*Johns Hopkins Bloomberg School of Public Health*

*coursera.org*

Data Science

*Notes form the Data Science Specialization Course*

*~ Cristina Lussiana ~*



Summary

[Useful resources 2](#_Toc436688659)

[Data Scientist’s Toolbox 3](#_Toc436688660)

[Week 1 3](#_Toc436688661)

[Week 2 3](#_Toc436688662)

[Week 3 5](#_Toc436688663)

[R Programming 8](#_Toc436688664)

[Week 1 8](#_Toc436688665)

[Week 2 16](#_Toc436688666)

[Week 3 18](#_Toc436688667)

[Week 4 20](#_Toc436688668)

[Tricks 21](#_Toc436688669)

## Useful resources

<https://www.datacamp.com> More than 200 practical exercises to get your hands on R and do some practice.

<http://stackoverflow.com/> Ask and it’ll be given to you. *The* hackers’ community to fix codes, clarify doubts, and not feeling alone in that very common feeling called desperation.

<https://www.gitbook.com/book/gdhorne/data-science-boot-camp-survival-manual/details> Data Science Bott Camp Survival Manual. Work in progress, I think.

<http://adv-r.had.co.nz/> Advanced R by Hadley Wickham. Extremely useful and much more clear than many Video Lectures.

<http://www.statmethods.net/> Quick R – Accessible Reference for R. A comprehensive overview of R.

<http://datasciencespecialization.github.io/> Data Science Specialization Community Site. A lot of useful documents to help you cope with the course materials!

## Data Scientist’s Toolbox

### Week 1

Hacking Skills + Math & Statistics Knowledge + Substantive Expertise

R programming <http://bit.ly/Ufaadn>,

<http://www.r-project.org/mail.html>,

<http://stackoverflow.com>,

<http://rseek.org>,

Data Analysis and Statistics CrossValidated: <http://stats.stackexchange.com/>

Or on Google: “[data type] R package” or “[data type] data analysis”

### Week 2

Version Control System (VCS): is a system that records changes to a file or a set of files overtime so that you can recall specific versions later.

#### Git

Git is a VCS. All it’s stored in local repositories in your pc and they’re called repos. You control it from the CLI.

**GitHub**: is a web-based hosting service for software development projects that use the Git VCS. Allows users to push and pull local repositories to and from remote repositories on the web and always saves a backup copy.

**GItHub Desktop**: an user-friendly version of GitHub that helps you pushing/pulling files by using an user-fiendly interface.

Git is the offline version of GitHub. Now that we’ve created the remote repo QoC, we make a copy on my laptop on a new directory. Open Git Bash (for Mac it’s Terminal), get git init and git remote add origin <https://github.com/cristinaelle/QoC.git> and we’ve linked the local copy with the remote version.

Fork allows you to create copies of others’ repos into your Git account, a kind of link.

git clone<https://etc.git>

Workspace – Index – Local repository – Remote repository (Git hub): from Local to Remote repository is push, the other way around is pull or rebase up to Workspace or fetch up to Local.

git add . add all new files

git add –u update tracking for files that changed names or were deleted

git add –a do both add . and –u

git checkout -b branchname a branch is another directory where you can make changes independently and other people don’t mix things up

git branch see which branch you are in

git checkout master switch back to the master from the branch

git commit –m «message» where message if a description of what you want to use. You use this command when you want to commit to save an intermediate version. And it only updates your local repos, not the remote

git pull pull files from the remote Git Hub to your local directory

git push save files on the remote Git Hub

If you have forked a repository called datascientist on Github and you want to bring the directory to your local computer: git clone https://github.com/username/datascientist.git

Sometimes you have to send a pull request because you want to save in your local repos some remote repos. So you select the branch you want to pull and send a request to the people who own that repos.

cristinaelle – cristinaelle@hotmail.it

repository: QoC, datasciencoursera, DHIS2

check out: git-scm.com/doc and help.github.com

**How to push/pull files**:

on GitHub I’ve created a repos called “datasciencecoursera”

on my hd I’ve open GitBash/Terminal and

mkdir datasciencecoursera

cd datasciencecoursera

touch HelloWorld.md

Echo ## This is a markdown file

git init

git remote add origin <https://github.com/cristinaelle/datasciencecoursera.git>

git add HelloWorld.md (or git add .)

git commit –m “first commit” (or git commit –m “merge from local to github”)

git push –u origin master and the file HelloWorld is on my remote repos.

If you want to do the other way around:

git pull and you find what you’ve modified on your remote repo in GitHub in your local repos.

Command Line Interface (CLI): for Windows it’s **Git Bash**. For Mac it’s **Terminal**.

Directories: the root directory is the upper directory that contains all other directories and it is usually represented as /. The home directory is where you host all the files and is usually represented as ˜.

Command Flags Argument:

Command = something like pwd

Flags = are options we give to the command to trigger certain behaviors, preceded by a –

Arguments = is what the command is going to modify, or other options for the command.

cd change directory and if you don’t type any argument, it takes you to your home directory

cd .. change directory to one level above your current level

clear clear all the commands

cp copy and you have to type first the file/folder you want to copy and then the destination (cp touch notes ADMIN-FINANCES).  cp the\_file\_to\_copy../the\_folder\_you\_want\_to\_copy. If the two folders share the same parent you need to sort of take a step backwards hence the "../" bit

cp –r copy one directory into another directory. Copy the content of one directory (first argument) into another (second argument) (cp –r me ISD). It copies the folder me into ISD

date print today date

echo print whatever you want (echo ciaociao)

exit shut down Git Bash

ls list files and folders in the current directory

ls –a list all hidden and not hidden files

ls –al list details for all hidden and not hidden files

ls –l list details

mkdir make a new directory (create a folder) and the argument is the name of the directory you’re creating (mkdir GIT)

mv move files from the directory where you are to another (mv notes ISD)

mv rename files (mv notes ciaociao)

pwd print working directory where you’re now

rm remove/delete

rm –r remove entire directory. Be careful! There’s no undo!

touch create an empty file (touch ciaociao)

**Markdown**: is a text file. Its extension is .md (readme.md).

Headings ## This is a secondary heading, ### This is a tertiary heading

Lists \* car, \* bicycle, \* foot

#### R packages

a <- available.packages() give you a list of available packages

head(rownames(a),3) give you the first 3 packages

install.packages(“slidify”) install the package solidify

install.packages(c(“slidify”, “ggplot2”, “devtools”)) install multiple packages

library(name) load packages into R and you use it after install to load it into R library(devtools)

search() you can use it right after having loaded a package to see all the functions of that package

#### RStudio

Go to Tools – Install Packages. After installing the package you have to load it.

You can also upload packages from web-based resources as bioconductor.

### Week 3

Descriptive analysis description can’t be generalized without a statistical model

Exploratory analysis find relationships you didn’t know about it. Good for discovering new connections. Correlation does not imply causation. The data is used for exploration, but not necessarily for confirming anything that you discovered

Inferential analysis use a relatively small sample of data to generalize something about a bigger population. This is the statistical model. Inference is more on estimating the quantity you care about and your uncertainty about the estimation

Statistical inference you’ve got a *population* and hence a *sample*. The sample is based on the *probability* that the sample represents your population. You conduct a *descriptive* *statistics* in your sample and from there you conduct an *inferential* *statistics* to say something about your *population*

Predictive analysis use the data on some object to predict the value of the data on another object. It heavily depends on measuring the right variables. More data and a simple model work pretty well

Causal analysis identify cause-relationship data. If I change the value of one variable, what happens to the other? Causal models are the gold standard for data analysis. They are calculated on average effects, but it might be that they don’t work on a bigger sample

Mechanistic analysis understand the exact change in variables that lead to changes in other variables for individual objects. It uses deterministic set of equations

“Data are values of qualitative or quantitative variables, belonging to a set of items.” Population!

* The question you’re trying to answer!
* Data! They can limit or modify the question, though. Having data can’t save you if you don’t have a question!

**Confounding**: you’re analyzing the relationship between two variables, A and B. The variable C can confound your analysis if variable C influences both. This is a cofounding. Correlation is not causation. What you get out a relationship that has a confounding variable is a spurious correlation. Even if variable A and B are correlated to each other, you have to prove that they are not correlated because of variable C!

**Fix a variable**: if you don’t, stratify it; if you can’t, randomize it. This will reduce the chances that your confounding variable affects your analysis.

**Prediction**: you’ve got two samples with 2 outcomes (i.e. responding and not-responding). We want to build a predict function to see if our subject will respond/not respond. Prediction is more challenging than inference because you compare 2 sample population and you have to be sure that these 2 sample populations are variable enough between themselves to say something. If they’re too overlapping, you can’t say much about your prediction. Prediction is not inference.

Prediction key quantities:

* Se = positive test/disease
* Sp = negative test/disease
* Positive predictive value = disease/positive test (false pos)
* Negative predictive value = no disease/negative test (false neg)
* Accuracy = correct outcome

**Data dredging**: you change so many times your hypothesis and restrict your tunnel vision that you come out with a correlation that is valid for your specific hypothesis based on what you left *out* of your sample!

## R Programming

### Week 1

Help: rhelp@r-project.org

Statistics With Interactive Learning = SWIRL

Data types: atomic classes (numeric, logical, character, integer, complex), vectors/lists, factors, missing values, dataframes/matrixes, names

#### Elements and objects

vector objects that contain data are data structure. Numeric vectors are the simplest. A single number is considered a vector of length 1. When given two vectors of the same length, R simply performs the specified arithmetic operation element-by-element. If the vectors are of different lengths, R 'recycles' the shorter vector until it is the same length as the longer vector

atomic vector contain one data type or class of objects (numeric, logical, character, integer and complex)

list is a vector that contains multiple data types. Ex.: list(1, “a”, TRUE, 1+4i)

number always 1.00

integer nL give you the integer

attributes objects have attributes: names, dimnames, dimensions (matrices, arrays, etc), class (numeric, etc), length, user-defined metadata. attributes()

coercion happen when R coerces a data type into another data type to make the function works. Ex.: paste(LETTERS, 1:4, sep = “-“) 1:4 is coerced into a character vector. Numeric into character, logical into numeric, logical into character. You can explicitly coerce with as.numeric(x), as.logical(x), as.character(x) to coerce the vector x into the desired class

subset vector you tell R you want to select particular elements from a vector. vector[subset conditions]. Four types of index vector: logical, vectors of positive integers, of negative integers, of character string. Ex:

* x[1:10] is the first 10 elements in x,
* x[x>0] is the objects in x >0,
* x[c(3, 5, 7)] gives you the 3rd, 5th and 7th objects of vector x,
* x[-2, -10] gives you all the objects but the 2nd and the 10th (or x[-c(2,10)],
* x[!is.na(x) & >0] combines 2 conditions and gives you all the objects in x that are not NA AND are >0,
* which(x>7) subsets a vector (not a matrix or dataframe!) according to a logical condition and returns what’s TRUE. vect[which(vect>5)] returns values in vectors >5

matrix vectors with a dimension attribute. Ex.: m<-matrix(nrow=2, ncol=3), dim(m) give you 2 and 3, attributes(m) give you $dim 2 and 3. Indexes = rows and columns

factor represent categorical data and can be unordered or ordered. Modelling functions like lm() and glm()

#### Algebraic operators

abs() absolute value

sqrt() square root

Inf infinity

x^2 x squared

NA quite confusing. Be careful with it because a single NA can derail the entire thing. They have classes (integer NA, character NA, etc.)

NaN not a number. NaN is NA, the converse is not true

#### Files and packages commands

args(function) tell you the arguments of a function, what it is composed of

dir.create("name") create a wd called name

getwd() tell you which wd you are in

file.copy(“name”, “name2”) make a copy of the file with a new name2

file.create(“name”) create a file

file.exists(“name”) verify if a file exists and report TRUE/FALSE

file.info(“name”) access info about the file

file.path(“name”) construct file and directory paths that are independent of your op system. file.path(“folder1”, “folder2”) create a platform-independent pathname (meta directories). For nested directories, recursive in dir.create must be TRUE. dir.create(file.path("testdir2", "testdir3"), recursive = TRUE) = create a new directory called testdir2 with a nested directory called testdir3

file.remove(“name”) delete the file

file.rename(“name”, “namenew”) change name of the file

ls() list the objects in the wd

list.files() or dir() list the files in the wd

library(package) load the package

package() start the package

name<-getwd() assign the name to the current wd

rm(list = ls()) clear workspace

setwd(“name”) set the wd to the one you’ve called name. with dir.create(“name”), go back to the dir called name

dir.create(“name”) create a new directory name into the working directory

unlink() delete a directory. Recursive must be TRUE in case there are nested directories and you want R to be aware that you are aware you’re deleting nested directories, too. unlink(“testdir”, recursive = TRUE)

?c open the help menu for the function, c or whatever

?”:” open the help menu for an operator, : or whatever

#### Generic commands

x print x

<- assignment operator. It assigns a value to a symbol

[] at the beginning of the row indicate that what’s right is a vector. If multiple rows, it always indicates the first object in the row

a:b or pi:10 move from a to b by +1. pi:10 = 3.14 4.14 5.14, etc., 9.14

as.numeric/character/logical(x) coerce vector x into numeric/character/logical class. If nonsense I get NAs

any(x><=!=y) return TRUE if one element of the logical function is TRUE

all(x><=!=y) return TRUE if all the elements of the logical function are TRUE

c() create a vector, c(1, 2, 4.67)

c(a=11, b=12, c=45) create a vector of 3 where each has an assigned number

c(1,2,3,4)+c(0,10) 1 12 3 14 concatenates vectors of different lengths where one divides by the other. It they don’t divide by each other, you get a warning message

identical(a, b) tell you if a is identical to b

ints(a) create a list of integers from 1 to length(a)

is.na(a) report TRUE/FALSE for each element of object a in case there are/not NA. It’s logical!

!is.na(a) report “is not NA” and it can be used to subset a vector and take NA off

is.nan(a) report TRUE/FALSE for each element of object a in case there are/not NaN

x[!bad] remove NA in case bad<-is.na(x) where x is a vector. Ex: mean(ozone[!is.na(ozone)]) give you the mean of the column Ozone once you converted the column into a vector

good<-complete.cases(x,y) remove NA from multiple things (dataframes and matrixes). Ex: good<-complete.cases(matrix) and then matrix[good, ]

length() report the length of the object

names(a) give the names of the object in a. Ex: names(x)-<c(“A”, “B”, “C”). Useful for list: x<-list(“Torino” = 1, “Milano” = 2, “Roma” = 3) and matrix: m<-matrix(1:4, nrow=2, ncol=2) and dimnames(m)<-list(c(“a”, “b”), 1:2) and dataframe where it gives you the names of the columns. Ex.: results<-c(12, 11, 13, 14, 15), names(results)<-c(“Mon”, “Tue”, “Wed”, “Thu”, “Fri”)

rep(a, times = b) repeat a b times. If a is a concatenate vector, it repeats the vector b times

rep(a, each = b) repeat a b times and if a is a concatenate vector, it repeats each object of a b times

sample(c(a,b),n) give you n elements from the concatenation of objects a and b

sample(c(anb),n, replace = TRUE) give you n elements from the vector c(a,b) replacing values

seq(a, b, by=c) give the seq of number from a to b by c

seq(a, b, length = c) give c numbers in seq from a to b

seq(along.with = a) generate a sequence of integers from 1 to the length of object a

1:length (a) generate a sequence of integers from 1 to the length of object a

seq\_along(a) generate a sequence of integers from 1 to the length of object a

sum(a) sum the values in object a. If a is composed of TRUE/FALSE, sum TRUE (1)

vector() empty vector

vector(class, attributes) vector(“numeric”, length =10) gives you 0 then times (the default numeric value is 0)

which(x><=!=y) return the position of the values of the vector x that verifies the condition x><=!=y

z<-c(1, 2, 4.67) result in 1 2 4.67

z\*2+100 give you 3 values, one for each of the object the vector is composed of

#### Vectors

character vector double quotes “” distinguish character objects (in these examples a is a character vector)

paste(a) report the objects in a. concatenate vectors after converting to characters

paste(a, collapse = “ “) report the objects in a with no “” but only a space between each other. collapse to the paste() function tells R that when we join together the elements of the character vector, we'd like to separate them with “ “

paste(a, sep = “ “) report the objects in a with a space between each other. sep argument tells R that we want to separate the joined elements with “ “

paste(a, b, sep = “”) report the pairs of the 2 vectors a and b separated by no space (2 vectors of same length. If it’s different, R recycles)

vectorized operations R vectorizes operations

vectorized matrix operations R vectorizes matrixes. x\*y, x/y multiply and divide element by element. If you want to really multiply a matrix times another matrix use %\*%

%[whatever]% binary operator to build your own combination of operators. Ex.: “%mult\_add\_one%" <- function(left, right){left \* right + 1} it multiplies left times right and adds 1. To call it use: 5 %mult\_add\_one% 6

logical vector if questioned (<a, >b, etc), R reports TRUE/FALSE for each object of the vector. Remember that AND is evaluated before OR

TRUE/FALSE 1 and 0. Case sensitive!

< > >= >= == != logical connectors

A | B whether at least one is TRUE (OR)

A || B whether what’s right is TRUE or the first element on left is TRUE. Ex: TRUE || c(TRUE, FALSE) is TRUE, while TRUE | c(TRUE, FALSE) is TRUE, TRUE because c is a vector

A & B whether they are both TRUE (AND)

A && B whether what’s right is TRUE and the first element on left is TRUE, too. Ex: TRUE && c(TRUE, FALSE) is TRUE, while TRUE & c(TRUE, FALSE) is TRUE, FALSE because c is a vector

!A negation of A and it is TRUE when A is FALSE and viceversa

isTRUE(x) evaluate if x is TRUE

xor(a,b) exclusive or. It’s TRUE if either a *or* b is TRUE. If both a *and* b are TRUE, xor(a,b) is FALSE. One argument must be TRUE and the other FALSE

== and ! and != The equals operator `==` tests whether two boolean values or numbers are equal, the not equals operator `!=` tests whether two boolean values or numbers are unequal, and the NOT operator `!` negates logical expressions so that TRUE expressions become FALSE and FALSE expressions become TRUE

⇒ First R analyzes &, then |, then &&, and then ||

#### Dataframes

dataframes can contain many different classes of data. They are a special type of list where every element of the list has the same length. Each element of the list is a column whose length is the number of rows

data.frame(a, b) add a column as the header of rows where a is the vector of your headers (and a the title of your column!) and b your matrix keeping the original object. Ex.: x<-data.frame(foo = 1:4, bar = c(T, T, F, F)), my\_matrix2<-data.frame(patients, my\_matrix) reports my matrix with the header patients in characters leaving the objects in my\_matrix in integers. Ex.: ciao<-data.frame(a = 1:4, b = 5:8, c = 9:12), then names(ciao)<-c(“1”, “2”, “3”) and row.names(ciao)<-100:103. It’s important to have a, b, c as the argument of names

name() is an attribute of dataframes, headers of the columns

row.names() is an attribute of dataframes, headers of the rows

read.table() or read.csv() create dataframe

data.matrix() convert dataframe to a matrix. Ex: c<-data.matrix(a)

ncol(a) and nrow(a) return number of columns and rows of a dataframe a

names(a) return the names of the columns of a dataframe a

#### Matrixes

matrix can only contain a single class of data. Data are first by column and then row

matrix(a, x, y) create a matrix of objects a of x rows and y columns. Ex.: d<-matrix(1:4, 2, 2)

cbind(a, b) add a column as the header of rows where a is the vector of your headers (and a the tilte of your columns!) and b your matrix. It works only for same-same class of objects (headers = integers, content of the matrix = integers)

rbind(a, b) add a row as the header of columns where a is the vector of your headers (and a the title of your rows!) and b your matrix. It works only for same-same class of objects and doesn’t add up to cbind

cbind/rbind in case a and b are 2 vectors you can bind them column/row wise. You tell R where you want to see your vectors: by row or column. It works only for vectors of same length

colnames(a)<-b add a row as the header of columns where a is the matrix and b is the vector that contains the header of your columns. Be careful to include the first column that is the header of the rows!! Ex.: colnames(my\_matrix)<-dose where dose is a vector whose objects are the headers of the columns

class(a) tell you the class of a (matrix, array)

dim(a) tell you the dimension of a

dim(a)<-c(x, y) allow you to get or set the dim attribute for an R object. Ex.: dim(my\_vector)<-c(4,5) gives the object my\_vector a dimension of 4 rows and 5 columns

dinmanes(a)<-list(c(x), c(y)) or c(x, y) assign to your matrix m the name of your rows (x) and columns (y)

complete.cases(a) keep only complete rows and eliminate NA from your matrix a. Subset your matrix for complete.cases(a). Ex.: good<-complete.cases(a) and a[good, ]

row<-nrow(a) assign to row the number of row in the table a and a[row,] returns the values of the last row, a[row-1,] returns the values of the next to last row

#### Subsetting

[ always return an object of the same class of the original and select more than 1 element of an object

[[ extract elements of a list or a dataframe and the classes of the objects may be different

$ extract elements of a list or a dataframe by name. It can be used only with literal names: the argument of $ must be a name, otherwise it doesn’t work. Ex.: x$bar return the bar element of the list x

[[$ partial matching, mean that I want to select something that matches perfectly

a[1, ] return the values of the first column where a is a vector/matrix

a[, 1] return the values of the first row where a is a vector/matrix

a[x,y] return the value in x column and y row where a is a vector/matrix. In case of matrix it always returns a vector of length 1, rather than a 1x1 matrix. To change this drop = FALSE (you don’t want to drop the dimension of the object, I want to keep it as a matrix). Ex.: x[1, , drop = FALSE] returns a matrix of the first row

a[a > “b”] return the value of the vector a > b where a is a vector/matrix

gnu$a return the value of a from the list gnu where gnu<-list(a = 1:4, b = c(“a”, “b”))

gnu$b[2] return the 2nd value of b from list gnu

gnu[c(1,2)] return the 1st and 2nd element of the list gnu. You can’t use the $ when you extract multiple elements of the list

gnu[[c(1,2)]] return the 3rd element in the 1st object of a *nested* list gnu

gnu[[1]][[3]] return the 3rd element in the 1st object of a *nested* list gnu

subset(a, subset, select] subset the object according to a logical operation subset and return the values within what selected. Ex.: subset(a, Temp == 10, Select=Ozone)

Dataframe data.frame(a, b) different types of objects, column-wise

Matrix matrix(a, b) same type of objects, column-wise

 rbind(a, b) or cbind(a, b) same type of objects. If only integers you can do calculations; if there are other type of objects, be careful because they are usually store as text so you can’t do calculations! You have to start from a dataframe, subset it, convert the subset in a matrix and do the calculations on the subset.

Object Set Column Names Set Row Names Works?

Dataframe names() and colnames() row.names() [, min, max, subset, complete.cases

Matrix colnames() rownames() [, mean, mix, max, complete.cases

*matrix or dataframe*

*(row, column)*

The big difference between a dataframe and a matrix is that you can’t do arithmetical calculations like *mean* with a dataframe. You have to convert the dataframe into a matrix and then use mean. If you have a dataframe, first you subset from the dataframe what you want, then you convert this into a data.matrix, and then you do the mean. Ex.: mean(c<-data.matrix(subset(hw1g, Ozone>41, select=Temp)))

If you have different types of objects you must use a dataframe. To create a dataframe: df<-data.frame(a, b, c) and R creates a dataframe by columns: it puts the 3 vectors a, b and c into 3 columns. The first row is the names of your vectors, in this case a, b, and c. If you want to do calculations you have to use a matrix, especially for mean. So it means that you have to do mean(m<-data.matrix(df[, 2]) and you have the mean of the second column of your dataframe df. If you only have integers, you are safe with a matrix because it recognizes all of them as integers and you can do calculations. However if there are texts, the class of objects of the matrix becomes texts and you can’t do calculations anymore.

To select things in a dataframe you can do length(df$a) where df is the dataframe and a is a column.

#### Factors

factor categorical data and R reports levels reported in alphabetical order, very important to know in modelling. If you want to change this, you have to crate a factor and then explicit the level you want with levels = c(“yes”, “no”)

x<-factor(c(a, b, c, d) read as categories the objects of vector c (Ex.: M, F, F, F). If then

table(x) give you how many objects for each category (Ex.: M F 1 3). If then

unclass(x) strip out the class for factors (Ex.: 1, 2, 2, 2)

#### Others

read.table or read.csv read tabular data (rows and columns) and return a dataframe

readLines read lines of a text line

source read R code

dget read R code

load read in saved workspace

unserialize read binary objects

write.table write tabular data to text files (es csv)

writeLines write character data line-by-line to text files

dump make up R code for multiple R objects

dput make up R code for a R object

save save arbitrary number of R objects in binary format

serialize covert R objet into a binary format

arguments of read.table

file name of the file

header indicate if the file has a header line or not. In case of csv is always TRUE

sep indicate how the columns are separated (, or spaces or tabs). In case of csv the separator is a comma and is automatically recognized by R for read.csv

colClasses character vector that indicates the class of *each* column

nrow number of rows

comment.char indicate what are the comments (after ##). Comments are ignored in R

skip in case you want to skip x lines

stringsAsFactors TRUE unless you set it as FALSE. Should character variables be coded as factors?

for large datasets when reading a table specify nrow and colClasses. It makes the upload faster

Memory each number of a dataframe (numeric) requires 8 bytes (remember that 1MB = 220 bytes). Calculate how much memory you need on your RAM before uploading large datasets. You usually have a 64 bit os

dumping for writing out textual format

dput write R code that can be used to reconstruct a R object

file open a connection to a file

gzfile open a connection to a gzip file

bzfile open a connection to a bzip file

url open a connection to a webpage

str(file) are the argument of a file where description is the name of the file, open specificies is a r, w, a file

readLines(file) read the file line by line. Ex.: con<-gzfile(“words.gz”), readLines(con, 10) read the first 10 lines

writeLines(a) take a character vector a and write each element of the vector one line at a time to a text file

url(link) read from a link. Ex.: head(readLines(con<-url("http://jhsph.edu", "r")))

### Week 2

#### Control Structures

if, else test a condition

for execute a loop a fixed number of times

while execute a loop while a condition is true

repeat break the execution of a loop

next skip an interaction of a loop

return exit a function

if if(condition) {do something} else {do something else}. Ex.: if(x>3) {y<-10} else {y<-0}

 you can add more conditions and the else clause is not necessary.

for for(i in condition) {do something}: Ex.: for(i 1:10) {print(i)}.

 they can be nested, but be careful because nesting beyond 2-3 levels is difficult to follow. Ex.: for (i in seq\_len(nrow(x))) {for (j in seq\_len(ncol(x))) {print(x[i, j]) } } where x is a matrix

while test a condition and will execute a value based on the condition (TRUE). Ex.: count<-0 while(count<10) {print(count) count<-count+1}}. In this case it stops at the 10th value because you’ve given a break (count +1). Be careful because while loops can easily go on forever unless you give them a limited time of execution. You can test multiple conditions: while(z>=3 && z>=10) {print(z) coin<-rbinom(1, 1, 0.5) if(coin == 1) z<-z+1} else {z<-z-1}}

repeat initiate an infinite loop. The only way to exit a repeat loop is to call break. Ex.: repeat {x1<-computeEstimate() if(abs(x1-x0)<tol {break} else {x0<-x1}}. There’s no guarantee it will stop, so better to use a for loop

next, return use when you want to skip an iteration. Ex.: for(i in 1:100) {if(i<=20) {next}} or for(i in 1:100) {if(i<=20) {next}}

stop retrieve an error and report a message. if(dis2==3) {stop(“invalid outcome”)}

#### Functions

above10<-function(above10) return the arguments of the function above 10

 for (i in 1:length(above10)) {

 if (above10[i]>10) print(above10[i])}

above <- function (x, n) return the arguments of the function above n

 {select<-x>n

 x[select]}

cmean <-function(y, removeNA = TRUE) { returns the means of each column in the dataframe or matrix.

mean() calculate the average of all of the numbers in a vector and then divide that sum by the length of the vector

nc<-ncol(y) report the number of columns in y

means<-numeric(nc) create the objet means as a numeric vector of length nc

 for(i in 1:nc) {means[i]<-mean(y[, i])}

 means }

 if you want just 1 number as the mean of all columns, add mn<-mean(means)

… is an argument that indicates a variable number of arguments that are passed on to other functions (plot, cat, paste, etc). Any arguments that appear after … on the argument list must be named explicitly and cannot be partially matched. The ellipses can be used to pass on arguments to other functions that are used within the function you're writing. Usually a function that has the ellipses as an argument has the ellipses as the last argument. Strict rule in R programming: all arguments after an ellipses must have default values!

#### Lexical scoping

workspace *global environment* is where R search for a symbol name, the first environment, and then search in each of the packages listed in search(), in a specific order that is not alphabetical, until the last environment, called *base package*. An environment is a collection of (symbol, value) pairs and environments have parent environment and can have many children. A function + an environment = a closure, or function closure

lexical scoping or static scoping. The scoping rules of a language determine how values are assigned to free variables. In R there’s the lexical scoping: the values of free variables are searched for in the environment where the function was defined. If it’s not found here, it it searched for in the parent environment, up to the empty environment, the one with no parent environment. It is different from the *dynamic scoping* where free variables are searched for in the environment where the function was called. In R you can define a function inside another function

 ls(environment(function)) gives you what is in a function’s environment. It lists objects or functions

 get(“a”, environment(function)) reports the value of object a in the function

optimization require you to pass a function whose argument is a vector of parameters

dates Date class. x<-as.Date(“1983-12-08”) and it counts the number of days from 1970, Jan 1

 strptime coerce character strings to Date/Time classes and give you the variables of a date: x<-strptime(date, “%B, %d, %Y”) gives Year-day-etc

times POSIXct class: a very large integer, useful for dataframes;

 POSIXlt lcass: is a list and it stores a lost of info (day of the week, day of the year, etc). Sys.time() report the current date

### Week 3

#### Loop functions

You want to execute a work on an object in an automatic way, especially for for and while loops.

Split-Apply-Combine strategy for data analysis: they split up some data into smaller pieces, apply a function to each piece and then combine the results.

lapply loop over a list and evaluate a function on each element and return a **list** of the same length as the original one. 3 arguments: a list, a function (mean, max, min, FUN), and the argument …. If the first argument is not a list, it will be coerced to a list using as.list and it always returns a list, regardless of the class of the input. You can coere a list into a numeric vector with as.numeric()

 Ex.: lapply(df, mean) give you the mean of the columns in dataframe

 Ex.: lapply(x, runif) generate random variables for x times

 Ex.: lapply(x, function(pink) pink[, 1]) report the first column of your list x: you can crate a function within the lapply arguments!

sapply simplify the result of lapply: it returns a *vector* if the result of lapply is a list of every element with length >1; It returns a *matrix* if every element of the result of lapply is of the same length (>1). Just substitute lapply with sapply

apply evaluate a function over the margins of a dataframe, matrix or array. It is used to apply a function to the rows or columns of a dataframe, matrix or array. It has 4 arguments: an array, a margin that is an integer vector indicating which margins should be retained, the function FUN, and the … argument.

 \* 2 dimensions for matrixes and dataframes: rows (1) and columns (2).

 \* array is a vector that has a dimension attached to it. 3 dimensions: rows (1), columns (2), and a third dimension (3).

 Ex.: apply(x, 2, mean) return the mean of the columns.

 Ex.: apply(x, 1, quantile, probs = c(0.25, 0.75)) return the 1t and 4th quinitles of rows.

 Alternative functions: rowSums, rowMeans, colSums, colMeans

mapply multivariate apply which applies a function in parallel over a set of arguments. It has 4 arguments: a function FUN, the argument …, MoreArgs is a list of other arguments to FUN, SIMPLIFY indicates if the results have to be simplified

tapply apply a function over subsets of a vector. it has 5 arguments: the first if the vector, then the INDEX that’s a factor or a list of factors, the function FUN, the argument …, and SIMPLIFY if you want it to simplify the result. If you don’t simplify the result, you get back a list

split take a vector or other objects and split in into group determined by a factor or list of factors. It has 3 arguments: the first is the vector or dataframe, then f the factor or list of factors, and drop indicates whether empty factor levels should be dropped.

 Used together with lapply: lapply(split(x, f), mean).

 Ex.: m<-split(dataframe, dataframe$colNameyouwanttosplit)

 Lapply(m, function(x) colMeans(x[, c(colNameyouwanthemean)])

 m<-split(airquality, airquality$Month)

 lapply<-m, function(x) colMeans(x[, c(“Ozone”, “Solar.R”, “Wind”)])) or

lapply(c, function(x) colMeans(subset(x, Day<=15)[, c("Ozone", "Solar.R", "Wind")], na.rm=TRUE))

split(airquality, airquality$Month) and you get n tables for n Months of the dataframe

interaction split on more than one level. Empty levels can be dropped with drop = TRUE

#### Debugging

type of errors message, warning, error, condition. Error is fatal, it stops the execution

debugging functions traceback: print out the function called stack after an error occurs and it gives you the most recent error

 debug: flag a function for debug mode and allows you to go line by line

 browser: suspend the execution of a function wherever it is called and puts the function in debug mode

 trace: allow you to insert debugging code

 recover: allow you to modify the error behavior

### Week 4

str diagnostic function to display what an object (or function) does

#### Simulations

Standard distributions are built in Normal (n), Poisson (p), Binomial (b), Exponential (e), etc.

r random number generation

d density

p cumulative distribution

q quantile

rnorm random normal variables with a given mean and sd. Use the arg replace = TRUE if you don’t want repetition of numbers

dnorm evaluate the normal probability density with a given mean and sd at a point

pnorm evaluate cumulative distribution function for a normal distribution

rpois random Poisson variables with a given rate

set.seed to ensure reproducibility of the same random numbers. Ex.: setseed(1), rnorm(5) you set the seed at this random distribution. Then rnorm(5) you get different random numbers, but then set.seed(1) and then rnorm(5), it recalls the random numbers generated the first time

#### Data plots

plot(x,y) if you define x and y (it might be a linear regression), R plots your value in a graph

hist(x,y) report an histogram for x and y

boxplot(x,y) report a box plot for x and y

plot(x=cars$speed, y=cars$dist) plot dataframe$speed on x axis and dataframe$dist on y axis

plot(x,y, xlab=”Names”, ylab=”Value”) define axis x and y titles

plot(x,y, main=”My graph”) set the title of the chart

plot(x,y, sub=”September 2015”) set the subtitle of the chart

plot(x,y, col=2) set color to red

plot(x,y, xlim=c(10,15)) set x axis limits to 10 and 15

#### Profiler

Profiling is a systematic way to examine how much time is spend in different part of a program.

system.time() take an R expression as input and gives you the time in seconds needed to execute an expression. If there’s an error, gives time until the error occurred. It gives 2 times: user time (CPU’s time) and elapsed time (wall clock time). Elapsed time is usually bigger than the user time when you analyze data from the web. Do not use with Rprof()

Rprof() start the profiler in R. Do not use with system.time()

summaryRprof() better than Rprof() because it summarizes the output and make them readable. by.total divides the time spend in each function by the total run time; by.self does the same but first subtract out time spent in functions above in call stack

### Tricks

Counting NA

bad<-is.na(hw1[, 1])

length(bad[bad==TRUE])

is.na(vector) reports TRUE FALSE if there is/not NA. every TRUE = 1, every FALSE = 0. sum(is.na(vector)) counts NA

Eliminating NA

good<-complete.cases(hw1)

hw1[good, ]

x[!is.na(x)] works for vectors

x[!is.na(x) & x>0] eliminates NA and reports positive

Print from 1 to n only if something

x<-1:10

> for (i in 1:length(x)) {n<-(x[[i]])

+ if(n>5) {n<-0}

+ print(n)}

Read and merge .csv files

ciao<-read.csv(“directory/name.csv”)

files\_list<-list.files(directory, full.names=TRUE)

Count files

list<-list.files(“specdata”, full.names=TRUE)

length(list)

Vectors of different lengths

c(1, 2, 3, 4) + c(0, 10)

1 12 3 14

c(1, 2, 3, 4) + c(0, 10, 100)

1 12 103 4 longer object length is not a multiple of shorter object length!!

Changing length of *character* vectors (merging elements)

my\_char<-c("My", "name", "is")

paste(my\_char, collapse = " ") My name is

paste(c(“x”, “y”, “z”), 1:3, sep = “” ) “x1” “y2” “z3”

paste(1:3, c("X", "Y", "Z"), sep = "") "1X" "2Y" "3Z"

paste(1:3, c("X", "Y", "Z"), collapse = " ") "1 X 2 Y 3 Z"

paste(1:3, c("X", "Y", "Z"), collapse = "") "1 X2 Y3 Z"

telegram <- function(...){ START ciao mamma STOP

paste("START", ..., "STOP", sep = " ", collapse = NULL)

}

mad\_libs <- function(...){

args<-list(...)

 place<-args[["place"]]

 adjective<-args[["adjective"]]

 noun<-args[["noun"]]

paste("News from", place, "today where", adjective, "students took to the streets in protest of the new", noun, "being installed on campus.")

}

we pass the arguments of the function as a list into the object args and then we subset the objects of that list. To call the function you have to write mad\_libs(place = “rome”, adjective = “bella”, noun = “mamma”)

Subsetting dataframes

vect

 1 2

foo 11 14

bar 2 15

norf NA NA

vect[2,] & vect[vect>10]

 1 2

bar TRUE NA

vect2<-vect[2,]

vect2[which(vect2>10)]

 2

bar 15

virginica<-subset(iris, Species=="virginica")

mean(c<-as.matrix(virginica[, 1])) returns mean of column 1 for Species = virginica in iris

answer<-hostdata[which.min(hostdata[, 2]), 1] which.min or which.max return the min or max value in the subset. If you use min or max only, it returns the value in min/max, not the row of the dataframe info with the corresponding info on what’s min/max

Subsetting using bracket notation: the left side of the comma is rows and the right side is columns:
DF[some\_condition, some\_columns]

If we want multiple conditions, we can also do this:

DF[condition1 & condition2, some\_columns]

You can use & and | (pipe) to combine conditions. The & is a logical AND and the | (pipe) is a logical OR. You can also do negation using !(some\_condition).

You can also use numeric values to select certain rows: df[1 , ] or df [1:10 , ]. If you want all rows, just leave the left side of the comma empty: df[, some\_columns]. If you want all columns, leave the right side of the comma empty: df [some\_condition , ].

When using bracket notation with dataframes, columns are required when the first parameter is a condition so the comma is also required. Columns can be numeric or column names. If you use column names, they must be quoted. So, c(1,3) and c("columnA", "columnB") are both acceptable. For a single column, you can leave out c(). Also, df[ , ]$columnA is the same as df[ , "columnA"] which is the same as df$columnA. However, you can use variables inside brackets but you can't use a variable with $.

If there is only a single value N without a comma, R returns a dataframe with single brackets: df[N] and a vector with double brackets: df[[N]] where N is a column number or a quoted column name.

There are two things to remember when using conditions in a dataframe:

1. The condition(s) go on the left side of the comma (since we're selecting rows)
2. The right side of the comma cannot be empty

df <- data.frame(X=1:10, Y=rep(1:2, 5))

Single Condition: df[df$Y == 2, ]

Combining conditions: AND df[df$Y == 2 & df$X <=5, ], df[df$Y == 2 & df$X %in% c(5:10), ], OR df[df$Y == 2 | df$X == 5, ], df[df$Y == 2 | df$X %in% c(1,9), ]

Which() returns a vector of indices where a given condition is TRUE: which(df$Y == 2), df[which(df$Y == 2), ]. Sometimes you might want to save some indices for use later: idx <- which(df$Y == 2), df[c(2,10), "Y"] <- 1. Using which() is functionally the same as putting the condition directly inside the brackets: df[which(df$Y == 2), ], df[df$Y==2, ]

Subsetting a Column: you can subset a single column using the $ sign and conditions. df$X[df$Y == 1], df$X[df$X < 5 | df$Y == 2]

Ordering dataframes

order.file<-order(file1[,2], file1[, 1])

ciao<-file1[order.file,c(1,2)]