R for Data Analytics Part 1, Lecture 3

Michèle Fille

Table of contents

# Lecture 3 – Data Structures for Data Science: Lists and Data Frames

## 3.1 Lists

### Exercise 3.1. Lists

#### Exercise 3.1 – Task 1: Creating and Accessing Lists

a) Create a vector my\_breakfast of everything you ate for breakfast.

my\_breakfast <- c("egg", "avocado", "sourdoughbread", "coffee", "oatmilk")

b) Create a vector my\_lunch of everything you ate (or will eat) for lunch.

my\_lunch <- c("crepes", "banana", "nutella")

c) Create a list meals\_list that contains your breakfast and lunch.

meals\_list <- list(breakfast = my\_breakfast, lunch = my\_lunch)  
  
meals\_list

$breakfast  
[1] "egg" "avocado" "sourdoughbread" "coffee"   
[5] "oatmilk"   
  
$lunch  
[1] "crepes" "banana" "nutella"

d) Add a dinner to your meals\_list list that holds what you plan to eat for dinner.

meals\_list$dinner <- c("onion rings", "cola zero", "oat schnitzel")  
  
meals\_list

$breakfast  
[1] "egg" "avocado" "sourdoughbread" "coffee"   
[5] "oatmilk"   
  
$lunch  
[1] "crepes" "banana" "nutella"  
  
$dinner  
[1] "onion rings" "cola zero" "oat schnitzel"

e) Use dollar notation to extract your dinner from your list and save it in a new vector called my\_dinner.

my\_dinner <- meals\_list$dinner

f) Use double-bracket notation to extract your lunch from your list and save it in your list as the element at index 5 (no reason beyond practice).

meals\_list[[5]] <- meals\_list[["lunch"]]  
  
meals\_list

$breakfast  
[1] "egg" "avocado" "sourdoughbread" "coffee"   
[5] "oatmilk"   
  
$lunch  
[1] "crepes" "banana" "nutella"  
  
$dinner  
[1] "onion rings" "cola zero" "oat schnitzel"  
  
[[4]]  
NULL  
  
[[5]]  
[1] "crepes" "banana" "nutella"

g) Use single-bracket notation to extract breakfast and lunch from your list and save them to a new list called early\_meals\_list.

early\_meals\_list <- meals\_list[1:2]  
  
early\_meals\_list

$breakfast  
[1] "egg" "avocado" "sourdoughbread" "coffee"   
[5] "oatmilk"   
  
$lunch  
[1] "crepes" "banana" "nutella"

#### Exercise 3.1 – Task 2: Using lapply()

a) Round the number pi to the nearest 0.1 (one decimal place) using the function round(). Use ?round, if needed.

round(pi, 1)

[1] 3.1

b) Create a list rnums of 10 random numbers.

* Hint: First, use the runif() function to create a vector of random numbers, then use as.list() to convert the result to a list.

Remark: runif(n, min, max) generates a vector of n random numbers between min and max from a uniform distribution.

rnums <- as.list(runif(10, 1, 100)) # 10 random numbers between 1 and 100, inclusive  
  
rnums

[[1]]  
[1] 46.30842  
  
[[2]]  
[1] 67.0513  
  
[[3]]  
[1] 95.01399  
  
[[4]]  
[1] 87.72148  
  
[[5]]  
[1] 44.78029  
  
[[6]]  
[1] 74.8713  
  
[[7]]  
[1] 83.6092  
  
[[8]]  
[1] 75.88092  
  
[[9]]  
[1] 52.20622  
  
[[10]]  
[1] 44.68786

c) Use lapply() to apply the round() function to each element of rnums, rounding it to the nearest 0.1 (one decimal place).

lapply(rnums, round, 1)

[[1]]  
[1] 46.3  
  
[[2]]  
[1] 67.1  
  
[[3]]  
[1] 95  
  
[[4]]  
[1] 87.7  
  
[[5]]  
[1] 44.8  
  
[[6]]  
[1] 74.9  
  
[[7]]  
[1] 83.6  
  
[[8]]  
[1] 75.9  
  
[[9]]  
[1] 52.2  
  
[[10]]  
[1] 44.7

#### Exercise 3.1 – Task 3: Using lapply() and sapply()

Create the list my\_list <- list(observationA = 16:8, observationB = exp(c(20:19, 6:12))).

a) Calculate the respective means of observationA and observationB . First use lapply, then use sapply(). What is the difference? Use class() to check the respective object classes.

my\_list <- list(observationA = 16:8, observationB = exp(c(20:19, 6:12)))  
  
my\_list

$observationA  
[1] 16 15 14 13 12 11 10 9 8  
  
$observationB  
[1] 4.851652e+08 1.784823e+08 4.034288e+02 1.096633e+03 2.980958e+03  
[6] 8.103084e+03 2.202647e+04 5.987414e+04 1.627548e+05

lapply(my\_list, mean) # returns a list

$observationA  
[1] 12  
  
$observationB  
[1] 73767193

sapply(my\_list, mean) # returns a vector (numeric)

observationA observationB   
 12 73767193

class(lapply(my\_list, mean))

[1] "list"

class(sapply(my\_list, mean))

[1] "numeric"

b) Calculate the respective quartiles of observationA and observationB . First use lapply, then use sapply(). What class are the respective output objects?

* Hint: You can get the quartiles using the function quantile().
* Remark: While mean returns a single value, quantile() returns a vector.

lapply(my\_list, quantile) # returns a list

$observationA  
 0% 25% 50% 75% 100%   
 8 10 12 14 16   
  
$observationB  
 0% 25% 50% 75% 100%   
4.034288e+02 2.980958e+03 2.202647e+04 1.627548e+05 4.851652e+08

sapply(my\_list, quantile) # returns a matrix

observationA observationB  
0% 8 4.034288e+02  
25% 10 2.980958e+03  
50% 12 2.202647e+04  
75% 14 1.627548e+05  
100% 16 4.851652e+08

class(lapply(my\_list, quantile))

[1] "list"

class(sapply(my\_list, quantile))

[1] "matrix" "array"

c) Apply the exponential function log() of to each element of observationB.

lapply(my\_list$observationB, log)

[[1]]  
[1] 20  
  
[[2]]  
[1] 19  
  
[[3]]  
[1] 6  
  
[[4]]  
[1] 7  
  
[[5]]  
[1] 8  
  
[[6]]  
[1] 9  
  
[[7]]  
[1] 10  
  
[[8]]  
[1] 11  
  
[[9]]  
[1] 12

# or  
log(my\_list[[2]])

[1] 20 19 6 7 8 9 10 11 12

d) Create the function my\_transformation <- function(x) { log(x) - 1 }. Apply my\_transformation() to each element of observationB. Try it first with vectorization, then with sapply().

my\_transformation <- function(x) { log(x) - 1 }  
  
lapply(my\_list$observationB, my\_transformation)

[[1]]  
[1] 19  
  
[[2]]  
[1] 18  
  
[[3]]  
[1] 5  
  
[[4]]  
[1] 6  
  
[[5]]  
[1] 7  
  
[[6]]  
[1] 8  
  
[[7]]  
[1] 9  
  
[[8]]  
[1] 10  
  
[[9]]  
[1] 11

# or  
sapply(my\_list$observationB, my\_transformation)

[1] 19 18 5 6 7 8 9 10 11

# or  
lapply(my\_list[[2]], my\_transformation)

[[1]]  
[1] 19  
  
[[2]]  
[1] 18  
  
[[3]]  
[1] 5  
  
[[4]]  
[1] 6  
  
[[5]]  
[1] 7  
  
[[6]]  
[1] 8  
  
[[7]]  
[1] 9  
  
[[8]]  
[1] 10  
  
[[9]]  
[1] 11

# or  
sapply(my\_list[[2]], my\_transformation)

[1] 19 18 5 6 7 8 9 10 11

# or  
my\_transformation(my\_list$observationB)

[1] 19 18 5 6 7 8 9 10 11

# or  
my\_transformation(my\_list[[2]])

[1] 19 18 5 6 7 8 9 10 11

### Self-Study 3.1. Lists

#### Self-Study 3.1 - Task 1: Using lapply()

a) Create a list yesterdays\_meals\_list of meals that you ate yesterday (breakfast, lunch, dinner).

* Remark: You can reuse your list from task 1, exercise 3.1.

yesterdays\_meals\_list <- list(breakfast = c("egg", "avocado", "sourdoughbread", "coffee", "oatmilk"),   
 lunch = c("crepes", "banana", "nutella"),   
 dinner = c("onion rings", "cola zero", "oat schnitzel"))  
  
yesterdays\_meals\_list

$breakfast  
[1] "egg" "avocado" "sourdoughbread" "coffee"   
[5] "oatmilk"   
  
$lunch  
[1] "crepes" "banana" "nutella"  
  
$dinner  
[1] "onion rings" "cola zero" "oat schnitzel"

b) Create a list that holds the number of items you ate for each meal .

* Hint: use the lappy() function to apply the length() function to each item.

lapply(yesterdays\_meals\_list, length)

$breakfast  
[1] 5  
  
$lunch  
[1] 3  
  
$dinner  
[1] 3

c) Write a function add\_schoggi that adds Schoggi (chocolate) to a given meal vector, and then returns the modified meal vector.

add\_schoggi <- function(meal){  
 meal <- c(meal, "schoggi") # add the schoggi  
 meal # returns the vector  
}

d) Create a vector better\_dinner that contains all the meals of yesterday’s dinner, but with schoggi added!

better\_dinner <- add\_schoggi(yesterdays\_meals\_list$dinner)  
  
better\_dinner

[1] "onion rings" "cola zero" "oat schnitzel" "schoggi"

e) Create a list better\_meals\_list that contains all the meals of yesterdays\_meals\_list, but with schoggi added.

better\_meals\_list <- lapply(yesterdays\_meals\_list, add\_schoggi)  
  
better\_meals\_list

$breakfast  
[1] "egg" "avocado" "sourdoughbread" "coffee"   
[5] "oatmilk" "schoggi"   
  
$lunch  
[1] "crepes" "banana" "nutella" "schoggi"  
  
$dinner  
[1] "onion rings" "cola zero" "oat schnitzel" "schoggi"

#### Self-Study 3.1 - Task 2: Using sapply()

a) Create a variable sentence that contains a sentence of text (something longish). Make the sentence lowercase; you can use a function to help.

sentence <- "Noe ist meine Beste Freundin und ich liebe sie und bin unendlich dankbar für sie!"  
  
sentence <- tolower(sentence)

b) Use the strsplit() function to split the sentence into a vector of letters.

* Hint: Split on “” to split every character.
* Note: This will return a list with 1 element ( - this element is the vector of letters).
* Remark: You don’t need to exclude punctuation marks.

letters\_list <- strsplit(sentence, "")  
  
letters\_list

[[1]]  
 [1] "n" "o" "e" " " "i" "s" "t" " " "m" "e" "i" "n" "e" " " "b" "e" "s" "t" "e"  
[20] " " "f" "r" "e" "u" "n" "d" "i" "n" " " "u" "n" "d" " " "i" "c" "h" " " "l"  
[39] "i" "e" "b" "e" " " "s" "i" "e" " " "u" "n" "d" " " "b" "i" "n" " " "u" "n"  
[58] "e" "n" "d" "l" "i" "c" "h" " " "d" "a" "n" "k" "b" "a" "r" " " "f" "ü" "r"  
[77] " " "s" "i" "e" "!"

c) Extract the vector of letters from the resulting list and store it in a variable called letters\_vector.

letters\_vector <- letters\_list[[1]]  
  
letters\_vector

[1] "n" "o" "e" " " "i" "s" "t" " " "m" "e" "i" "n" "e" " " "b" "e" "s" "t" "e"  
[20] " " "f" "r" "e" "u" "n" "d" "i" "n" " " "u" "n" "d" " " "i" "c" "h" " " "l"  
[39] "i" "e" "b" "e" " " "s" "i" "e" " " "u" "n" "d" " " "b" "i" "n" " " "u" "n"  
[58] "e" "n" "d" "l" "i" "c" "h" " " "d" "a" "n" "k" "b" "a" "r" " " "f" "ü" "r"  
[77] " " "s" "i" "e" "!"

d) Use the unique() function to get a vector of unique letters. Store it in the variable letters\_unique.

letters\_unique <- unique(letters\_vector)  
  
letters\_unique

[1] "n" "o" "e" " " "i" "s" "t" "m" "b" "f" "r" "u" "d" "c" "h" "l" "a" "k" "ü"  
[20] "!"

e) Count how many different letters occur in your sentence by counting the number of elements in letters\_unique.

* Hint: Use the function length(). (Notice that this includes punctuation marks! Just leave them in.)

length(letters\_unique)

[1] 20

f) How often does the letter ‘a’ occur in your sentence? (Don’t use loops but work with vectorization!)

* Remark: To find out, filter letters\_vector for the letter ‘a’. Then use the function length() on the filtered vector.
* Remark: Remember the lecture on vectors: You can filter a vector by using a vector of logicals (“logical subsetting”). To get the vector of logicals that you need for this task, specify the logical test that compares a given letter with the letter ‘a’. Apply this test to your letters\_vector. (Recycling vectorizes your test automatically!).

length(letters\_vector[letters\_vector == 'a'])

[1] 2

g) Now define a function count\_occurrences that takes two parameters: an arbitrary letter and a sentence. The function should return the number of times letter occurs in sentence.

* Remark: Test your function with your sentence and the letter “a”.

count\_occurrences <- function(letter, sentence) {  
 sentence <- tolower(sentence)  
 letters\_list <- strsplit(sentence, "")  
 letters\_vector <- letters\_list[[1]]  
 length(letters\_vector[letters\_vector == letter])  
}  
  
  
count\_occurrences("a", sentence)

[1] 2

h) Call your count\_occurrences() function to see how many times the letter ‘i’ is in your sentence.

count\_occurrences("i", sentence)

[1] 9

i) Use sapply() to apply your count\_occurrences() function to each unique letter in the vector to determine their frequencies.

sapply(letters\_unique, count\_occurrences, sentence)

n o e i s t m b f r u d c h l a k ü !   
10 1 11 14 9 4 2 1 4 2 3 4 5 2 2 2 2 1 1 1

j) Convert the result into a list (using as.list()). Print the resulting list of frequencies.

frequencies <- as.list(sapply(letters\_unique, count\_occurrences, sentence))  
  
print(frequencies)

$n  
[1] 10  
  
$o  
[1] 1  
  
$e  
[1] 11  
  
$` `  
[1] 14  
  
$i  
[1] 9  
  
$s  
[1] 4  
  
$t  
[1] 2  
  
$m  
[1] 1  
  
$b  
[1] 4  
  
$f  
[1] 2  
  
$r  
[1] 3  
  
$u  
[1] 4  
  
$d  
[1] 5  
  
$c  
[1] 2  
  
$h  
[1] 2  
  
$l  
[1] 2  
  
$a  
[1] 2  
  
$k  
[1] 1  
  
$ü  
[1] 1  
  
$`!`  
[1] 1

## 3.2. Data Frames

### Exercise 3.2. Data Frames

#### Exercise 3.2 – Task 1: Creating Data Frames

a) Create a vector of 100 employees (“Employee 1”, “Employee 2”, … “Employee 100”).

* Hint: use the paste() function and vector recycling to add a number to the word “Employee”.

employees <- paste("Employee", 1:100)

b) Create a vector of 100 random salaries for the year 2017. Use the runif() function to pick random numbers between 40’000 and 50’000.

salaries\_2017 <- runif(100, 40000, 50000)

c) Create a vector of 100 annual salary adjustments between -5’000 and +10’000. (A negative number represents a salary decrease.) Again use the runif() function to pick 100 random numbers in that range.

salary\_adjustment <- runif(100, -5000, 10000)

d) Create a data frame salaries by combining the 3 vectors you just made.

salaries <- data.frame(employees, salaries\_2017, salary\_adjustment)

e) Add a column to the salaries data frame that represents each person’s salary in 2018 (e.g., with the salary adjustment added in).

salaries$salaries\_2018 <- salaries$salaries\_2017 + salaries$salary\_adjustment

f) Add a column to the salaries data frame that has a value TRUE if the person got a raise (their salary went up).

salaries$got\_raise <- salaries$salaries\_2018 > salaries$salaries\_2017  
  
head(salaries)

employees salaries\_2017 salary\_adjustment salaries\_2018 got\_raise  
1 Employee 1 45758.45 -3809.482 41948.97 FALSE  
2 Employee 2 46571.47 -2673.457 43898.01 FALSE  
3 Employee 3 42255.22 9396.138 51651.36 TRUE  
4 Employee 4 45437.21 3620.772 49057.98 TRUE  
5 Employee 5 43252.45 -3900.214 39352.24 FALSE  
6 Employee 6 49899.00 1462.680 51361.68 TRUE

#### Exercise 3.2 – Task 2: Working with Data Frames

Retrieve values from your data frame salaries to answer the following questions.

* Note: You should get the value as specific as possible (e.g., a single cell rather than the whole row).

i. What was the 2018 salary of Employee 57

salaries[salaries$employees == "Employee 57", "salaries\_2018"]

[1] 44774.41

ii. How many employees got a raise?

nrow(salaries[salaries$got\_raise == TRUE, ])

[1] 66

iii. What was the dollar value of the highest raise?

max(salaries$salary\_adjustment)

[1] 9396.138

iv. What was the “name” of the employee who received the highest raise?

salaries[salaries$salary\_adjustment == max(salaries$salary\_adjustment) , "employees"]

[1] "Employee 3"

v. What was the largest decrease in salaries between the two years?

biggest\_paycut <- min(salaries$salary\_adjustment)  
  
biggest\_paycut

[1] -4479.127

vi. What was the name of the employee who received the largest decrease in salary?

salaries[salaries$salary\_adjustment == biggest\_paycut, "employees"]

[1] "Employee 50"

vii. What was the average salary change?

mean(salaries$salary\_adjustment)

[1] 2405.541

viii. For people who did not get a raise, how much money did they lose on average?

* Consider: Do the above averages match what you expected them to be based on how you generated the salaries?

mean(salaries$salary\_adjustment[salaries$got\_raise == FALSE])

[1] -2156.596

ix. Write a .csv file of your salary data to your working directory.

write.csv(salaries, "salaries.csv")

### Self-Study 3.2. Data Frames

#### Self-Study 3.2 - Task 1: Built-In Data Sets: US Personal Expenditures

a) Load R’s “USPersonalExpenditure” dataset using the data() function.

This will produce a data frame called USPersonalExpenditure.

The variable USPersonalExpenditure is now accessible to you.

Unfortunately, it’s not a data frame (it’s a matrix). Test this using the is.data.frame()and class() functions.

Luckily, you can pass the USPersonalExpenditure variable as an argument to the data.frame() function to convert it a data frame. Do this, storing the result in a new variable.

data("USPersonalExpenditure")  
  
is.data.frame(USPersonalExpenditure)

[1] FALSE

class(USPersonalExpenditure)

[1] "matrix" "array"

USPE <- data.frame(USPersonalExpenditure)

i. What are the column names of your data frame?

colnames(USPE)

[1] "X1940" "X1945" "X1950" "X1955" "X1960"

ii. Why are they so strange? Think about whether you could use a number like 1940 with dollar notation!

*An X is added automatically, so the columns can be used with the $.*

iii. What are the row names of your data frame?

rownames(USPE)

[1] "Food and Tobacco" "Household Operation" "Medical and Health"   
[4] "Personal Care" "Private Education"

b) Add a column category to your data frame that contains the rownames.

USPE$category <- rownames(USPE)

i. How much money was spent on personal care in 1940?

care\_1940 <- USPE["Personal Care", "X1940"]  
care\_1940

[1] 1.04

ii. How much money was spent on Food and Tobacco in 1960?

food\_1960 <- USPE["Food and Tobacco", "X1960"]  
food\_1960

[1] 86.8

iii. What was the highest expenditure category in 1960?

highest\_1960 <- USPE$category[USPE$X1960 == max(USPE$X1960)]  
highest\_1960

[1] "Food and Tobacco"

c) Define a function lowest\_category that takes in a year as a parameter, and returns the lowest spending category of that year.

i. Using your function, determine the lowest spending category of each year.

* Hint: Use the sapply() function to apply your function to a vector of years.

lowest\_category <- function(year) {  
 col <- paste0("X", year)  
 USPE$category[USPE[, col] == min(USPE[, col])]  
}  
  
lowest <- sapply(seq(1940, 1960, 5), lowest\_category)  
lowest

[1] "Private Education" "Private Education" "Private Education"  
[4] "Private Education" "Private Education"

#### Self-Study 3.2 - Task 2: External Data Sets: Gates Foundation Educational Grants

a) Use the read.csv() function to read the data file gates\_money.csv from Moodle into a variable called grants.

* Remark: The dataset holds data on Gates Foundation Educational Grants.
* Hint: Be sure to set your working directory in Rstudio. Use the View function to look at the loaded data

grants <- read.csv("gates\_money.csv")

b) Create a variable organization that contains the organization column of the dataset.

* Hint: Confirm that the “organization” column is a vector using the is.vector() function. This is a useful debugging tip if you hit errors later!

organization <- grants$organization  
  
is.vector(organization)

[1] TRUE

c) Now you can ask some interesting questions about the dataset:

i. What was the mean grant value?

mean(grants$total\_amount)

[1] 2600197

ii. What was the dollar amount of the largest grant?

max(grants$total\_amount)

[1] 100000000

iii. What was the dollar amount of the smallest grant?

min(grants$total\_amount)

[1] 5000

iv. Which organization received the largest grant?

grants$organization[grants$total\_amount == max(grants$total\_amount)]

[1] "Hillsborough County Public Schools"

v. Which organization received the smallest grant?

grants$organization[grants$total\_amount == min(grants$total\_amount)]

[1] "New Mexico Business Roundtable for Educational Excellence"

vi. How many grants were awarded in 2010?

length(grants$total\_amount[grants$start\_year == 2010])

[1] 18

#### Self-Study 3.2 - Task 3: Large Data Sets: Female Baby Names

a) Read in the female baby names data file found on Moodle (see lecture 1) into a variable called names.

names <- read.csv("female\_names.csv")  
is.data.frame(names)

[1] TRUE

b) Create a data frame names\_2013 that contains only the rows for the year 2013. What was the most popular female name in 2013?

names\_2013 <- names[names$year == 2013, ]  
  
names\_2013[names\_2013$prop == max(names\_2013$prop), "name"]

[1] "Sophia"

c) Write a function most\_popular\_in\_year that takes in a year as a value and returns the most popular name in that year. What was the most popular female name in 1994?

most\_popular\_in\_year <- function(year) {  
 names\_year <- names[names$year == year, ]  
 most\_popular <- names\_year[names$prop == max(names\_year$prop), "name"]  
 most\_popular  
}

d) Write a function number\_in\_million that takes in a name and a year, and returns statistically how many babies out of 1 million born that year have that name.

* Hint: Get the popularity percentage, and take that percentage out of 1 million.

number\_in\_million <- function(name, year){  
 name\_popularity <- names[names$year == year & names$name == name, "prop"]  
 round(name\_popularity \* 1000000, 1)  
}

i. How many babies out of 1 million had the name Laura in 1995?

ii. How many babies out of 1 million had your name in the year you were born? Consider: What does this tell you about how easy it is to identify you with just your name and birth year?

number\_in\_million ("Laura", 1995)

[1] 3125.3

number\_in\_million ("Michele", 1995)

[1] 245.7