Data Analysis and Visualization in R (IN2339)

Exercise Session 1

Daniela Klaproth-Andrade, Julien Gagneur

Quizzes (from the lecture)

The following quizzes will be solved orally by the students and the professor during the lecture.

1. What is the output of `rep(6:9, 2)` ?

   a. 6 7 8 9 6 7 8 9
   b. 6 6 7 7 8 8 9 9
   c. "6", "7", "8", "9", "6", "7", "8", "9"
   d. "6", "6", "7", "7", "8", "8", "9", "9"

2. What is the output of the following code: `c(3 != sqrt(9), TRUE == (3 > 8))`?

   a. FALSE TRUE
   b. TRUE
   c. FALSE FALSE
   d. FALSE

4. Let `x <- c(1, 6, 3, 2)`. What is the output of `sort(x)`?

   a. 1 2 3 6
   b. 6 3 2 1
   c. 1 4 3 2
   d. 2 3 4 1

Tutorial

The following exercises will be solved during the tutorial sessions.

Section 00 - Getting ready

1. Make sure you have already installed and loaded the library `dslabs` by running the following commands:

```
install.packages("dslabs")  # execute only once!
library(dslabs)   # execute in every new script
```

Section 01 - R basics

1. What is the sum of the first 100 positive integers? The formula for the sum of integers from 1 to n is $n(n+1)/2$. Define $n = 100$ and then use R to compute the sum from 1 to 100 using the formula. What is the value of the sum?

2. Now use the same formula to compute the sum of the integers from 1 to 1,000.

3. Look at the result of evaluating the following code into R:

```
n <- 1000
x <- seq(1, n)
sum(x)
```

```
## [1] 500500
```

Based on the result, what do you think the functions `seq` and `sum` do? You can use `help`.

a. `sum` creates a vector of consecutive numbers and `seq` adds them up.
b. `seq` creates a vector of consecutive numbers and `sum` adds them up.
c. `seq` creates a random vector and `sum` computes the sum of 1 to n.
d. `sum` always returns the same number.

4. In math and programming, we say that we evaluate a function when we replace the argument with a given value. So if we type `sqrt(4)`, we evaluate the `sqrt` function. In R, you can evaluate a function inside another function. The evaluations happen from the inside out. Use one line of code to compute the logarithm, in base 10, of the square root of 100.

5. Assuming `x` is numeric, which of the following will always return the value stored in `x`? You can try out examples and use the help system if you want.

a. `log(10^x)`
b. `log10(x^10)`
c. `log(exp(x))`
d. `exp(log(x, base = 2))`

6. What is the outcome of the following sum $1 + 1/2^2 + 1/3^2 + \cdots + 1/100^2$? Thanks to Euler, we know it approximates to $\pi^2/6$. Compute the sum and check that it is close to the approximation. Note that R has a variable for π stored as `pi`.

Section 02 - Basic data

1. Load the US murders dataset from the `dslabs` package executing the following command:

```
data(murders)
```

Use the function `str` to examine the structure of the `murders` object. Which of the following best describes the variables represented in this data frame?

a. The 50 states and DC.
b. The murder rates for all 50 states and DC.
c. The state name, the abbreviation of the state name, the state's region, and the state's population and total number of murders for 2010.
d. `str` shows no relevant information.

2. What are the column names used by the data frame for these five variables?

3. Use the accessor `$` to extract the state abbreviations and assign them to the variable `a`. What is the class of this object?

4. We saw that the `region` column stores a factor. You can corroborate this by typing:

```
class(murders$region)
```

With one line of code, use the function `levels` and `length` to determine the number of regions defined by this dataset.

5. The function `table` takes one or multiple vectors and returns the frequency of each element. For example:

```
x <- c('DC', 'Alabama', 'Florida', 'Florida', 'DC', 'DC')
table(x)
```

You can quickly see how many states are in each region by applying this function. Use this function in one line of code to create a table of states per region.

6. Which state has the highest percentage of murders?

Section 03 - Working with vectors

1. Use the function `c` to create a vector with the average high temperatures in January for Beijing, Lagos, Paris, Rio de Janeiro, San Juan, and Toronto, which are 35, 88, 42, 84, 81, and 30 degrees Fahrenheit. Call the object `temp`.

2. Now create a vector with the city names and call the object `city`.

3. Use the `names` function and the objects defined in the previous exercises to associate the temperature data with its corresponding city.

4. Use the `[` and `:` operators to access the temperature of the first three cities on the list.

5. Use the `[` operator to access the temperature of Paris and San Juan.

6. Use the `:` operator to create a sequence of numbers $12, 13, 14, \ldots, 73$.

7. Create a vector containing all the positive odd numbers smaller than 100.

8. Create a vector of numbers that starts at 6, does not pass 55, and adds numbers in increments of 4/7: 6, 6 + 4/7, 6 + 8/7, and so on. How many numbers does the vector have? Hint: use `seq` and `length`.

9. What is the class of the following object `a <- seq(1, 10, 0.5)`?

10. What is the class of the following object `a <- seq(1, 10)`?

Homework

Solve the exercises below at home. The solutions will be discussed in the central exercise.

Section 04 - sorting and ranking

For these exercises we will use the US murders dataset. Make sure you load it before starting.

```
library(dslabs)
data("murders")
```

1. Use the `$` operator to access the population size and store it as the object `pop`. Then use the `sort` function to redefine `pop` so that it is sorted. Finally, use the `[` operator to report the smallest population size. Confirm this is the minimum value using the `min` function.

2. Now instead of the smallest population size, find the index of the entry with the smallest population size. Hint: use `order` instead of `sort`.

3. We can actually perform the same operation as in the previous exercise using the function `which.min`. Write one line of code that does this.

4. Now we know what the smallest population is and the row which contains it. What is the name of this state?

5. You can create a data frame using the `data.frame` function. Here is a quick example:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
          "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Use the `rank` function to determine the population rank of each state from smallest population size to biggest. Save these ranks in an object called `ranks`, then create a data frame with the state name and its rank. Call the data frame `my_df`.

Section 05 - Vector arithmetics

1. Have a look a the following lines of code:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
          "San Juan", "Toronto")
names(temp) <- city
```

Use the `temp` vector to create a different vector `temp_F` that converts the temperature from Fahrenheit to Celsius. The conversion is $C = \frac{5}{9} \times (F - 32)$.

2. The `na_example` vector represents a series of counts. You can quickly examine the object using:

```
data("na_example")
str(na_example)
```

However, when we compute the average with the function `mean`, we obtain `NA`:

```
mean(na_example)
```

The `is.na` function returns a logical vector that tells us which entries are `NA`. Assign this logical vector to an object called `ind` and determine how many `NA`s does `na_example` have.

3. Now compute the average again, but only for the entries that are not `NA`.

Section 06 - Factors and sorting

How do the returned values of each of these calls differ? Explain why.

```
sort(factor(c("red", "green", "blue")))

sort(factor(c("red", "green", "blue"), levels = c("red", "green", "blue")))
```