The R Language

MY 470, Week 8: Introduction and Data Structures

Kenneth Benoit

2019-11-18

About me

- Professor of Computational Social Science. Dept. of Methodology
- My research
  - Political communication, party competition
  - Text as data methods, natural language processing
  - Author of R packages to analyze text (incl. **quanteda**)
- Contact: K.R.Benoit@lse.ac.uk
  - Office hours: Tuesdays 15:30-17.00, Wednesdays 10-11:00

Introduction to R

R in a nutshell

R is a versatile, open source programming/scripting language that's useful both for statistics but also data science. Inspired by the programming language S.

- Open source software under GPL.
- Superior (if not just comparable) to commercial alternatives. R has over 10,000 user contributed packages (CRAN) and many more elsewhere.
- Available on all platforms.
- Not just for statistics, but also general purpose programming, graphics, network analysis, machine learning, web scraping...
- Is object oriented and functional.
- Large and growing community of peers.

How to run R

You can run R interactively or in batch mode.

Interactive mode*

e.g. type in R from the shell. The window that appears is called the R console. Any command you type into the prompt is interpreted by the R kernel. An output may or may not be printed to the screen depending on the types of commands that you run.

From the R GUI console

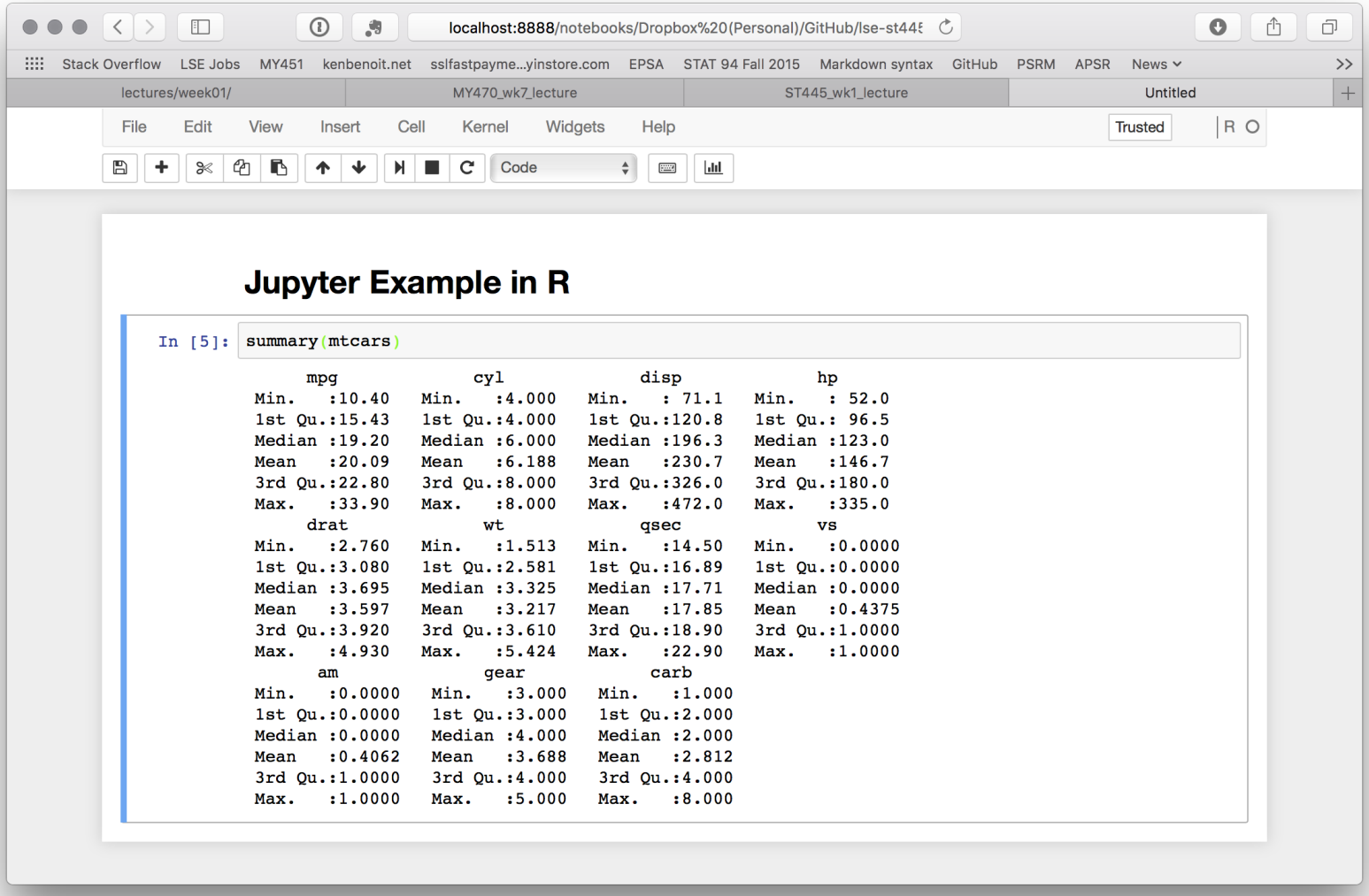
This is installed when you install R.

Preferred: RStudio

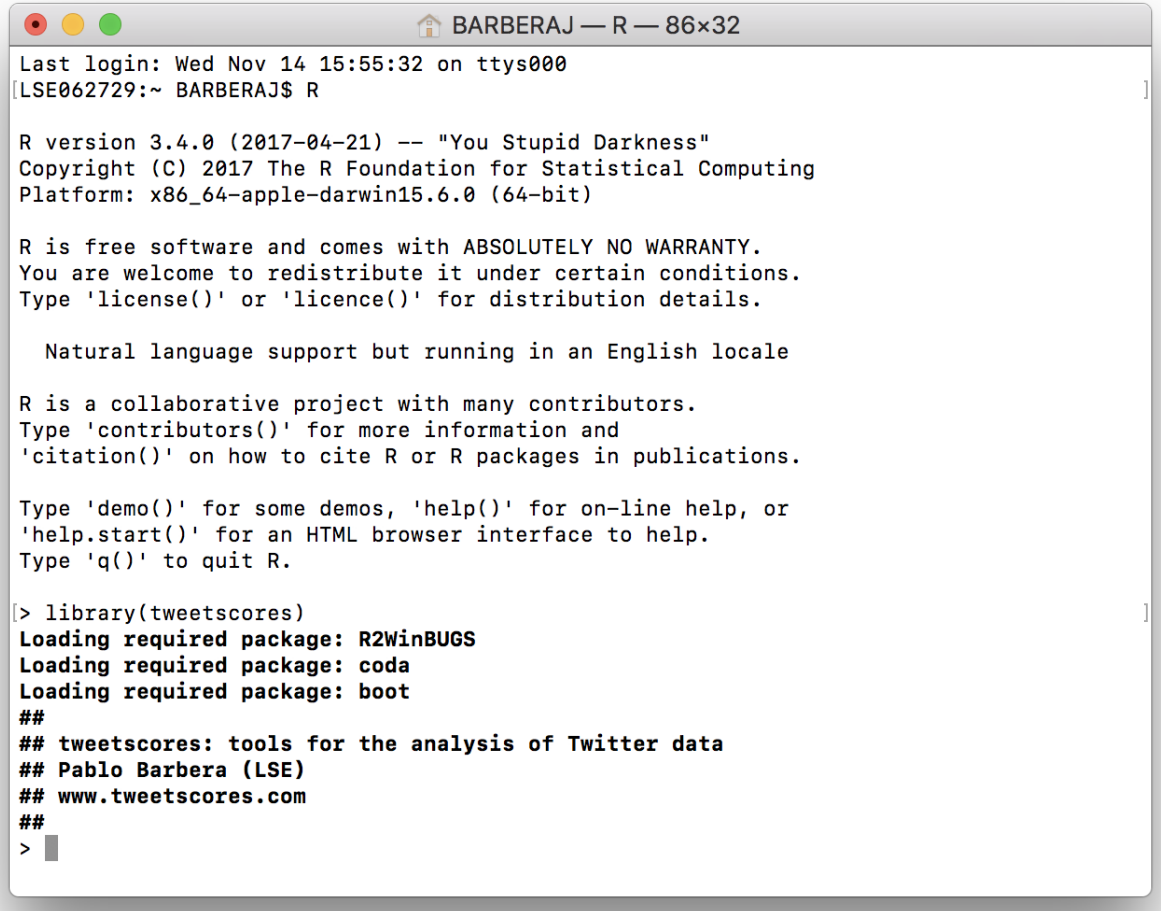
RStudio is an IDE (Integrated Development Environment) that makes everything fantastically easier.

From Jupyter

You need to change the kernel to R.



Terminal


BARBERAJ — R — 86×32
Last login: Wed Nov 14 15:55:32 on ttys000
LSE062729:~ BARBERAJ$ R
R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> library(tweetscores)
Loading required package: R2WinBUGS
Loading required package: coda
Loading required package: boot
##
tweetscores: tools for the analysis of Twitter data
Pablo Barbera (LSE)
www.tweetscores.com
##
>

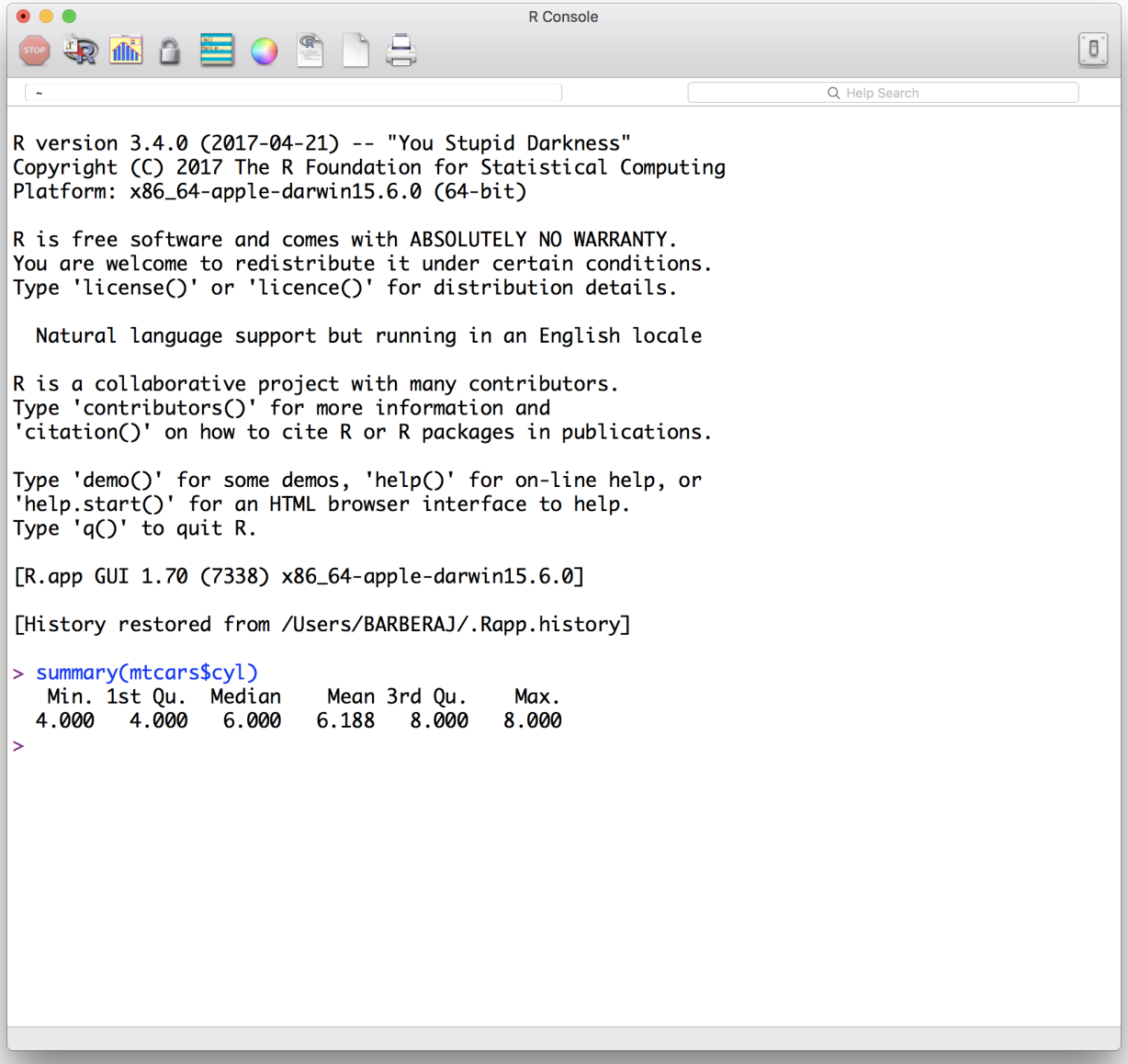
Batch mode

You can also run one or more R scripts in batch mode.

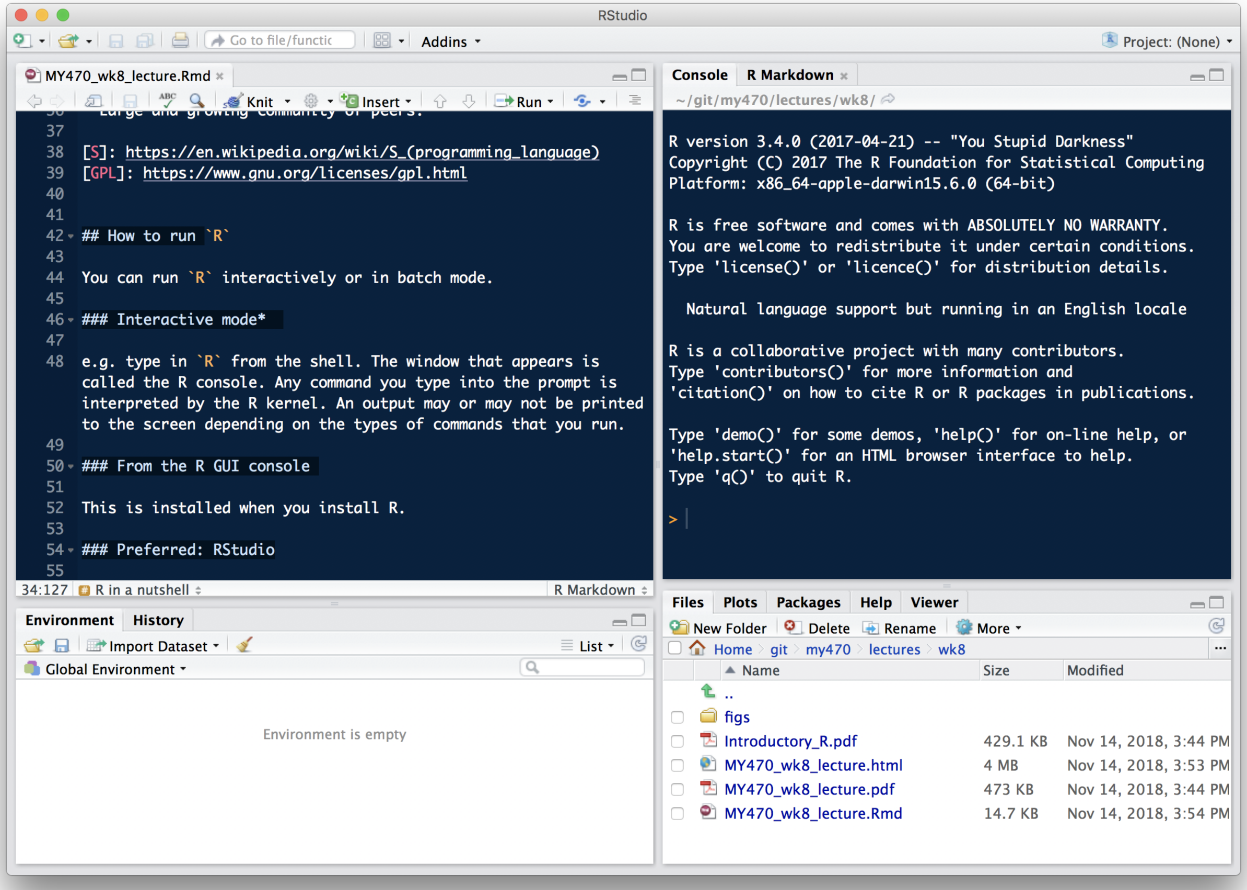
```
R CMD BATCH script_1.R script_2.R
```

You can also script inline using `Rscript -e`.

R GUI Console



RStudio


RStudio
Addins
Project: (None)
MY470_wk8_lecture.Rmd
Knit
Insert
Run
[S]: https://en.wikipedia.org/wiki/S_(programming_language)
[GPL]: https://www.gnu.org/licenses/gpl.html
How to run `R`
You can run `R` interactively or in batch mode.
Interactive mode*
e.g. type in `R` from the shell. The window that appears is called the R console. Any command you type into the prompt is interpreted by the R kernel. An output may or may not be printed to the screen depending on the types of commands that you run.
From the R GUI console
This is installed when you install R.
Preferred: RStudio
34:127
R in a nutshell
R Markdown
Console
R Markdown
~/git/my470/lectures/wk8/
R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment
History
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
git
my470
lectures
wk8
Name
Size
Modified
..
figs
Introductory_R.pdf
429.1 KB
Nov 14, 2018, 3:44 PM
MY470_wk8_lecture.html
4 MB
Nov 14, 2018, 3:53 PM
MY470_wk8_lecture.pdf
473 KB
Nov 14, 2018, 3:44 PM
MY470_wk8_lecture.Rmd
14.7 KB
Nov 14, 2018, 3:54 PM

R Operators

Basic operations in R

```
1 + 1
```

```
## [1] 2
```

```
5 - 3
```

```
## [1] 2
```

```
6 / 2
```

```
## [1] 3
```

```
4 * 4
```

```
## [1] 16
```

Basic operations in R

Other typical mathematical functions are also hard-coded:

```
log(<number>)
exp(<number>)
sqrt(<number>)
mean(<numbers>)
sum(<numbers>)
```

Basic operations in R

R also understands logical operators:

- `<` : less than
- `>` : greater than
- `==` : equal to (note, not =)
- `>=` : greater than or equal to
- `<=` : less than or equal to
- `!=` : not equal to
- `&` : and
- `|` : or

R objects

- R is an object-oriented programming language
- The entities that R creates and manipulates are known as objects
- These may be variables, arrays of numbers, character strings, functions
- Objects are created using the *assignment operator*: <-
- Once created, an object is stored in your current *workspace*

```
my_object <- 10
print(my_object)
```

```
## [1] 10
```

```
my_other_object <- 4
print(my_object - my_other_object)
```

```
## [1] 6
```

Viewing objects in your global environment and how to clean them up

List objects in your current environment

```
ls()
```

remove objects from your current environment

```
x <- 5
ls()
```

```
## [1] "my_object"       "my_other_object" "x"
```

```
rm(x)
ls()
```

```
## [1] "my_object"       "my_other_object"
```

remove all objects from your current environment

```
rm(list = ls())
```

- Notice that we have nested one function inside another.

Package management

- `install.packages("package-name")` will download a package from one of the CRAN mirrors assuming that a binary is available for your operating system. If you have not set a preferred CRAN mirror in your `options()`, then a menu will pop up asking you to choose a location.
- Use `old.packages()` to list all your locally installed packages that are now out of date. `update.packages()` - will update all packages in the known libraries interactively. This can take a while if you haven't done it recently. To update everything without any user intervention, use the `ask = FALSE` argument.

```
update.packages(ask = FALSE)
```

Quitting R

type in `quit()` or `q()` and answer `y` to quit.

R versus Python

Data Types

Python class	Immutable	Description	R cla
bool	Yes	Boolean value	logica
int	Yes	integer number	intege
float	Yes	floating-point number	numer
list	No	mutable sequence of objects	list
tuple	Yes	immutable sequence of objects	-
str	Yes	character string	charac
set	No	unordered set of distinct objects	-
frozenset	Yes	immutable form of set class	-
dict	No	dictionary	(named)

Indexing

R indexes from 1 (not 0)

- where an index begins counting, when addressing elements of a data object
- most languages index from 0
- human ages - do they index from 0? what about months of the year?
- buildings in Europe vs the US

R indexes include the ending index

- so `myvector[1:3]` includes the first, second, and third elements in R
- whereas in Python, `[1:3]` is up to, but not including, the ending element index

R does not index individual character elements of a string

Python

```
letters = "abcdefghijklmnopqrztuv"
print letters[0:4]
```

```
## abcd
```

R

```
letters[1:5]
```

```
## [1] "a" "b" "c" "d" "e"
```

```
firstletters <- "abcdefg"
firstletters[1:3]
```

```
## [1] "abcdefg" NA        NA
```

Why did this happen?

Data Structures

The Importance of Understanding Object Types

To make the best of the R language, you'll need a strong understanding of the basic data types and data structures and how to operate on those.

It is **critical** to understand because these are the objects you will manipulate on a day-to-day basis in R. Dealing with object conversions is one of the most common sources of frustration for beginners.

> *To understand computations in R, two slogans are helpful: - Everything that exists is an object. - Everything that happens is a function call. – John Chambers*

Atomic Classes

`R` has 6 (excluding the `raw` class for this lecture) atomic classes.

- character
- numeric (real or decimal)
- integer
- logical
- complex

Example	Type
“a”, “swc”	character
2, 15.5	numeric
2L (Must add a `L` at end to denote integer)	integer
`TRUE`, `FALSE`	logical
1+4i	complex

Data Structures

R has many data structures. These include:

- atomic vector
- list
- matrix
- data frame
- factors

`vector` - the basic building block of R

A vector is a collection of entites which all share the same type.

A vector is the most common and basic data structure in `R` and is pretty much the workhorse of R. Technically, vectors can be one of two types:

- atomic vectors
- lists

We use the `c()` function to concatenate observations into a single vector.

Numeric

```
num_vec <- c(5, 4, 2, 100, 7.65)
print(num_vec)
```

```
## [1]   5.00   4.00   2.00 100.00   7.65
```

Character

```
char_vec <- c("apple", "pear", "plumb", "pineapple", "straw
print(char_vec)
```

```
## [1] "apple"     "pear"      "plumb"     "pineapple"
```

Atomic Vectors

A vector can be a vector of elements that are most commonly `character`, `logical`, `integer` or `numeric`.

You can create an empty vector with `vector()` (By default the mode is `logical`. You can be more explicit as shown in the examples below.) It is more common to use direct constructors such as `character()`, `numeric()`, etc.

```
x <- vector()
# with a length and type
vector("character", length = 10)
```

```
##  [1] "" "" "" "" "" "" "" "" "" ""
```

```r
vector("character", length = 10)
```

```
##  [1] "" "" "" "" "" "" "" "" "" ""
```

```r
character(5) ## character vector of length 5
```

```
## [1] "" "" "" "" ""
```

```r
numeric(5)
```

```
## [1] 0 0 0 0 0
```

```r
logical(5)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

Various examples:

```
x <- c(1, 2, 3)
x
```

```
## [1] 1 2 3
```

```
length(x)
```

```
## [1] 3
```

x is a numeric vector. These are the most common kind. They are numeric objects and are treated as double precision real numbers. To explicitly create integers, add an L at the end.

```
x1 <- c(1L, 2L, 3L)
```

You can also have logical vectors.

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

Finally you can have character vectors:

```
z <- c("Pablo", "Milena", "Tom", "Jon")
z
```

```
## [1] "Pablo"  "Milena" "Tom"    "Jon"
```

Examine your vector

```
typeof(z)
```

```
## [1] "character"
```

```
length(z)
```

```
## [1] 4
```

```
class(z)
```

```
## [1] "character"
```

```
str(z)
```

```
##  chr [1:4] "Pablo" "Milena" "Tom" "Jon"
```

Adding elements

```
z <- c(z, "Anett")
z
```

```
## [1] "Pablo"  "Milena" "Tom"    "Jon"    "Anett"
```

More examples of vectors

```
x <- c(0.5, 0.7)
x <- c(TRUE, FALSE)
x <- c("a", "b", "c", "d", "e")
x <- 9:100
x <- c(1+0i, 2+4i)
```

You can also create vectors as a sequence of numbers

```
series <- 1:10
seq(10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
seq(1, 10, by = 0.1)
```

```
##  [1]  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9
## [15]  2.4  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3
## [29]  3.8  3.9  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7
## [43]  5.2  5.3  5.4  5.5  5.6  5.7  5.8  5.9  6.0  6.1
## [57]  6.6  6.7  6.8  6.9  7.0  7.1  7.2  7.3  7.4  7.5
## [71]  8.0  8.1  8.2  8.3  8.4  8.5  8.6  8.7  8.8  8.9
## [85]  9.4  9.5  9.6  9.7  9.8  9.9 10.0
```

Other objects

`Inf` is infinity. You can have either positive or negative infinity.

```
1 / 0
```

```
## [1] Inf
```

```
1 / Inf
```

```
## [1] 0
```

`NaN` means Not a number. It's an undefined value.

```
0 / 0
```

```
## [1] NaN
```

Object Attributes and Classes

Object Attributes

Each object can have attributes. Attribues can be part of an object of R. These include:

- names
- dimnames
- dim
- class
- attributes (contain metadata)

You can also glean other attribute-like information such as length (works on vectors and lists) or number of characters (for character strings).

```
length(1:10)
```

```
## [1] 10
```

```
nchar("London")
```

```
## [1] 6
```

Querying object attributes

```
typeof()      # what is it?
length()      # how long is it? What about two dimensional o
attributes() # does it have any metadata?
```

Many objects have attributes attached to them that are necessary to their definition.

Example: `matrix`

A `matrix` object in R is simply a vector with row and column attributes.

```
m <- matrix(1:6, nrow = 2)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
attributes(m)
```

```
## $dim
## [1] 2 3
```

```
length(m)
```

```
## [1] 6
```

Example

```
x <- "dataset"
typeof(x)
```

```
## [1] "character"
```

```
attributes(x)
```

```
## NULL
```

```
y <- 1:10
typeof(y)
```

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

```
length(y)
```

```
## [1] 10
```

```r
attributes(y)
```

```
## NULL
```

```r
z <- c(1L, 2L, 3L)
typeof(z)
```

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

Factors: Vectors with labels

A factor is similar to a character vector, but here each unique element is associated with a numerical value which represents a category:

```
fac_vec <- as.factor(c("a", "b","c","a","b","c"))
fac_vec
```

```
## [1] a b c a b c
## Levels: a b c
```

```
as.numeric(fac_vec)
```

```
## [1] 1 2 3 1 2 3
```

Vectorized operations

vector subsetting

To subset a `vector`, use square parenthesis to index the elements you would like via `object[index]`:

- Numerical subsetting

```
char_vec
```

```
## [1] "apple"     "pear"      "plumb"     "pineapple"
```

```
char_vec[1:3]
```

```
## [1] "apple" "pear"  "plumb"
```

vector subsetting

- Logical subsetting

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

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

```
char_vec[logical_vec]
```

```
## [1] "apple"     "pineapple"
```

vector operations

In R, mathematical operations on vectors occur elementwise:

```
fib <- c(1, 1, 2, 3, 5, 8, 13, 21)
fib[1:7]
```

```
## [1]  1  1  2  3  5  8 13
```

```
fib[2:8]
```

```
## [1]  1  2  3  5  8 13 21
```

```
fib[1:7] + fib[2:8]
```

```
## [1]  2  3  5  8 13 21 34
```

Recycling

R likes to operate on vectors of the same length, so if it encounters two vectors of different lengths in a binary operation, it *replicates* (recycles) the smaller vector until it is the same length as the longest vector, then it does the operation.

If the recycled smaller vector has to be "chopped off" to make it the length of the longer vector, you will get a warning, but it will still return a result:

```
x <- c(1, 2, 3)
y <- c(1, 10)
x * y
```

```
## Warning in x * y: longer object length is not a multiple
## length
```

```
## [1]  1 20  3
```

(recyling continued)

Recycling occurs in other contexts too:

- With mathematical operators (as above)
- Whenever two or more vectors get operated on elementwise, not just with comparison operators
- With indexing operators when indexing by logical vectors

```r
x <- 1:20

x * c(1, 0)     # turns the even numbers to 0
```

```
##  [1]  1  0  3  0  5  0  7  0  9  0 11  0 13  0 15  0 17
```

```r
x * c(0, 0, 1) # turns non-multiples of 3 to 0
```

```
## Warning in x * c(0, 0, 1): longer object length is not a
## shorter object length
```

```
##  [1]  0  0  3  0  0  6  0  0  9  0  0 12  0  0 15  0  0
```

```r
x < ((1:4) ^ 2)  # recycling c(1, 4, 9, 16)
```

```
##  [1] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FAI
## [12]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAI
```

vector operations

It is also possible to perform logical operations on vectors:

```
fib <- c(1, 1, 2, 3, 5, 8, 13, 21)
fib_greater_five <- fib > 5
print(fib_greater_five)
```

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

And we can combine logical operators

```
fib_greater_five_less_two <- fib > 5 | fib < 2
print(fib_greater_five_less_two)
```

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

matrix

A matrix is just a collection of vectors! I.e. it is a multidimensional vector where all elements are of the same type

```
my_matrix <- matrix(data = 1:100, nrow = 10, ncol = 10)
my_matrix
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1   11   21   31   41   51   61   71   81    91
##  [2,]    2   12   22   32   42   52   62   72   82    92
##  [3,]    3   13   23   33   43   53   63   73   83    93
##  [4,]    4   14   24   34   44   54   64   74   84    94
##  [5,]    5   15   25   35   45   55   65   75   85    95
##  [6,]    6   16   26   36   46   56   66   76   86    96
##  [7,]    7   17   27   37   47   57   67   77   87    97
##  [8,]    8   18   28   38   48   58   68   78   88    98
##  [9,]    9   19   29   39   49   59   69   79   89    99
## [10,]   10   20   30   40   50   60   70   80   90   100
```

data.frame

A `data.frame` is a matrix-like R object in which the columns can be of different types

```
my_data_frame <- data.frame(numbers = num_vec, fruits = cha
my_data_frame
```

```
##   numbers      fruits logical
## 1    5.00       apple    TRUE
## 2    4.00        pear   FALSE
## 3    2.00       plumb   FALSE
## 4  100.00  pineapple    TRUE
## 5    7.65 strawberry   FALSE
```

matrix and data.frame subsetting

To subset a matrix or data.frame, you need to specify both rows and columns:

```
my_matrix[1:3, 1:3]
```

```
##      [,1] [,2] [,3]
## [1,]    1   11   21
## [2,]    2   12   22
## [3,]    3   13   23
```

```
my_data_frame[1, ]
```

```
##   numbers fruits logical
## 1       5  apple    TRUE
```

matrix and data.frame subsetting

You can filter data frames using variables' names:

```
my_data_frame$numbers
```

```
## [1]   5.00   4.00   2.00 100.00   7.65
```

```
my_data_frame$numbers[1:3]
```

```
## [1] 5 4 2
```

matrix and data.frame subsetting

We can also subset to remove rows or columns that we do not want to see by using the - operator applied to the c function:

```
my_matrix[-c(1:3), -c(1:3)]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]   34   44   54   64   74   84   94
## [2,]   35   45   55   65   75   85   95
## [3,]   36   46   56   66   76   86   96
## [4,]   37   47   57   67   77   87   97
## [5,]   38   48   58   68   78   88   98
## [6,]   39   49   59   69   79   89   99
## [7,]   40   50   60   70   80   90  100
```

In the code, 1:3 creates a vector of the integers 1, 2 and 3, and the - operator negates these. We wrap the vector in the c function so that - applies to each element, and not just the first element.

list

A `list` is a collection of any set of object types

```
my_list <- list(something = num_vec,
                another_thing = my_matrix[1:3,1:3],
                something_else = "ken")
my_list
```

```
## $something
## [1]   5.00   4.00   2.00 100.00   7.65
##
## $another_thing
##      [,1] [,2] [,3]
## [1,]    1   11   21
## [2,]    2   12   22
## [3,]    3   13   23
##
## $something_else
## [1] "ken"
```

Style Guide

Formatting code nicely in R

- The *Tidyverse Style Guide*
- The **lintr** package
- Works great from RStudio

Exercise Week 8 Preview