

Let's Practice

Using R as a calculator

1:1 (Top Level) R Script

Blanks: Using R as a calculator

Line 15

```
3  *  5
```

Challenge #1

Run the following equation into R code and run it in the source pane or console.

$$5^2 + (8 * 5) + 3$$

Solution

```
>  (5^2)+(8*5)+3
[1]  68
```

Storing data as objects

```
> 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
7           4.6         3.4          1.4         0.3     setosa
8           5.0         3.4          1.5         0.2     setosa
9           4.4         2.9          1.4         0.2     setosa
10          4.9         3.1          1.5         0.1     setosa
11          5.4         3.7          1.5         0.2     setosa
12          4.8         3.4          1.6         0.2     setosa
13          4.8         3.0          1.4         0.1     setosa
14          4.3         3.0          1.1         0.1     setosa
15          5.8         4.0          1.2         0.2     setosa
16          5.7         4.4          1.5         0.4     setosa
17          5.4         3.9          1.3         0.4     setosa
18          5.1         3.5          1.4         0.3     setosa
19          5.7         3.8          1.7         0.3     setosa
20          5.1         3.8          1.5         0.3     setosa
21          5.4         3.4          1.7         0.2     setosa
22          5.1         3.7          1.5         0.4     setosa
23          4.6         3.6          1.0         0.2     setosa
24          5.1         3.3          1.7         0.5     setosa
25          4.8         3.4          1.9         0.2     setosa
26          5.0         3.0          1.6         0.2     setosa
27          5.0         3.4          1.6         0.4     setosa
28          5.2         3.5          1.5         0.2     setosa
29          5.2         3.4          1.4         0.2     setosa
30          4.7         3.2          1.6         0.2     setosa
```

```
> co2
         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug
1959 315.42 316.31 316.50 317.56 318.13 318.00 316.39 314.65
1960 316.27 316.81 317.42 318.87 319.87 319.43 318.01 315.74
1961 316.73 317.54 318.38 319.31 320.42 319.61 318.42 316.63
1962 317.78 318.40 319.53 320.42 320.85 320.45 319.45 317.25
1963 318.58 318.92 319.70 321.22 322.08 321.31 319.58 317.61
1964 319.41 320.07 320.74 321.40 322.06 321.73 320.27 318.54
1965 319.27 320.28 320.73 321.97 322.00 321.71 321.05 318.71
```

```
> airmiles[1:length(airmiles)]
 [1]   412   480   683  1052  1385  1418  1634  2178  3362  5948  6109  5981  6753  8003 10566 12528 14760 16769 19819 22362 25340 25343 29269 3
```

```
> cars
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
33    18   56
34    18   76
35    18   84
36    19   36
37    19   46
38    19   68
```

```
> Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
```

```
> volcano
      [,1] [,2] [,3] [,4]
 [1,]  100  100  101  101
 [2,]  101  101  102  102
 [3,]  102  102  103  103
 [4,]  103  103  104  104
 [5,]  104  104  105  105
 [6,]  105  105  105  106
 [7,]  105  106  106  107
 [8,]  106  107  107  108
 [9,]  107  108  108  109
[10,]  108  109  109  110
[11,]  109  110  110  111
[12,]  110  110  111  113
[13,]  110  111  113  115
[14,]  111  113  115  117
[15,]  114  115  117  117
[16,]  116  118  118  118
      [,36] [,37] [,38] [,
 [1,]   110   110   110
 [2,]   114   116   115
 [3,]   [illegible]   121   121
 [4,]   [illegible]   127   127
 [5,]   131   132   132
 [6,]   135   136   136
 [7,]   140   140   140
 [8,]   144   144   144
 [9,]   149   148   148
[10,]   158   154   151
[11,]   164   160   157
[12,]   168   166   162
[13,]   172   171   169
[14,]   176   176   174
[15,]   183   180   178
[16,]   184   184   181
```

Object assignment

R is an **object-based** environment; data is assigned to names with the assignment operator (`<-`) .

`Object_name <- value(s)`

Object name gets value(s)

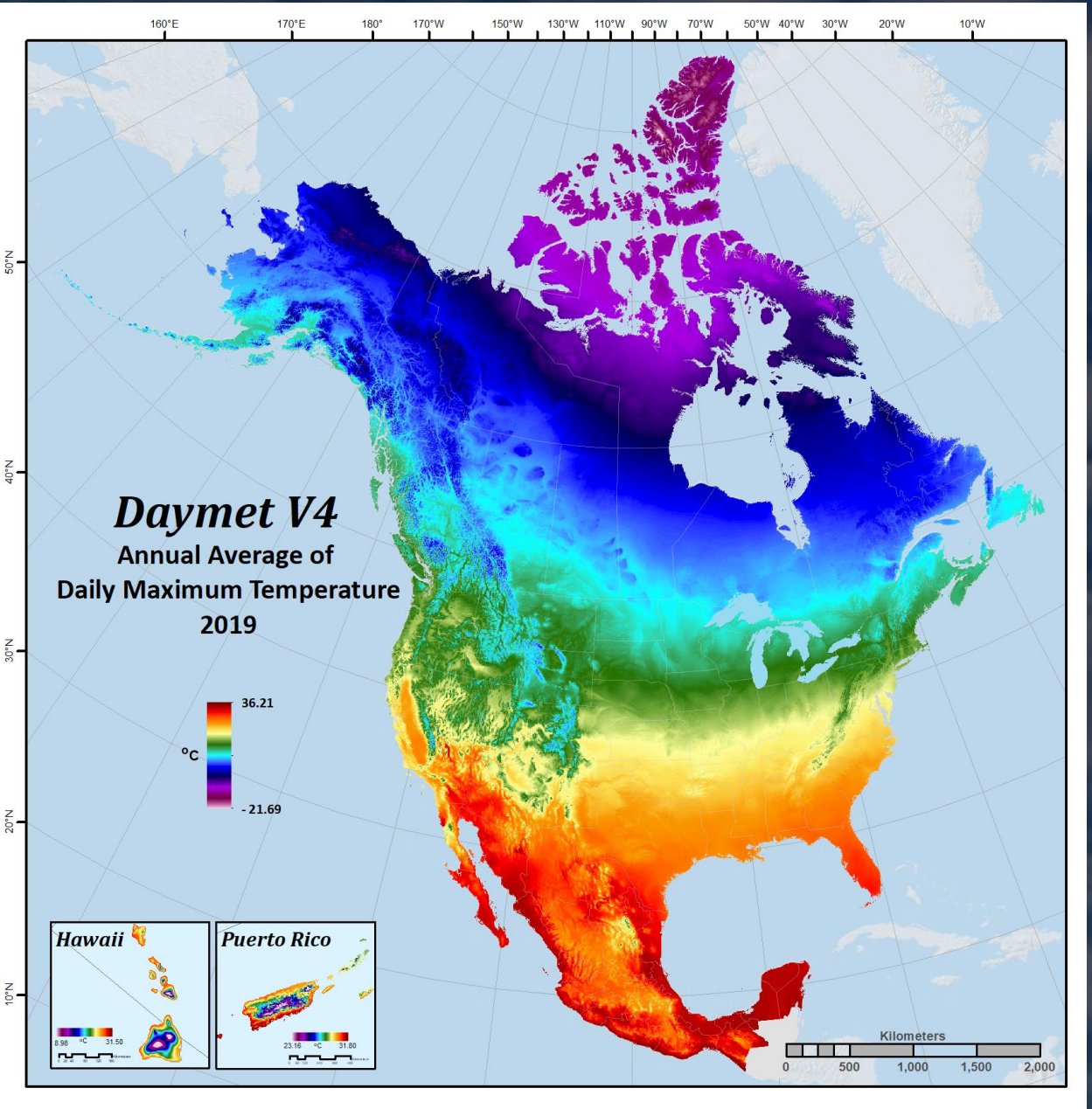
```
> Weather <- "Sunny"
> Weather
[1] "Sunny"
```

Tip: Use `Alt` + `-` (`Option` + `-` on Mac) to inset an arrow (`<-`)

Daymet: Daily weather data

We will explore objects by applying daily weather data for Peterborough, ON from [Daymet](#).

Daymet provides daily weather data on a 1 km x 1 km grid of North America. Weather parameters generated include daily surfaces of minimum and maximum temperature, precipitation, vapor pressure, radiation, snow water equivalent, and day length.



Daymet: Daily Surface Weather Data on a 1-km Grid for North America, Version 4
R1 https://doi.org/10.3334/ORNLDAAC/2129

Major data types

Numeric (`<num>`): Numbers with decimal values

- Also referred to as **Double** (`<dbl>`)

Integer (`<int>`): Numbers without decimal values

```
> 5.1
[1] 5.1
> 4L
[1] 4
```

Character (`<chr>`): String (text) values, always entered between quotations, `' '` or `" "`

- **Factor** (`<Factor>`) categories for statistical analysis
- **Date** (`<Date>`) calendar and time values

```
> "Sunny"
[1] "Sunny"
```

Logical (`<logi>`): Boolean values (TRUE or FALSE), always all capitals without quotations

Missing (`<logi>`): Not available / Missing values

```
> TRUE
[1] TRUE
> NA
[1] NA
```

What's in a name?

Short and **descriptive** names are recommended

- Use common abbreviations when possible

```
> C <- 4
```

Not descriptive

```
> MaxTemp_PTBO <- 36
> MaxTemp_PTBO
[1] 36
```

Clear descriptive name

camelCase

```
> MaxTempPTBO <- 36
```

kebab-case

Source: [Junior to Export]

snake_case

```
> Max-Temp-PTBO <- 36
```

```
> Max_Temp_PTBO <- 36
```

What's in a name?

Short and descriptive names are recommended

- Use common abbreviations when possible

Names must:

- Start with a letter
- Contain only letters, numbers, underscores (`_`), and periods (`.`)
- Not be one of the reserved keywords (e.g., `if`, `else`, `TRUE`, `FALSE`)

```
> 2019_Precip <- 727.1
Error: unexpected
input in "2019_"
```

Starts with a number

```
> 2019@Precip <- 727.1
Error in 2019@Precip : no
applicable method for `@` …
```

Unaccepted special character (@)

```
> if <- "a"
Error: unexpected
assignment in "if <-"
```

Reserved keyword

Let's Practice

Storing data as objects

Blanks: Storing data as objects

Line 44 `Prcp_mm_Dec312022 <- 13.2`

Line 47 `Prcp_mm_Dec312022`

Line 57 `MeanTemp_C_Mar012023 <- -3.9`

Line 62 `n_Ptbo_weatherstations <- 13L`

Line 67 `Data_Source <- "Daymet"`

Line 72 `Precip_Mar112022 <- TRUE`

Challenge #2

Create the following three objects:

- An object named `x` with a value of 5
- An object named `b` with a value of 8
- An object named `c` with a value of 3

Then, use the objects to run the provided equation.

Solution

```
> x <- 5
> b <- 8
> c <- 3
>(x^2) + (b * x) + c
[1] 68
```

Functions and Packages

Functions

Functions allows for execution of simple to complex tasks.

```
function_name(arg1 = val1, arg2 = val2, ...)
```

```
> abs(-1)
[1] 1
```

Absolute value

```
> substr("GCCTAG", start=1, stop=3)
[1] "GCC"
```

Subset character value(s) from specified start to finish characters

Each function usually consists of one or more data inputs and/or arguments. Inputs and/or arguments can be optional with a set default value.

Not sure which function to use?

Google

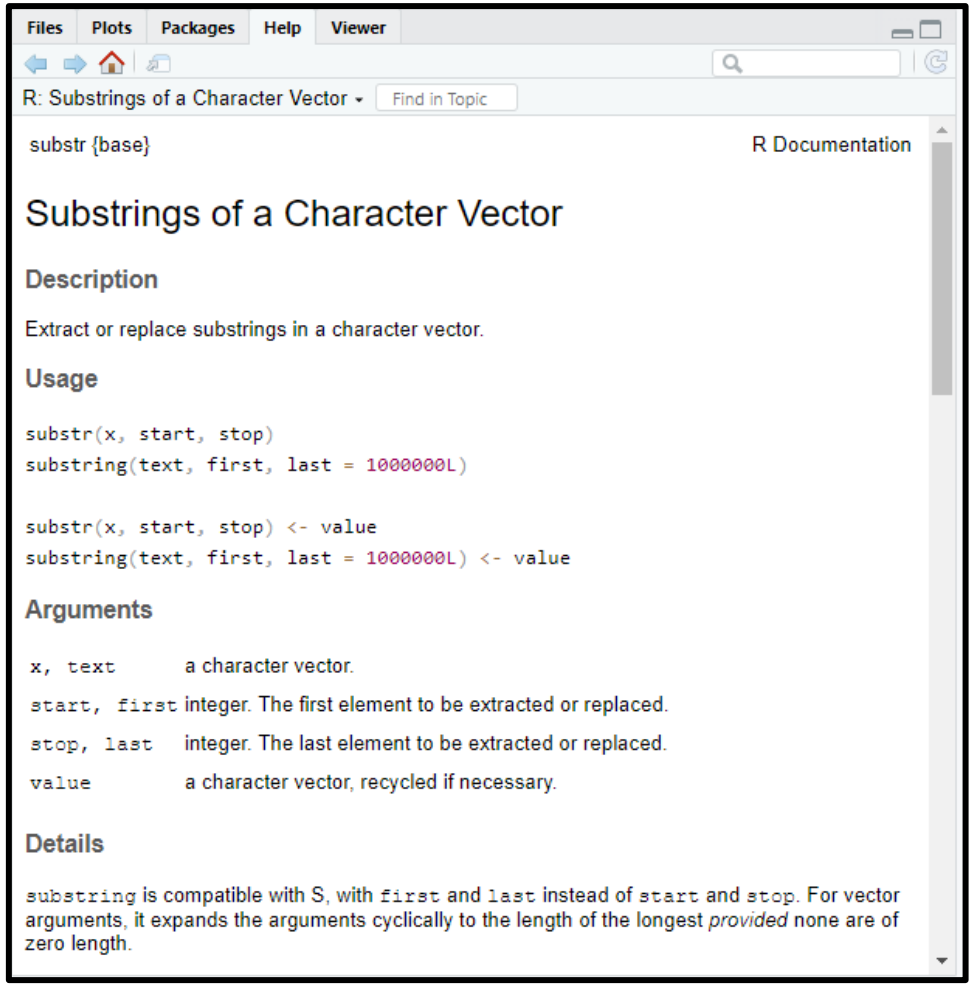
madgichelp@trentu.ca

Functions: Getting help

Get details on a function's use with `?` or `??`:

- `?` searches documentation within loaded packages

```
> ?substr
```



- `??` searches all R documentation

```
> ??gather
```

Opens the documentation page in the **Help tab** of **Files/Plots/Packages/Help** panel

R Documentation from ? function

Files | Plots | Packages | Help | Viewer

R: Substrings of a Character Vector — Find in Topic

substr {base} — Where the function is from

R Documentation

Substrings of a Character Vector — Function name

Description

What the function is/does

Extract or replace substrings in a character vector.

Usage

How to use a function and similar functions

```
substr(x, start, stop)
substring(text, first, last = 1000000L)

substr(x, start, stop) <- value
substring(text, first, last = 1000000L) <- value
```

Arguments

Description of what function arguments

`x, text`	a character vector.
`start, first`	integer. The first element to be extracted or replaced.
`stop, last`	integer. The last element to be extracted or replaced.
`value`	a character vector, recycled if necessary.

Details

In-depth description of arguments

`substring` is compatible with S, with `first` and `last` instead of `start` and `stop`. For vector arguments, it expands the arguments cyclically to the length of the longest *provided* none are of zero length.

Files | Plots | Packages | Help | Viewer

R: Substrings of a Character Vector — Find in Topic

Value

What the function returns

For `substr`, a character vector of the same length and with the same attributes as `x` (after possible coercion).

For `substring`, a character vector of length the longest of the arguments. This will have names taken from `x` (if it has any after coercion, repeated as needed), and other attributes copied from `x` if it is the longest of the arguments).

For the replacement functions, a character vector of the same length as `x` or `text`, with `attributes` such as `names` preserved.

Elements of `x` or `text` with a declared encoding (see `Encoding`) will be returned with the same encoding.

Examples

Run examples

Examples of function use

```
substr("abcdef", 2, 4)
substring("abcdef", 1:6, 1:6)
## strsplit() is more efficient ...

substr(rep("abcdef", 4), 1:4, 4:5)
x <- c("asfef", "qwerty", "yuiop[", "b", "stuff.blah.yech")
substr(x, 2, 5)
substring(x, 2, 4:6)

X <- x
names(X) <- LETTERS[seq_along(x)]
comment(X) <- noquote("is a named vector")
```

Argument name vs. position

Function arguments can be provided by name…

```
> substr(x="GCCTAG", start=1, stop=3)
[1] "GCC"
```

… or position…

```
> substr("GCCTAG", 1, 3)
[1] "GCC"
```

… or both

```
> substr("GCCTAG", start=1,stop=3)
[1] "GCC"
```

All return the same results, **if** arguments defined properly.

```
> substr(3, 1, "GCCTAG")
[1]NA
```

When in doubt, define arguments by name.

Source on Save Run Source

1

Let's Practice

Functions

1:1 (Top Level) R Script

Blanks: Functions

Line 117
```
abs(-3)
```

Line 119
```
toupper("Hello world")
```

Line 128
```
abs(MeanTemp_C_Mar012023)
```

Line 146
```
?round
```

Line 159
```
round(MeanAnnualTemp_degC_2022, digits = 2)
```

Line 183
```
rep(replicate_this, times = 3
```

Line 186
```
rep(replicate_this, each = 3
```

Challenge #3

From the information provided on the `seq()` help page, which command(s) would return the following values:

```
[1] 3 7 11 15 19 23 27
```

A)
```
> seq(3, 4, 7)
```

B)
```
> seq(from = 3, by = 4, length.out = 7)
```

C)
```
> seq(from = 3, to = 30, by = 4)
```

D)
```
> seq(from = 3, to = 32, by = 4)
```

R packages

Most functions in R are stored within **packages**.

Packages may be specialized to a:

- specific task (e.g. data wrangling)
- field (e.g. biological assays)
- data type (e.g. dates)

Other packages contain miscellaneous functions based on common evaluations (e.g. mode is not a base R function).

Install and Load Packages

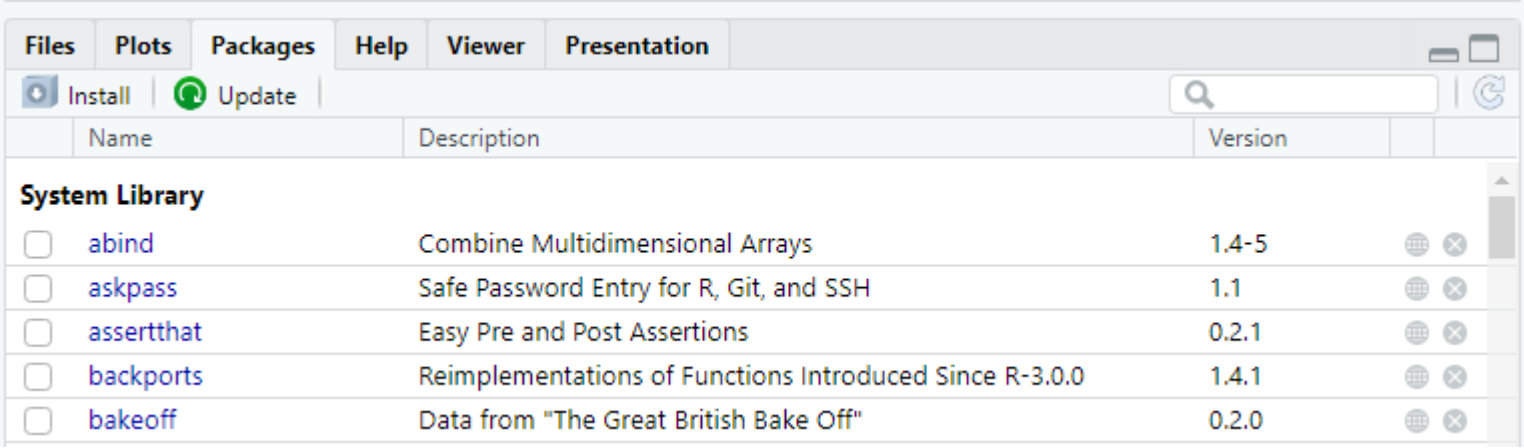
Packages may be installed or loaded by:

Code

```
> install.packages("package")
> library(package)
```

OR

RStudio Pane



Both methods provide the same result

Install and Load Packages by code

Packages must be installed ONCE per computer.

```
> install.packages("dplyr")
```

R packages are then loaded into the session with `library()`.
Packages must be loaded ONCE per session.

```
> library("dplyr")
```

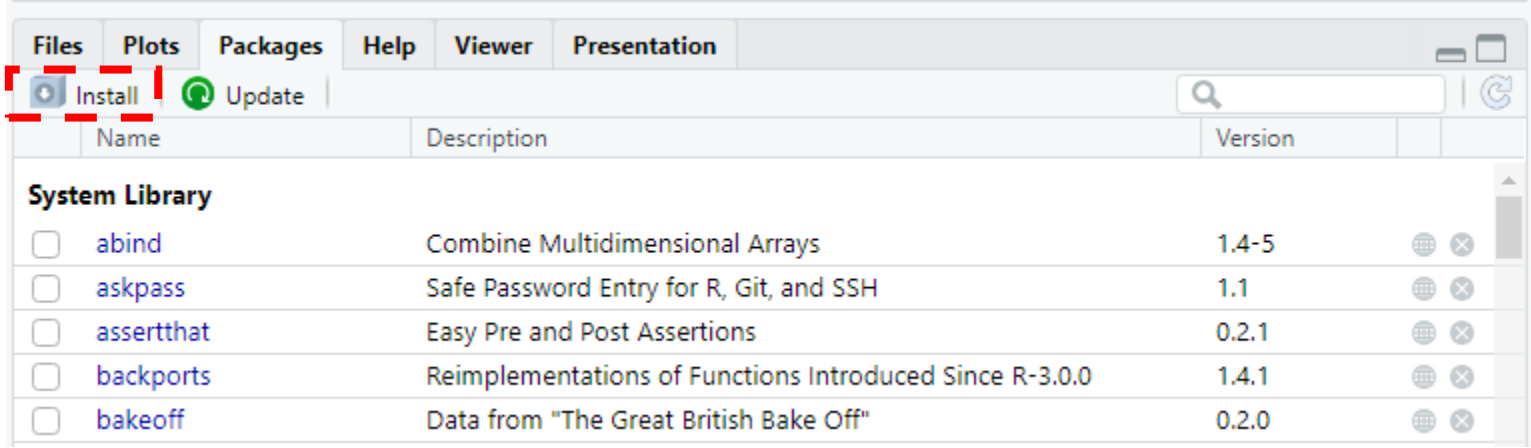
=

```
> library(dplyr)
```

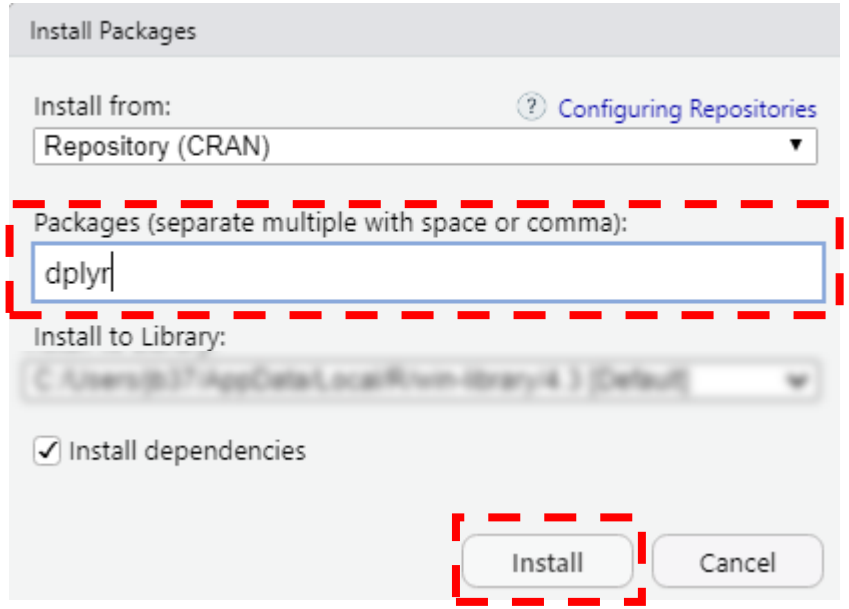
Install Packages in the Package pane

Packages may also be installed by the **Project panel**

Select **Package > Install**

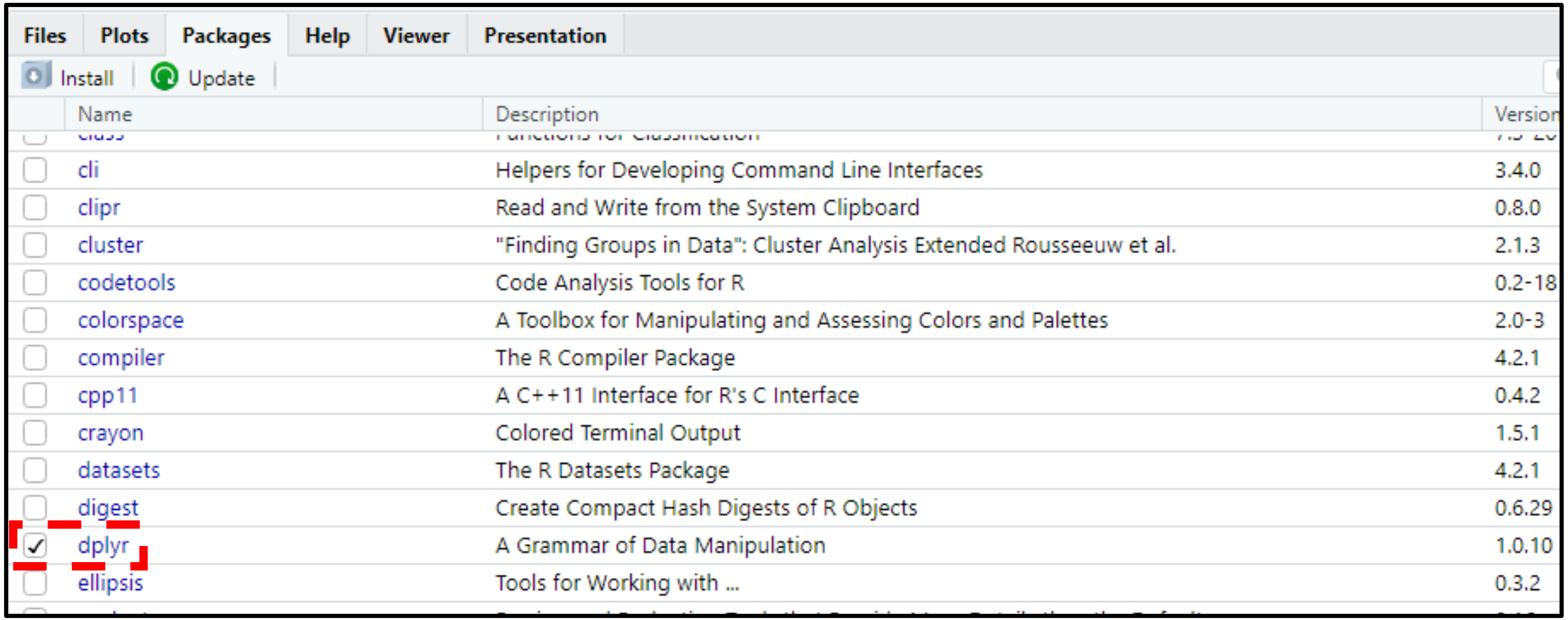


Search and install the desired package.



Load Packages in the Package pane

Load packages by checking the corresponding box in the Project panel.




```
> library(dplyr)
```

R automatically writes the corresponding code needed to install and load packages through the Project pane

Getting help: Packages

The Comprehensive R Archive Network (CRAN) Documentation

- Provides a description of the package and functions
- Full R documentation of all associated functions
- Available for all R packages

```
help(package = "stringr")
```

String Operations

Documentation for package 'stringr' version 1.4.1

- DESCRIPTION file.
- User guides, package vignettes and other documentation.

Help Pages

boundary	Control matching behaviour with modifier functions.
case	Convert case of a string.
coll	Control matching behaviour with modifier functions.
fixed	Control matching behaviour with modifier functions.
fruit	Sample character vectors for practicing string manipulations.
invert_match	Switch location of matches to location of non-matches.
modifiers	Control matching behaviour with modifier functions.
regex	Control matching behaviour with modifier functions.
sentences	Sample character vectors for practicing string manipulations.
stringr-data	Sample character vectors for practicing string manipulations.
str_c	Join multiple strings into a single string.
str_conv	Specify the encoding of a string.
str_count	Count the number of matches in a string.
str_detect	Detect the presence or absence of a pattern in a string.

Getting help: Packages

Vignettes

- Tutorials and example work-flow for the package
- Not available for all packages

```
browseVignettes("stringr")
```

Introduction to stringr

There are four main families of functions in stringr:

1. Character manipulation: these functions allow you to manipulate individual characters within the strings in character vectors.
2. Whitespace tools to add, remove, and manipulate whitespace.
3. Locale sensitive operations whose operations will vary from locale to locale.
4. Pattern matching functions. These recognise four engines of pattern description. The most common is regular expressions, but there are three other tools.

Getting and setting individual characters

You can get the length of the string with `str_length()`:

```
str_length("abc")
#> [1] 3
```

This is now equivalent to the base R function `nchar()`. Previously it was needed to work around issues with `nchar()` such as the fact that it returned 2 for `nchar(NA)`. This has been fixed as of R 3.3.0, so it is no longer so important.

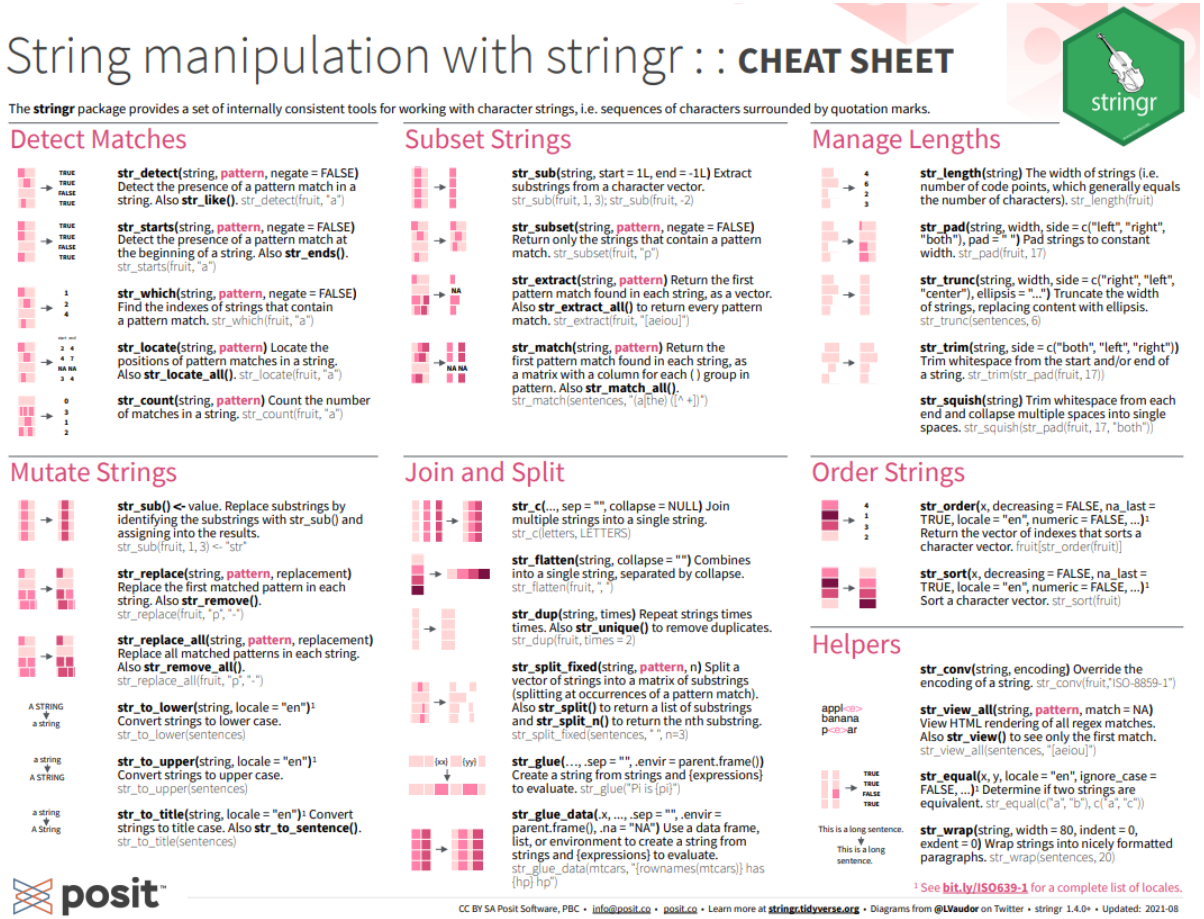
You can access individual character using `str_sub()`. It takes three arguments: a character vector, a `start` position and an `end` position. Either position can either be a positive integer, which counts from the left, or a negative integer which counts from the right. The positions are inclusive, and if longer than the string, will be silently truncated.

```
x <- c("abcdef", "ghifjk")

# The 3rd letter
str_sub(x, 3, 3)
#> [1] "c" "i"

# The 2nd to 2nd-to-last character
str_sub(x, 2, -2)
#> [1] "bcde" "hifj"
```

Getting help: Packages

Package specific cheat sheets

- The most popular R packages have cheat sheets
- Be sure to check for a cheat sheet when using a package!



String manipulation with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

str_detect(string, pattern, negate = FALSE) Detect the presence of a pattern match in a string. Also **str_like()**. str_detect(fruit, "a")

str_starts(string, pattern, negate = FALSE) Detect the presence of a pattern match at the beginning of a string. Also **str_ends()**. str_starts(fruit, "a")

str_which(string, pattern, negate = FALSE) Find the indexes of strings that contain a pattern match. str_which(fruit, "a")

str_locate(string, pattern) Locate the positions of pattern matches in a string. Also **str_locate_all()**. str_locate(fruit, "a")

str_count(string, pattern) Count the number of matches in a string. str_count(fruit, "a")

Subset Strings

str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. str_sub(fruit, 1, 3); str_sub(fruit, -2)

str_subset(string, pattern, negate = FALSE) Return only the strings that contain a pattern match. str_subset(fruit, "p")

str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also **str_extract_all()** to return every pattern match. str_extract(fruit, "[aeiou]")

str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all()**. str_match(sentences, "(a|the) ([^ +])")

Manage Lengths

str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). str_length(fruit)

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. str_pad(fruit, 17)

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. str_trunc(sentences, 6)

str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. str_trim(str_pad(fruit, 17))

str_squish(string) Trim whitespace from each end and collapse multiple spaces into single spaces. str_squish(str_pad(fruit, 17, "both"))

Mutate Strings

str_sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. str_sub(fruit, 1, 3) <- "str"

str_replace(string, pattern, replacement) Replace the first matched pattern in each string. Also **str_remove()**. str_replace(fruit, "p", "-")

str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. Also **str_remove_all()**. str_replace_all(fruit, "p", "-")

str_to_lower(string, locale = "en")[1] Convert strings to lower case. str_to_lower(sentences)

str_to_upper(string, locale = "en")[1] Convert strings to upper case. str_to_upper(sentences)

str_to_title(string, locale = "en")[1] Convert strings to title case. Also **str_to_sentence()**. str_to_title(sentences)

Join and Split

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. str_c(letters, LETTERS)

str_flatten(string, collapse = "") Combines into a single string, separated by collapse. str_flatten(fruit, ", ")

str_dup(string, times) Repeat strings times times. Also **str_unique()** to remove duplicates. str_dup(fruit, times = 2)

str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split()** to return a list of substrings and **str_split_n()** to return the nth substring. str_split_fixed(sentences, " ", n=3)

str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. str_glue("Pi is {pi}")

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. fruit[str_order(fruit)]

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. str_sort(fruit)

Helpers

str_conv(string, encoding) Override the encoding of a string. str_conv(fruit,"ISO-8859-1")

str_view_all(string, pattern, match = NA) View HTML rendering of all regex matches. Also **str_view()** to see only the first match. str_view_all(sentences, "[aeiou]")

str_equal(x, y, locale = "en", ignore_case = FALSE, ...)[1] Determine if two strings are equivalent. str_equal(c("a", "b"), c("a", "c"))

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. str_wrap(sentences, 20)

[1] See bit.ly/ISO639-1 for a complete list of locales.

posit

CC BY SA Posit Software, PBC • info@posit.co • posit.co • Learn more at stringr.tidyverse.org • Diagrams from @LVaudor on Twitter • stringr 1.4.0+ • Updated: 2021-08

Available from:

posit™

Source on Save Run Source

1

Let's Practice

Packages

1:1 (Top Level) R Script

Blanks: Packages

Line 226
```
install.packages("dplyr")
```

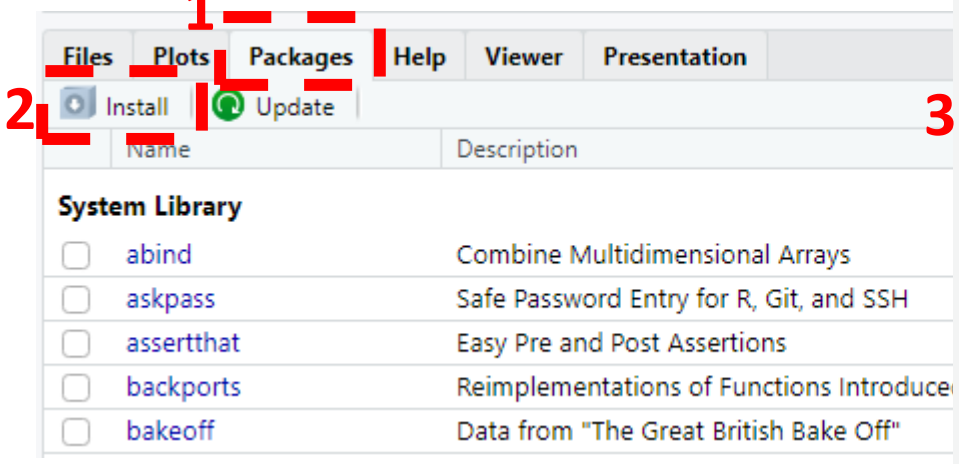
Line 228
```
library(dplyr)
```

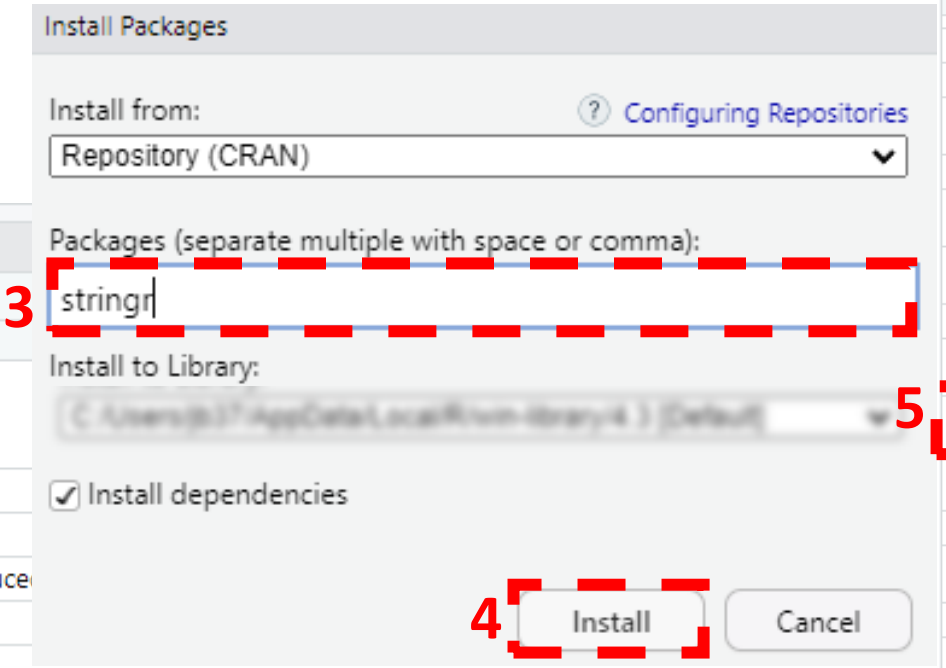
Install and Load Packages

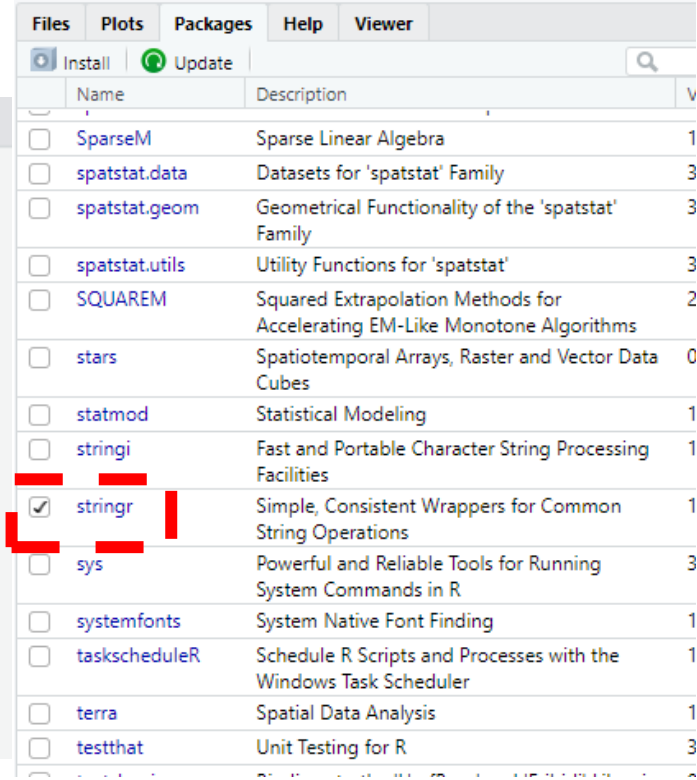
Code

```
> install.packages("dplyr")
> library(dplyr)
```

RStudio package pane







Note: Both methods provide the same result

Challenge #4

Install and load the package `beepr`
(by script or Project pane)

Solution (by code)

```
> install.packages("beepr")
> library("beepr") #or library(beepr)
> beep()#creates a "ping" sound
```

Installing beepr by the package pane also acceptable

Data structures

```
> 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
7           4.6         3.4          1.4         0.3     setosa
8           5.0         3.4          1.5         0.2     setosa
9           4.4         2.9          1.4         0.2     setosa
10          4.9         3.1          1.5         0.1     setosa
11          5.4         3.7          1.5         0.2     setosa
12          4.8         3.4          1.6         0.2     setosa
13          4.8         3.0          1.4         0.1     setosa
14          4.3         3.0          1.1         0.1     setosa
15          5.8         4.0          1.2         0.2     setosa
16          5.7         4.4          1.5         0.4     setosa
17          5.4         3.9          1.3         0.4     setosa
18          5.1         3.5          1.4         0.3     setosa
19          5.7         3.8          1.7         0.3     setosa
20          5.1         3.8          1.5         0.3     setosa
21          5.4         3.4          1.7         0.2     setosa
22          5.1         3.7          1.5         0.4     setosa
23          4.6         3.6          1.0         0.2     setosa
24          5.1         3.3          1.7         0.5     setosa
25          4.8         3.4          1.9         0.2     setosa
26          5.0         3.0          1.6         0.2     setosa
27          5.0         3.4          1.6         0.4     setosa
28          5.2         3.5          1.5         0.2     setosa
29          5.2         3.4          1.4         0.2     setosa
30          4.7         3.2          1.6         0.2     setosa
```

```
> cars
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
33    18   56
34    18   76
35    18   84
36    19   36
37    19   46
38    19   68
```

```
> Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

```
> volcano
      [,1] [,2] [,3] [,4]
 [1,]  100  100  101  101
 [2,]  101  101  102  102
 [3,]  102  102  103  103
 [4,]  103  103  104  104
 [5,]  104  104  105  105
 [6,]  105  105  105  106
 [7,]  105  106  106  107
 [8,]  106  107  107  108
 [9,]  107  108  108  109
[10,]  108  109  109  110
[11,]  109  110  110  111
[12,]  110  110  111  113
[13,]  110  111  113  115
[14,]  111  113  115  117
[15,]  114  115  117  117
[16,]  116  118  118  118
      [,36] [,37] [,38] [,
 [1,]   110   110   110
 [2,]   114   116   115
 [3,]   119   121   121
 [4,]   126   127   127
 [5,]   131   132   132
 [6,]   135   136   136
 [7,]   140   140   140
 [8,]   144   144   144
 [9,]   149   148   148
[10,]   158   154   151
[11,]   164   160   157
[12,]   168   166   162
[13,]   172   171   169
[14,]   176   176   174
[15,]   183   180   178
[16,]   184   184   181
```

```
> co2
         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug
1959 315.42 316.31 316.50 317.56 318.13 318.00 316.39 314.65
1960 316.27 316.81 317.42 318.87 319.87 319.43 318.01 315.74
1961 316.73 317.54 318.38 319.31 320.42 319.61 318.42 316.63
1962 317.78 318.40 319.53 320.42 320.85 320.45 319.45 317.25
1963 318.58 318.92 319.70 321.22 322.08 321.31 319.58 317.61
1964 319.41 320.07 320.74 321.40 322.06 321.73 320.27 318.54
1965 319.27 320.28 320.73 321.97 322.00 321.71 321.05 318.71
```

```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
```

```
> airmiles[1:length(airmiles)]
 [1]   412   480   683  1052  1385  1418  1634  2178  3362  5948  6109  5981  6753  8003 10566 12528 14760 16769 19819 22362 25340 25343 29269 3
```

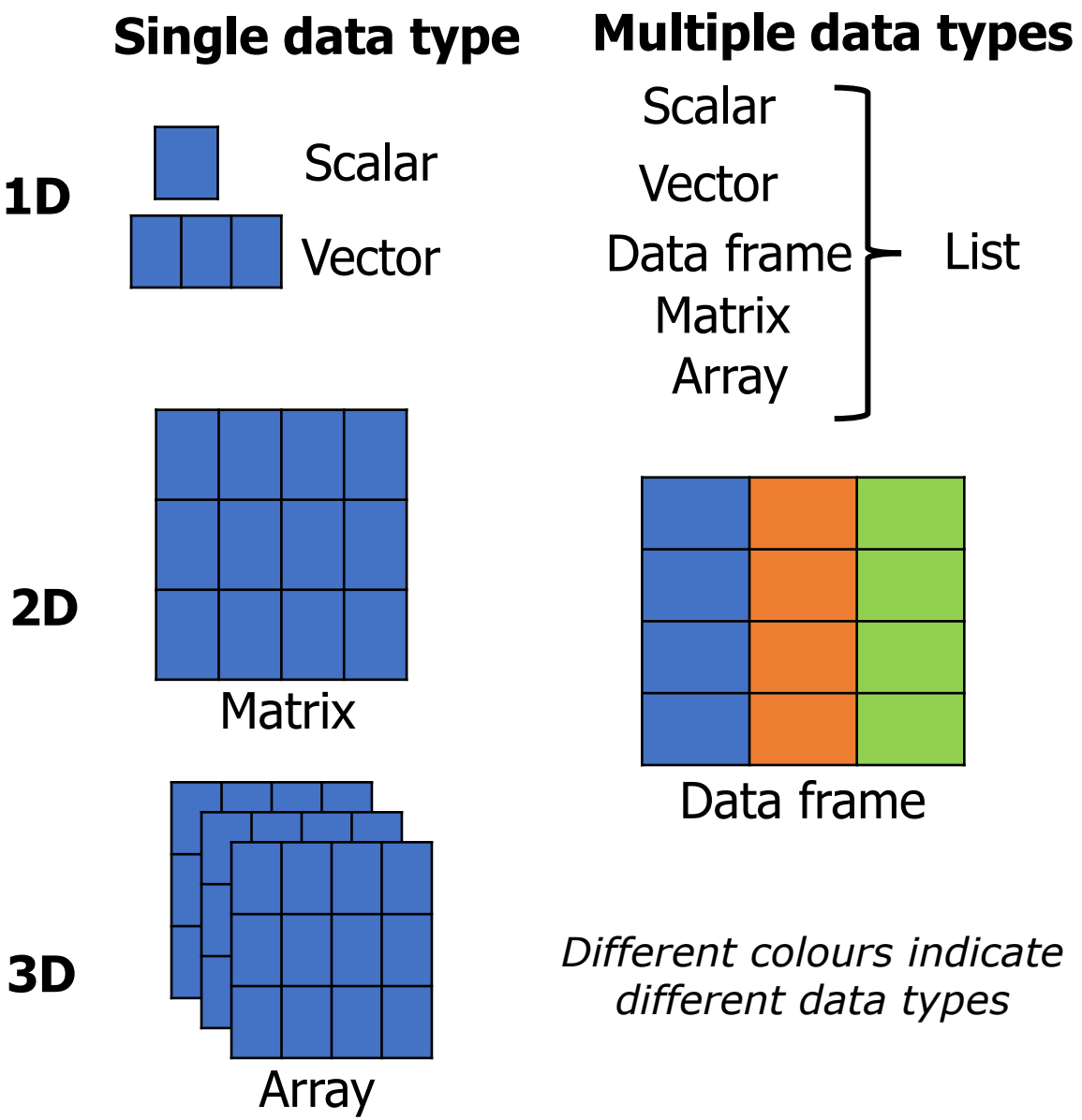
Data Structures

So far, we have only seen **scalar** values, an object with a single value.

Rarely will one work exclusively with scalars. Rather data is compiled into a structure.

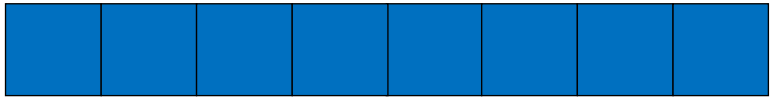
Each data structure is required for different functions, analyses, and visualizations.

Vectors and **Data frames** are most common.



Vectors

- The basic data structure, a **one-dimensional** structure containing elements of the **same data type**.



- Building blocks of all other data structures

- Used by most basic functions

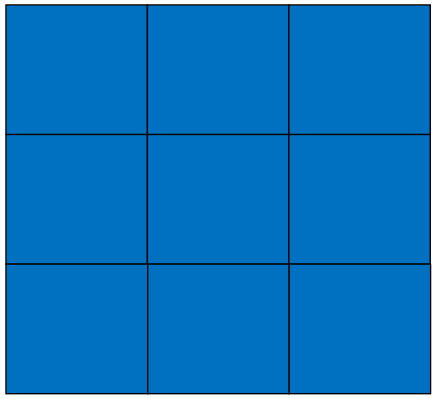
Vectors are created with the combine function, `c()`.

Place the vector elements separated by a comma between parentheses.

```
> AvgDailyAnnual_prcp_mm <- c(2.4, 2.6, 2)
> AvgDailyAnnual_prcp_mm
[1] 2.4 2.6 2.0
> mean(AvgDailyAnnual_prcp_mm)
[1] 2.333333
```

Matrices and Arrays

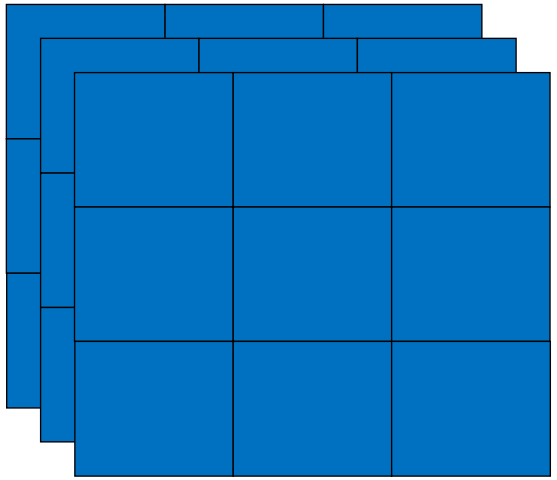
Matrix: A two-dimensional (rows and columns) structure with elements of the same data type.



```
> temp_matrix
        2011  2012  2013
May 1   11.6   7.7  16
May 2    9.6   9.2  15.1
May 3    5.9  14.5  15.2
```

Array: A multi-dimensional structure with two or more matrices of the same size and data type

```
> weather_array
, , Temp
        2011  2012  2013
May 1   11.6   7.7  16
May 2    9.6   9.2  15.1
May 3    5.9  14.5  15.2

, , Precip
        2011  2012  2013
May 1    0.8   0.3   0
May 2    2.8   2.8   0
May 3    6.1  14.3   0
```

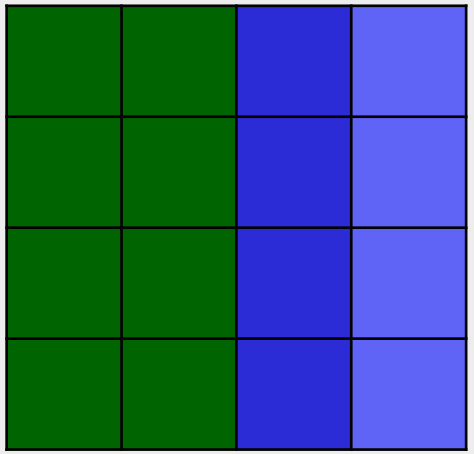


Matrices and arrays are commonly used as part of the mathematical machinery of statistics

Data Frame

Two dimensional (rows and columns) with elements of the same or different data types.

```
> Date <- c("2021/12/01", "2021/12/01", "2021/12/02, "2021/12/02")
> Location <-c("Ptbo A", "Trent U", "Ptbo A", "Trent U")
> Temp_C <- c(-0.5, -1, 4.3, 3.4)
> Precip_mm <- c(TRUE, TRUE, TRUE, TRUE)
> data.frame(Date, Station, Temp_C, Precip_mm)
        Date  Station Temp_C Precip_mm
1 2021/12/01  Ptbo A   -0.5    TRUE
2 2021/12/01  Trent U  -1      TRUE
3 2021/12/02  Ptbo A   4.3     TRUE
4 2021/12/02  Trent U  3.4     TRUE
```



- R's version of spreadsheets (aka Microsoft Excel)
- Most data imported into R from another source will default to a data frame
- **Essential for most statistical analysis and visualizations**

Lists

A data structure having components of same or mixed data structures and types

```
> weather_list
$Weather Description
[1] "Sunny"    "Cloudy" "Thunderstorm"

$Temperature
        2011  2012  2013
May 1   11.6   7.7  16
May 2    9.6   9.2  15.1
May 3    5.9  14.5  15.2

$Totals
        Date   Station Temp_C Precip_mm
1 2021/12/01   Ptbo A   -0.5    TRUE
2 2021/12/01   Trent U  -1      TRUE
3 2021/12/02   Ptbo A   4.3     TRUE
```

Vector

Matrix

Data frame

- Often used to compile the results of statistical output or related data of different structure

Inspect the data

Once data is created, you may inspect the data.
You can view the entire, parts, or attributes of a data structure.

Function	Description
`df`	Look at the entire data frame in the console
`head(df)`	Displays the first 6 rows or values
`tail(df)`	Displays the last 6 rows or values
`View(df)`	Look at the whole data frame in a new window or tab
`str(df)`	Displays the data frame structure (column names, data types, and first few values)
`summary(df)`	Descriptive statistics of each column

Let's Practice

Data structures

Blanks: Data structures (Vectors)

Lines 313-314

```
Months <- c("Jan", "Feb", "Mar", "Apr",
"May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec")
```

Line 328

```
head(MaxMonthtemp_C_Ptbo_2022, n = 6)
```

Line 331

```
tail(MaxMonthtemp_C_Ptbo_2022, n = 6)
```

Line 353

```
names(MaxMonthtemp_C_Ptbo_2022) <- Months
```

Line 370

```
?sd
```

Line 373

```
sd(MaxMonthtemp_C_Ptbo_2022, na.rm = TRUE)
```

Blanks: Data structures (Data frames)

Line 418

```
mean(Ptbo_Temps$MeanTemp_C, na.rm = TRUE)
```

All workshop materials are available on [MADGIC's Github](#).

- Unfilled workshop script
- Filled workshop script
- Completed annotated workshop script
- Filled slides with challenge answers
- Cheat sheet
- Example data

References

Daymet: Daily Surface Weather Data on a 1-km Grid for North America, Version 4 R1 https://doi.org/10.3334/ORNLDAAC/2129

Feather Icons v4.10.0 by Cole Bemis (https://feathericons.com/) licensed under MIT License

Font Awesome Icons by Dave Gandy (https://fontawesome.com/) licensed under Creative Commons Attribution 4.0 International