Para limpiar R console: Ctrl + L

Para ver el directorio de trabajo: getwd ()

Para cambiar directorio de trabajo: setwd (“working\_directory”) o ir a Session en RStudio

Para ver toda la lista de variables en el global enviroment: ls()

Para eliminar una variable del workspace: rm (variable)

Para limpiar todas las variables de la memoria de R: rm (list = ls ())

Para ver qué tipo de variable es: class (variable)

Para asignar una secuencia a una variable: x <- seq (1, 35, by = 2)

Para leer una tabla con espacios en blanco: read.table (“table\_1.txt”, header = TRUE, sep = “\t”)

Para rellenar una tabla (evitar el error de distinto número de columnas): read.table (“table\_1.txt”, fill = TRUE)

Cómo actualizar R: install package “installr” 🡪 load package installr 🡪 Type in the console updateR ()

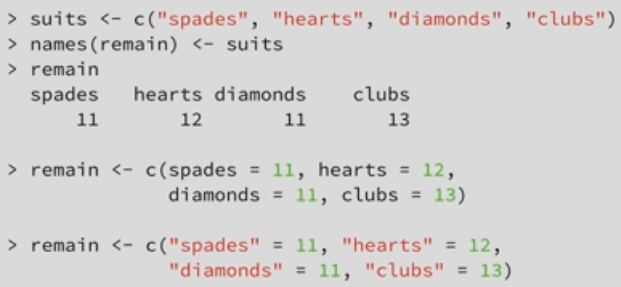
~ = “C:\Users\JET\Documents”

\*NA = Not Available

is.\* () is used to see whether variables are of a certain type

as.\*() is used to transform the type of a variable to another type

(ATOMIC) VECTORS



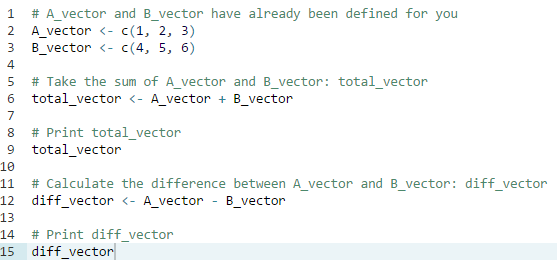
SINGLE VALUE = VECTOR

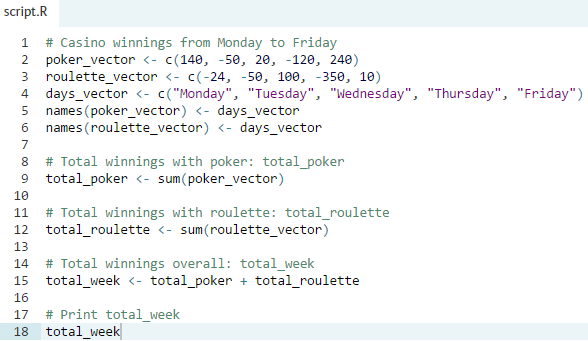
length (vector) para saber la longitud del vector

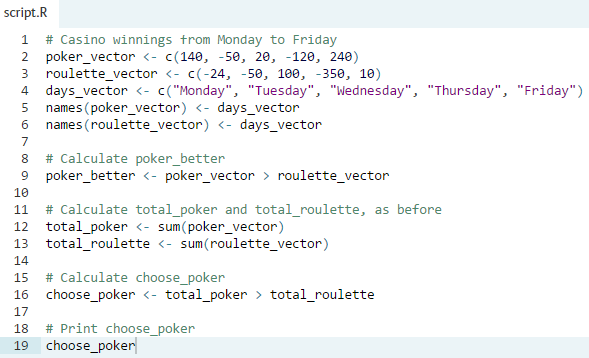
R performs automatic coercion for atomic vectors

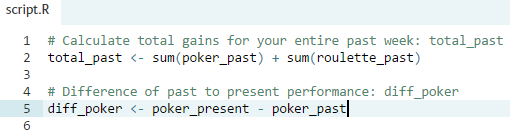
sum (vector) para calcular la suma de los elementos del vector

To calculate the sum of 2 vectors of equal length, R takes the sum of each element of the vectors and returns a new vector of the same length.



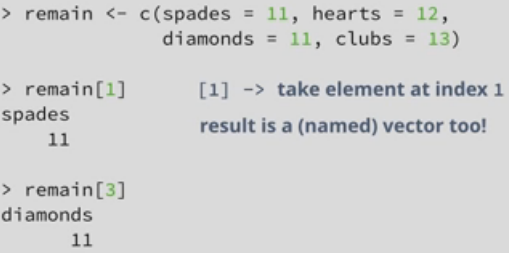




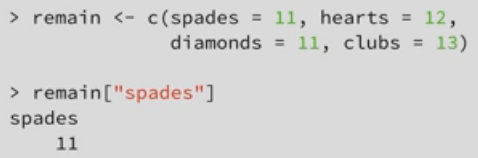


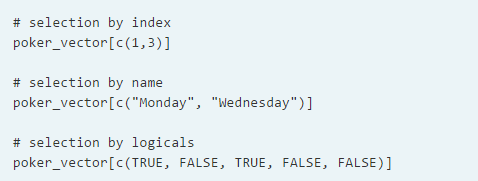
Subsetting

-By index

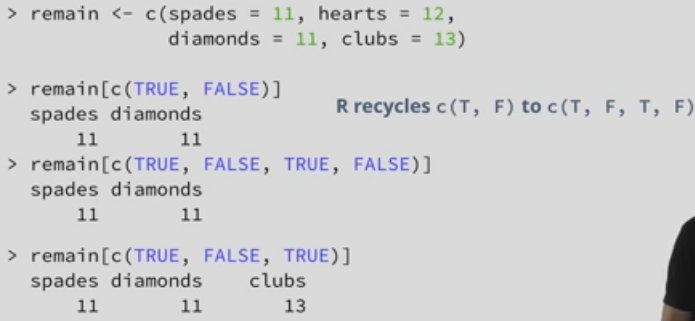


-By name

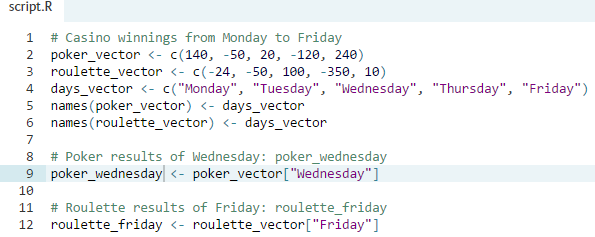


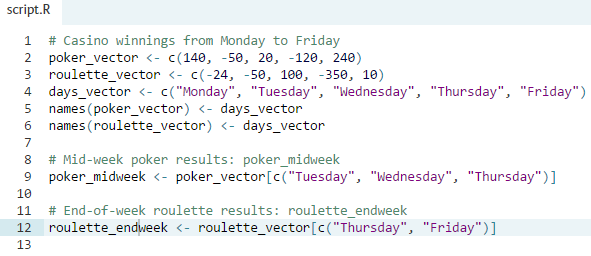


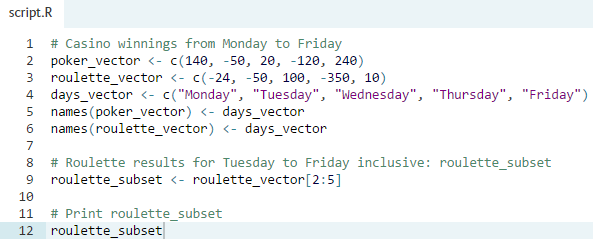
Recycling: R is smart enough to see that the vector of logicals you passed it is shorter than the `remain` vector, so it repeats the contents of the vector until it has the same length as `remain`. This means that, behind the scenes, this line of code is executed, giving the result that we've observed before. Even if you use a vector of length 3 to do the selection, the vector is recycled to end up with a vector of length 4, thus appending the first element again.

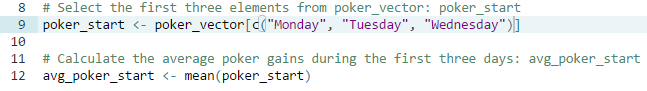


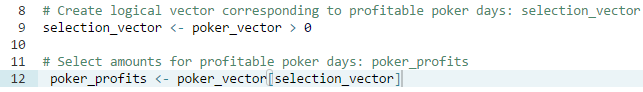
Exercises

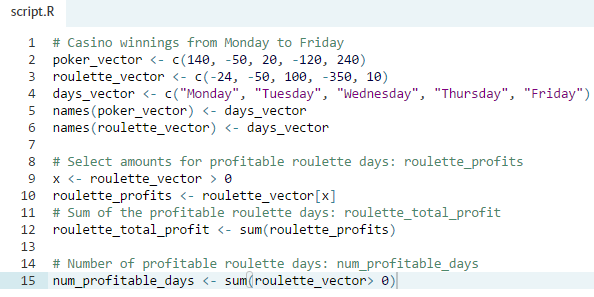


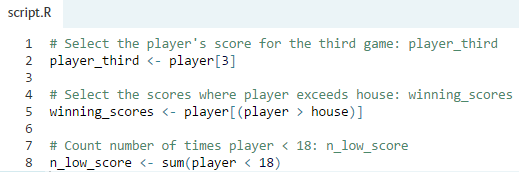




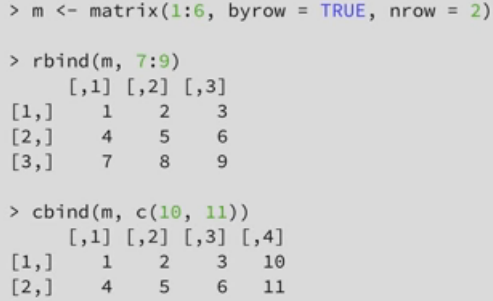




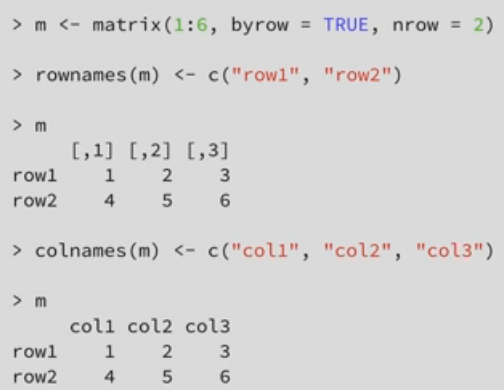




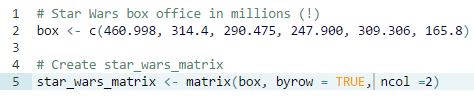
MATRICES



Naming a matrix: rownames (), colnames ()

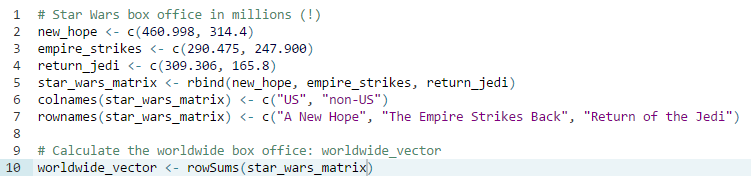


Exercises



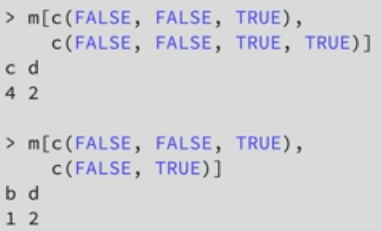
In R, the function rowSums() conveniently calculates the totals for each row of a matrix. This function creates a new vector:

sum\_of\_rows\_vector <- rowSums(my\_matrix)

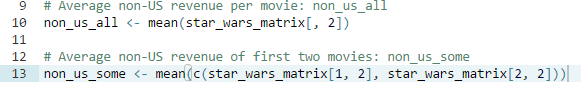




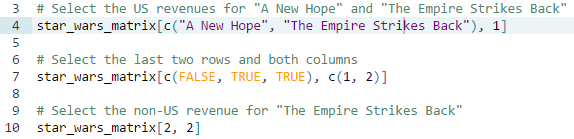


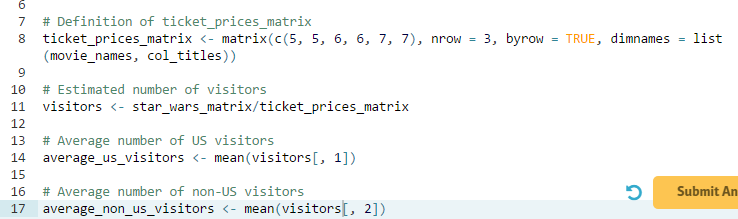


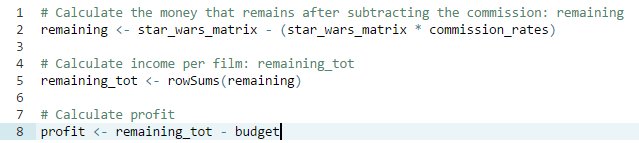
Exercises





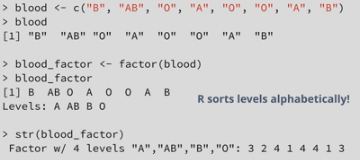






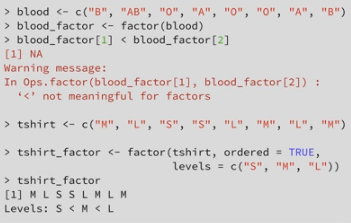
FACTORS

R basically does two things when you call the factor function on a character vector: first of all, it scans through the vector to see the different categories that are in there. In this case, that's "A", "AB", "B" and "O". Notice here that R sorts the levels alphabetically. Next, it converts the character vector, blood in this example, to a vector of integer values. These integers correspond to a set of character values to use when the factor is displayed. Inspecting the structure reveals this: We're dealing with a factor with 4 levels. The "A"'s are encoded as 1, because it's the first level, "AB" is encoded as 2, "B" as 3 and "O" as 4.

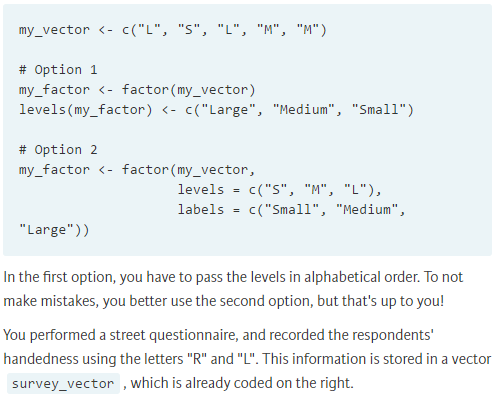


Factors are actually integer vectors, where each integer corresponds to a category, or a level.

Normal vs Ordinal

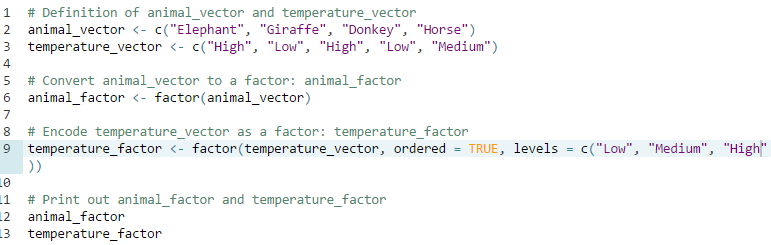


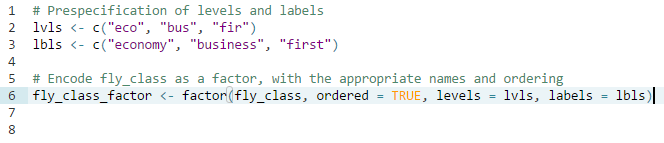
You can change the factor levels using the levels function or by using the labels argument inside the factor function.



R allows you to make the difference between ordered and unordered factors, thus catering to both nominal and ordinal variables.

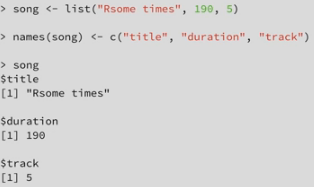
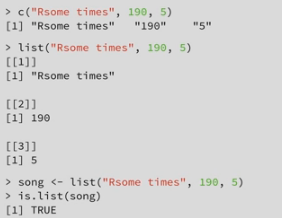
Exercises

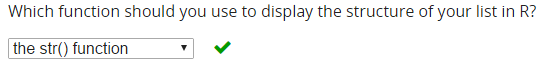




LISTS

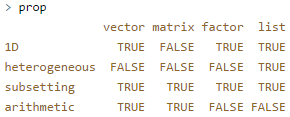
A list can contain all kinds of R objects, such as vectors and matrices, but also other R objects, such as dates, data frames, factors and many more. All of this can be stored in a single list without R having to perform coercion to enforce the same type.

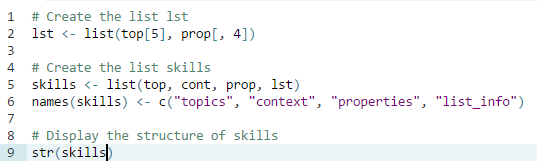




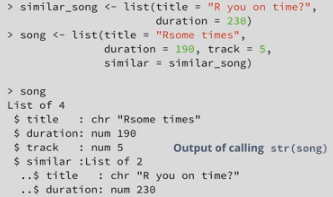
Exercises







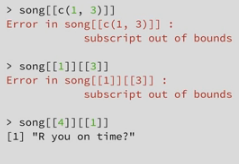
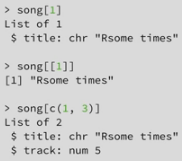
Subset and extend lists



* [ vs. [[

If you use single brackets on lists, you subset the list, but also a list gets returned. If you want to select the actual title from `song`, so the character string R sometimes, you'll need double square brackets instead of single ones: That looks more like it! The difference between single brackets and double brackets is not big on your keyboard, but it sure is important from an R perspective.

Just remember this: subsetting lists using single brackets results in lists, while you can only access a single element using double brackets. This difference doesn't limit you from supercharging your subsetting operations of course.



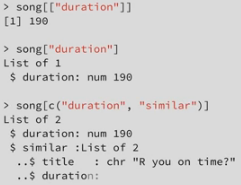
Double brackets are only to select single elements from a list.

This command means: take the first element from the song list, and from that list, take the third element. But the first element from song is simply a character vector of length 1, so there's no way of selecting the third element from it.

* Subset by names

To select the second element from song, for example, you can just as well use the string "duration" inside double brackets.

This subsetting by names of course also works with single brackets, both to build a one-element list as well as to select multiple elements.

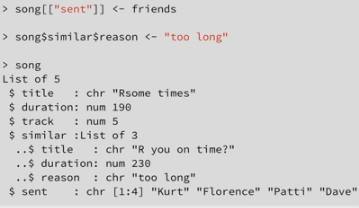
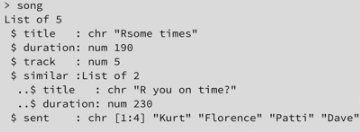


* $ sign to select an element from a list

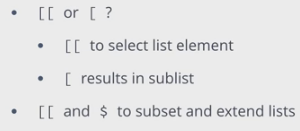
It works just the same as the double brackets but only works on named lists.



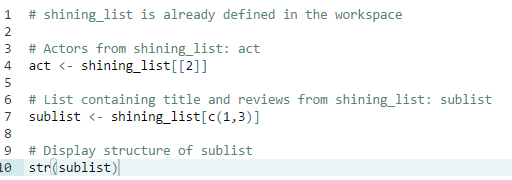
* Extend lists

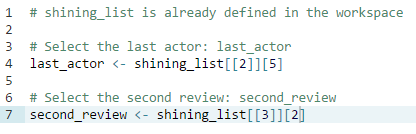


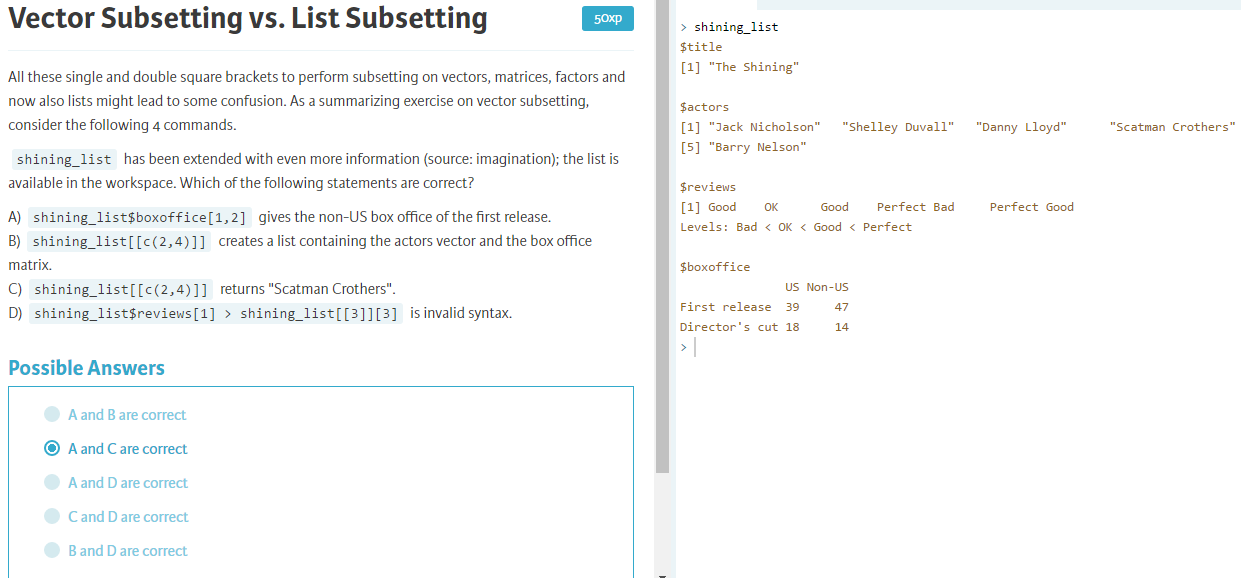
Summary

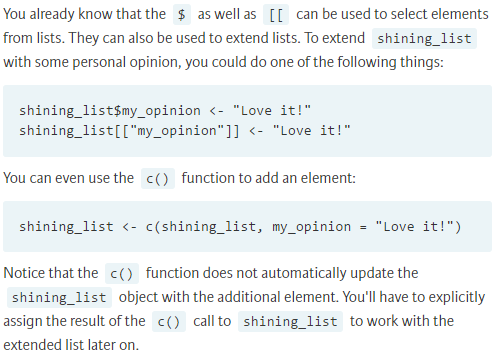


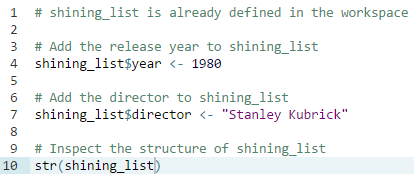
Exercises

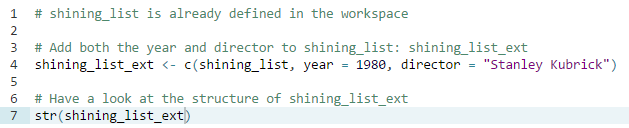


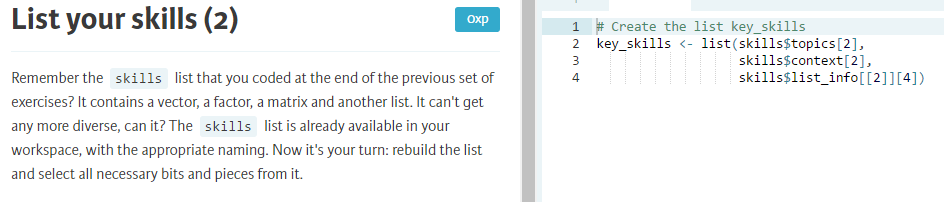








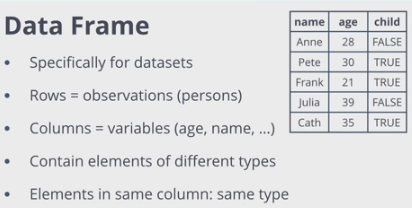




DATA FRAMES

A data frame is the fundamental data structure to store typical data sets.

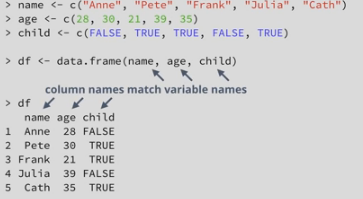
It's pretty similar to a matrix, because it also has rows and columns. Also for data frames, the rows correspond to the observations, the persons in our example, while the columns correspond to the variables, or the properties of each of these persons. The big difference with matrices is that a data frame can contain elements of different types. One column can contain characters, another one numerics and yet another one logicals. That's exactly what we need to store our persons' information in the dataset.



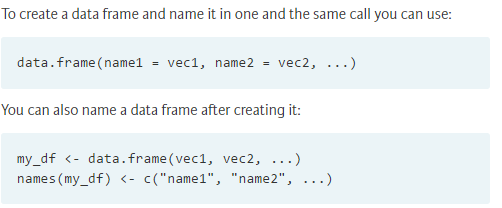
In most cases, you don't create a data frame yourself. Instead, you typically import data from another source. This could be a csv file, a relational database (SQL), but also come from other software packages like Excel or SPSS.

Otherwise:

data.frame () function.

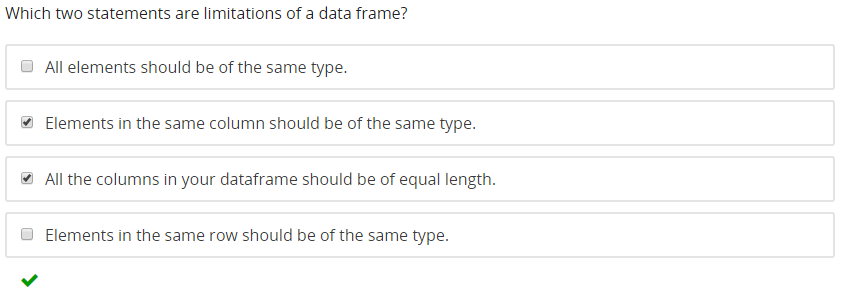


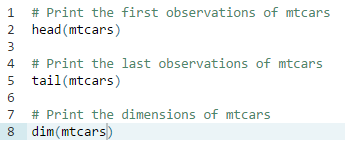
To name the data frame:



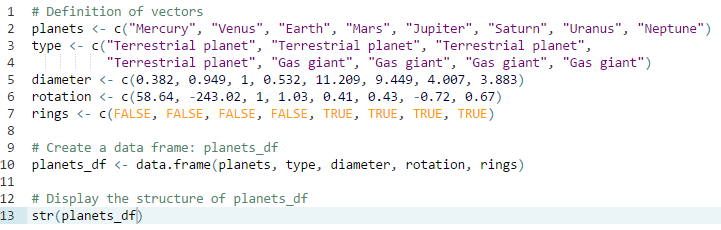
IN THE END, DATA FRAME ARE ALSO LISTS

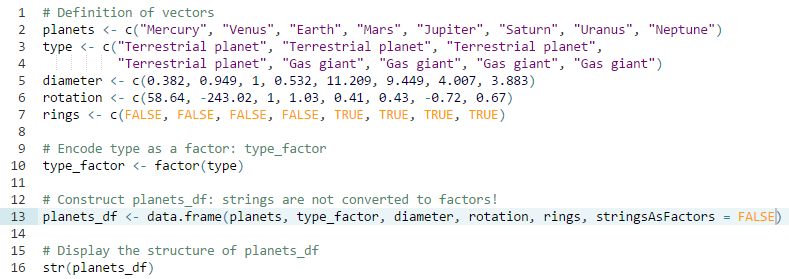
Exercises

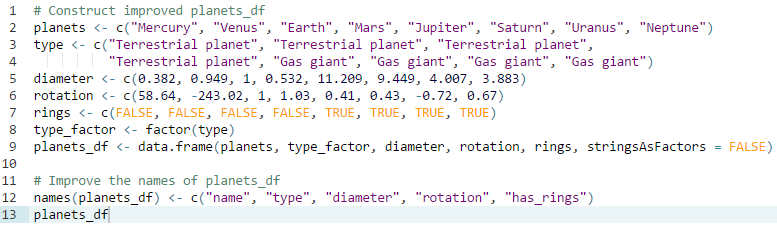


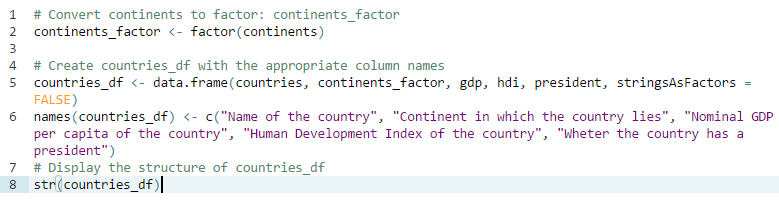






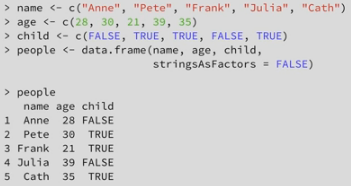


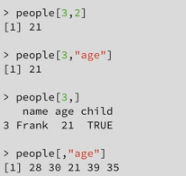




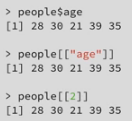
Subset, extend, and sort data frame

Essentially the same as in matrices. The only difference occurs when you specify only one index inside `people`. In the matrix case, R would go through each column from left to right to find the index you specified. In the data frame case, you simply end up with a new data frame, that only contains the column you specified.

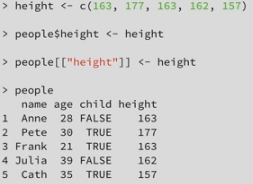
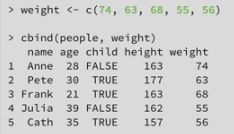




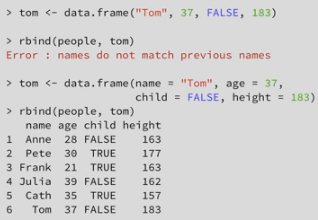
You can also use the list subsetting:



Extend Data Frame

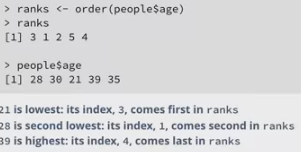
 

You can use `rbind()` to add new rows to your observations. Suppose you want to add the information of another person, Tom, to the data frame. Simply creating a vector with the name, age, height etc, won't work, because a vector can't contain elements of different types. You'll have to create a new data frame containing only a single observation, and add that to the data frame using rbind.



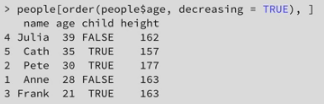
Sorting

We'll need the `order()` function to help us out. Let's test the order function on the age column: Order returns a vector `ranks` with the ranked position of each element. To see what this means, let's print the age column and try to reproduce the result.

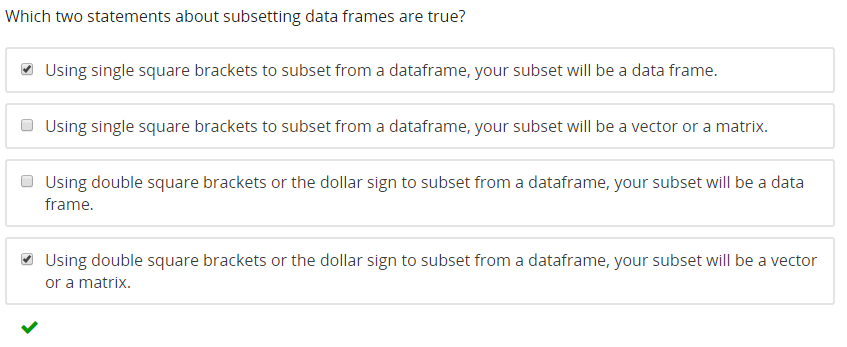


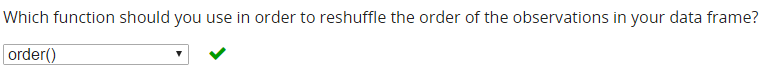
The ranks vector contains indices that are now used to perform a selection in a scrambled order, such that the result is an ordered data frame.

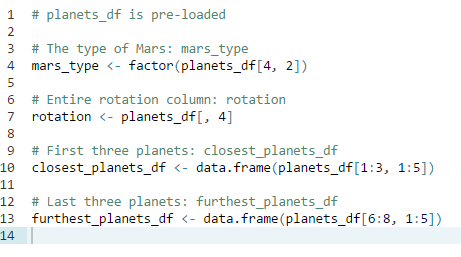
To sort in descending order, you can set the decreasing argument to TRUE

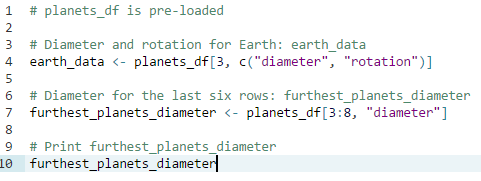


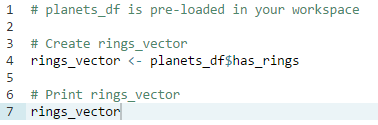
Exercises

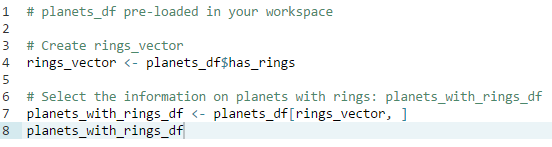


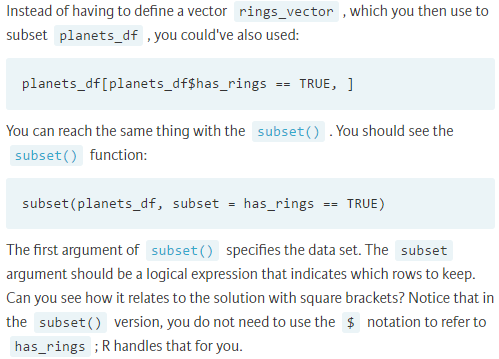


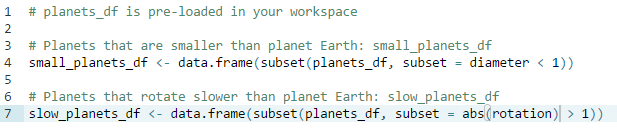


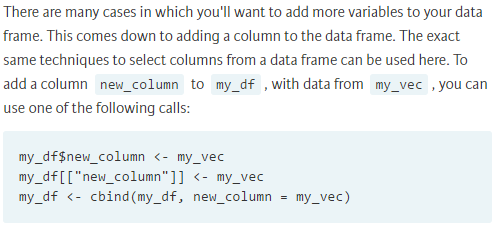


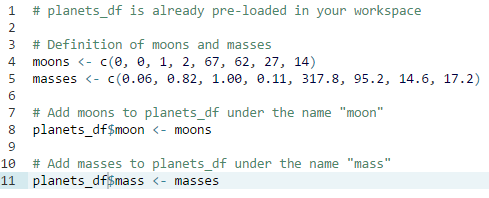


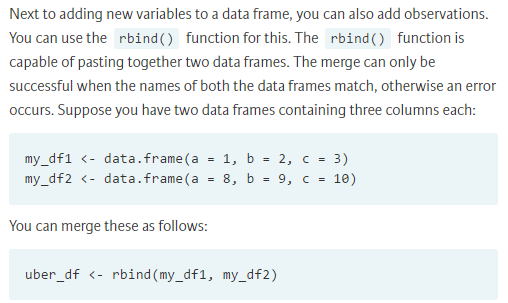


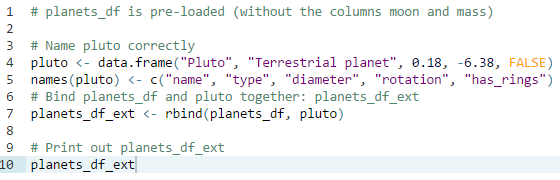


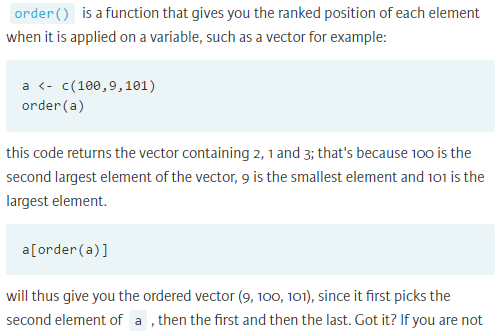


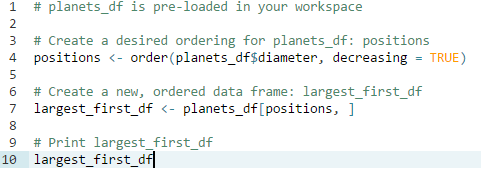




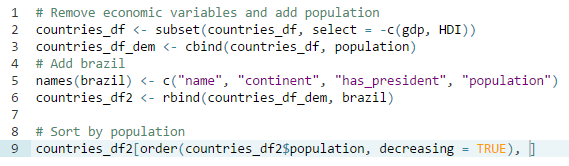








**the variable names would NOT be specified in quotes when using subset() function**



GRAPHICS

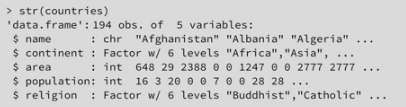
-Create plots with code

-Replication and modification easy

-Reproducibility

-*graphics* package: plot (), hist(), barplot (), boxplot (), pairs ()…

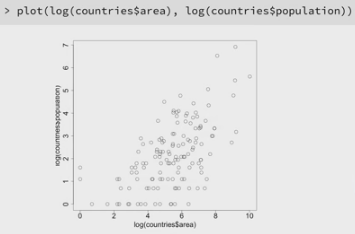
-*ggplot2, ggvis, lattice*



To create basic visualizations in R, a great place to start is the plot() function. This function is very generic, as the graphs it creates are different depending on the type of input information.

If you try to plot factors, for example, plot() will automagically create a bar chart that nicely displays the number of occurences of each factor.

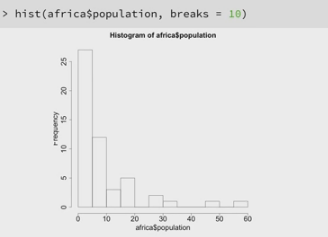
You can also pass plot() vectors; the data corresponding to the first argument is then shown on the horizontal x-axis, while the second vector will be plotted on the y-axis.



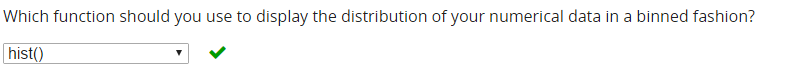
Basically, the hist() function visualizes the distribution of your data by placing all values in bins and displaying the bin frequencies of each bin. Have a look at the usage of the hist() function:

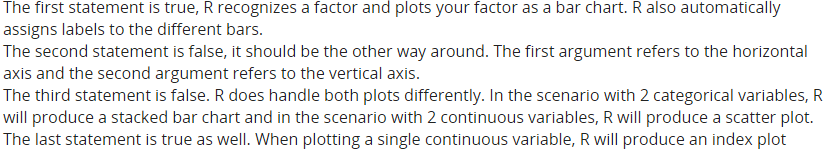
hist (x, breaks = "Sturges")

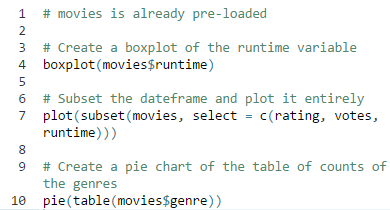
x is the vector of values for which you want to create a histogram. The breaks argument specifies how many bins, or histogram cells, your plot should contain. By default, the "Sturges" algorithm is used to automatically determine the number of bins.

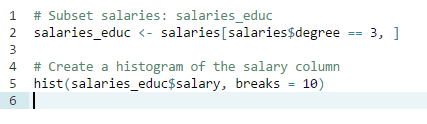


Exercises

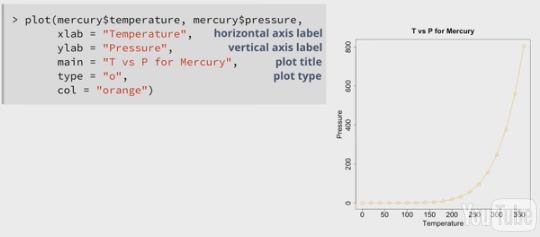








Customizing plots

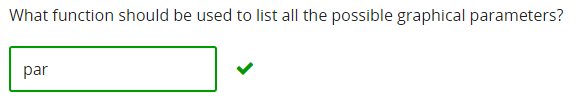


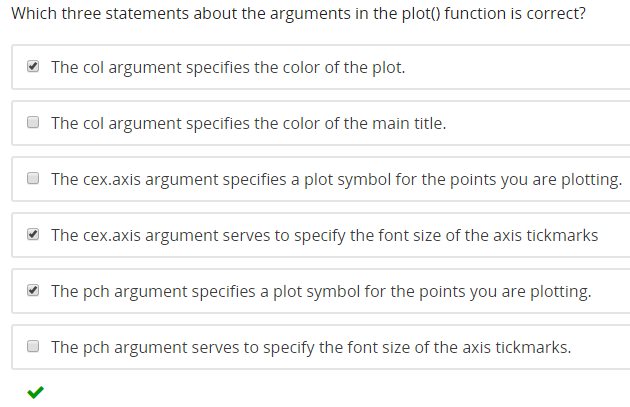
If you only want a line, you can type = "l"

To add a subtitle to your plot, use *sub*

The size of the axis ticks' font size is 150% of the overall font size

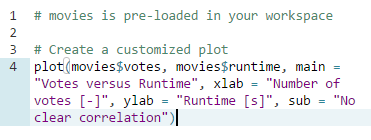
?par 🡪 Information about all of the parameters that you can specify. Parameters specified with `par()` are maintained for different plotting operations.





Exercises

* *plot*



By default, R prints rather dark colors: scatter plots have dots, axis, labels and titles that are all black. Histograms have a grayish fill if you do not manually specify the options.

If you want to specify colors, you can choose simple strings such as "red" and "blue", but also you can also use indices and hexadecimal codes. For example, col = 1, col = "white", and col = "#FFFFFF" are all equivalent.

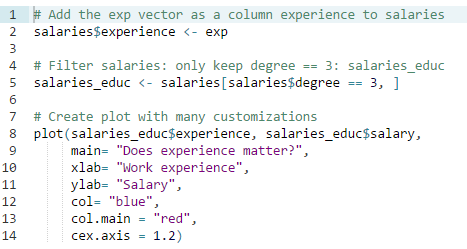
With cex.axis for example, you can specify how much the font of the axis ticks should be made smaller or larger with respect to the general font size. You can do similar things with cex.main and cex.lab.

* *hist*

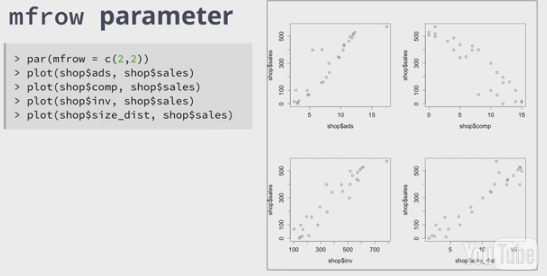
Limits of the x-axis to c(90, 220): *xlim*

The fill of the bars is "cyan" (*col* argument)

The color of the bars' borders is "red" (*border* argument)



Multiple plots



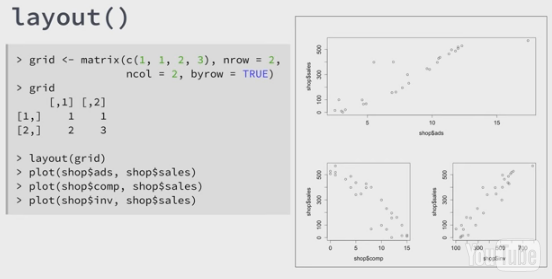
The graphs are added in a row-wise fashion.

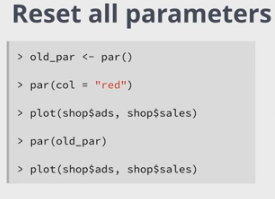
You can use the par() function to specify the mfrow argument. If you for example execute:

par(mfrow = c(1,2))

and next build two plots, these will appear next to each other in the same figure. c(1,2) corresponds to a grid of 1 row and 2 columns.

With the layout() function, you can define more complex plot arrangements. In its most minimal use, layout() simply requires a matrix, in which you specify the location of the figures on the output device.







Functions such as abline(), text(), lines(), segments() add graphical elements to the current figure, instead of creating a new figure. These are great to add more information to your plots.

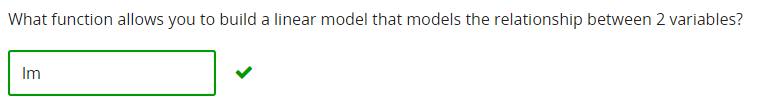
Only for the `lines()` function it's important that you pass the data points in the right order. It is important to use “~” instead “,” in the argument of the lm function.

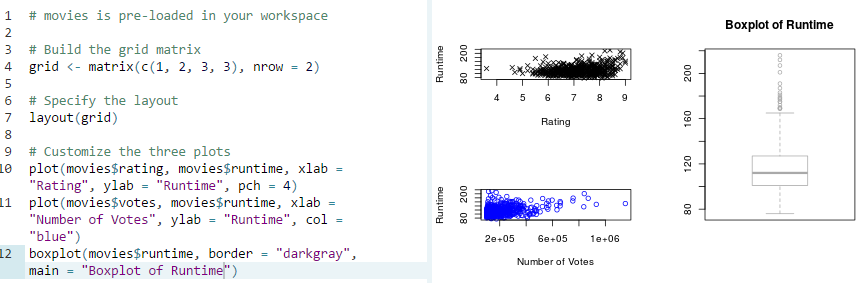
Once you add something to your plot, such as the mess of lines going from left to right, you cannot remove it anymore. Plotting in R works like painting things on a canvas; you can't take it off again. If you want to remove elements, you'll have to start over with a plot generating function like `plot()` and generally add more layers to it with the functions I've just introduced.

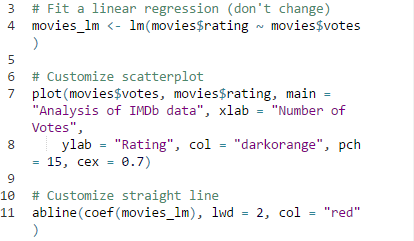
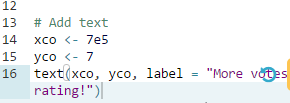
-lm () 🡪 fit a linear regression

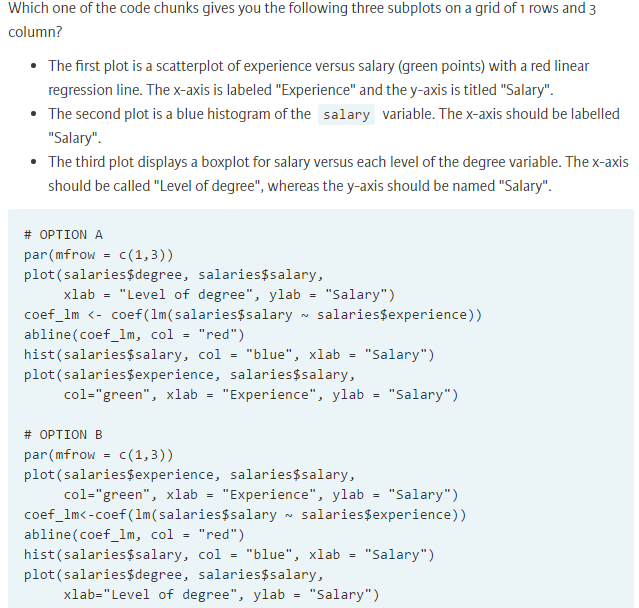
-abline () 🡪 add a atraight line to the plot. Yo have to pass the coeficients of the linear regression to it, usin coef ().

Exercises







Los tipos de punto en R se especifican con un vector de dos puntos. Por ejemplo, c (1, 2) son triángulos.