Session 2
Computing and R

R for Social Data Science

Jeffrey Ziegler, PhD

Assistant Professor in Political Science & Data Science
Trinity College Dublin

Fall 2022

Road map for today

- Backstory of R
- R operators and objects
- Data structures and types

R background

- **S** (for **S**tatistics) is a programming language for statistical analysis developed in 1976 in AT&T Bell Labs
- Original S language and its extention S-PLUS were closed source
- In 1991 **R**oss Ihaka and **R**obert Gentleman began developing R, an open-source alternative to **S**

R BASICS

- R is an *interpreted* language (like Python and Stata)
- It is geared towards statistical analysis
- R is often used for interactive data analysis (one command at a time)
- But it also permits to execute entire scripts in *batch* mode

```
1 print("Hello World!")
```

```
[1] "Hello World!"
```

Operators

Key operators ("infix" functions) in R are:

- Arithmetic (‘+‘, ‘-‘, ‘*‘, ‘^‘, ‘/‘, ‘%/%‘, ‘%%‘, ‘%*%‘)
- Boolean (‘&‘, ‘&&‘, ‘|‘, ‘||‘, ‘¡)
- Relational (‘==‘, ‘!=‘, ‘>‘, ‘>=‘, ‘<‘, ‘<=‘)
- Assignment (‘<-‘, ‘«-‘, ‘=‘)
- Membership (‘%in%‘)

Basic mathematical operations in R

```
1+1
```

```
[1] 2
```

```
5-3
```

```
[1] 2
```

```
6/2
```

```
[1] 3
```

```
4*4
```

```
[1] 16
```

Advanced mathematical operations in R

```r
# Let's try exponentation
# Note that 2 ** 4 also works, but is not recommended 2^4
2^4
```

```
[1] 16
```

```r
# Integer division, equivalent to Python's '//'
7%/%3
```

```
[1] 2
```

```r
# Modulo operation (remainder of division), equivalent to
    Python's '%'
7%%3
```

```
[1] 1
```

Basic logical operations in R

```r
3!=1 # Not equal
```

```
[1] TRUE
```

```r
3>3 # Greater than
```

```
[1] FALSE
```

```r
FALSE|TRUE # True if either 1st or 2nd operand is T
```

```
[1] TRUE
```

```r
F|T # R also treats F and T as Boolean, not recommended
```

```
[1] TRUE
```

```r
3>3|3>=3 # Combining 3 Boolean expressions
```

```
[1] TRUE
```

R objects

Everything is an object.

- John Chambers

- Everything you are dealing with in R is an **object**
- That includes individual variables, datasets, functions, and many other classes of objects
- The key reference to an object is its *name*
- Typically, reference is established through assignment operation

Assignment operations

- ‘<-‘ is standard assignment operator in R
- While ‘=‘ is also supported, not recommended
- As it hides the difference between ‘<-‘ and ‘«-‘ (deep assignment)

Extra: R Documentation on assignment

```

x <- 3
```

```
[1] 3
```

```
f <- function(){
  x<<-1 # Modifies existing var in parent namespace
  }
f()
x
```

```
[1] 1
```

Membership operations

Operator ‘%in%‘ returns ‘TRUE‘ if an object of the left side is in a sequence on the right

```
"a" %in% "abc"# Note that R strings are not sequences
```

```
[1] FALSE
```

```
3 %in% c(1,2,3) # c(1, 2, 3) is a vector
```

```
[1] TRUE
```

```
!(3 %in% c(1,2,3))
```

```
[1] FALSE
```

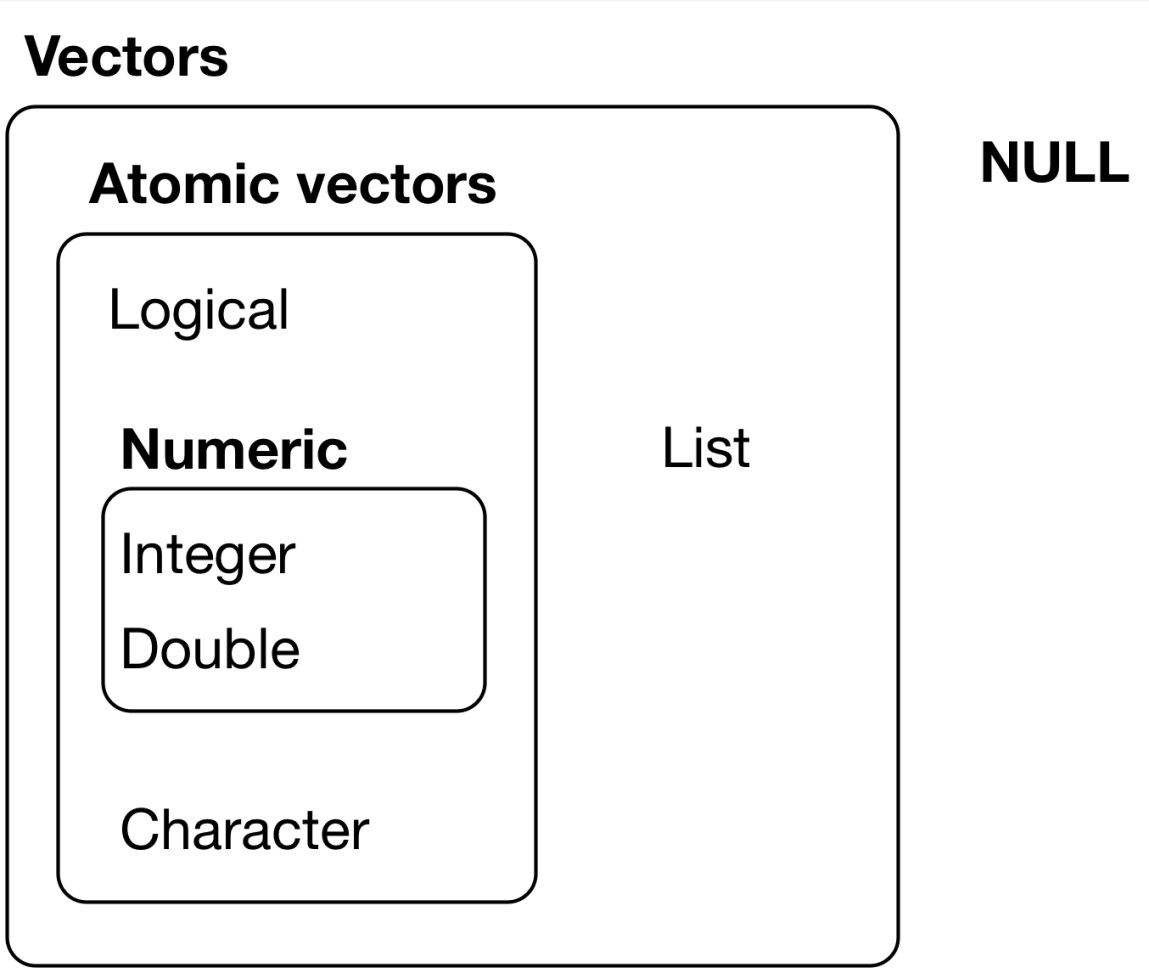
Data structures

- Base R data structures can be classified along their *dimensionality* and *homogeneity*
- 5 main built-in data structures in R:
  - Atomic vector (‘vector‘)
  - Matrix (‘matrix‘)
  - Array (‘array‘)
  - List (‘list‘)
  - Data frame (‘data.frame‘)

Summary of data structures in R

Structure	Description	Dimensionality	Data Type
vector	Atomic vector (scalar)	1d	homogenous
matrix	Matrix	2d	homogenous
array	One-, two or n-dimensional array	1d/2d/nd	homogenous
list	List	1d	heterogeneous
data.frame	Rectangular data	2s	heterogeneous

Vectors in R



Source: R for Data Science

Atomic vectors

- *Vector* is core building block of R
- R has no scalars (they are just vectors of length 1)
- Vectors can be created with ‘c()‘ function (short for **c**oncatenate)

```
v <- c(8,10,12)
v
```

```
[1] FALSE
```

```
v <- c(v,14) # Always flattened (even when nested)
v
```

```
[1] FALSE
```

Data types

4 common data types that are contained in R structures:

- Character (‘character‘)
- Integer (‘integer‘)
- Double/numeric (‘double‘/‘numeric‘)
- Logical/boolean (‘logical‘)

Character vector

```r
char_vec <- c("apple","banana","watermelon")
char_vec
```

```
[1] "apple"      "banana"     "watermelon"
```

```r
length(char_vec)
```

```
[1] 3
```

```r
is.character(char_vec)
```

```
[1] TRUE
```

Integer vector

```
# Note 'L' suffix to get an integer, not double
int_vec <- c(300L,200L,4L)
```

```
[1] 300 200   4
```

```
typeof(int_vec)
```

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

```
is.integer(int_vec)
```

```
[1] TRUE
```

Double vector

```
# Note that even without decimal part R treats these
    numbers as doubled
dbl_vec <- c(300,200,4)
dbl_vec
```

```
[1] 300 200   4
```

```
typeof(dbl_vec)
```

```
[1] "double"
```

```
is.double(dbl_vec)
```

```
[1] TRUE
```

```
is.numeric(int_vec)
```

```
[1] TRUE
```

Integer vs double

- Integers are used to store whole numbers (e.g. counts)
- Double is a floating-point numbers with double precisions
- 32-bit integer: $2^{32} = 4,294,967,296$
- Signed 32-bit integer: $[-2,147,483,648 ., . 2,147,483,648]$

Extra: More on integer overflow on YouTube

Logical vector

```
log_vec <- c(FALSE,FALSE,TRUE)
log_vec
```

```
[1] FALSE FALSE TRUE
```

```
# While more concise, using T/F instead of TRUE/FALSE can
    be confusing
log_vec2 <- c(F,F,T)
log_vec2
```

```
[1] FALSE FALSE TRUE
```

```
typeof(log_vec)
```

```
[1] "logical"
```

Type coercion in vectors

- All elements of a vector must be of the same type
- If you try to combine vectors of different types, their elements will be *coerced* to most flexible type

```
# Note that logical vector get coerced to 0/1 for F/T
c(dbl_vec,log_vec)
```

```
[1] 300 200   4   0   0   1
```

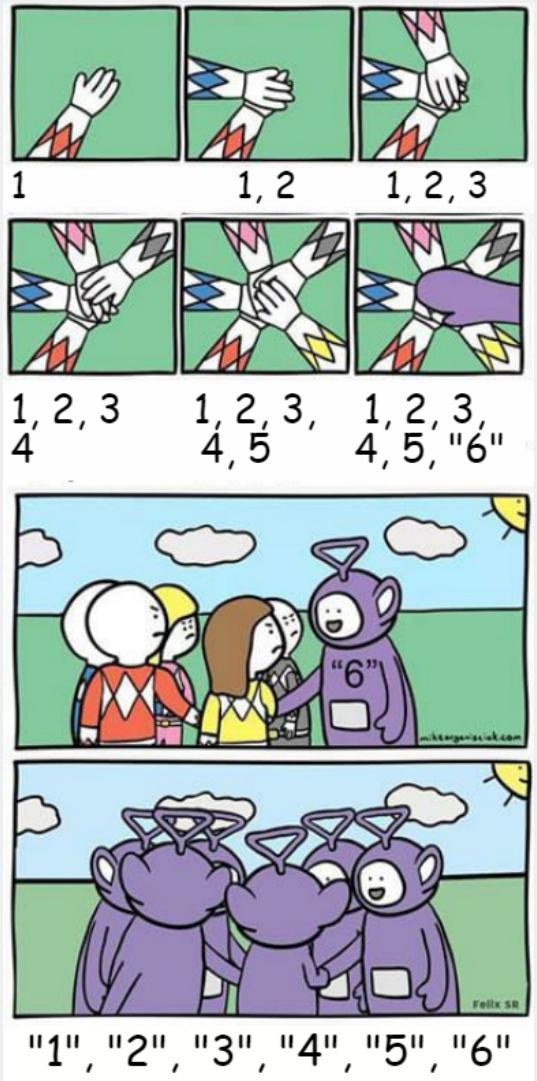
```
c(char_vec,int_vec)
```

```
[1] "apple" "banana" "watermelon" "300" "200" "4"
```

```
# If no way of conversion exists, NAs are introduced
as.numeric(char_vec)
```

```
[1] NA NA NA
Warning message: NAs introduced by coerc
```

Implicit type coercion



Extra: Twitter

NA and NULL values

- In Python we encountered ‘None‘ value
- R makes a distinction between:
  - ‘NA‘ - value exists, but is unknown (e.g. survey non-response)
  - ‘NULL‘ - object does not exist
- ‘NA‘’s are defined for each data type (integer, character, numeric, etc.))

Extra: R Documentation on NA

NA and NULL example

```
na_vec <- c(NA,NA,NA)
na_vec
```

```
[1] NA NA NA
```

```
length(na_vec)
```

```
[1] 3
```

```
null_vec <- c(NULL,NULL,NULL)
null_vec
```

```
[1] NULL
```

```
length(null_vec)
```

```
[1] 0
```

Working with NAs

```
v_na<-c(1,2,3,NA,5)
mean(v_na)
```

```
[1] NA
```

```
# NAs should be treated specially
mean(v_na,na.rm=TRUE)
```

```
[1] 2.75
```

```
# Remember NAs are missing values
NA==NA
```

```
[1] NA
```

Working with NAs

```
is.na(v_na)
```

```
[1] FALSE FALSE FALSE  TRUE FALSE
```

```
# We can use such logical vectors for subsetting (more
    below)
v_na[!is.na(v_na)]
```

```
[1] 1 2 3 5
```

"Tutorial": R Script

- Usually you want to have a record of what analysis was done and how you did it
- So, instead of writing all your R commands in the interactive console
- You can create an R script, write them there and run then together or one at a time
- R script is a file with .R extension and contains a collection of valid R commands

Naming conventions

- Even while allowed in R, do not use ‘.‘ in variable names (it works as an object attribute in Python)
- Do not name give objects the names of existing functions and variables (e.g. ‘c‘, ‘T‘, ‘list‘, ‘mean‘)
- Use **UPPER_CASE_WITH_UNDERSCORE** for named constants (e.g. variables that remain fixed and unmodified)
- Use **lower_case_with_underscores** for function and variable names

Extra: http://adv-r.had.co.nz/Style.html

Code layout

- Limit all lines to a maximum of 79 characters
- Break up longer lines

```
my_long_vector <- c(
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60
)

long_function_name <- function(a = "a long argument",
b = "another argument",
c = "another long argument") {
# As usual code is indented by two spaces
}
```

Reserved words

There are 14 (plus some variations of them) reserved words in R that cannot be used as identifiers

- ‘break‘
- ‘NA‘
- ‘else‘
- ‘NaN‘
- ‘FALSE‘
- ‘next‘
- ‘for‘
- ‘NULL‘
- ‘function‘
- ‘repeat‘
- ‘if‘
- ‘TRUE‘
- ‘Inf‘
- ‘while‘

Exercise 1: Vector manipulation

- Load built-in R object ‘letters‘ (lower-case letters of the Roman alphabet)
- Calculate its length
- Generate a vector of integers that starts from 1 and has the same length as ‘letters‘
- Assign each integer corresponding lower-case letter as name
- Use these names to subset all vowels
- Now, repeat subsetting, but using indices not names

Tabulation and crosstabulation in R

- R function ‘table()‘ provides an easy way of summarizing categorical variables
- Note that implicitly variables represented as character vectors are converted to factors"

```
1 settlements < -c("Dublin","Belfast","Cork","Limerick","
    Derry","Galway","Newtownabbey","Bangor","Waterford","
    Lisburn")
2 provinces <- c("Leinster","Ulster","Munster","Munster","
    Ulster","Connacht","Ulster","Ulster","Munster","Ulster"
    )
3 # Given that each town appears only once, crosstabs might
    not be informative
4 table(top_10_settlements,provinces)
```

Source: Top 10 most populous settlements on the island of Ireland

Exercise 2: Working with attributes and factors

- As you note the output of ‘table(provinces)‘ is sorted alphabetically
- Change this to reflect the actual counts
- First, let’s store the result of tabulation for later re-use
- Start from exploring the structure of this object with ‘str()‘
- What are the 2 main parts of this object? How are they stored?

Exercise 2: Working with attributes and factors

- Extract the relevant parts from the stored object
- Save them as a named vector with provinces as names and counts as values
- Use ‘sort()‘ function to sort the vector in a decreasing order (from largest to smallest)
- Convert the original ‘provinces‘ vector into a factor with the levels ordered accordingly
- Re-run ‘table(provinces)‘

Week 2 "Takehome" Exercise

- Save a ‘letters‘ object under a different name
- Convert saved object into a matrix of 13 rows and 2 columns
- Subset letter ’f’ using indices
- Concatenate 3 copies of ‘letters‘ object together in a single character vector
- Convert it into a 3-dimensional array, where each dimension appears as a matrix above
- Subset all letters ’f’ across all 3 dimensions "

Class business

Today, we talked about...

- Backstory
- R operators and objects
- Data structures and types

Next week...

- Indexing and subsetting
- Attributes