MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

1º Semestre - 2020

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

NÃO ESQUEÇA DE INICIAR A GRAVAÇÃO

MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

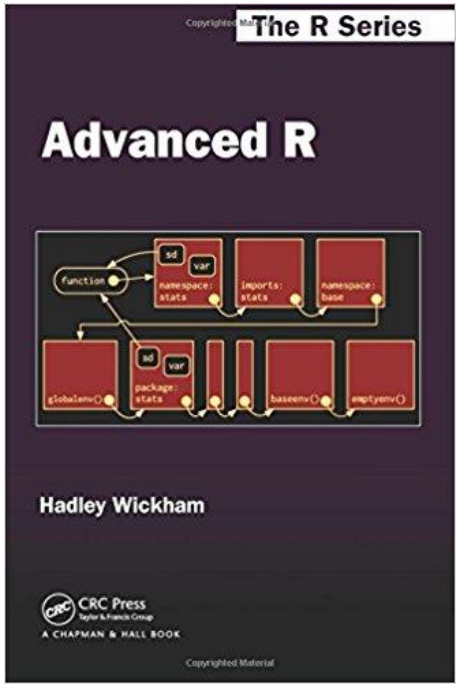
1º Semestre - 2020

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

ROTEIRO (continuação)

Material Prof. Roger Peng
http://www.biostat.jhsph.edu/~rpeng/



Esse material é fortemente baseado no livro

Advanced R (Chapman & Hall/CRC The R Series)

de Hadley Wickham (http://hadley.nz/) o Cientista-chefe do RStudio

Data structures

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

Subsetting

Subsetting

R's subsetting operators are powerful and fast. Mastery of subsetting allows you to succinctly express complex operations in a way that few other languages can match. Subsetting is hard to learn because you need to master a number of interrelated concepts:

- The three subsetting operators.
- The six types of subsetting.
- Important differences in behaviour for different objects (e.g., vectors, lists, factors, matrices, and data frames).
- The use of subsetting in conjunction with assignment.

Subsetting

There are a number of operators that can be used to extract subsets of R objects.

- `[` always returns an object of the same class as the original; can be used to select more than one element (there is one exception)
- `[[` is used to extract elements of a list or a data frame; it can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame
- `$` is used to extract elements of a list or data frame by name; semantics are similar to that of `[[`.

Subsetting

```
> x <- c("a", "b", "c", "c", "d", "a")
> x[1]
[1] "a"
> x[2]
[1] "b"
> x[1:4]
[1] "a" "b" "c" "c"
> x[x > "a"]
[1] "b" "c" "c" "d"
> u <- x > "a"
> u
[1] FALSE TRUE TRUE TRUE TRUE FALSE
> x[u]
[1] "b" "c" "c" "d"
```

Atomic vectors

```
x <- c(2.1, 4.2, 3.3, 5.4)
```

- **Positive integers** return elements at the specified positions:

```
x[c(3, 1)]
#> [1] 3.3 2.1
x[order(x)]
#> [1] 2.1 3.3 4.2 5.4

# Duplicated indices yield duplicated values
x[c(1, 1)]
#> [1] 2.1 2.1

# Real numbers are silently truncated to integers
x[c(2.1, 2.9)]
#> [1] 4.2 4.2
```

```
> x <- c(2.1,4.2,3.3,5.4)
> order(x)
[1] 1 3 2 4
>
```

```
x <- c(2.1, 4.2, 3.3, 5.4)
```

- **Negative integers** omit elements at the specified positions:

```
x[-c(3, 1)]
#> [1] 4.2 5.4
```

You can't mix positive and negative integers in a single subset:

```
x[c(-1, 2)]
#> Error: only 0's may be mixed with negative subscripts
```

```
x <- c(2.1, 4.2, 3.3, 5.4)
```

- **Logical vectors** select elements where the corresponding logical value is `TRUE`. This is probably the most useful type of subsetting because you write the expression that creates the logical vector:

```
x[c(TRUE, TRUE, FALSE, FALSE)]
#> [1] 2.1 4.2
x[x > 3]
#> [1] 4.2 3.3 5.4
```

If the logical vector is shorter than the vector being subsetted, it will be *recycled* to be the same length.

```
x[c(TRUE, FALSE)]
#> [1] 2.1 3.3
# Equivalent to
x[c(TRUE, FALSE, TRUE, FALSE)]
#> [1] 2.1 3.3
```

```
x <- c(2.1, 4.2, 3.3, 5.4)
```

A missing value in the index always yields a missing value in the output:

```
x[c(TRUE, TRUE, NA, FALSE)]
#> [1] 2.1 4.2  NA
```

- **Nothing** returns the original vector. This is not useful for vectors but is very useful for matrices, data frames, and arrays. It can also be useful in conjunction with assignment.

```
x[]
#> [1] 2.1 4.2 3.3 5.4
```

- **Zero** returns a zero-length vector. This is not something you usually do on purpose, but it can be helpful for generating test data.

```
x[0]
#> numeric(0)
```

```
x <- c(2.1, 4.2, 3.3, 5.4)
```

If the vector is named, you can also use:

- **Character vectors** to return elements with matching names.

```
(y <- setNames(x, letters[1:4]))
#>   a   b   c   d
#> 2.1 4.2 3.3 5.4
y[c("d", "c", "a")]
#>   d   c   a
#> 5.4 3.3 2.1

# Like integer indices, you can repeat indices
y[c("a", "a", "a")]
#>   a   a   a
#> 2.1 2.1 2.1

# When subsetting with [ names are always matched exactly
z <- c(abc = 1, def = 2)
z[c("a", "d")]
#> <NA> <NA>
#>   NA   NA
```

3.1.2 Lists

Subsetting a list works in the same way as subsetting an atomic vector. Using `[` will always return a list; `[[` and `$`, as described below, let you pull out the components of the list.

Subsetting Lists

```
> x <- list(foo = 1:4, bar = 0.6)
> x[1]
$foo
[1] 1 2 3 4

> x[[1]]
[1] 1 2 3 4

> x$bar
[1] 0.6
> x[["bar"]]
[1] 0.6
> x["bar"]
$bar
[1] 0.6
```

Console Terminal

```
> x <- list(foo=1:4, bar=0.6)
> a <- x[1]
> a
$foo
[1] 1 2 3 4

> class(a)
[1] "list"
> b<- x[[1]]
> b
[1] 1 2 3 4
> class(b)
[1] "integer"
>
```

Subsetting Lists

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> x[c(1, 3)]
$foo
[1] 1 2 3 4

$baz
[1] "hello"
```

Subsetting Lists

The `[[` operator can be used with *computed* indices; `$` can only be used with literal names.

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> name <- "foo"
> x[[name]]  ## computed index for 'foo'
[1] 1 2 3 4
> x$name     ## element 'name' doesn't exist!
NULL
> x$foo
[1] 1 2 3 4  ## element 'foo' does exist
```

Subsetting Nested Elements of a List

The `[[` can take an integer sequence.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
> x[[c(1, 3)]]
[1] 14
> x[[1]][[3]]
[1] 14

> x[[c(2, 1)]]
[1] 3.14
```

Console Terminal

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
> aa <- x[1]
> aa
$a
$a[[1]]
[1] 10

$a[[2]]
[1] 12

$a[[3]]
[1] 14


> class(aa)
[1] "list"
> bb<- x[2]
> bb
$b
[1] 3.14 2.81

> class(bb)
[1] "list"
> cc <- x[[2]]
> cc
[1] 3.14 2.81
> class(cc)
[1] "numeric"
>
```

Console Terminal

~/

```
> x <- list(a=list(10,12,14),b=c(3.14,2.81))
> x[[c(1,3)]]
[1] 14
> x[c(1,3)]
$a
$a[[1]]
[1] 10

$a[[2]]
[1] 12

$a[[3]]
[1] 14


$<NA>
NULL

>
```

O que aconteceu ?

3.1.3 Matrices and arrays

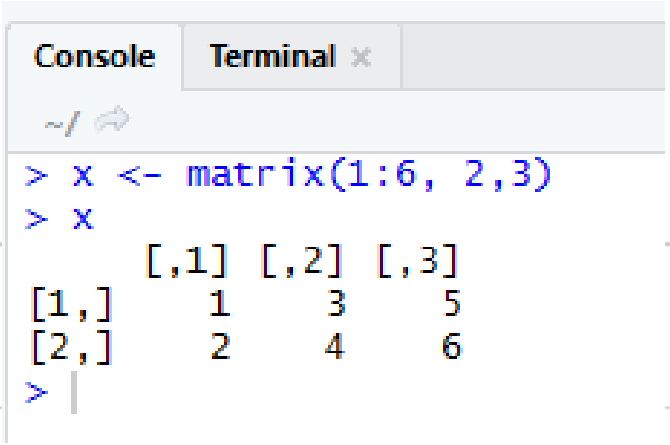
You can subset higher-dimensional structures in three ways:

- With multiple vectors.
- With a single vector.
- With a matrix.

Subsetting a Matrix

Matrices can be subsetted in the usual way with (*i,j*) type indices.

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
[1] 3
> x[2, 1]
[1] 2
```



Indices can also be missing.

```
> x[1, ]
[1] 1 3 5
> x[, 2]
[1] 3 4
```

Subsetting a Matrix

By default, when a single element of a matrix is retrieved, it is returned as a vector of length 1 rather than a 1 × 1 matrix. This behavior can be turned off by setting `drop = FALSE`.

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
[1] 3
> x[1, 2, drop = FALSE]
     [,1]
[1,] 3
```

Subsetting a Matrix

Similarly, subsetting a single column or a single row will give you a vector, not a matrix (by default).

```
> x <- matrix(1:6, 2, 3)
> x[1, ]
[1] 1 3 5
> x[1, , drop = FALSE]
     [,1] [,2] [,3]
[1,]    1    3    5
```

3.1.4 Data frames

Data frames possess the characteristics of both lists and matrices: if you subset with a single vector, they behave like lists; if you subset with two vectors, they behave like matrices.

Console Terminal

```
~/
> df <- data.frame(x = 1:3, y=3:1, z = letters[1:3])
> df
  x y z
1 1 3 a
2 2 2 b
3 3 1 c
> df[df$x == 2,]
  x y z
2 2 2 b
> df[c(1,3)]
  x z
1 1 a
2 2 b
3 3 c
>
```

```
> a <- df[df$x == 2,]
> class(a)
[1] "data.frame"
> b <- df[c(1,3)]
> class(b)
[1] "data.frame"
>
```

```
# There are two ways to select columns from a data frame

# Like a list:
df[c("x", "z")]
#>   x z
#> 1 1 a
#> 2 2 b
#> 3 3 c
# Like a matrix
df[, c("x", "z")]
#>   x z
#> 1 1 a
#> 2 2 b
#> 3 3 c

# There's an important difference if you select a single
# column: matrix subsetting simplifies by default, list
# subsetting does not.
str(df["x"])
#> 'data.frame':    3 obs. of  1 variable:
#>  $ x: int  1 2 3
str(df[, "x"])
#>  int [1:3] 1 2 3
```

Partial Matching

Partial matching of names is allowed with `[[` and `$`.

```
> x <- list(aardvark = 1:5)
> x$a
[1] 1 2 3 4 5
> x[["a"]]
NULL
> x[["a", exact = FALSE]]
[1] 1 2 3 4 5
```

Removing NA Values

A common task is to remove missing values (**NA**s).

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x)
> x[!bad]
[1] 1 2 4 5
```

```
x <- c(1, 2, NA, 4, NA, 5)
> is.na(x)
[1] FALSE FALSE TRUE FALSE TRUE FALSE
> bad <- is.na(x)
> bad
[1] FALSE FALSE TRUE FALSE TRUE FALSE
> !bad
[1] TRUE TRUE FALSE TRUE FALSE TRUE
> x[!bad]
[1] 1 2 4 5
```

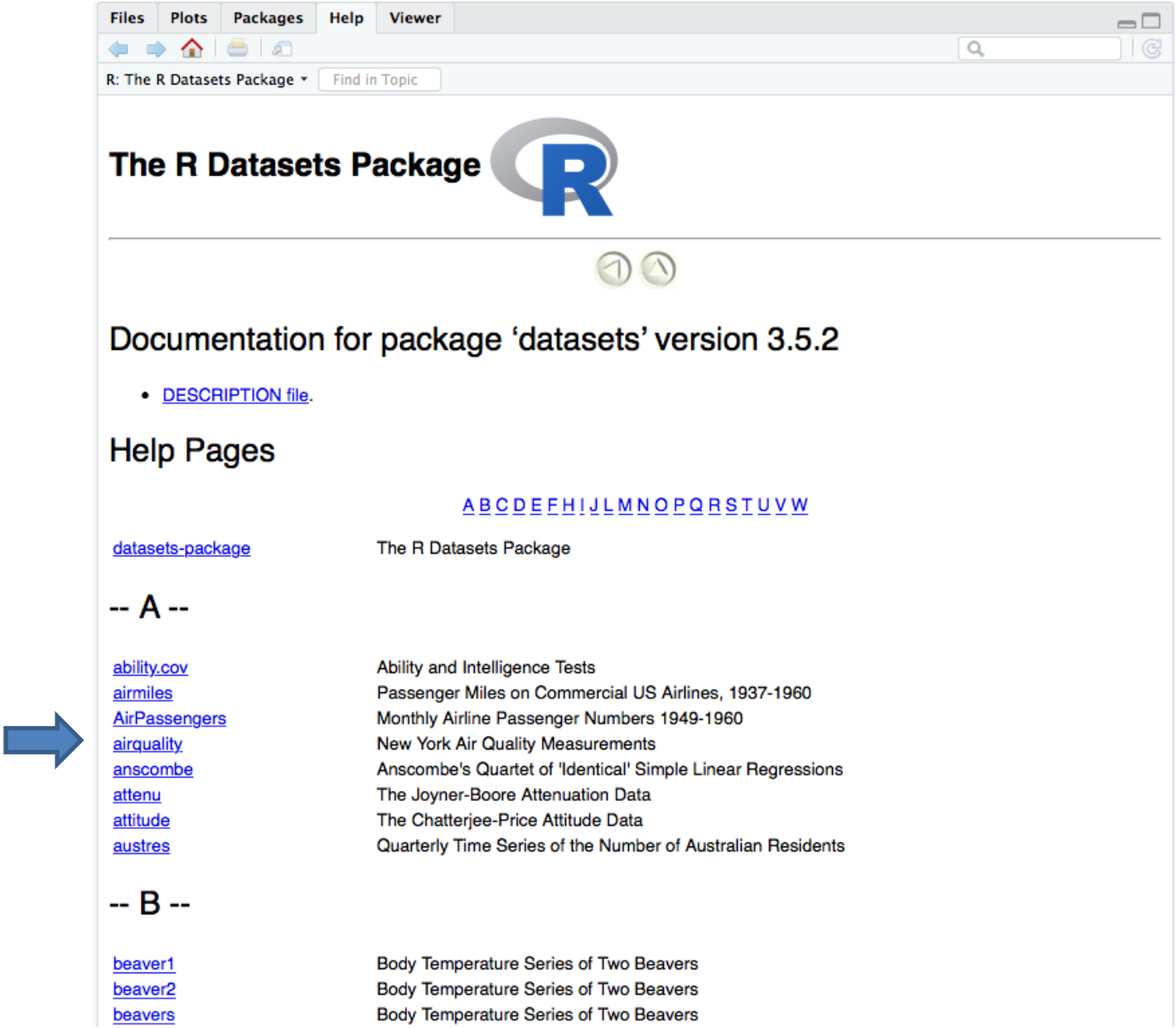
Removing NA Values

What if there are multiple things and you want to take the subset with no missing values?

```
> x <- c(1, 2, NA, 4, NA, 5)
> y <- c("a", "b", NA, "d", NA, "f")
> good <- complete.cases(x, y)
> good
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
> x[good]
[1] 1 2 4 5
> y[good]
[1] "a" "b" "d" "f"
```

Removing NA Values

```
> airquality[1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> good <- complete.cases(airquality)
> airquality[good, ][1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
```


Files
Plots
Packages
Help
Viewer
R: The R Datasets Package
Find in Topic
The R Datasets Package
Documentation for package ‘datasets’ version 3.5.2
DESCRIPTION file.
Help Pages
A B C D E F H I J L M N O P Q R S T U V W
datasets-package The R Datasets Package
-- A --
ability.cov Ability and Intelligence Tests
airmiles Passenger Miles on Commercial US Airlines, 1937-1960
AirPassengers Monthly Airline Passenger Numbers 1949-1960
airquality New York Air Quality Measurements
anscombe Anscombe's Quartet of 'Identical' Simple Linear Regressions
attenu The Joyner-Boore Attenuation Data
attitude The Chatterjee-Price Attitude Data
austres Quarterly Time Series of the Number of Australian Residents
-- B --
beaver1 Body Temperature Series of Two Beavers
beaver2 Body Temperature Series of Two Beavers
beavers Body Temperature Series of Two Beavers

Files | Plots | Packages | Help | Viewer

R: New York Air Quality Measurements — Find in Topic

Usage

```
airquality
```

Format

A data frame with 154 observations on 6 variables.

`[,1]`	`Ozone`	numeric Ozone (ppb)
`[,2]`	`Solar.R`	numeric Solar R (lang)
`[,3]`	`Wind`	numeric Wind (mph)
`[,4]`	`Temp`	numeric Temperature (degrees F)
`[,5]`	`Month`	numeric Month (1--12)
`[,6]`	`Day`	numeric Day of month (1--31)

Details

Daily readings of the following air quality values for May 1, 1973 (a Tuesday) to September 30, 1973.

- `Ozone`: Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
- `Solar.R`: Solar radiation in Langleys in the frequency band 4000–7700 Angstroms from 0800 to 1200 hours at Central Park
- `Wind`: Average wind speed in miles per hour at 0700 and 1000 hours at LaGuardia Airport
- `Temp`: Maximum daily temperature in degrees Fahrenheit at La Guardia Airport.

Source

The data were obtained from the New York State Department of Conservation (ozone data) and the National Weather Service (meteorological data).

References

Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983) *Graphical Methods for Data Analysis*. Belmont, CA: Wadsworth.

Examples

```
require(graphics)
pairs(airquality, panel = panel.smooth, main = "airquality data")
```

Console Terminal

~/

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> tail(airquality)
    Ozone Solar.R Wind Temp Month Day
148    14      20 16.6   63     9  25
149    30     193  6.9   70     9  26
150    NA     145 13.2   77     9  27
151    14     191 14.3   75     9  28
152    18     131  8.0   76     9  29
153    20     223 11.5   68     9  30
>
```

Próxima Semana

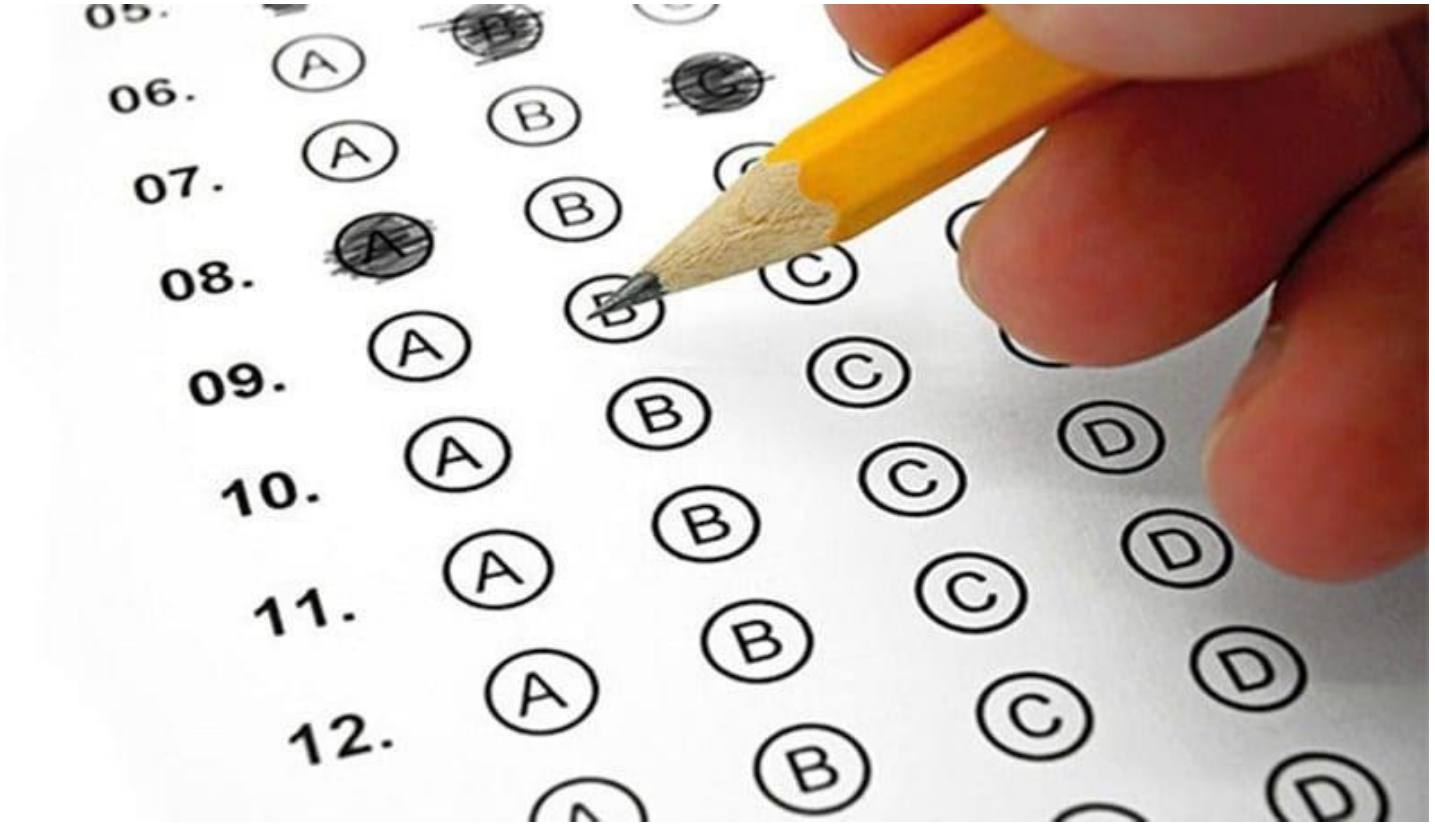
EXERCÍCIOS NA MONITORIA

Horário alternativo

TBD

No dia 28/4/2020

TESTE através do Moodle



Cheat Sheets – “cola oficial”

https://www.rstudio.com/resources/cheatsheets/

Base R

http://github.com/rstudio/cheatsheets/raw/master/base-r.pdf

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.
help.search('weighted mean')
Search the help files for a word or phrase.
help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.
class(iris)
Find the class an object belongs to.

Using Packages

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)
Return x sorted.
rev(x)
Return x reversed.
table(x)
See counts of values.
unique(x)
See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

While Loop

```
while (condition){
   Do something
}
```

Example

```
while (i < 5){
   print(i)
   i <- i + 1
}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){
   Do something
   return(new_variable)
}
```

Example

```
square <- function(x){
   squared <- x*x
   return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions

`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** package.

Data Frames

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x` `df[[2]]`

Understanding a data frame

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.

`rbind` - Bind rows.

Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

`lm(y ~ x, data=df)`
Linear model.

`glm(y ~ x, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Perform a t-test for difference between means.

`pairwise.t.test`
Perform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** package.

`plot(x)` Values of x in order.

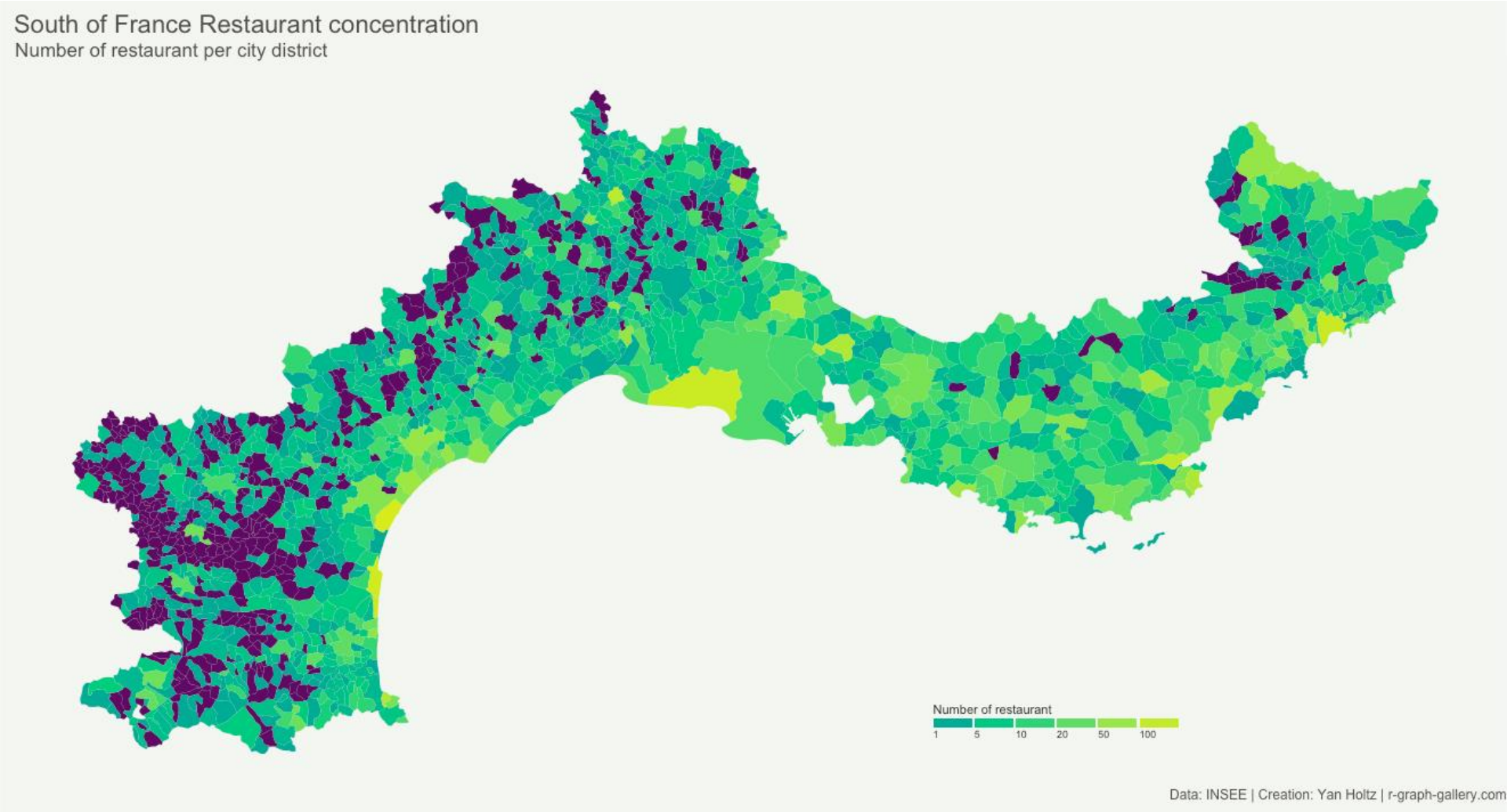
`plot(x, y)` Values of x against y.

`hist(x)` Histogram of x.

Dates

See the **lubridate** package.

MAP2112



https://www.r-graph-gallery.com/327-chloropleth-map-from-geojson-with-ggplot2/