An introduction to R

Jorge Cimentada and Basilio Moreno

6th of July 2019

Subsetting in R

Alright, so far we have seen vectors, matrices and data frames.

- What is subsetting?
- Is it the same for all objects?

```
x <- sample(1:10)
x
```

```
 [1]  8  3  1  4  6  7  5  9 10  2
```

We have 10 random numbers.

Their positions are:

```
 1  2  3  4  5  6  7  8  9 10
 8  3  1  4  6  7  5  9 10  2
```

Subsetting in R

If `x` is:

```
 [1]  8  3  1  4  6  7  5  9 10  2
```

what is the result of:

```
x[c(1, 3, 8)] #Watch out for square brackets.

x[c(-1, -5)]

x[seq(1, 8, 2)]

x[NA]

x[]
```

Write it down without running it!

Subsetting in R

Do these subsetting rules apply the same for all types of vectors?

```
char <- letters[1:10]
lgl <- c(TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE)
gender <- factor(sample(c("female", "male"), 10, replace = T))
```

What about these ones?

```
char[c(1, 1, 1)]
lgl[c(TRUE, 5, 1)]
gender[c(1:3, TRUE)]
```

Super test:

```
super_vector <- c(char, gender, lgl)
super_vector[c(1, 11, 27)]
```

Subsetting in R

Subsetting rules are the same for all types of vectors.

Exceptions are:

- matrices
- data frames
- lists

Let's go through each one…

Subsetting in R

If you remember correctly, matrices are a vector with rows and columns.

```
x_matrix <- matrix(1:10, 5, 2) # 5 rows and 2 columns
x_matrix
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

Building on the previous examples, what wouldl be the result of this?

```
x_matrix[c(1, 4, 6)]
```

To confuse you even more, what do you think would be the result of this?

```
x_matrix[2:3, ]
```

Subsetting in R

A matrix can be thought of as two things:

- A numeric vector:

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

- Or a numeric vector with rows and columns

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

- Both things come from the same thing and can be subsetted differently!

Subsetting in R

Now that you know… What are the results of:

```
x_matrix[1:5, 2]

x_matrix[, 2]

x_matrix[1, 1]

x_matrix[1:10, 2]

x_matrix[, 1:2]
```

Subsetting in R

Now, data frame are very similar to matrices.

```
our_df <- data.frame(letters = letters[1:10], age = sample(25:50, 10),
                     lgl = sample(c(TRUE, FALSE), 10, replace = T))
our_df
```

```
   letters age   lgl
1        a  25 FALSE
2        b  27 FALSE
3        c  34 FALSE
4        d  39 FALSE
5        e  40  TRUE
6        f  45 FALSE
7        g  35 FALSE
8        h  43  TRUE
9        i  28 FALSE
10       j  48  TRUE
```

- But if we remember correctly we can have different variables in a data frame.
- Data frames are like the combination of lists and matrices.
- How do we subset these?

Subsetting in R

The same way matrices are subsetted!

```
# First 3 rows for all columns
our_df[1:3, ]

# Only the first and 8th row for first two columns
our_df[c(1, 8), 1:2]

# The 5th column three times for the third column
our_df[c(5, 5, 5), 3]
```

What? Why is the last one a vector?

Subsetting in R

So far we saw how to subset the same way we subset matrices.

- Data frames are lists, remember?
- They also have similar subsetting rules to lists.

```
# We lose the data frame dimensions using this method.
our_df[["age"]]

# We get a data frame with this one.
our_df["age"]

# We don't get a data frame here.
our_df$age
```

Subsetting in R

Following the 'list' subsetting rules for data frames:

- Give me the positions of the 3rd, 4th and 9th element of the age variable.
- It should be a numeric vector.
- It should have no dimensions.

The result should be:

```
[1] 34 39 28
```

Subsetting in R

Well, now that we're at it… How does it work for lists?

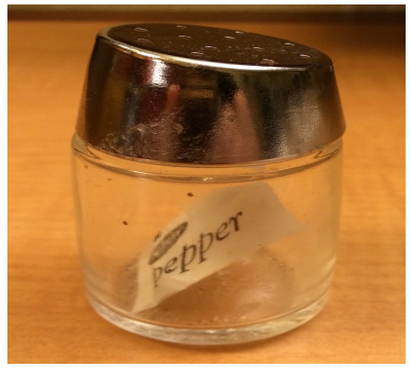

```
our_list <- list(data = our_df, x_matrix, gnd = gender)
```

Explanation

Subsetting in R


```
ourlist
```

Subsetting in R

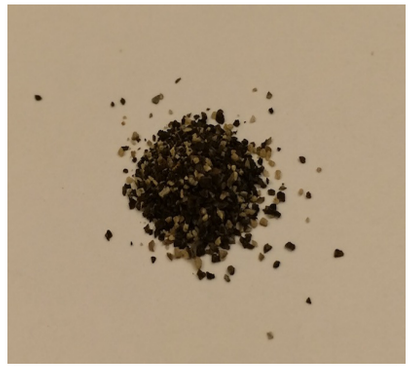



```
ourlist[1]
```

Subsetting in R


```
ourlist[[1]]
```

Subsetting in R



```
ourlist[[1]][[1]]
```

How do we create variables inside data frames or matrices?

Subsetting in R

What does this return?

```
our_df[["our_variable"]]

our_df["our_variable"]

our_df$our_variable
```

- Nothing!
- We're subsetting a variable that doesn't exist
- What is missing to create this variable?

Subsetting in R

Three ways of creating a variable:

```
our_df[["our_variable"]] <- 1:10

our_df["our_variable"] <- 11:20

our_df$our_variable <- seq(1, 20, 2)
```

There's one other way of doing it… Think hard about `[]` and the `,` to divide rows and columns

```
our_df[, "our_variable"] <- "this repeats until end"
```

Add two variables to the `our_df` data frame from any of the options above.

- A logical vector the states `TRUE` for when age is above or equal to 35.
- An addition of `our_df$age` and `our_df$lgl`.

Call them whatever you want.

```
our_df$lgl_two <- our_df$age >= 35
our_df$add <- our_df$age + our_df$lgl
```

Subsetting in R

When whe subset we almost always don't subset like we've been doing.

- We never choose rows 1, 2 and 7, for example.
- Instead, we want things like where gender equals 'Male'.
- Or for people over ages 40.

You have all the tools to achieve this, can you tell me how to do this?

Subsetting in R

Ok, we only want people with ages below 40 years old.

- First, we need a logical statement.

```
age < 40
```

Everything set!

Subsetting in R

- But `age` is not a variable out there in our environment!
- We have to call variables inside data frame as their first names

```
our_df$age < 40
```

```
 [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE
```

- Only positions `c(2, 4, 7, 8, 10)` comply with the logical statement.
- We could try only subsetting these numbers.

Subsetting in R

```
our_df[c(2, 4, 7, 8, 10), ]
```

```
   letters age   lgl           our_variable lgl_two add
2        b  27 FALSE this repeats until end   FALSE  27
4        d  39 FALSE this repeats until end    TRUE  39
7        g  35 FALSE this repeats until end    TRUE  35
8        h  43  TRUE this repeats until end    TRUE  44
10       j  48  TRUE this repeats until end    TRUE  49
```

- However, this is too problematic. What if we had 2,000 rows?

```
our_df[our_df$age < 40, ]
```

```
  letters age   lgl           our_variable lgl_two add
1       a  25 FALSE this repeats until end   FALSE  25
2       b  27 FALSE this repeats until end   FALSE  27
3       c  34 FALSE this repeats until end   FALSE  34
4       d  39 FALSE this repeats until end    TRUE  39
7       g  35 FALSE this repeats until end    TRUE  35
9       i  28 FALSE this repeats until end   FALSE  28
```

- Much better!

Subsetting in R

We can subset pretty much anything with logical vectors.

```
gender[gender == "female"]
lgl[lgl == TRUE]
```

Always think about the details!

```
gender == "female" # is a logical statement
```

```
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
```

We could've written:

```
gender[c(FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE, TRUE, TRUE, FALSE)]
```

```
[1] male   male   male   female male   female
Levels: female male
```

But that's too long.

Functions in R

Let's move on to functions.

What are functions?

- Objects
- Commands
- Black boxes

All at the same time!

Functions in R

For example, take the `sd` function (standard deviation).

```
class(x)
```

```
[1] "integer"
```

```
class(sd)
```

```
[1] "function"
```

- They're both of different classes
- What happens if you print them?

Functions in R

```
x
```

```
 [1]  8  3  1  4  6  7  5  9 10  2
```

```
sd
```

```
function (x, na.rm = FALSE)
sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
    na.rm = na.rm))
<bytecode: 0x0000000013de4e78>
<environment: namespace:stats>
```

- For the vector we get its contents
- For the function we get it's source code

Functions in R

- Functions are commands that accept something and return something

```
sd(x)
```

returns the standard deviation of a variable

When you have questions about a function type `?`

`function_name`

Functions in R

```
x <- rnorm(100)
y <- x + rnorm(100, mean = 1, sd = 1)
```

- Check what `?rnorm` does.
- Use `?cor` to calculate the correlation between x and y
- Set the `method` argument to be “spearman”

```
cor(x, y, method = "spearman")
```

```
[1] 0.7328173
```

To be continued….