Introduction to R

COMM 205 - Lecture 16 - R1

Hasan Cavusoglu

2020

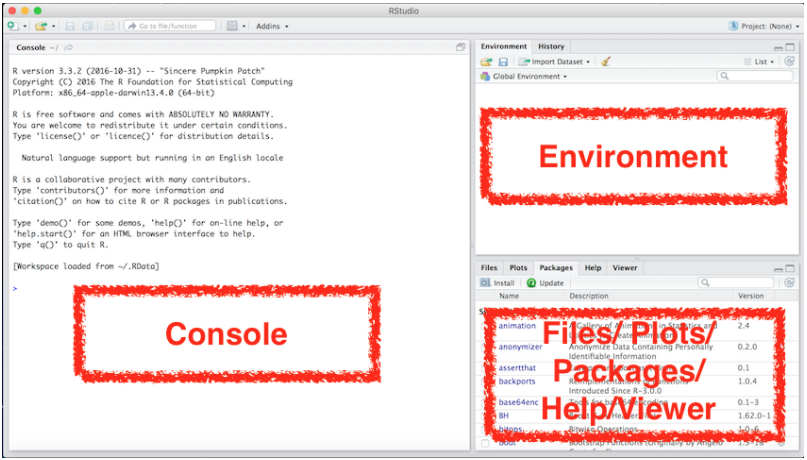
Agenda

- RStudio interface
- R conventions (naming, assignment. . .)
- Data types
- Basic operations
- Introduction to Implicit Coercion

How to use R

- We will be using R through an IDE (Integrated Development Environment) called R Studio.
- Once R Studio is launched (and there is nothing saved in the workspace), you should see three windows (i.e., panes). These are:
  - **Console** (on the left): This is where you can execute a piece of R code interactively
  - **Environment** (on the top right): You will see objects which are created during the session
  - **Files/Plots/Packages/Help/Viewer** (on the bottom right): You can navigate the directories, help documents and packages. Some output is also displayed there.

RStudio


RStudio
Console
Environment
History
Environment
Files
Plots
Packages
Help
Viewer
Console
Files/ Plots/
Packages/
Help/Viewer

Executing a code in R Studio

- There are various ways to execute (i.e., run) a code in R Studio. We will **first** cover how to execute a code from the **Console**.
- If you want to run a piece of R code, you can directly type it to the console. Once properly executed, the output of the code, if any, will be displayed.
- However, you will **not** be able to save your code if you run in on Console. We will also cover how to execute a code via a file later in the course.

Executing R Code on Console

- On console, after > prompt, if you type a piece of R code and hit `enter` (the code is syntactically correct), the R will execute the code.
- The results will be displayed either on console or on plot/viewer pane. For example, let's type the following on Console:

```
3 + 4 + 5
```

- Once you hit `enter`, you should see the following output on the console.

```
[1] 12
```

Case Sensitivity

- Before we learn how to execute commands in R, please note that almost everything in R is **case sensitive**. That is, you must enter the commands, variables, and options using the correct case. Let's display the working directory by typing `version` on Console:

```
version
```

- However, if you do **not** conform to right cases of a function name, you will be given an error message.

```
Version
```

```
Error: object 'Version' not found
```

- Names of the most of the R functions are all **lower case**. It is a convention, *not* a hard-and-fast rule. There are some functions whose names are not all lower case.

Arithmetic

- You can use R as your desktop calculator.
- The following order of arithmetic operations is used in R:

1. parentheses,
2. exponents,
3. multiplication/division,
4. addition/ subtraction.

- Operators of equal precedence are evaluated from left to right

Examples (Order of Aritmetic Operations)

```
> 3+4*2/2^2
[1] 5
```

1. 2^2 will be resolved. Then, we 3+4*2/4.
2. * and / are at the same level of precedence. They (4*2/4) are executed from left to right. First 4*2, then 8/4.
3. '3+2

- The complete list of precedence that R gives to the operators can be found in R manual.

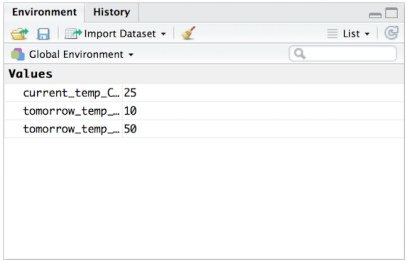
Assigning Objects

- If you want to retain the result of a calculation, you need to *assign* the results of a given computation to an *object* in the current workspace. Each object created should have a **unique** name.
- You can specify an assignment in R in two ways. But, by following the convention that almost all R users adhere to, we will do the assignment by:
  - using **arrow notation (<-)**
- Example:
  - When you get ready for school today, yo?u have heard on the radio that the weather tomorrow will be 15^oC cooler than today. You look at the dashboard, the current temperature is 25^oC. What would be the temperature tomorrow in Fahrenheit?

```
current_temp_Celsius <- 25
tomorrow_temp_Celsius <- current_temp_Celsius - 15
tomorrow_temp_Fahrenheit <- 1.8 * tomorrow_temp_Celsius + 32
```

Finding/Displaying Objects Created in the Session (1)

- Note that when a computation is assigned to an object, the result of the computation is not displayed
- However, you can confirm that the three objects were created at the Environment pane (top right window).



Finding/Displaying Objects Created in the Session (2)

- If you want to display the object, you can simply type its name or use a function call `print()`.

```
> tomorrow_temp_Fahrenheit
[1] 50
> print(tomorrow_temp_Fahrenheit)
[1] 50
```

- As you can see, both approaches will display the object (in this case, `tomorrow_temp_Fahrenheit` which 50).
- **Shortcut for inserting assignment operator** is `Alt` + `-` in `Windows` and `Option` + `-` in Mac OS.
- Almost all R coders prefer arrow notation for the assignment. Hence, we will use the arrow notation in this course. A single equal sign is used when the assignment is made within a function (which will see later).

Rules for naming Identifiers in R

R is very *liberal* when it comes to names for objects and functions. But, there are still some rules:

- Name can be a combination of letters, digits, period (.) and underscore (_).
- It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
- Reserved words in R cannot be used as identifiers (such as `if`, `else`, `TRUE`, `FALSE`, `NA`,...).

```
?reserved
```

Introduction to functions

- In programming, a named piece of code that performs a specific task is called **function**.
  - R comes with its own set of functions called base R functions.
  - However, there are many other libraries which provide many more functions.
  - If you cannot find a particular function that you want, you can write your own function to perform specialized tasks (*outside of the scope of the course*).
- Each function has a name. If you want to call a function, you need to follow its usage guideline.
- On the `Console`, if you type `?` followed by a function's name without any space between them, R documentation pertaining the function will be displayed in the help pane. There, you can obtain detailed information about the function, including its usage guideline.

Syntax

Generic Syntax of an R Function

$$function\_name(argument_1, .., argument_i = default_i, ..)$$

- When a function is executed, R uses the inputs, if provided by the user, and performs the steps and procedures prescribed in the function to deliver an output. Inputs to a function are called *arguments*.
  - Arguments should be listed within the parantheses after function.
  - Some functions do not take any argument. To call (i.e., use) those functions which do not take any argument, you type the function's name followed by `()`.
  - If a function requires a number of arguments, you need to provide them when you are calling the function.
  - For some functions, there are default values for some of the arguments.
  - $argument_i = default_i$ means if you do not provide a value for that argument when you call, R will use the default value defined by the function. If you specify a value for the argument, R will use the value you specified instead of default value.

Syntax Examples

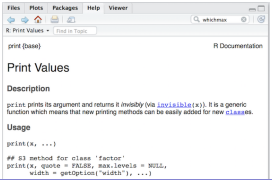
- Recall, just before this section, we called `print` function and passed `tomorrow_temp_Fahrenheit` as the argument as follows:

```
> print(tomorrow_temp_Fahrenheit)
[1] 50
```

- If you want to access to the help documents associated with any function, you can type ? followed bt the name of the function.

```
?print
```

- The help document, if it exists, will be displayed in the lower left pane as follows:



Basic Data Types in R

Basic data types in R.

- Numeric data (double, integer),
- String data (character), and
- Logical data (logical)

There are other types which are outside of our scope.

Numeric

- -20, -1.234, 0, 0.001, 20000000 are examples of **numeric** data.
- Default numeric type in R is called **double**.
- If **integer** numeric type is desired, it can be done by explicitly putting `L` after the number.
- Note that `typeof()` is a base R function which display the data type of the argument passed.

```
> my_age <- 40
> number_of_children <- 2L
> typeof(my_age)
[1] "double"
> typeof(number_of_children)
[1] "integer"
```

String

- "Hasan", "Adam", "COMM 205", "$%^&" are examples of **character**. Any string is of character data type.

```
> my_name <- "Hasan Cavusoglu"
> my_address <- '2053 Main Mall, Vancouver, BC V6T 1Z2'
> typeof(my_name)
[1] "character"
> typeof(my_address)
[1] "character"
```

- Note that character data type should be within quotes, *either* double quotes *or* single quotes. But, in the environment, character data type is stored within double quotes.

Logical

- TRUE and FALSE are the two possible values of **logical** data.

```
> is_slim <- FALSE
> typeof(is_slim)
[1] "logical"
```

- We can use comparison operators on numbers, characters, and logical data. "Logical" comparison operators available in R are:

Operator	Explanation
$<$	for less than
$>$	for greater than
$<=$	for less than or equal to
$>=$	for greater than or equal to
$==$	for equal to each other
$!=$	not equal to each other

- The result of a logical comparison is **logical data**.

Examples (1)

- Let's see some examples of logical comparison operators. Let's start with numeric data.

```
> 1 <= 2 # The result output should be TRUE
[1] TRUE
> 1 != 2
[1] TRUE
> 1 >= 2
[1] FALSE
> 1 == 1
[1] TRUE
```

- Note that # is the character to add comment to R code. If you add # after a piece of code, whatever comes in that line after # will be ignored by the R.

Examples (2)

Let's look at the logical data

```
> TRUE == TRUE
[1] TRUE
> TRUE != FALSE
[1] TRUE
> TRUE >= FALSE
[1] TRUE
```

Example (3)

Let's look at the character data

- If character type objects are contrasted, the *alphabetical ordering* is used. This is the order that you will see in a dictionary. If the cases of the strings are *not* the same, strings are compared assuming that they are all in lower case.
- If the string has a number in it, a digit is smaller than a letter.

```
> "abc" < "abd"      # results in TRUE
> "www" >= "http"    # results in TRUE
> "AAA" < "abc"      # results in TRUE
> "105X" < "Apple"   # results in TRUE
```

- **EXCEPTION** If there is a tie when everything is assumed lower case, R will consider the cases: The upper case is larger than the lower case. R compares the characters starting from the first one until it breaks the tie.

```
> "abc" == "Abc"     # results in FALSE
> "aBc" < "aBC"      # results in TRUE
```

Implicit Coercion

- If you are using an inappropriate data type in an operation or in a function, R can coerce the inappropriate type into an appropriate one.
  - A logical type will be coerce into number type if it is used in an arithmetic operation. `FALSE` will be coerced into `0` and `TRUE` will be coerced into `1`.

```
> typeof(TRUE + FALSE)
[1] "integer"
> typeof(FALSE + 2L)
[1] "integer"
> typeof(FALSE + 2)
[1] "double"
```

- When an arithmetic operation is applied *only* on logical data types, then logical data types are coerced into `integer`. However, if an arithmetic operation is applied on logical data as well as numeric data, the logical data will be coerced into the type of the number (either double or integer).

R cannot always Coerce

If R *cannot* resolve implicit coercion, it will give an error message. This generally happens with *nonsensical* cases. For example:

```
"a" + 1
```

will result in the following error message.

```
Error in "a" + 1 : non-numeric argument to binary operator
```

Similar error will be given if you try to execute the following pieces of code.

```
"a" + "b"
"a" + TRUE
```

Thank you!

Thank you.
See you next time!