Lecture 7: Getting Started with R and Its Data Types

Chiheb Ben Hammouda

Mathematical Institute, Utrecht University

Python and R Course (WISB153)

June 3, 2025

Plan of Lecture 7

1. Introduction to R
   - Motivation
   - R and RStudio
   - R Markdown
2. R data types
   - Vectors and Matrices
   - Data frames
   - Lists
3. Practicalities and Further Resources

Python and R

Python

- General purpose language.
- Python is designed for readability and flexibility.
- Easy to learn.

R

- Statistical programming language.
- Easy to manipulate data (clean, import visualize and process data, ...).
- Has many functionalities for data and statistical analysis, and econometrics.

Transition: Python vs R Syntax

Python	**R**
x = numpy.array([1, 2, 3])	x <- c(1, 2, 3)
x[0]	x[1]
for i in range(3):	for (i in 1:3) { ... }
def fun(x):	fun <- function(x) { ... }
if x > 0:	if (x > 0) { ... }

⚠ Transition from Python to R: Book: An Introduction to R and Python For Data Analysis: A Side By Side Approach

On R

R is a programming language

- Published in 1993 (two years after Python's release)
- Often used by statisticians: you can do any kind of statistical analysis with it.
- With many ready-to-use packages: there are about 20,000 user-contributed packages that perform all kinds of specialized analyses.
- Very similar to Python, but is easier to use for
  - manipulating and processing data
  - visualising data
  - fitting statistical models

⚠ Unfortunately, R is not the most user-friendly software package. That is why the vast majority of users work with R through RStudio.

On RStudio

An integrated development environment (IDE) for R

Additional functionality

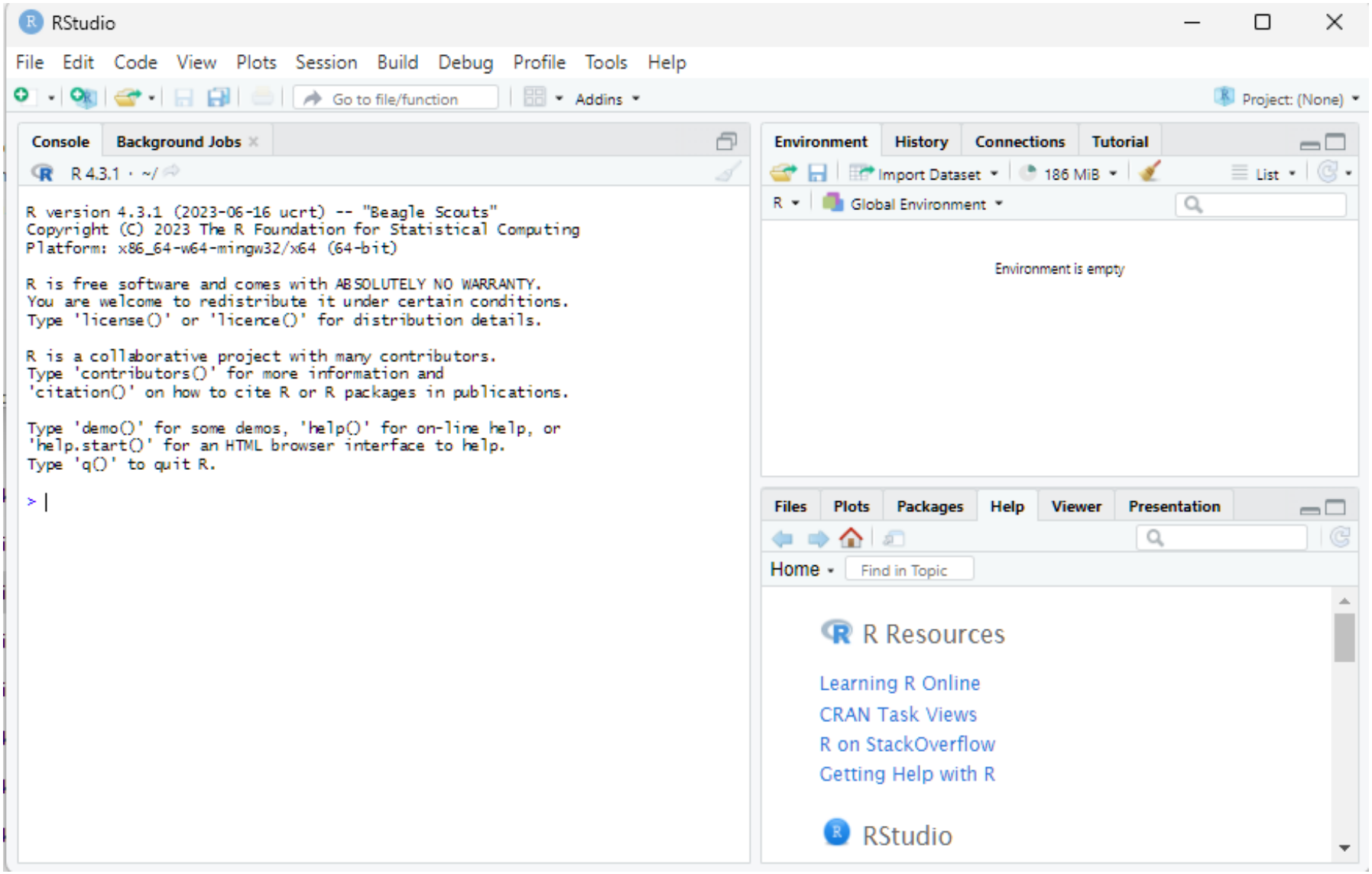
- editor with syntax highlighting
- integrated help functions
- debugging tools
- plotting tools
- and much more

Using RStudio makes working with R much easier!

To install R and RStudio:
https://rstudio-education.github.io/hopr/starting.html

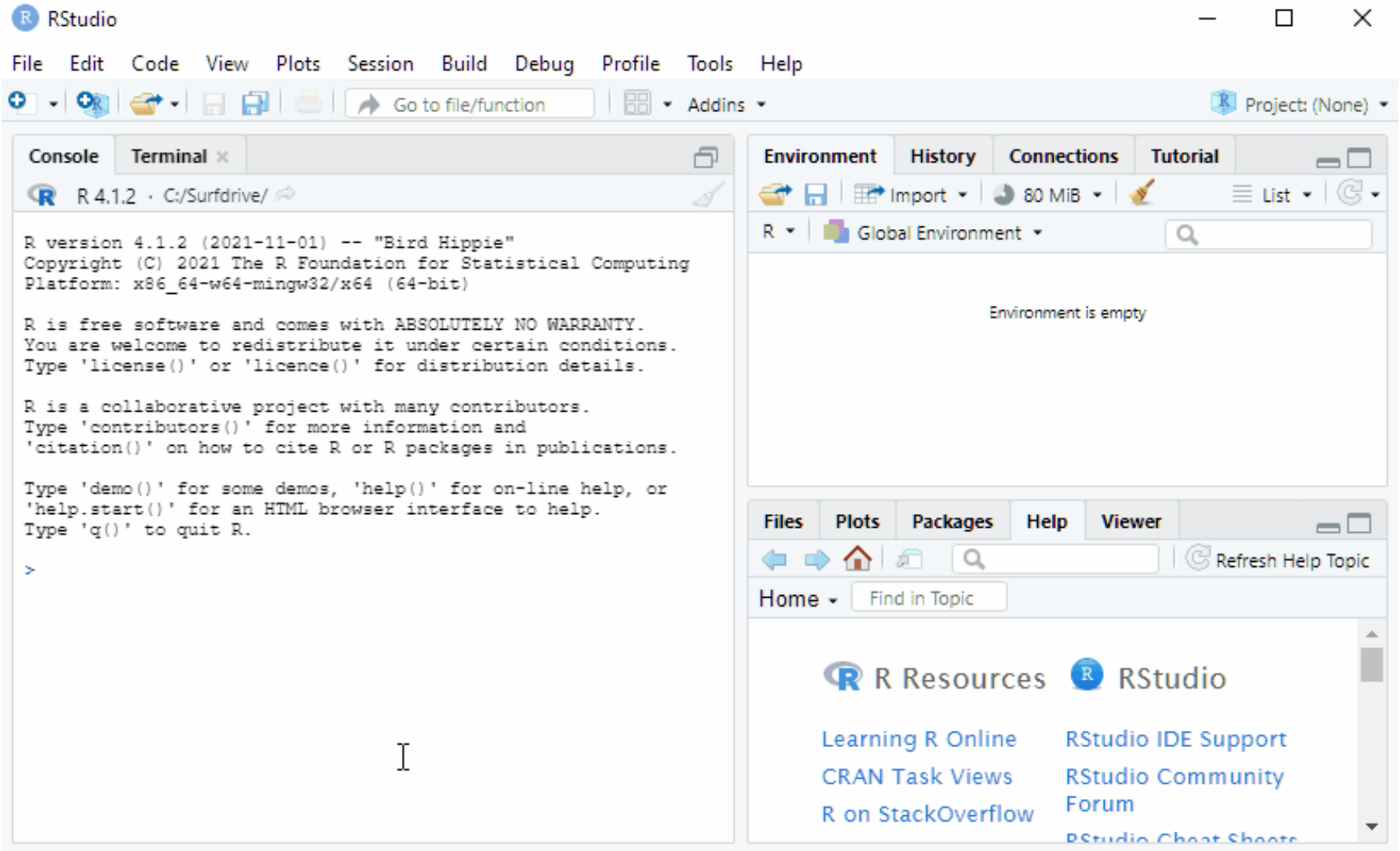
RStudio

Three important panels: Console, Environment and Help



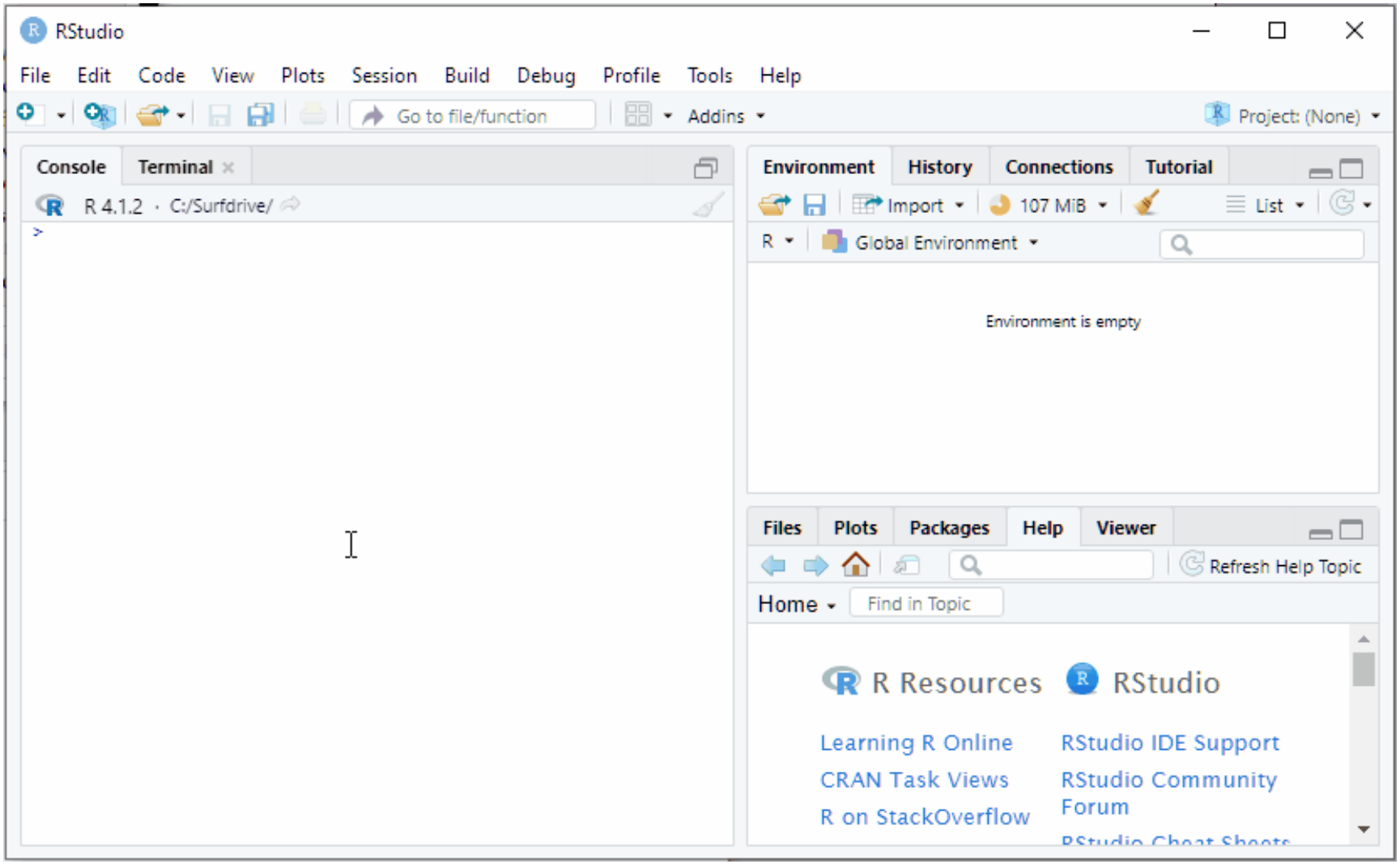
Console

Executes commands and shows output



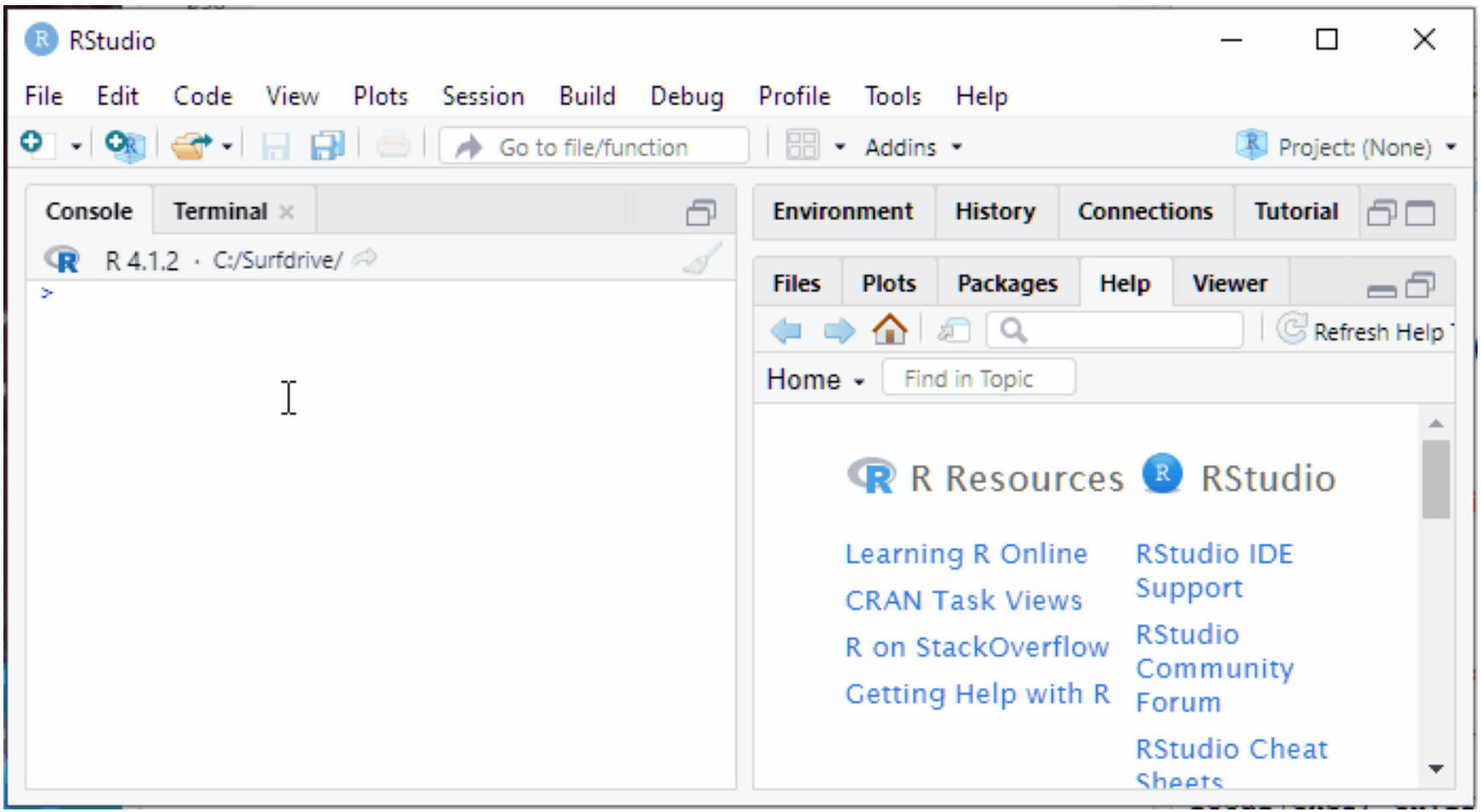
Global environment (workspace)

Stores user-defined objects and loaded R packages



Getting help

Various ways to get help on a function.



Base R Packages

- R packages are documented collections of functions (and data)
- Base R packages `stats`, `graphics`, ..., `base`:
  - automatically loaded in workspace
  - You can do almost everything with the base packages.

```
# Run the following command in the R console to check
installed packages
installed.packages()

# Sample output
                Package      Version      Priority
base            "base"       "4.0.2"      "base"
graphics        "graphics"   "4.0.2"      "base"
stats           "stats"      "4.0.2"      "base"
```

User-contributed packages

There are almost 20,000 user-contributed R packages

- they perform specialized operations
- need to be installed first (only once)
- need to be loaded in workspace at each new session

- Install and load packages commands in console
  - install a package with
    - run `install.packages("name package")`
  - Load a package with `library(<name package>)`
- Example:

```
# Install the markdown package
install.packages("markdown")

# Load the markdown package
library(markdown)
```

What is R Markdown?

RStudio facilitates the use of R Markdown:

- a file format for making (static and dynamic) documents that combine text with R code and results.
- file with extension `.Rmd`
- various output formats: useful for sharing, communicating and presenting statistical analyses, e.g.,
  - documents (html, word, pdf)
  - presentations (ioslides, slidy, powerpoint, revealjs, beamer)
  - articles (templates for various journals)
  - websites
- Install and load the packages `markdown` and `rmarkdown`.
- Use both text and code in one notebook. Then 'knit' the notebook to a PDF, HTML or Word file.

R Markdown Universe

- Books
- Journal articles
- Long form documents

Slide presentations

- ioslides
- slidy
- Beamer
- PowerPoint
- reveal js

Document + code

- HTML document
- PDF
- LaTeX
- Interactive notebooks
- Dashboards

- Websites/blogs

Rmd in Source Mode

demo.Rmd*

Knit on Save | Knit | Run

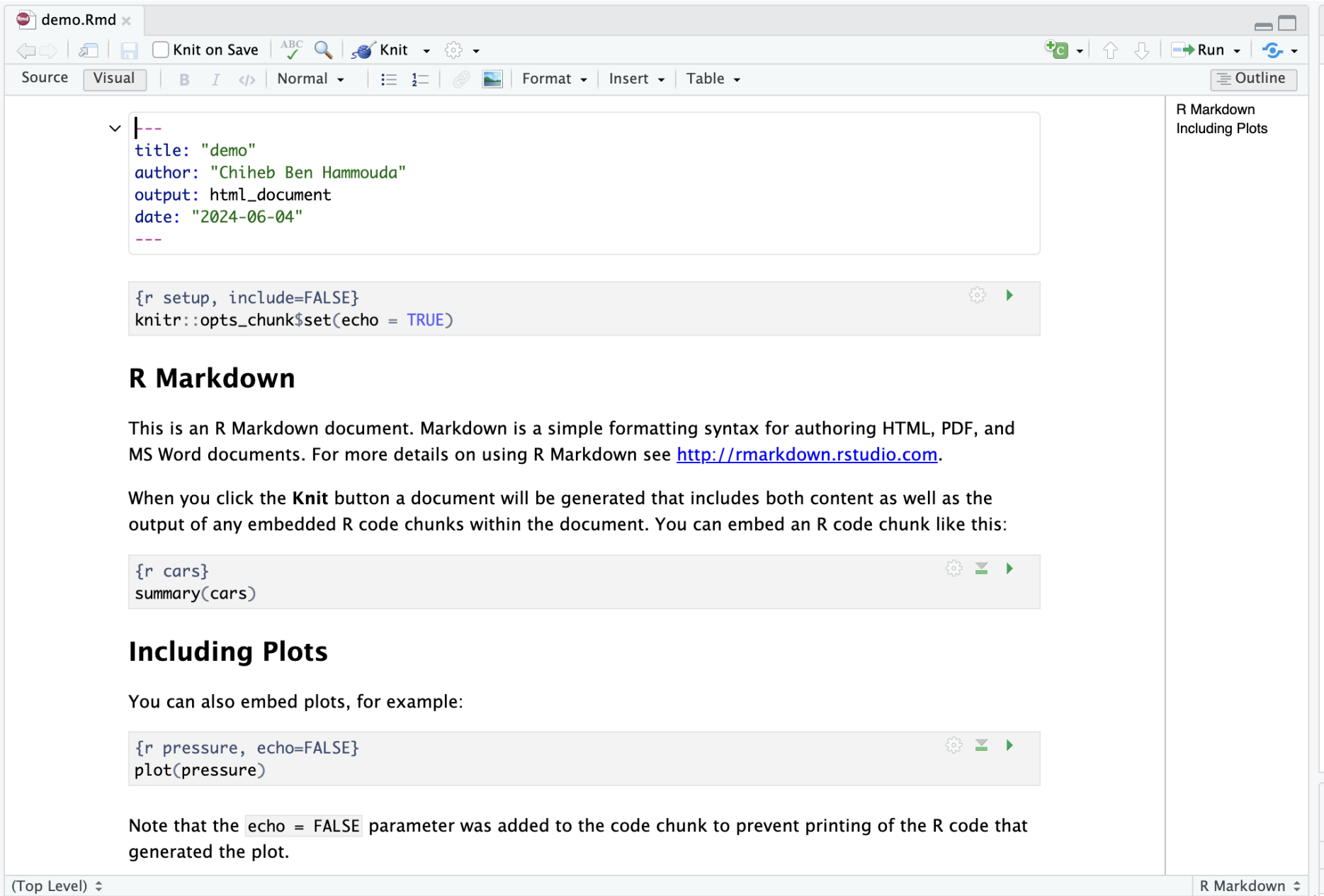
Source | Visual | Outline

```
1  ---
2  title: "demo"
3  author: "Chiheb Ben Hammouda"
4  output: html_document
5  date: "2024-06-04"
6  ---
7
8  ```{r setup, include=FALSE}
9  knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word
   documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
15
16 When you click the **Knit** button a document will be generated that includes both content as well as the output of
   any embedded R code chunks within the document. You can embed an R code chunk like this:
17
18 ```{r cars}
19 summary(cars)
20 ```
21
22 ## Including Plots
23
24 You can also embed plots, for example:
25
26 ```{r pressure, echo=FALSE}
27 plot(pressure)
28 ```
29
30 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated
   the plot.
31
```

R Markdown
Including Plots

1:1 | demo | R Markdown

Rmd in Visual Mode



Structure Rmd files

YAML: at top of document embedded in 3 dashes

```yaml
---
title: "demo"
author: "Chiheb Ben Hammouda"
output: html_document
date: "2024-06-04"
---
```

R chunks: insert with shortcut keys `Ctrl + Alt + I` or `Cmd + Option + I`

````
```{r pressure}
plot(pressure)
```
````

Chunk options: modify default behavior (e.g. `echo=TRUE`)

````
```{r pressure, echo=FALSE}
plot(pressure)
```
````

Other Chunk Options

Option	Meaning when chaning the default
`include`	if `FALSE`, run but do not show the R code and output
`echo`	if `FALSE`, hide the R code but show the output
`eval`	if `FALSE`, show the code but do not run it
`message/warning`	if `FALSE`, do show R messages/warnings
`error`	if `TRUE`, knit the document with incorrect R code
`fig.width`	set the width of a figure/plot (default = 7)
`fig.height`	set the height of a figure/plot (default = 5)

R Markdown: Examples and Tutorials

- For what you can do with R markdown, check out
  - R Markdown Gallery
- Tutorials
  - R Markdown Tutorial
  - R Markdown Quick tour

Basic data types in R

- R has many data types
  - Basic data types: `numeric (float in Python)`, `character (string in Python)` and `logical (boolean in Python)` (`TRUE`, `FALSE`)
    - Python

```
print(type('a'), type(1), type(1.3))
## <class 'str'> <class 'int'> <class 'float'>
```

    - R

```
# cat() is kind of like print()
cat(typeof('a'), typeof(1L), typeof(1.3))
## character integer double
```

  - vectors (Numpy array in Python), matrices (Numpy ndarray in Python), data frames and lists

- The assignment operator <- assigns a name to a data object, e.g. `x <- 3`

  ⚠ You can use x=3 but it is less common in R community.

Vectors

- A single entity consisting of a collection of things (*R manual*)
- The function `c(...)` combines its arguments into a vector, e.g. `x <- c(3, 5)`

class	elements	example
`numeric`	numbers	`c(1, 2, 3)`
`character`	strings	`c("a", "b", "c")`
`logical`	TRUE/FALSE	`c(TRUE, FALSE)`

Vectors are Atomic

You can't mix classes within a vector

```
x <- c("cat", "dog")   # character vector
y <- c(1, 2)           # numeric vector
z <- c(TRUE, FALSE)    # logical vector

# Check the class of each individual vector
class(x)  # [1] "character"
class(y)  # [1] "numeric"
class(z)  # [1] "logical"

# Combine the vectors and check the class
combined <- c(x, y, z)
class(combined)  # [1] "character"

# View the combined vector
combined  # "cat" "dog" "1" "2" "TRUE" "FALSE"
```

Why is the Combined Vector Character?

- R uses a coercion hierarchy: logical $<$ numeric $<$ character.
- When combining different types, R coerces them to the least restrictive type.
- Here, combining characters, numerics, and logicals results in a character vector.
- This is because characters can represent any type as a string.

Data Generating Vectors

Functions `c()`, `seq()`, and `rep()`

```
# Create a numeric vector using c()
c(1, 2, 3, 4, 5)
[1] 1 2 3 4 5

# Generate a sequence using seq()
seq(from = 1, to = 5)
[1] 1 2 3 4 5

# Generate a sequence using colon operator
1:5
[1] 1 2 3 4 5

# Generate a sequence with a specified increment
seq(1, 5, by = 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

# Repeat a sequence a specified number of times
rep(1:5, times = 2)
[1] 1 2 3 4 5 1 2 3 4 5

# Repeat each element of a sequence a specified number of times
rep(1:5, each = 2)
[1] 1 1 2 2 3 3 4 4 5 5
```

Naming Vector Elements

Assign a name to a vector element with = (not <-)!

```
# Create a named numeric vector
c(dog = 1, cat = 2, rabbit = 3)
dog     cat rabbit
1       2       3

# Create a numeric vector using assignment
c(dog <- 1, cat <- 2, rabbit <- 3)
[1] 1 2 3
```

Subsetting Vectors

`vector[i]` selects the element(s) in position `i` from a vector

- `i` can be a number, a vector with position numbers or a name
- the minus sign - before a number or vector deselects

```
# Create a named numeric vector
v <- c(dog = 1, cat = 2, rabbit = 3)

# Access the second element
v[2]
cat
2

# Access the first and third elements
v[c(1, 3)]
dog rabbit
  1      3

Access the element named "rabbit"
v["rabbit"]
rabbit
3

# Access all elements except the first one
v[-1]
cat rabbit
2      3
```

Matrices: Two-dimensional arrays with rows and columns

```
# Create a matrix with default parameters
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
               dimnames = NULL)
```

Examples:

```
# Create a matrix with data from 1 to 6, 2 rows, and 3 columns
matrix(data = 1:6, nrow = 2, ncol = 3)
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
# Create a matrix with data from 1 to 6, 3 rows, 2 columns,
#by row,  and named dimensions
matrix(data = 1:6, nrow = 3, ncol = 2, byrow = TRUE,
           dimnames = list(c("a", "b", "c"), c("A", "B")))
```

```
  A B
a 1 2
b 3 4
c 5 6
```

Other matrix functions

Matrices can also be made from the vectors of the same length with:

- `rbind(...)`: the vectors are combined as rows

```
# Create the matrix using rbind()
matrix <- rbind(c(1, 1), c(1, 2), c(1, 3))
matrix
```

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

- `cbind(...)`: the vectors are combined as columns

```
# Create the matrix using cbind()
matrix <- cbind(c(1, 1, 1), c(1, 2, 3))
matrix
```

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

Subsetting matrices

Matrices are also atomic (one type of data). Subsetting as for vectors, with two indices `[i, j]` for rows and columns:

```
# Create individual vectors
x <- c("cat", "dog")
y <- c(1, 2)
z <- c(TRUE, FALSE)

# Combine vectors into a matrix using cbind()
xyz <- cbind(x, y, z)
xyz
          x     y      z
[1,] "cat" "1"  "TRUE"
[2,] "dog" "2" "FALSE"

# Access element in the second row and second column
xyz[-1, 2]
[1] "2"

# Access element in the second row and column named "z"
xyz[2, "z"]
[1] "FALSE"
```

Subsetting a row or column

Empty spaces as in `[, j]` or `[i, ]` select the entire row/column

- returns a vector when selecting a single row/columns unless `drop = FALSE`

```
# Subset the matrix to include columns "y" and "z"
xyz[, c("y", "z")]
     y     z
[1,] "1"   "TRUE"
[2,] "2"  "FALSE"

# Subset the matrix to include only the column "z"
xyz[, "z"]
[1]   "TRUE" "FALSE"

# Subset the matrix to include only the column "z"
#and keep it as a matrix
xyz[, "z", drop = FALSE]
     z
[1,]   "TRUE"
[2,]   "FALSE"
```

Data frames

Two-dimensional array with observations and variables

```
# Create a data frame with observations and variables
data.frame(..., row.names = NULL)
```

Data frames are not atomic, variables can be of different class

```
# Combine vectors into a data frame
d <- data.frame(x, y, z)
d
    x   y     z
1 cat   1  TRUE
2 dog   2 FALSE

# Check the class of each variable in the data frame
sapply(d, class)
        x         y         z
"character" "numeric" "logical"
```

Subsetting data frames

As for matrices with `[i, j]` or with `$` as in `dataframe$variable`

```
# Subset the first row
d[1, ]
      x   y    z
 1   cat  1 TRUE

# Access the element in the first row and "x" column
d[1, "x"]
[1] "cat"

# Access the "x" column
d$x
[1] "cat" "dog"
```

Example

- R contains built-in data sets in the form of data frames.
- The `mtcars` dataset contains data extracted from the 1974 Motor Trend US magazine and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
  - **mpg**: Miles/(US) gallon
  - **cyl**: Number of cylinders
  - **disp**: Displacement (cu.in.)
  - **hp**: Gross horsepower
  - **drat**: Rear axle ratio
  - **wt**: Weight (1000 lbs)
  - **qsec**: 1/4 mile time
  - **vs**: Engine (0 = V-shaped, 1 = straight)
  - **am**: Transmission (0 = automatic, 1 = manual)
  - **gear**: Number of forward gears
  - **carb**: Number of carburetors

Displaying the First 6 Rows of `mtcars` and Classes of Variables

```
# Display the first 6 rows of the mtcars data frame
head(mtcars,6)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

```
# Display the classes of the variables in the mtcars data frame
sapply(mtcars, class)
```

```
      mpg       cyl      disp        hp      drat        wt      qsec
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
       vs        am      gear      carb
"numeric" "numeric" "numeric" "numeric"
```

Commands to List Available Datasets

```r
# List all available datasets in all loaded packages
data()

# List datasets available in the 'datasets' package
data(package = "datasets")

# Load a specific dataset
data(mtcars)

# Display help page for the dataset
?mtcars
```

1. Introduction to R
   - Motivation
   - R and RStudio
   - R Markdown

2. R data types
   - Vectors and Matrices
   - Data frames
   - Lists

3. Practicalities and Further Resources

Lists

- Collection of R objects of any kind (either named or unnamed, with different lengths)

```
# Create individual vectors
x <- c("cat", "dog")
y <- c(1, 2)
z <- c(TRUE, FALSE)
d <- data.frame(x, y, z)

# Create a list
my_list <- list(x = x, y, c = d)
my_list
$x
[1] "cat" "dog"

[[2]]
[1] 1 2

$c

    x  y     z
1 cat  1  TRUE
2 dog  2 FALSE
```

Subsetting Lists

- List element with
  - Double square brackets, e.g. `my_list[[...]]`
  - Dollar sign with list and element names, e.g. `my_list$x`

```
# Access the elements using different methods
my_list[["x"]]
[1] "cat" "dog"

my_list[[1]]
[1] "cat" "dog"

my_list[[1]][2]
[1] "dog"

my_list$x[2]
[1] "dog"
```

When to Use Each Data Type in R

- **Vectors**: Use when you have a single variable with all elements of the same type.
- **Matrices**: Use for 2D data where all entries are of the same type — great for numeric computations.
- **Data Frames**: Use for tabular data with columns of different types (e.g., names, ages, and test scores). This is your default for real-world datasets.
- **Lists**: Use when you need to group different objects — vectors, models, or even data frames — into a single container (e.g., model output + data + plot).

1. Introduction to R
   - Motivation
   - R and RStudio
   - R Markdown

2. R data types
   - Vectors and Matrices
   - Data frames
   - Lists

3. Practicalities and Further Resources

Practicalities and Further Resources

- Deadlines:
  - R assignment part1 exercises: Friday June 13, 3 pm.
- Related complementary material:
  - Book chapters (for transition from Python to R): Ch1, Ch2, Ch3.1-6, Ch4, Ch5, Ch6, Ch8 in "An Introduction to R and Python For Data Analysis: A Side By Side Approach" book.
  - Book chapters (Intro to R and data types): Ch2, Ch4, Ch5, Ch6, Ch7, Ch8 of "YaRrr! The Pirate's Guide to R" book.
  - Video lectures: Introduction to R and Data
- Interactive Tutorial:
  - Swirl: `https://swirlstats.com/students.html`
  - For Setup: `https://github.com/swirldev/R_Programming_E`
- Cheatsheets: `https://iqss.github.io/dss-workshops/R/Rintro/base-r-cheat-sheet.pdf`