Apuntes R

Pilar Amat Rodrigo

7/12/2017

Contents

Introduction to R

Sources : KSchool y Data Camp Material : https://github.com/joseramoncajide/master_data_science ##Basics R works with numerous data types. Some basic types are:

```
Decimals values -> like 5,6 are called numerics.
Natural numbers -> like 5 are called integers. Integers are also numerics.
Boolean values -> (TRUE or FALSE) are called logical.
Text (or string) -> values are called characters.
```

Note how the quotation marks on the right indicate that "some text" is a character.

If you want to check out the contents of the workspace, you can type ls() in the console

NOTE: R is CASE SENSITIVE

What's that data type?

Do you remember that when you added 5 + "four", you got an error due to a mismatch in data types? You can avoid such embarrassing situations by checking the data type of a variable beforehand. You can do this with the class() function, as the code on the right shows.

```
# Declare variables of different types
my_numeric <- 33
my_character <- "window"
my_logical <- TRUE

# Check class of my_numeric
class(my_numeric)
```

```
## [1] "numeric"
```

```
# Check class of my_character
class(my_character)
```

```
## [1] "character"
```

```
# Check class of my_logical
class(my_logical)
```

```
## [1] "logical"
```

Vectors

In R, you create a vector with the combine function c(). You place the vector elements separated by a comma between the parentheses. For example:

```
numeric_vector <- c(1, 2, 3)
character_vector <- c("a", "b", "c")
```

Once you have created these vectors in R, you can use them to do calculations.

Name vectors

You can give a name to the elements of a vector with the names() function. Have a look at this example:

```
some_vector <- c("John Doe", "poker player")
names(some_vector) <- c("Name", "Profession")
```

This code first creates a vector some_vector and then gives the two elements a name. The first element is assigned the name Name, while the second element is labeled Profession. Printing the contents to the console yields following output:

```
      Name     Profession
"John Doe" "poker player"
```

You can also create a variable that contains the days of the week. This way you can use and re-use it.

The variable days_vector

```
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
```

Assign the names of the day to roulette_vector and poker_vector

```
names(some_vector) <- days_vector
names(some_vector2) <- days_vector
```

Arithmetic calculations on vectors

It is important to know that if you sum two vectors in R, it takes the element-wise sum. For example, the following three statements are completely equivalent:

```
c(1, 2, 3) + c(4, 5, 6)
c(1 + 4, 2 + 5, 3 + 6)
c(5, 7, 9)
```

You can also do the calculations with variables that represent vectors:

```
a <- c(1, 2, 3)
b <- c(4, 5, 6)
c <- a + b
c
```

```
## [1] 5 7 9
```

SUM()

A function that helps you to answer this question is sum(). It calculates the sum of all elements of a vector. For example, to calculate the total amount of money you have lost/won with poker you do:

```
total_poker <- sum(poker_vector)
```

Vector selection

In other words, our goal is to select specific elements of the vector. To select elements of a vector (and later matrices, data frames, ...), you can use square brackets. Between the square brackets, you indicate what elements to select. For example, to select the first element of the vector, you type poker_vector[1]. To select the second element of the vector, you type poker_vector[2], etc. Notice that the first element in a vector has index 1, not 0 as in many other programming languages.

To select multiple elements from a vector, you can add square brackets at the end of it. You can indicate between the brackets what elements should be selected. For example: suppose you want to select the first and the fifth day of the week: use the vector c(1, 5) between the square brackets. For example, the code below selects the first and fifth element of poker_vector:

```
poker_vector[c(1, 5)]
```

So, another way to find the mid-week results is poker_vector[2:4]. Notice how the vector 2:4 is placed between the square brackets to select element 2 up to 4.

Another way is by using the names of the vector elements (Monday, Tuesday, ...) instead of their numeric positions. For example,

```
poker_vector["Monday"]
```

will select the first element of poker_vector since "Monday" is the name of that first element.

Just like you did in the previous exercise with numerics, you can also use the element names to select multiple elements, for example:

```
poker_vector[c("Monday","Tuesday")]
```

Selection by comparison

The (logical) comparison operators known to R are:

```
< for less than
> for greater than
<= for less than or equal to
>= for greater than or equal to
== for equal to each other
!= not equal to each other
```

As seen in the previous chapter, stating $6 > 5$ returns TRUE. The nice thing about R is that you can use these comparison operators also on vectors. For example:

```
> c(4, 5, 6) > 5
[1] FALSE FALSE TRUE
```

This command tests for every element of the vector if the condition stated by the comparison operator is TRUE or FALSE.

```r
# Poker and roulette winnings from Monday to Friday:
poker_vector <- c(140, -50, 20, -120, 240)
roulette_vector <- c(-24, -50, 100, -350, 10)
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
names(poker_vector) <- days_vector
names(roulette_vector) <- days_vector

# Which days did you make money on poker?
selection_vector <- poker_vector>0
```

```
# Print out selection_vector
selection_vector
```

```
##    Monday   Tuesday Wednesday  Thursday    Friday
##      TRUE     FALSE      TRUE     FALSE      TRUE
```

In the previous exercises you used selection_vector <- poker_vector > 0 to find the days on which you had a positive poker return. Now, you would like to know not only the days on which you won, but also how much you won on those days.

You can select the desired elements, by putting selection_vector between the square brackets that follow poker_vector:

```
poker_vector[selection_vector]
```

Matrix

In R, a matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns. Since you are only working with rows and columns, a matrix is called two-dimensional.

You can construct a matrix in R with the matrix() function. Consider the following example:

```
matrix(1:9, byrow = TRUE, nrow = 3)
```

In the matrix() function:

- The first argument is the collection of elements that R will arrange into the rows and columns of the matrix. Here, we use 1:9 which is a shortcut for c(1, 2, 3, 4, 5, 6, 7, 8, 9).
- The argument byrow indicates that the matrix is filled by the rows. If we want the matrix to be filled by the columns, we just place byrow = FALSE.
- The third argument nrow indicates that the matrix should have three rows.

Name Matrix

Similar to vectors, you can add names for the rows and the columns of a matrix

```
rownames(my_matrix) <- row_names_vector
colnames(my_matrix) <- col_names_vector
```

```
# Box office Star Wars (in millions!)
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.8)

# Construct matrix
star_wars_matrix <- matrix(c(new_hope, empire_strikes, return_jedi), nrow = 3, byrow = TRUE)

# Vectors region and titles, used for naming
region <- c("US", "non-US")
titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")

# Name the columns with region
colnames(star_wars_matrix)<- region

# Name the rows with titles
rownames(star_wars_matrix)<- titles
```

```
# Print out star_wars_matrix
star_wars_matrix
```

```
##                             US non-US
## A New Hope              460.998  314.4
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
```

Aritmetic calculations

In R, the function rowSums() conveniently calculates the totals for each row of a matrix. This function creates a new vector, idem for columns

```
rowSums(my_matrix)
rowSums(my_matrix)
```

Adding parts

You can add a column or multiple columns to a matrix with the cbind() function, which merges matrices and/or vectors together by column. For example:

```
big_matrix <- cbind(matrix1, matrix2, vector1 ...)
```

Just like every action has a reaction, every cbind() has an rbind().

Selection of matrix elements

Similar to vectors, you can use the square brackets [ ] to select one or multiple elements from a matrix. Whereas vectors have one dimension, matrices have two dimensions. You should therefore use a comma to separate that what to select from the rows from that what you want to select from the columns. For example:

```
my_matrix[1,2] selects the element at the first row and second column.
my_matrix[1:3,2:4] results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3, 4.
```

If you want to select all elements of a row or a column, no number is needed before or after the comma, respectively:

```
my_matrix[,1] selects all elements of the first column.
my_matrix[1,] selects all elements of the first row.
```

Example

```
                         US non-US
A New Hope              461.0  314.4
The Empire Strikes Back 290.5  247.9
Return of the Jedi      309.3  165.8
The Phantom Menace      474.5  552.5
Attack of the Clones    310.7  338.7
Revenge of the Sith     380.3  468.5

*Select the non-US revenue for all movies
non_us_all <- all_wars_matrix[,2]

*Select the non-US revenue for first two movies
non_us_some <- all_wars_matrix[1:2,2]
```

Factors

To create factors in R, you make use of the function factor(). First thing that you have to do is create a vector that contains all the observations that belong to a limited number of categories. For example, gender_vector contains the sex of 5 different individuals:

```
gender_vector <- c("Male","Female","Female","Male","Male")
```

There are two types of categorical variables: a nominal categorical variable and an ordinal categorical variable.

Factor levels

When you first get a data set, you will often notice that it contains factors with specific factor levels. However, sometimes you will want to change the names of these levels for clarity or other reasons. R allows you to do this with the function levels():

```
levels(factor_vector) <- c("name1", "name2",...)
```

A good illustration is the raw data that is provided to you by a survey. A standard question for every questionnaire is the gender of the respondent. You remember from the previous question that this is a factor and when performing the questionnaire on the streets its levels are often coded as "M" and "F".

```
survey_vector <- c("M", "F", "F", "M", "M")
```

Next, when you want to start your data analysis, your main concern is to keep a nice overview of all the variables and what they mean. At that point, you will often want to change the factor levels to "Male" and "Female" instead of "M" and "F" to make your life easier.

Watch out: the order with which you assign the levels is important. If you type levels(factor_survey_vector), you'll see that it outputs [1] "F" "M". If you don't specify the levels of the factor when creating the vector, R will automatically assign them alphabetically. To correctly map "F" to "Female" and "M" to "Male", the levels should be set to c("Female", "Male"), in this order.

Example:

```
# Code to build factor_survey_vector
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)

# Specify the levels of factor_survey_vector
levels(factor_survey_vector)<- c("Female", "Male")
```

Summarizing a factor

After finishing this course, one of your favorite functions in R will be summary(). This will give you a quick overview of the contents of a variable:

```
summary(my_var)
```

Example:

```
# Build factor_survey_vector with clean levels
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)
levels(factor_survey_vector) <- c("Female", "Male")
factor_survey_vector
```

```
## [1] Male   Female Female Male   Male
## Levels: Female Male
```

```r
# Generate summary for survey_vector
summary(survey_vector)
```

```
##    Length     Class      Mode
##         5 character character
```

```r
# Generate summary for factor_survey_vector
summary(factor_survey_vector)
```

```
## Female   Male
##      2      3
```

Ordered factors

To create an ordered factor, you have to add two additional arguments: ordered and levels.

```r
factor(some_vector,
   ordered = TRUE,
   levels = c("lev1", "lev2" ...))
```

By setting the argument ordered to TRUE in the function factor(), you indicate that the factor is ordered. With the argument levels you give the values of the factor in the correct order.

```r
# Create speed_vector
speed_vector <- c("fast", "slow", "slow", "fast", "insane")

# Convert speed_vector to ordered factor vector
factor_speed_vector <- factor(speed_vector, ordered=TRUE, levels=c("slow", "fast", "insane"))

# Print factor_speed_vector
factor_speed_vector
```

```
## [1] fast   slow   slow   fast   insane
## Levels: slow < fast < insane
```

Data Frames

You will often find yourself working with data sets that contain different data types instead of only one (numeric).

A data frame has the variables of a data set as columns and the observations as rows.

Overview

he function head() enables you to show the first observations of a data frame. Similarly, the function tail() prints out the last observations in your data set.

Both head() and tail() print a top line called the 'header', which contains the names of the different variables in your data set.

Another method that is often used to get a rapid overview of your data is the function str(). The function str() shows you the structure of your data set. For a data frame it tells you:

The total number of observations (e.g. 32 car types) The total number of variables (e.g. 11 car features) A full list of the variables names (e.g. mpg, cyl ...) The data type of each variable (e.g. num) The first observations

Create one

Read the example

```
# Definition of vectors
name <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
          "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)

# Create a data frame from the vectors
planets_df <- data.frame(name,type,diameter,rotation,rings)
```

Select elements

from a data frame with the help of square brackets [ ]. By using a comma, you can indicate what to select from the rows and the columns respectively. For example:

`my_df[1,2] selects the value at the first row and select element in my_df.`
`my_df[1:3,2:4] selects rows 1, 2, 3 and columns 2, 3, 4 in my_df.`

Sometimes you want to select all elements of a row or column. For example, my_df[1,] A possible disadvantage of this approach is that you have to know (or look up) the column number of type, which gets hard if you have a lot of variables. It is often easier to just make use of the variable name:

`planets_df[1:3,"type"] (type is the name of the column)`

However, there is a short-cut. If your columns have names, you can use the $ sign:

`planets_df$diameter`

Subset

To select only a specific part

`subset(my_df, subset = some_condition)`

The first argument of subset() specifies the data set for which you want a subset. By adding the second argument, you give R the necessary information and conditions to select the correct subset.

`subset(planets_df, subset = rings)`

Sorting

In data analysis you can sort your data according to a certain variable in the data set. In R, this is done with the help of the function order().

order() is a function that gives you the ranked position of each element when it is applied on a variable, such as a vector for example:

```
    a <- c(100, 10, 1000)
    order(a)
```

```
## [1] 2 1 3
```

10, which is the second element in a, is the smallest element, so 2 comes first in the output of order(a). 100, which is the first element in a is the second smallest element, so 1 comes second in the output of order(a).

This means we can use the output of order(a) to reshuffle a:

```
a[order(a)]
```

```
## [1]   10  100 1000
```

Example

```
# Play around with the order function in the console
premios<- c( 100, 200, 30, 50, 230, 4500)
order(premios)
```

```
## [1] 3 4 1 2 5 6
```

```
premios[order(premios)]
```

```
## [1]   30   50  100  200  230 4500
```

Lists

A list in R allows you to gather a variety of objects under one name (that is, the name of the list) in an ordered way. These objects can be matrices, vectors, data frames, even other lists, etc. It is not even required that these objects are related to each other in any way.

You could say that a list is some kind super data type: you can store practically any piece of information in it!

To construct a list you use the function list():

```
my_list <- list(comp1, comp2 ...)
```

The arguments to the list function are the list components. Remember, these components can be matrices, vectors, other lists, ...

Name

That is why you should give names to them:

```
my_list <- list(name1 = your_comp1,
                name2 = your_comp2)
```

This creates a list with components that are named name1, name2, and so on. If you want to name your lists after you've created them, you can use the names() function as you did with vectors. The following commands are fully equivalent to the assignment above:

```
my_list <- list(your_comp1, your_comp2)
names(my_list) <- c("name1", "name2")
```

Selecting elements from a list

Your list will often be built out of numerous elements and components. Therefore, getting a single element, multiple elements, or a component out of it is not always straightforward.

One way to select a component is using the numbered position of that component. For example, to "grab" the first component of shining_list you type

```
shining_list[[1]]
```

A quick way to check this out is typing it in the console. Important to remember: to select elements from vectors, you use single square brackets: [ ]. Don't mix them up!

You can also refer to the names of the components, with [[ ]] or with the $ sign. Both will select the data frame representing the reviews:

```
shining_list[["reviews"]]
shining_list$reviews
```

Adding new information

To conveniently add elements to lists you can use the c() function, that you also used to build vectors:

```
ext_list <- c(my_list , my_val)
```

This will simply extend the original list, my_list, with the component my_val. This component gets appended to the end of the list. If you want to give the new list item a name, you just add the name as you did before:

```
ext_list <- c(my_list, my_name = my_val)
```