Practical: R’s data objects

BaselRBootcamp 2017

Slides

Here a link to the lecture slides for this session: **LINK**
(https://therbootcamp.github.io/_sessions/D1S2_Objects/Objects.html)

Overview

In this practical you’ll learn how to use R’s basic data objects. By the end of this practical you will know how to:

1. Create data objects of different kinds
2. How to inspect objects
3. Change object types
4. Access elements from data objects

Functions

Here are the main functions for object creation.

Function	Description
`c(), rep(), seq(), numeric(), etc.`	Create a vector
`matrix(), cbind()`	Create a matrix
`data.frame()`	Create a data.frame
`list()`	Create a list

Here are the main functions for object inspection.

Function	Description
`head(), tail()`	Inspect first or last elements of object
`str()`	Inspect the structure of the data object
`View()`	To access an Excel like data interface

Here are the main functions for object selection.

Function	Description
`[ ]`	Single brackets: Select individual elements from `vector`
`[,]`	Single brackets 2d: Select rows, columns, or elements from `matrix` or `data.frame`
`[[ ]]`	Double brackets: Select element/variable in list or data.frame
`[[$]]`	Dollar: Select named element/variable from list or data.frame

Tasks

Vectors

1. Create a numeric (double), a character, and a logical vector each with 10 elements using `c()` and store them as `dbl_vec` , `log_vec` , and `chr_vec` (using the assignment operator, i.e., `name <- c())` .
2. Ensure the type and length of the vectors using `typeof()` and `length()` . Simply place the name of the vectors into the parenthesis and execute.
3. Expand each of the vectors using `rep()` to two times their length. `rep()` is a general purpose repeat function for vectors. It takes at least two arguments: `x` the vector and `times` , which can be a single integer, e.g., `2` . Two expand the vectors use the repeat function and reassign the output to the object á la `object <- function(object, arguments)` .

Matrices & Data frames

4. Use the 3 vectors to create a 3xN `matrix` where X is the length of the vectors. Matrices are usually created in one of two ways. First, a matrix can be created using the `matrix()` function. The easier way, however, is, second, via the column bind function `cbind()` . `cbind` simply takes vectors (of equal length) as input and binds them together. For instance, consider `example <- cbind(c(1,2),c(1,2))` . Try this using your three vectors and call the resulting matrix `my_mat` . Then look at the matrix using `head()` , and test its type (`typeof()`) and dimension (`dim()`). Why is the type what it is?
5. Use the 3 vectors to create a 3xN `data.frame` . To create a data frame use the `data.frame()` function. It works just like `cbind` . Call the data frame `my_df` . Now, look at the data frame using `head()` , and test its type (`typeof`) and dimension (`dim()`). Why is the type what it is?
6. Inspect the types of the columns of `my_df` . To do this try several of the different options to access columns: (1) Using double brackets and index, e.g., `[[index]]` , (2) using double brackets and name `[['column_name']]` , (3) using 2d single brackets and index, e.g., `[,index]` , (4) using using 2d single brackets and name, e.g., `[,'column_name']` , using the dollor operator `$` , e.g., `$column_name` . To see the names of the columns use `names()` . Finally, use `str(my_df)` to verify the types of the columns.
7. Transform `my_mat` into a data frame using `as.data.frame()` . Call it `my_df_2` and valuate its contents using `str()` . Now change the columns to their appropriate type using `as.double()` and `as.logical()` . To do this you need to select the appropriate column on both sides of the assignment. E.g., `mat[,1] <- as.numeric(mat[,1])` . Of course, you can also use the other ways of selecting a column. Afterwards reevaluate the types using `str()` .

Lists

8. Use the 3 vectors to create a list of length 3 using `list()` . `list()` works exactly as `c()` (or `data.frame` and `cbind`). Call the object `my_list` and inspect it using `str()` . Compare the output of `str()` to the output for `my_df` . What is different, and why?
9. Transform `my_list` into a data frame using `as.data.frame()` . Call the object `my_df_3` . Inspect again using `str()` . Rename the columns to the original names of the vectors. To do this assign to `names(my_df_3)` a character vector of length 3 containing those names. Inspect and compare names to those of `my_df` and `my_df2`

Logical comparisons

10. An important tool of working with data are logical comparisons. Logical comparisons can be used to conveniently select parts of the data. They can also be used to make checks throughout script. For instance, we could use logical comparisons to compare whether the names of, e.g., `my_df` and `my_3` are now equal. To do this the two name vectors need to be compared using the is-equal-to operator

`==` , e.g., `vec_a == vec_b` . Such logical operators will iterate throuh every index in the two involved vectors (beginning with 1) and compare whether the elements at the present location are equal. The result is a logical vector of the same length as the other two vectors. To evaluate whether all elements are equal one can conveniently use simple arithmetic functions such as `sum()` or `mean()` . Try now to check whether the names of these to data frames are indeed equal now. Remember `TRUE` is coerced to 1 and `False` to 0.

11. A second important use for logicals is subsetting. All object accessing using brackets, e.g., `[]` , can be used with logical vectors (provided that the dimensions match). E.g., `c(1, 2, 3)[c(TRUE, FALSE, TRUE)]` returns the first and the third element of the vector. The program thus iterates through every index and returns those elements for which the logical vector is `TRUE` . Try to use this now on the three vectors. Use `log_vec` to subset the elements in `dbl_vec` and `chr_vec` .

12. Logical vectors become especially useful, when they are used to subset based on specific conditions. For instance, we may be interested in retrieving all values in `chr_vec` for which `dbl_vec` is larger than some value. This can be easily accomplished by coercing `dbl_vec` to a logical vector using `>` . Consider `c(1, 4, 7, 2) > 3` . Try this for `chr_vec` and `dbl_vec` with an appropriate cut-off value.

13. Another convenient aspect of working with logical vectors is that they can be conveniently combined using the logical AND operator `&` and the logical `OR` operator `|` . Consider `c(1, 4, 7, 2) > 3 & c(1, 4, 7, 2) < 6` and `c(1, 4, 7, 2) > 3 | c(1, 4, 7, 2) < 6` . Try now combining the logical vector coerced from `dbl_vec` with `log_vec` to subset `chr_vec` .

Non-flat objects

14. Create a list containing `my_df` and the three vectors. Try to verify (using the list structure) that each of the columns of `my_df` is equal the respective vector. To access the columns in `my_df` you have to combine selectors. Can be any of `my_list[[index]][[index]]` , `my_list[[index]]$name` , `my_list$name$name` , etc.

Additional reading

- For more details on all steps of data analysis check out Hadley Wickham's R for Data Science (http://r4ds.had.co.nz/).

- For more advanced content on objects check out Hadley Wickham's Advanced R (http://adv-r.had.co.nz/).

- For more on pirates and data analysis check out the respective chapters in YaRrr! The Pirate's Guide to R YaRrr! Chapter Link (https://bookdown.org/ndphillips/YaRrr/htests.html)