R Basics

The R Studio User Interface
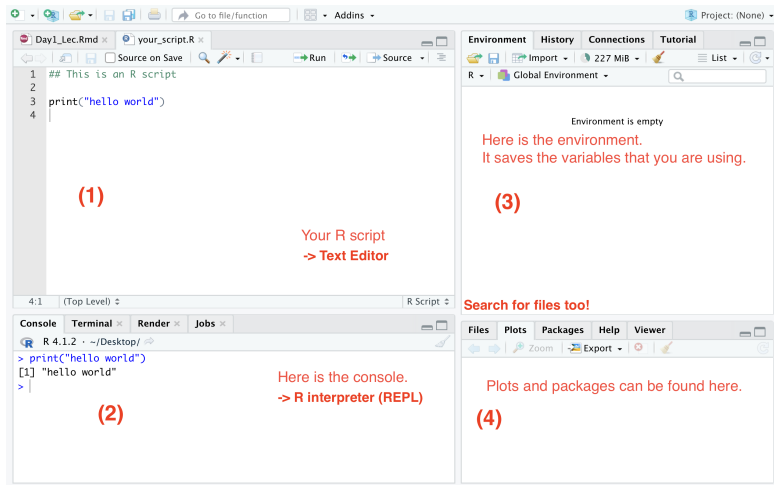
- RStudio is an IDE designed for R

Figure 1: R Studio

R Console

- R console is the R interpreter itself
- Execute code by typing on this console screen directly
- The normal prompt > denotes that R is ready for the new input.

R as a calculator: Basic numerical operations

```
2 + 3
```

```
## [1] 5
```
```
2 * 3 # asterisk
```

```
## [1] 6
```
```
4 - 1 # subtraction
```

```
## [1] 3
```
```
6 / 4 # slash
```

```
## [1] 1.5
```

R as a calculator: Basic numerical operations

```
2 ^ 3 # caret
```

```
## [1] 8
```
```
2 ** 3
```

```
## [1] 8
```
```
# '^'(2,3)
```

R as a calculator: Basic numerical operations

```
3 + 3 * 20
```

```
## [1] 63
```

```
(3 + 3) * 20
```

```
## [1] 120
```

Incomplete Commands

- If you type an incomplete command and press Enter, R will display a + prompt
- Either finish the command or hit Escape to start over:

console example

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

Errors

- When R recognizes **Errors**, then it **stops** executing the code and will return an error message.

```
> 3 % 5
Error: unexpected input in "3 % 5"
>
# Here the user is trying to do modulus in R, but fails to use the correct
# operator for modulus %%.
```

Warnings

- When you encounter **Warnings**, R usually finishes execution, but she wants to tell you that there are somethings you might want to take a look. Here is an example.

```
setwd("~/")
Warning message:
In in_dir(input_dir(), expr) :
  You changed the working directory to ~/
  (probably via setwd()).
  It will be restored to the default working directory.
  See the Note section in ?knitr::knit
```

Comments in R

- Comments are always helpful for humans (and LLMs)
- A single comment is written using # in the beginning of the statement.
- There is no multiline comments support.

```
# This is a comment.
## This is also a comment.
### This is still a comment.
### Comment again. #######
```

- Multiple Comments: Block select the lines and then,
  - On Windows and Linux, Ctrl + Shift + C.
  - On macOS, Cmd + Shift + C.

In-class exercises:

Let's try doing these simple tasks. If you execute everything correctly, you should end up with the same number that you started with:

1. Choose any number and add 2 to it.
2. Multiply the result by 3.
3. Subtract 6 from the answer.
4. Divide what you get by 3.

In-class exercises:

```r
3 + 2 # add 2
```

```
## [1] 5
```

```r
5 * 3 # multiply by 3
```

```
## [1] 15
```

```r
15 - 6 # substract 6
```

```
## [1] 9
```

```r
9 / 3 # divide by 3
```

```
## [1] 3
```

R Objects

- How to create objects
- Data types
- Data structures

Creating an Object

- We can give names that can carry meanings

```
a <- 3
print(a)
```

```
## [1] 3
```

```
print(class(a))
```

```
## [1] "numeric"
```

- *Assign* values by using two operators: <- and =.
- However, = is usually reserved to be used inside functions.
- R automatically detect types

Five Basic Data Types

There are **5** major data types in R. That said, most data you encounter will fall into these five categories. You can check the type of any object using the class() function.

1. Numeric (double)
2. Integer
3. Character
4. Logical
5. Factor

Numeric (Double)

- Default for any number
- Includes decimals (floats)

```
weight <- 70.25
print(weight)
```

```
## [1] 70.25
```

```
print(class(weight))
```

```
## [1] "numeric"
```

Integer

- Whole numbers can be contained as integer.
- In R, you must add L suffix to force numbers to be integer
- Rarely used?

```r
# Naive assignment - Numeric
height_num <- 6
print(class(height_num))
```

```
## [1] "numeric"
```

```r
# Force it to be integer
height_int <- 6L
print(class(height_int))
```

```
## [1] "integer"
```

Character (Strings)

- Text data
- "double quotes" or 'single quotes'

```
city <- "St. Louis"
print(class(city))
```

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

```
city <- 'St. Louis'
print(class(city))
```

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

Logical (Boolean)

- TRUE and FALSE
- TRUE = 1 FALSE = 0
- T and F

```
isNY <- city == "New York"
print(isNY)
```

```
## [1] FALSE
```

```
isSTL <- city == "St. Louis"
print(isSTL)
```

```
## [1] TRUE
```

```
print(isNY + isSTL)
```

```
## [1] 1
```

Factor (categorical values)

- Categories (e.g., "High", "Medium", "Low" or "Male", "Female")
- Integers with labels(levels)

```
gender <- factor(c("Male", "Female", "Female"))
class(gender) # Returns "factor"
```

```
## [1] "factor"
```

```
levels(gender) # Shows the unique categories
```

```
## [1] "Female" "Male"
```

Data Structures

- Let's explore some of the major types (There are different data structures: `tibble`, `data.table`)
  1. Vector
  2. List
  3. Dataframe
  4. Matrix

Vector

- Vector is a *list* of elements of the *same type*.
- c() function.
- R is specialized in vector level operations

```r
# A numeric vector
heights <- c(150, 162, 180, 155)

# A character vector
colors <- c("red", "blue", "green")

# IMPORTANT: If you mix types, R will "coerce" them to the most flexible ty
mixed <- c(1, "apple", TRUE)
class(mixed)

## [1] "character"
```

List

- Group of *any kinds of* R objects
- list()

```
list1 <- list(100:130, "R", list(TRUE, FALSE))
list1

## [[1]]
##  [1] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
## [20] 119 120 121 122 123 124 125 126 127 128 129 130
##
## [[2]]
## [1] "R"
##
## [[3]]
## [[3]][[1]]
## [1] TRUE
##
## [[3]][[2]]
## [1] FALSE
```

List Operations

- **double-bracketed indices** → *element*
- **single-bracket indices** → *sub-element*

```
list1[[2]]
```

```
## [1] "R"
```

```
list1[2]
```

```
## [[1]]
## [1] "R"
```

In-class exercises:

1. Use a list to store a single playing card, like the ace of hearts, which has a point value of one. The list should save the face of the card, the suit, and the point value in separate elements.

In-class exercises:

```r
card <- list("ace", "hearts", 1)
card

## [[1]]
## [1] "ace"
##
## [[2]]
## [1] "hearts"
##
## [[3]]
## [1] 1
```

Dataframe

- A data frame is a collection of vectors of the **same length**
- data.frame(column_name = values)

```r
df <- data.frame(HappyFace = c("ace", "two", "six"),
                 Suit = c("clubs", "clubs", "clubs"),
                 Value = c(1, 2, 3))
head(df)
```

```
##   HappyFace  Suit Value
## 1       ace clubs     1
## 2       two clubs     2
## 3       six clubs     3
```

Dataframe

- str(): investigate structure

```
typeof(df)
```

```
## [1] "list"
```

```
class(df)
```

```
## [1] "data.frame"
```

```
str(df) #structure
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ HappyFace: chr  "ace" "two" "six"
##  $ Suit     : chr  "clubs" "clubs" "clubs"
##  $ Value    : num  1 2 3
```

Example

```
deck <- data.frame(
  face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
    "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
    "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
    "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five"
    "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
    "eight", "seven", "six", "five", "four", "three", "two", "ace"),
  suit = c("spades", "spades", "spades", "spades", "spades", "spades",
    "spades", "spades", "spades", "spades", "spades", "spades", "spades",
    "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
    "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
    "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
    "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
    "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
    "hearts", "hearts", "hearts", "hearts", "hearts"),
  value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8
    7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12,
    10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
)
# View(deck) # This will open R studio stock viewer
```

In-class exercises:

1. Create an empty dataframe named df.
2. Create a dataframe using the four given vectors. Name the dataframe df.
3. Get the structure of df.
4. Open df in the data viewer in R.

In-class exercises:

```
# 2.
name = c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',
         'Matthew', 'Laura', 'Kevin', 'Jonas')
score = c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19)
attempts = c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1)
qualify = c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes'

df <- data.frame(cbind(name,score,attempts,qualify))
df <- data.frame(name,score,attempts,qualify)
```

Matrix

- Array
- matrix(values, nrow and or ncol) an

```
die = 1:6
die
```

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

```
m <- matrix(die, nrow = 2, byrow = FALSE)
m
```

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

matrix will fill up the matrix column by column by default, but you can fill the matrix row by row if you include the argument byrow = TRUE:

```
m <- matrix(die, nrow = 2, byrow = TRUE)
m
```

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

Matrix Operations

- There are few useful functions to manipulate matrices

Checking the Dimension

```
A <- matrix(c(2,3,-2,1,2,2),3,2)
A
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    3    2
## [3,]   -2    2
```

```
print(dim(A))
```

```
## [1] 3 2
```

- Can also be used with datamframes

```
df <- data.frame(names = c("charles", "amy"), age = c(15, 68))
print(dim(df))
```

```
## [1] 2 2
```

Multiplication by a Scalar

```
3 * A
```

```
##      [,1] [,2]
## [1,]    6    3
## [2,]    9    6
## [3,]   -6    6
```

Matrix Addition & Subtraction

```r
B <- matrix(c(1,4,-2,1,2,1),3,2)
A - B
```

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

```r
A + B
```

```
##      [,1] [,2]
## [1,]    3    2
## [2,]    7    4
## [3,]   -4    3
```

Matrix Multiplication

```
C <- matrix(c(2,-2,1,2,3,1),2,3)
dim(C)

## [1] 2 3

dim(A)

## [1] 3 2

C %*% A

##      [,1] [,2]
## [1,]    1   10
## [2,]    0    4

A %*% C

##      [,1] [,2] [,3]
## [1,]    2    4    7
## [2,]    2    7   11
## [3,]   -8    2   -4
```

Transpose of a Matrix

```
A
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    3    2
## [3,]   -2    2
```

```
t(A)
```

```
##      [,1] [,2] [,3]
## [1,]    2    3   -2
## [2,]    1    2    2
```

```
t(t(A))
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    3    2
## [3,]   -2    2
```

Inverse of a Matrix

```
D <- matrix(c(4,4,-2,2,6,2,2,8,4),3,3)
D
```

```
##      [,1] [,2] [,3]
## [1,]    4    2    2
## [2,]    4    6    8
## [3,]   -2    2    4
```

```
solve(D)
```

```
##       [,1] [,2] [,3]
## [1,]  1.0 -0.5  0.5
## [2,] -4.0  2.5 -3.0
## [3,]  2.5 -1.5  2.0
```

Determinant of a Matrix

```
D
```

```
##      [,1] [,2] [,3]
## [1,]    4    2    2
## [2,]    4    6    8
## [3,]   -2    2    4
det(D)
```

```
## [1] 8
```

Horizontal Concatenation

```
A
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    3    2
## [3,]   -2    2
```

```
B
```

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

```
cbind(A,B)
```

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

- can also be used with dataframes and vectors

Vertical Concatenation

```
rbind(A,B)
```

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

- can also be used with dataframes and vectors

In-class exercises:

1. Create a 3 by 3 identity matrix I.
2. Create another 3 by 3 matrix M with numbers 1 through 9, i.e. the first row should be 1, 2, 3, etc.
3. Print the diagonal of M.
4. Multiply M by 10, and save the new matrix as N.
5. Find the determinant of M.
6. Transpose M.
7. Does M have an inverse? If so, find the inverse of M. If not, explain why.

Common Attributes

- An attribute is a piece of information (metadata) that you can attach to R object
- attributes()
- names, dimensions, and classes (data types)

```
die <- 1:6
attributes(die)
```

```
## NULL
# R uses NULL to represent the null set, an empty object.
# NULL is often returned by functions whose values are undefined.
# You can create a NULL object by typing NULL in capital letters.
```

Names

```
names(die)
```

```
## NULL
```

- NULL means that die does not have a names attribute.

  - You can given a "names" attribute to die like this:

```
# The vector should include one name for each element in die
names(die) <- c("one", "two", "three", "four", "five", "six")

# Now die has a names attribute
attributes(die)
```

```
## $names
## [1] "one"   "two"   "three" "four"  "five"  "six"
```

```
head(die)
```

```
##   one   two three  four  five   six
##     1     2     3     4     5     6
```

  - To remove the names attribute, set it to NULL:

```
names(die) <- NULL
```

Dimension

- 1-dimensional: length().

```
length(die)
```

```
## [1] 6
```

- You can transform an atomic vector into an n-dimensional array by giving it a dimensions attribute with dim().

```
dim(die) <- c(2, 3) # values by default added column-wise
head(die)
```

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

Functions

- Take inputs, process these inputs, and return outputs
- Arguments is a value that you pass to a function (= inputs)
- Some examples: round() and factorial()

```
# The following two functions come with base R.
# You do not need to install and load new packages to use them.
round(3.1415)
```

```
## [1] 3
```

```
factorial(3)
```

```
## [1] 6
```

Functions

- R will work **from the innermost function to the outermost**.

```
die <- 1:6
die
```

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

```
mean(1:6)
```

```
## [1] 3.5
```

```
mean(die)
```

```
## [1] 3.5
```

```
round(mean(die))
```

```
## [1] 4
```

```
ceiling(mean(die)) # rounds up
```

```
## [1] 4
```

```
floor(mean(die)) # rounds down
```

```
## [1] 3
```

Function Arguments

- How do you know which argument names to use?
- args()

```
args(round)
```

```
## function (x, digits = 0, ...)
## NULL
```

```
round(3.14, digits = 1)
```

```
## [1] 3.1
```

```
args(ceiling)
```

```
## function (x)
## NULL
```

```
args(floor)
```

```
## function (x)
## NULL
```

Function Arguments

- Functions may have default values: digis=0
- For example,

```
round(3.1415)
```

```
## [1] 3
```

```
round(3.1415, digits = 2)
```

```
## [1] 3.14
```

Function Arguments

- If arguments names are not provided, R follows it
- If arguments names are not specified, R uses default order

```r
round(digits = 2, x = 3.1415) # reversed order, specify argument name
```

```
## [1] 3.14
```

```r
round(2, 3.1415) # R treat this as round(x = 2, digits=3.1415)
```

```
## [1] 2
```