R for Data Analytics Part 1, Lecture 3

Michèle Fille

Table of contents

Lecture 3 – Data Structures for Data Science: Lists and Data Frames **1**
3.1 Lists 1
Exercise 3.1. Lists 1
Exercise 3.1 – Task 1: Creating and Accessing Lists 1
Exercise 3.1 – Task 2: Using lapply() 4
Exercise 3.1 – Task 3: Using lapply() and sapply() 6
Self-Study 3.1. Lists 11
Self-Study 3.1 - Task 1: Using lapply() 11
Self-Study 3.1 - Task 2: Using sapply() 12
3.2. Data Frames 17
Exercise 3.2. Data Frames 17
Exercise 3.2 – Task 1: Creating Data Frames 17
Exercise 3.2 – Task 2: Working with Data Frames 18
Self-Study 3.2. Data Frames 20
Self-Study 3.2 - Task 1: Built-In Data Sets: US Personal Expenditures . 20
Self-Study 3.2 - Task 2: External Data Sets: Gates Foundation Educational Grants 22
Self-Study 3.2 - Task 3: Large Data Sets: Female Baby Names 23

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.

```r
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).

```r
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.

```r
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.

```r
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.

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

rnums
```

```
[[1]]
[1] 70.48019

[[2]]
[1] 42.57768

[[3]]
[1] 32.4652

[[4]]
[1] 66.56008

[[5]]
[1] 83.0012

[[6]]
[1] 48.78605

[[7]]
[1] 68.28374
```

```
[[8]]
[1] 60.8599

[[9]]
[1] 24.09873

[[10]]
[1] 34.11738
```

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

```r
lapply(rnums, round, 1)
```

```
[[1]]
[1] 70.5

[[2]]
[1] 42.6

[[3]]
[1] 32.5

[[4]]
[1] 66.6

[[5]]
[1] 83

[[6]]
[1] 48.8

[[7]]
[1] 68.3

[[8]]
[1] 60.9

[[9]]
[1] 24.1

[[10]]
```

```
[1] 34.1
```

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.

```r
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
```

```r
lapply(my_list, mean) # returns a list
```

```
$observationA
[1] 12

$observationB
[1] 73767193
```

```r
sapply(my_list, mean) # returns a vector (numeric)
```

```
observationA observationB
          12     73767193
```

```r
class(lapply(my_list, mean))
```

```
[1] "list"
```

```r
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.

```r
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 
```

```r
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
```

```r
class(lapply(my_list, quantile))
```

```
[1] "list"
```

```r
class(sapply(my_list, quantile))
```

```
[1] "matrix" "array" 
```

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

```r
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
```

```r
# 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().

```r
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
```

```r
# or
sapply(my_list$observationB, my_transformation)
```

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

```r
# 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",
                              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.

```r
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!

```r
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.

```r
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.

```r
sentence <- "Noe ist meine Beste Freundin und ich liebe sie und bin unendlich dankbar für

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.

```r
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.

```r
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.

```r
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      46801.80        8154.93064      54956.73      TRUE
2 Employee 2      43246.12         322.57572      43568.70      TRUE
```

```
3 Employee 3      42527.68         2200.81211      44728.49      TRUE
4 Employee 4      48313.93           96.38663      48410.31      TRUE
5 Employee 5      46600.91         4781.65592      51382.56      TRUE
6 Employee 6      44315.09         1495.54344      45810.63      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

```r
salaries[salaries$employees == "Employee 57", "salaries_2018"]
```

```
[1] 46741.06
```

ii. How many employees got a raise?

```r
nrow(salaries[salaries$got_raise == TRUE, ])
```

```
[1] 71
```

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

```r
max(salaries$salary_adjustment)
```

```
[1] 9971.201
```

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

```r
salaries[salaries$salary_adjustment == max(salaries$salary_adjustment) , "employees"]
```

```
[1] "Employee 29"
```

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

```r
biggest_paycut <- min(salaries$salary_adjustment)

biggest_paycut
```

```
[1] -4907.873
```

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

```r
salaries[salaries$salary_adjustment == biggest_paycut, "employees"]
```

```
[1] "Employee 27"
```

vii. What was the average salary change?

```r
mean(salaries$salary_adjustment)
```

```
[1] 3173.747
```

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?

```r
mean(salaries$salary_adjustment[salaries$got_raise == FALSE])
```

```
[1] -2609.731
```

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

```r
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.

```r
data("USPersonalExpenditure")

is.data.frame(USPersonalExpenditure)
```

```
[1] FALSE
```

```r
class(USPersonalExpenditure)
```

```
[1] "matrix" "array"
```

```r
USPE <- data.frame(USPersonalExpenditure)
```

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

```r
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?

```r
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])]
}
```

```r
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

```r
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!

```r
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?

```r
mean(grants$total_amount)
```

```
[1] 2600197
```

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

```r
max(grants$total_amount)
```

```
[1] 100000000
```

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

```r
min(grants$total_amount)
```

```
[1] 5000
```

iv. Which organization received the largest grant?

```r
grants$organization[grants$total_amount == max(grants$total_amount)]
```

```
[1] "Hillsborough County Public Schools"
```

v. Which organization received the smallest grant?

```r
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?

```r
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.

```r
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?

```r
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?

```r
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.

```r
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?

```r
number_in_million ("Laura", 1995)
```

```
[1] 3125.3
```

```r
number_in_million ("Michele", 1995)
```

```
[1] 245.7
```