Managing and Manipulating Data Using R

Introduction

Ozan Jaquette

1. Course overview

2. Introduction to R

Course overview

Introduction to R

R as a calculator

```r
5
#> [1] 5
5+2
#> [1] 7
10*3
#> [1] 30
```

Executing commands in R

```
5
#> [1] 5
5+2
#> [1] 7
10*3
#> [1] 30
```

Three ways to execute commands in R

1. Type/copy commands directly into the “console”
2. ‘code chunks’ in RMarkdown (.Rmd files)
    - Can execute one command at a time, one chunk at a time, or “knit” the entire document
3. R scripts (.R files)
    - This is just a text file full of R commands
    - Can execute one command at a time, several commands at a time, or the entire script

Shortcuts you should learn for executing commands

```
5+2
#> [1] 7
10*3
#> [1] 30
```

Three ways to execute commands in R

1. Type/copy commands directly into the “console”
2. ‘code chunks’ in RMarkdown (.Rmd files)
    - **Cmd/Ctrl + Enter**: execute highlighted line(s) within chunk
    - **Cmd/Ctrl + Shift + k**: “knit” entire document
3. R scripts (.R files)
    - **Cmd/Ctrl + Enter**: execute highlighted line(s)
    - **Cmd/Ctrl + Shift + Enter** (without highlighting any lines): run entire script

Assignment

Assignment means creating a variable – or more generally, an “object” – and assigning values to it

- `<-` is the assignment operator
  - in other languages `=` is the assignment operator
- good practice to put a space before and after assignment operator

```r
# Create an object and assign value
a <- 5
a
#> [1] 5

b <- "yay!"
b
#> [1] "yay!"
```

Objects

Most statistics software (e.g., SPSS, Stata) operates on datasets, which consist of rows of observations and columns of variables

- Usually, these packages can open only one dataset at a time

R is an “object-oriented” programming language

- “Objects are like boxes in which we can put things: data, functions, and even other objects.” - Ben Skinner
- There are several different “types” of objects in R
  - A dataset is just one type of object in R
  - There is no limit to the number of objects R can hold (except memory)
  - R “functions” do different things to different types of objects

Vectors

The fundamental object in R is the “vector”

- A vector is a collection of values
- The individual values within a vector are called “elements”
- The values in a vector can be numeric, character (e.g., “Apple”), or any other type

Create a numeric vector that contains three elements

```
x <- c(4, 7, 9)
x
#> [1] 4 7 9
# examine help file for c() function
```

Vector where the elements are characters

```
animals <- c("lions", "tigers", "bears", "oh my")
animals
#> [1] "lions"  "tigers" "bears"  "oh my"
```

Formal classification of vectors in R

More formally, there are two broad types of vectors

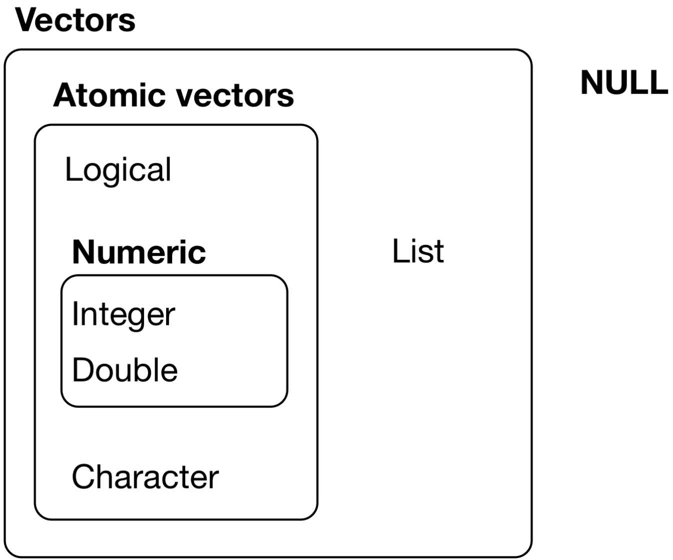
1. **Atomic vectors**. There are six types:
   - ▷ **logical**, **integer**, **double**, **character**, **complex**, and **raw**.
     - – **Integer** and **double** vectors are collectively known as **numeric** vectors.
2. **Lists**, which are sometimes called recursive vectors because lists can contain other lists.

Difference between atomic vectors and lists

- ○ atomic vectors are **homogeneous**: all elements within atomic vector must be of the same type
- ○ lists can be **heterogeneous**: e.g., one element can be an integer and another element can be character

Formal classification of vectors in R

PUT PIC ON WEBSITE AND PROVIDE LINK



Vectors

Atomic vectors

Logical

Numeric

Integer

Double

Character

List

NULL

Let’s develop an intuitive understanding of vector types

Technically, **lists** are a type of **vector**, but most people think of **atomic vectors** and **lists** as fundamentally different things

From now on, I’ll use the term **vector** to refer to atomic vectors

Rule for (atomic) vectors:

- all elements within a vector must have the same data “type”
- data types we will focus on in class:
    - numeric (integer and double)
    - character
    - logical

“Length” of a vector is the number of elements

Use `length()` function to examine vector length

```
x
#> [1] 4 7 9
length(x)
#> [1] 3

animals
#> [1] "lions"  "tigers" "bears"  "oh my"
length(animals)
#> [1] 4
```

A single number [or string] is a vector of length=1

```
z <- 5
length(z)
#> [1] 1
length("Tommy")
#> [1] 1
```

Aside: Sequences

A vector that contains a “sequence” of numbers (e.g., 1, 2, 3, 4) can be created with the notation `start:end`

```
0:4
#> [1] 0 1 2 3 4
99:104
#> [1]  99 100 101 102 103 104
w <- c(10:15)
w
#> [1] 10 11 12 13 14 15
length(w)
#> [1] 6
```

Data type of a vector

Three “types” of vectors, where type refers to the elements within the vector

- numeric: can be “integer” (e.g., 5) or “double” (e.g., 5.5)
- character (e.g., “ozan”)
- logical: TRUE or FALSE; more on this later

Use `typeof()` function to examine vector type

```
x
#> [1] 4 7 9
typeof(x)
#> [1] "double"

p <- c(1.5, 1.6)
p
#> [1] 1.5 1.6
typeof(p)
#> [1] "double"

animals
#> [1] "lions"  "tigers" "bears"  "oh my"
typeof(animals)
#> [1] "character"
```

Data type of a vector, numeric

Numeric vectors can be “integer” (e.g., 5) or “double” (e.g., 5.5)

```
typeof(1.5)
#> [1] "double"
```

R stores numbers as doubles by default.

```
x
#> [1] 4 7 9
typeof(x)
#> [1] "double"
```

To make an integer, place an `L` after the number:

```
typeof(5)
#> [1] "double"
typeof(5L)
#> [1] "integer"
```

Vector math

Most mathematical operations operate on each element of the vector

- e.g., add a single value to a vector and that value added to each element of the vector

```
1:3
#> [1] 1 2 3
1:3+.5
#> [1] 1.5 2.5 3.5
(1:3)*2
#> [1] 2 4 6
```

Mathematical operations involving two vectors have the same length

- DESCRIBE IN WORDS

```
c(1,1,1)+c(1,0,2)
#> [1] 2 1 3
c(1,1,1)*c(1,0,2)
#> [1] 1 0 2
```

All elements in (atomic) vector must have same data type.

“When you try and create a vector containing multiple types with `c()`: the most complex type always wins” - Wickham

```
1:3
#> [1] 1 2 3
typeof(1:3)
#> [1] "integer"

mix <- c(1:3, "hi!")
mix
#> [1] "1"   "2"   "3"   "hi!"
typeof(mix)
#> [1] "character"
```

Data type of a vector, logical

Logical vectors can take three possible values: `TRUE` , `FALSE` , `NA`

- `TRUE` , `FALSE` , `NA` are special keywords, different from the character strings `"TRUE"` , `"FALSE"` , `"NA"`
- Don’t worry about `"NA"` for now

```
typeof(TRUE)
#> [1] "logical"
typeof("TRUE")
#> [1] "character"

typeof(c(TRUE,FALSE,NA))
#> [1] "logical"
typeof(c(TRUE,FALSE,NA,"FALSE"))
#> [1] "character"
```

We’ll learn more about logical vectors later

Lists

What is a **list**?

- Like (atomic) vectors, a list is an object that contains **elements**
- Unlike vectors, data types can differ across elements within a list
- An element within a list can be another list
  - this characteristic makes lists more complicated than vectors
  - suitable for representing hierarchical data

Lists are more complicated than vectors; today we'll just provide a basic introduction

Create lists using `list()` function

Review: a vector

```
a <- c(1,2,3)
typeof(a)
#> [1] "double"
length(a)
#> [1] 3
```

A list

```
b <- list(1,2,3)
typeof(b)
#> [1] "list"
length(b)
#> [1] 3
b # print list is awkward
#> [[1]]
#> [1] 1
#>
#> [[2]]
#> [1] 2
#>
#> [[3]]
#> [1] 3
```

Investigate structure of lists using `str()` function

```
b <- list(1,2,3)
typeof(b)
#> [1] "list"
length(b)
#> [1] 3
str(b)
#> List of 3
#>  $ : num 1
#>  $ : num 2
#>  $ : num 3
```

Can also apply `str()` to vectors

```
a
#> [1] 1 2 3
str(a)
#>  num [1:3] 1 2 3
```

Elements within list can have different data types

```r
b <- list(1,2,"apple")
typeof(b)
#> [1] "list"
str(b)
#> List of 3
#>  $ : num 1
#>  $ : num 2
#>  $ : chr "apple"
```

Vector

```r
a <- c(1,2,"apple")
typeof(a)
#> [1] "character"
str(a)
#>  chr [1:3] "1" "2" "apple"
```

Lists can contain other lists

```
x1 <- list(1, list("apple", "orange"), list(1, 2, 3))

typeof(x1)
#> [1] "list"

str(x1)
#> List of 3
#>  $ : num 1
#>  $ :List of 2
#>   ..$ : chr "apple"
#>   ..$ : chr "orange"
#>  $ :List of 3
#>   ..$ : num 1
#>   ..$ : num 2
#>   ..$ : num 3
```

You can name each element in the list

```
x2 <- list(a=1, b=list("apple", "orange"), c=list(1, 2, 3))

str(x2)
#> List of 3
#>  $ a: num 1
#>  $ b:List of 2
#>   ..$ : chr "apple"
#>   ..$ : chr "orange"
#>  $ c:List of 3
#>   ..$ : num 1
#>   ..$ : num 2
#>   ..$ : num 3
```

`names()` function shows names of elements in the list

```
names(x2) # has names
#> [1] "a" "b" "c"
names(x1) # no names
#> NULL
```

Access individual elements in a “named” list

Syntax: `list_name$element_name`

```r
x2 <- list(a=1, b=list("apple", "orange"), c=list(1, 2, 3))
typeof(x2$a)
#> [1] "double"
length(x2$a)
#> [1] 1

typeof(x2$b)
#> [1] "list"
length(x2$b)
#> [1] 2

typeof(x2$c)
#> [1] "list"
length(x2$c)
#> [1] 3
```

Note: length can differ across elements within a list

Compare structure of list to structure of element within a list

```r
str(x2)
#> List of 3
#>  $ a: num 1
#>  $ b:List of 2
#>   ..$ : chr "apple"
#>   ..$ : chr "orange"
#>  $ c:List of 3
#>   ..$ : num 1
#>   ..$ : num 2
#>   ..$ : num 3

str(x2$c)
#> List of 3
#>  $ : num 1
#>  $ : num 2
#>  $ : num 3
```

A dataset is just a list!

A data frame is a list with the following characteristics:

- Data type can differ across elements (like all lists)
- Each **element** in data frame must be a **vector**, not a **list**
  - Each element (column) is a variable
- Each **element** in a data frame must have the same length
  - The length of an element is the number of observations (rows)
  - so each variable in data frame must have same number of observations

```
names(df)
#> [1] "mpg" "cyl" "hp"
head(df, n=5) # print first 5 rows
#> # A tibble: 5 x 3
#>     mpg   cyl    hp
#> * <dbl> <dbl> <dbl>
#> 1  21       6   110
#> 2  21       6   110
#> 3  22.8     4    93
#> 4  21.4     6   110
#> 5  18.7     8   175
```

A data frame is a named list

```
typeof(df)
#> [1] "list"
names(df)
#> [1] "mpg" "cyl" "hp"
length(df) # length=number of variables
#> [1] 3
str(df)
#> 'data.frame':    32 obs. of  3 variables:
#>  $ mpg: num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#>  $ cyl: num  6 6 4 6 8 6 8 4 4 6 ...
#>  $ hp : num  110 110 93 110 175 105 245 62 95 123 ...
```

Like any named list, can examine the elements

```
typeof(df$mpg)
#> [1] "double"
length(df$mpg) # length=number of rows/obs
#> [1] 32
str(df$mpg)
#>  num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
```

Main takeaways on types of objects – vectors and lists

Basic data stuctures

1. **(Atomic) vectors**: **logical**, **integer**, **double**, **character**.
   - each element in vector must have same data type
2. **Lists**:
   - Data type can differ across elements

Takeaways

- These concepts are difficult; ok to feel confused
- I will reinforce these concepts throughout the course
- Good practice: run simple diagnostics on any new object
  - `length()` : how many **elements** in the object
  - `typeof()` : what **type** of data is the object
  - `str()` : hierarchical structure of the object

Main takeaways on types of objects – vectors and lists

Basic data stuctures

1. **(Atomic) vectors**: **logical**, **integer**, **double**, **character**.
   - each element in vector must have same data type
2. **Lists**:
   - Data type can differ across elements

Takeaways

- These data structures (vectors, lists) and data types (e.g., character, numeric, logical) are the basic building blocks of all object oriented programming languages
- Application to statistical analysis
  - Datasets are just lists
  - The individual elements – columns/variables – within a dataset are just vectors
- These structures and data types are foundational for all "data science" applications, e.g.,:
  - maapping, webscraping, network analysis, etc.