Week 2: Data Structures

POP88162 Introduction to Quantitative Research Methods

Tom Paskhalis

Department of Political Science, Trinity College Dublin

So Far

- R is a free and open-source programming language with a focus on data analysis.
- Code can be distributed as R scripts and R Markdown.
- Everything in R is an object, the references to which are established through assignment.

Topics for Today

- Data structures
- Data types
- Missing values
- Vector subsetting

Review: Assignment Operations

- `<-` is the standard assignment operator in R
- While `=` is also supported it is not recommended

```
1 x <- 3
2 x
```

```
[1] 3
```

```
1 democracy_2020 <- read.csv("../data/democracy_2020.csv")
```

> **Extra**
>
> R Documentation on assignment

Data Structures

Base R data structures can be classified along their:

- **dimensionality**
- **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	2d	heterogeneous

Atomic Vectors

- *Vector* is the core building block of R.
- Vectors can be created with `c()` function (short for combine)
- Individual variables are stored as vectors.

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

```
[1]  8 10 12
```

```
1 v <- c(v, 14) # Vectors are always flattened (even when nested)
2 v
```

```
[1]  8 10 12 14
```

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) # length() function gives the length of an R object
```

```
[1] 3
```

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

```
[1] TRUE
```

Numeric Vector

```r
num_vec <- c(300, 200, 4)
num_vec
```

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

```r
typeof(num_vec) # typeof() function returns the type of an R object
```

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

```r
is.numeric(num_vec)
```

```
[1] TRUE
```

Logical Vector

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

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

```r
# 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
```

```r
typeof(log_vec)
```

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

```r
is.logical(log_vec)
```

```
[1] TRUE
```

```r
# Compare two vectors element-by-element
log_vec == log_vec2
```

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

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 the most flexible type

```
# Note that logical vector get coerced to 0/1 for FALSE/TRUE
c(num_vec, log_vec)
```

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

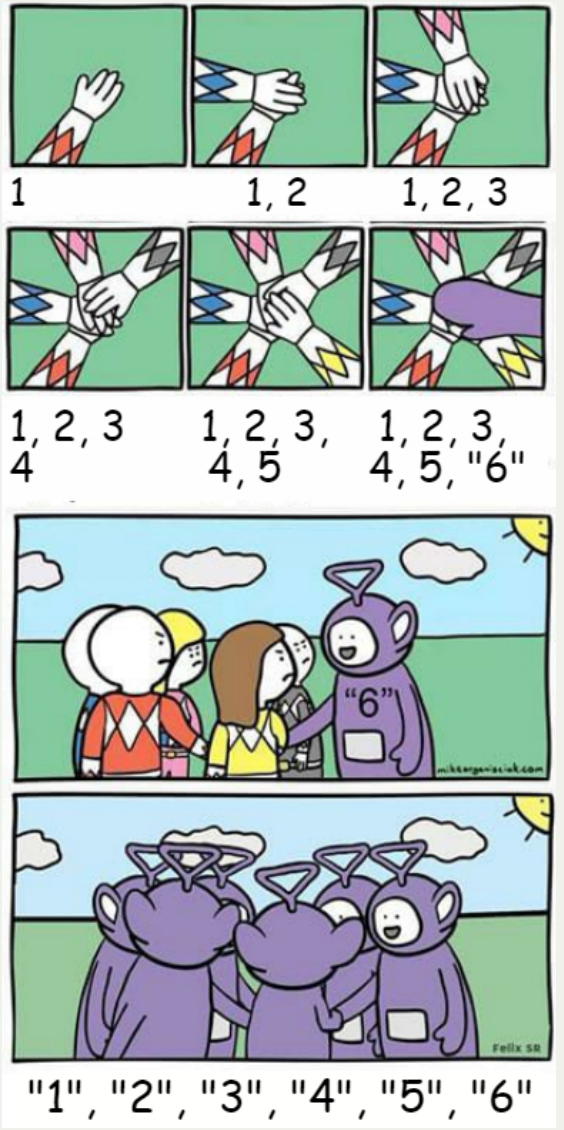
```
c(char_vec, num_vec)
```

```
[1] "apple"      "banana"     "watermelon" "300"        "200"
[6] "4"
```

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

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

Implicit Type Coercion



X (Twitter)

Missing Values in R

- Missing values have a special type in R.
- R makes a distinction between:
  - `NA` - value exists, but is unknown (e.g. survey non-response)
  - `NULL` - object does not exist

> **Extra**
>
> R Documentation on NA

NA and NULL Example

```
1 na <- c(NA, NA, NA)
2 na
```

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

```
1 length(na)
```

```
[1] 3
```

```
1 null <- c(NULL, NULL, NULL)
2 null
```

```
NULL
```

```
1 length(null)
```

```
[1] 0
```

Working with NAs

```r
# Presence of NAs can lead to unexpected results
v_na <- c(1, 2, 3, NA, 5)
mean(v_na)
```

```
[1] NA
```

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

```
[1] 2.75
```

```r
# Remember NAs are missing values
# Thus result of comparing them is unknown
NA == NA
```

```
[1] NA
```

```r
# is.na() is a special function that checks whether value is missing (NA)
is.na(v_na)
```

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

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

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

Vector Indexing and Subsetting

- To subset a vector, use `[]` to index the elements you would like to select:

```
vector[index]
```

```
1 num_vec[1]
```

```
[1] 300
```

```
1 num_vec[c(1,3)]
```

```
[1] 300   4
```

Summary of Vector Subsetting

Value	Example	Description
Positive integers	`v[c(3, 1)]`	Returns elements at specified positions
Negative integers	`v[-c(3, 1)]`	Omits elements at specified positions
Logical vectors	`v[c(FALSE, TRUE)]`	Returns elements where corresponding logical value is `TRUE`
Character vector	`v[c("c", "a")]`	Returns elements with matching names (only for named vectors)
Nothing	`v[]`	Returns the original vector
0 (Zero)	`v[0]`	Returns a zero-length vector

Generating Sequences for Subsetting

- You can use `:` operator to generate vectors of indices for subsetting.
- `seq()` function provides a generalization of `:` for generating arithmetic progressions.

```
1 2:4
```

```
[1] 2 3 4
```

```
1 seq(from = 1, to = 4, by = 2)
```

```
[1] 1 3
```

Vector Subsetting Examples

```
# v <- c(8, 10, 12, 14)
v[2:4]
```

```
[1] 10 12 14
```

```
# Argument names in seq() can be omitted
v[seq(1,4,2)]
```

```
[1]  8 12
```

```
# All but the last element
v[-length(v)]
```

```
[1]  8 10 12
```

```
# Reverse order
v[seq(length(v),1,-1)]
```

```
[1] 14 12 10  8
```

Vector Recycling

For operations that require vectors to be of the same length R recycles (reuses) the shorter one:

```
1 c(0, 1) + c(1, 2, 3, 4)
```

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

```
1 5 * c(1, 2, 3, 4)
```

```
[1]  5 10 15 20
```

```
1 c(1, 2, 3, 4)[c(TRUE, FALSE)]
```

```
[1] 1 3
```

Lists

- As opposed to vectors, **lists** can contain elements of any type.
- List can also have nested lists within it.
- Lists are constructed using `list()` function in R.
- Data frames are just lists with some special characteristics!

```r
# We can combine different data types in a list and, optionally, name elements (e.g. B below)
l <- list(2:4, list("a"), B = c(TRUE, FALSE, FALSE))
l
```

```
[[1]]
[1] 2 3 4

[[2]]
[[2]][[1]]
[1] "a"


$B
[1]  TRUE FALSE FALSE
```

R object structure

- `str()` - one of the most useful functions in R.
- It shows the **str**ucture of an arbitrary R object.

```
1 str(l)
```

```
List of 3
 $  : int [1:3] 2 3 4
 $  :List of 1
  ..$ : chr "a"
 $ B: logi [1:3] TRUE FALSE FALSE
```

List subsetting

- As with vectors you can use `[]` to subset lists.
- This will return a list of length one.
- Components of the list can be individually extracted using `[[` and `$` operators.

```
list[index]
list[[index]]
list$name
```

List subsetting examples

```
1 l
```

```
[[1]]
[1] 2 3 4

[[2]]
[[2]][[1]]
[1] "a"


$B
[1]  TRUE FALSE FALSE
```

```
1 l[3]
```

```
$B
[1]  TRUE FALSE FALSE
```

```
1 str(l[3])
```

```
List of 1
 $ B: logi [1:3] TRUE FALSE FALSE
```

```
1 l[[3]]
```

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

```
1 # Only works with named elements
2 l$B
```

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

Next

- Tutorial:
  - Working with variables
- 1 R Assignment due:
  - **08:59 Tuesday, 4 February**
- Next week:
  - Probability distributions