Lecture 1.5 – Types of Data in R

Specific Learning Objectives:

1.1.9 – Create vectors, arrays, matrices, lists, and data frames.

1.1.10 – Understand vectors and vectorized calculations.

1.1.11 – Understand the data classes of R.

1.1.12 – Learn how to index vectors, arrays, matrices, lists, and data frames.

Try this

- A new way to assign a value to a variable:

```
assign(variable, value)
```

Try it:

```
> assign(y, 8)
Error in assign(y, 8) : object 'y' not found
```

What does this error mean?

Here's how to fix it:

```
> assign("y", 8)
>
```

How is "y" different from y?

Try looking at help: `?assign`

Data Types in R

- There are many types of data in R!

  - Some data accept different operations and have different meanings. Ex: “1” (character), 1 (numeric), and 1 (logical) all have different properties, and can accept different operations!

  - Without different data types, R can’t distinguish y (that carries the numeric value 8) and the letter “y”.

- For our example: `assign()` needs its first argument to be character so that it can create an object with that name. The second argument can be anything!

```
> assign(y, 8)
Error in assign
```

R is looking for object y and not finding it.

```
> assign("y", 8)
```

y is a character, so it knows to use y to create an object and assign it value 8.

Data Types in R

- **Class** is a ways of distinguishing different types of data.

  - 5 atomic classes of objects:
    - numeric
    - integer
    - complex
    - logical
    - character
  - Other specialized classes:
    - factor
    - matrix
    - data frame
    - list

- Check class using **`class(obj)`**:

```
> y<-seq(1,8)
> class(y)
[1] "integer"
```

Numerical data

- **Numerical data types:** those that deal with number values

  - **numeric** (or double): positive or negative real, numbers.

    Examples: **1/3** **4.0** **π** **5.9** **16/7**

    Test with **`is.numeric()`:**

    ```
    > is.numeric(2.1)
    [1] TRUE
    ```

  - **integer**: positive or negative real, numbers that are also whole.

    Examples: **1** **1100** **3**

    Test with **`is.integer()`:**

    ```
    > is.integer(2.1)
    [1] FALSE
    ```

    ```
    > is.integer(2L)
    [1] TRUE
    ```

  - **complex**: imaginary numbers (those that have $i = \sqrt{-1}$).

    Examples: **1+*i*** **2+3*i*** ***4i***

    Test with **`is.complex()`:**

    ```
    > is.complex(2.1)
    [1] FALSE
    ```

    ```
    > is.complex(2i)
    [1] TRUE
    ```

Logical data

- **Logical:** conditional value TRUE or FALSE, based on on/off binary

Examples: **1** **TRUE** **0** **FALSE**

- Useful for telling whether or not a condition is satisfied

```
> 2 > -3
[1] TRUE
```

```
> -1 < -10
[1] FALSE
```

- Output of many testing functions, like **`is.numeric()`**

```
> is.numeric(2.1)
[1] TRUE
```

```
> is.integer(2.1)
[1] FALSE
```

- Assign by using TRUE or FALSE, T or F.

```
> blip <- TRUE
> blip <- T
> blip <- true
Error: object 'true' not found
>
```

Not successfully assigned!

Logical data

- **Relational operators**: used to compare values, returns logical

Operator	Interpretation
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- Relational operators are vectorized:

```
> foo <- seq(1,5, by=0.5)
> foo > 2
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
>
```

Logical data

- **Logical operators**: used to compare logical values, returns logical

Operator	Interpretation
& **(ampersand)**	AND (element-wise)
&&	AND (single comparison)
| **(pipe)**	OR (element-wise)
||	OR (single comparison)
!	NOT

- Single logical operators are vectorized and will do element-wise comparisons. Double operators will only do one comparison (only the first positions in vectors)

Logical data

- **Logical can also be numeric!** R will automatically coerce logicals into numerics in some functions.

```
> foo <- seq(1,5, by=0.5)
> foo > 2
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
>
     0     0     0     1     1     1     1     1     1

> sum(foo > 2)
[1] 6
```

The correct code is:
`anyvec[!c(...)] <- 0`
see notes.R!

- **Logicals make sorting and replacement easy.** Entering logicals in square brackets will pull a value at any position that is TRUE.

```
> anyvec <- c(3, 10, 12, 30, 45, 60)
> anyvec[c(T,F,T,F,T,F)]
[1]  3 12 45
```

```
> foo[foo > 2]
[1] 2.5 3.0 3.5 4.0 4.5 5.0
```

Replaces true values with 0!

```
> anyvec[c(T,F,T,F,T,F)] <- 0
> anyvec
[1]  0 10  0 30  0 60
```

Replaces false values with 0!

~~`> anyvec[-c(T,F,T,F,T,F)] <- 0`~~
~~`> anyvec`~~
~~`[1] 0 0 0 0 0 0`~~

Check Your Understanding!

Will the following line of code work without an error?

```
mean(c(T, F, F, F, T, T, T, F))
```

Correct answer

a) Yes!

a) No!

Write about why this is true.

Character data

- **Characters** or strings: written alphanumeric text.

Examples: **“7-27-2021”** **“1”** **“Steve”** **“h8rs”**

- Character strings can be used to record dates, names, file locations, etc.

- Character strings are indicated by double quotes: “”

```
> blop <- "This is a string."
> blop
[1] "This is a string."
```

- R treats strings as a single object

```
> length(blop)
[1] 1
```

Character data

- Remember that character strings are NOT numeric!

```
> mynumber <- "35.4"
> mynumber*2
Error in mynumber * 2 : non-numeric argument to binary operator
```

- A lot of relational operators still work.

- Be careful of using the backslash \, it is an escape character which controls how things are printed on the screen!

Character data

- **Concatenating characters:** several options, depending on how you want things combined.

  - For **`c()`**, we can create a vector of strings:

```
> my.string <- c("This", "is", "a", "string", ".")
> my.string
[1] "This"   "is"     "a"      "string" "."
> length(my.string)
[1] 5
```

  - For **`paste()`**, we can create a single string out of several separate strings:

```
> paste(my.string[1],my.string[2], my.string[3], my.string[4], my.string[5])
[1] "This is a string ."
```

    - **You can assign the output of `paste()` into its own variable!**

```
> filename <- paste0("./results/gpc-surrogates/",
+                    "results-",Sys.Date(), ".csv")
> filename
[1] "./results/gpc-surrogates/results-2021-07-27.csv"
```

Check Your Understanding!

Assign the following value to a:

```
a <- "y"
```

What class is object `a` and how can you test this to make sure?

a) numeric
`is.numeric(a)`

b) logical
`is.logical(a)`

Correct answer

c) character
`is.character(a)`

d) complex
`is.complex(a)`

What is R's output if you tried a, b, or d?

Troubleshooting Corner

- Mixing data types will sometimes go sideways!
  - If you do this, the errors might look like this:

```
> y<-"a"
> b<-2
> y+b
Error in y + b : non-numeric argument to binary operator
```

```
> y<-3"b"
Error: unexpected string constant in "y<-3"b""
```

- Functions have limitations on what types of data they can use.
  - If you use the wrong type of data, the errors might look like this:

```
> sum(y)
Error in sum(y) : invalid 'type' (character) of argument
```

```
> assign(y, 8)
Error in assign
```

Troubleshooting Corner

- Relational operators ALWAYS need two things to compare.

```
> b<-2
> b < 3
[1] TRUE
> b < 3 & > 7
Error: unexpected '>' in "b < 3 & >"

> b < 3 & b > 7
[1] FALSE
```

Comparison fails because of syntax

Correct syntax fixes error

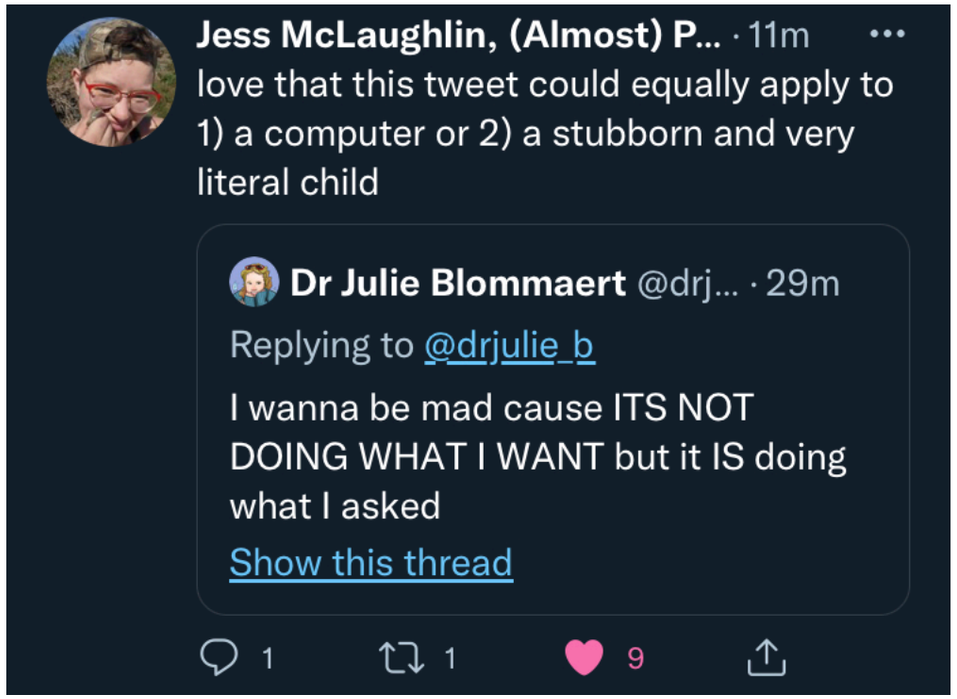
```
> randos <- runif(10)
> randos[>0.5]
Error: unexpected '>' in "randos[>"

> randos[randos>0.5]
[1] 0.5372260 0.7084477 0.7552380 0.8837865 0.6889207 0.9151532 0.8237162
```

Troubleshooting Corner

- Remember to take breaks! Don’t leave your assignments to the last minute! Errors are normal!

- Remember to treat R like that time your friend took everything extremely literally just to annoy you. That’s like how R is *every day*.



Action Items

1. **Complete assignment 1.6.**

2. **Read Davies Ch. 2 for next time.**