R as a Calculating Environment

# Basic Operators in R

We will first start by writing codes in an R code chunk (how to insert a code chunk will be covered later in the lecture). Note how the code chunk is shaded grey. Only information that you write in the code chunk is registered as codes. Anything written outside the code chunk, A.K.A the R Markdown chunk, will be registered as non-codes by RStudio.

Note also how the we have the # before the blue text “Basic Operators in R”. Having this # symbol in the R Markdown chunk tells R that the text should act as a heading. If you want to create a subheading or a sub-subheading, simply add more # symbols in front of the text - for example, ## heading 2 and ### heading 3.

The same # symbol has a very different meaning when written in an R code chunk. Anything written after # in a code chunk are registered by RStudio as comments and do not run as codes - they are like extra notes that tell you, or someone you share this file to, more information about what the code does.

To learn more about a function, you can place a ? before the function name (e.g. ?mean) or search for it directly in the help tab in the bottom right window. This built-in function of R will give you information regarding any code function such as which package it belongs to, its arguments, and examples on how to use it. References and related functions are also provided for your own independent exploration.

As you may have recalled from our synchronous session, you can create vectors with the operator c(). To save a vector and assign it to a variable, we use <- (the shortcut for this symbol is Alt+-(minus sign) for Windows or Option+-(minus sign) for Mac). R also has other [data types](https://www.statmethods.net/input/datatypes.html) that you can explore on your own time.

?mean  
x <- c(0:10, 50)  
x

## [1] 0 1 2 3 4 5 6 7 8 9 10 50

# getting help in R  
?plot

# Running codes

If you want to run a whole code chunk, simply click on the green play button on the top right corner of the R chunk. If you only want to run a single line of code, you can highlight or place your curser on only that line, then click “run” (on the top right corner of the code editor window) and then “run selected lines”.

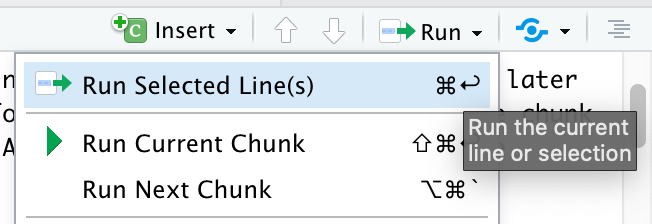


Figure 1. How to Run Selected Lines

A shortcut for running single code lines is Ctrl+Enter for Windows and Cmd+Return/Enter for Mac. A shortcut for running a whole code chunk is Shift+Ctrl+Enter for Windows and Shift+Cmd+Return/Enter for Mac.

As you can see in the code chunk below, we can also store vectors in variables that are actual words instead of just a single letter. It is noteworthy that variable names need to be 1 single, connected word. For example, red\_apple is okay, but red apple is not! Variable names also cannot start with a number, R is picky this way. For instance, six\_apples or apples\_6 are okay, but 6\_apples is not!

The function ls() is especially helpful if you want to check how many and what variables you’ve created. Another way to check this is to go to your Environment on the top right corner of RStudio.

storks <- c(2,6,3,6,2,8)  
storks

## [1] 2 6 3 6 2 8

babies <-c(6,8,5,9,4,9)   
babies

## [1] 6 8 5 9 4 9

ls()

## [1] "babies" "storks" "x"

# Insert a new chunk

To insert a new code chunk, you can go to insert and then click R.

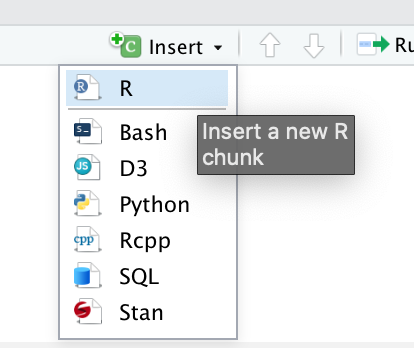


Figure 2. Inserting a Code Chunk

Or you can also type this ({r}) manually. Note that you can only see the following blank code chunk if you’re viewing this file on RStudio.

# Writing and saving codes

Another data type in R is data frame. You can join several vectors together to create a data frame using the function data.frame(). Similarly, you can also save this data frame by storing it in your environment using <-. Also, a data frame is similar to a table.

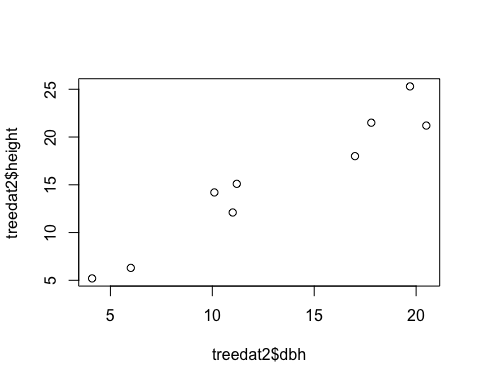
To check all of the names of a data frame’s variables, you can use the function names(). In this case, a variable is equivalent to a column in a data frame. Pro-tip: if you want to change the names of a data frame’s variable/column, you can use the function names() followed by <- and then a vector of the new names in quotation marks "". For example names(treedat2) <- c("DBH", "Height") - try it! (But note that you have to turn the names back to its original form for the rest of the codes of this document to work as **R is case-sensitive**)

For now, you are only expected to know that plot() is the function that graphs your data. plot() needs you to specify the data frame followed by a $ and then the variable name. The only way to avoid typing the data frame name every time is to use attach() - we will go over this function later. There are two ways for you to type your arguments in plot():

* using ~ to separate the x and y variables and
* using , to separate the x and y variables.

We will explore plot() in MUCH more detail in the next module: R II: Graphical Presentation in R.

## [1] "dbh" "height"



### Challenge:

Can you spot any differences between the 2 graphs? How can you change the code of either one of them to make the two match?

# R Important Rules

In this section, we will go over some important rules in R and how to troubleshoot common mistakes.

* Again, it is important to note that **R is case-sensitive**. In other words, Dbh is VERY different from dbh!!
* There will be times when you need to copy and paste a folder pathway (e.g. when importing a file into R or when setting a new working directory using setwd()). If you are using a Mac, R uses / instead of , so make sure to change all of your  to / when copying a folder pathway (we will talk more about this later).
* If you are using Windows, R uses \ instead of , so make sure to change all of your  to \ when copying a folder pathway (we will talk more about this later).
* Everything following # is a comment that does not code for anything.

# Saving the Workspace

You can choose to save the workspace so that RStudio remembers everything that you ran in an earlier session.

# Cleaning up

For the sake of organization, you can choose to clear the workspace - this is especially helpful when your environment is swarmed with unecessary/intermediary variables and dataframes that you don’t need. On the top left corner of your environment, there is a broom that you can click to clear your environment.

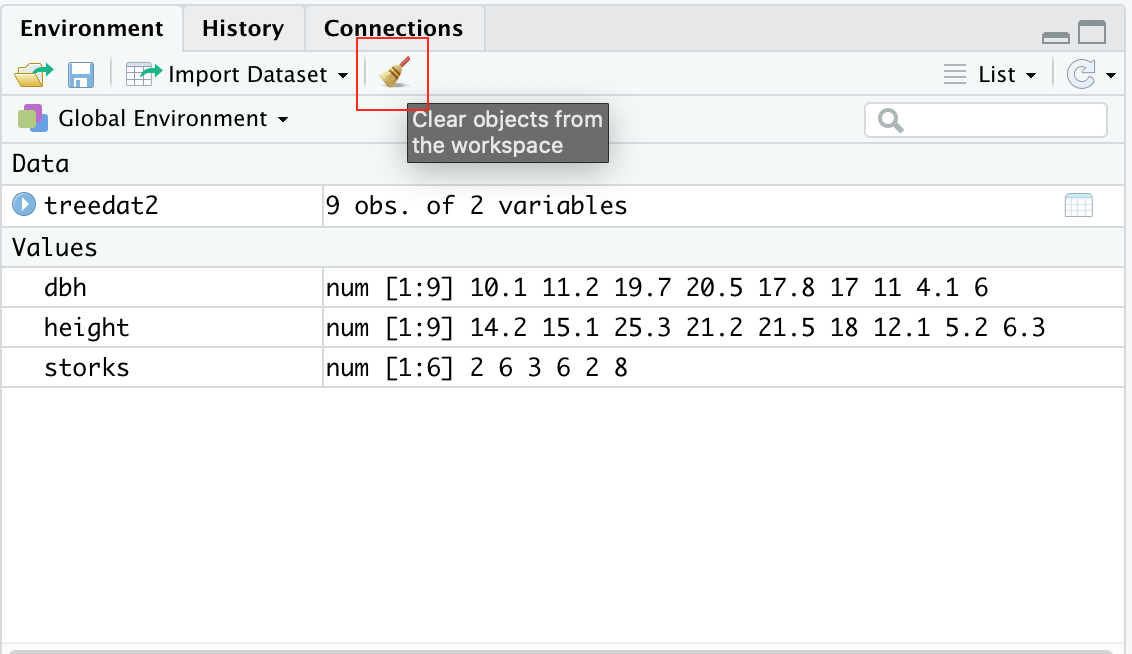


Figure 3. Clearing Your Environment

# Types of Objects

As mentioned earlier, R contains many types of data or types of objects. We have previously covered what a vector is. Once you’ve created and stored a vector, you can also check its length using length() and its class using class(). A length of an object is simply how many variables/components it contains. For example c(1, 2, 3)’s length is three and 1’s length is one. The class of an object tells us the type of information an object contains. For example, the class of c(1, 2, 3) is numeric whereas the class of c("male", "female", "other") is character.

A factor is used to categorize data and can store both strings and integers. You can convert a vector into a factor by using as.factor().

A matrix is a two-dimensional collection of similar data type. You can use cbind() (column bind) or rbind() (row bind) to bind vectors together into a matrix. A data frame is a more complex table that can contain different types of data. It can be difficult at first to distinguish between a matrix and a dataframe. You can visit this [site](https://www.geeksforgeeks.org/matrix-vs-dataframe-in-r/) to learn more about their differences.

You don’t have to know too much about modelling. The main takeaway is that the function lm() is used to fit a [linear model](https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/lm). In other words, the code lm(v2~v3) tells R that we want to predict the values of v2 knowing v3 values using a linear model. After creating a model, you can check its structure by using the function str(). This function will lets you know all of the parameters and outputs of your model.

The very last function that we will cover in this .Rmd file is summary(). This is an extremely powerful function that tells you all of the basic statistics of your object. This includes mean, median, min, max, quartiles, and identify any special values such as NA values.

#Examples:  
 # Vectors:  
v1 <- 1  
v2 <- c(1,7.5,2.0)  
v3 <- c(1,2,3)  
v4 <- c(11,NA,13) # one value is NA (missing)  
length(v1)

## [1] 1

length(v2)

## [1] 3

class(v3)

## [1] "numeric"

class(v4)

## [1] "numeric"

#Factors:  
f1 <- "spaghetti" # called a string-in quotes  
f2 <- c("male", "female")  
class(f2)

## [1] "character"

#Change from vector to factor:  
f3 <- as.factor(v3) # changes v3 from a vector to a factor  
class(f3)

## [1] "factor"

#Matrix:  
m1 <- rbind(v2,v3,v4)  
dim(m1)

## [1] 3 3

class(m1)

## [1] "matrix"

#Dataframe:  
df1 <- data.frame(m1,f3) # changes the info to a dataframe  
class(df1)

## [1] "data.frame"

#Model:  
model1 <- lm(v2~v3)   
str(model1) # tells you what is in model1 - lots of stuff

## List of 12  
## $ coefficients : Named num [1:2] 2.5 0.5  
## ..- attr(\*, "names")= chr [1:2] "(Intercept)" "v3"  
## $ residuals : Named num [1:3] -2 4 -2  
## ..- attr(\*, "names")= chr [1:3] "1" "2" "3"  
## $ effects : Named num [1:3] -6.062 -0.707 -4.899  
## ..- attr(\*, "names")= chr [1:3] "(Intercept)" "v3" ""  
## $ rank : int 2  
## $ fitted.values: Named num [1:3] 3 3.5 4  
## ..- attr(\*, "names")= chr [1:3] "1" "2" "3"  
## $ assign : int [1:2] 0 1  
## $ qr :List of 5  
## ..$ qr : num [1:3, 1:2] -1.732 0.577 0.577 -3.464 -1.414 ...  
## .. ..- attr(\*, "dimnames")=List of 2  
## .. .. ..$ : chr [1:3] "1" "2" "3"  
## .. .. ..$ : chr [1:2] "(Intercept)" "v3"  
## .. ..- attr(\*, "assign")= int [1:2] 0 1  
## ..$ qraux: num [1:2] 1.58 1.26  
## ..$ pivot: int [1:2] 1 2  
## ..$ tol : num 1e-07  
## ..$ rank : int 2  
## ..- attr(\*, "class")= chr "qr"  
## $ df.residual : int 1  
## $ xlevels : Named list()  
## $ call : language lm(formula = v2 ~ v3)  
## $ terms :Classes 'terms', 'formula' language v2 ~ v3  
## .. ..- attr(\*, "variables")= language list(v2, v3)  
## .. ..- attr(\*, "factors")= int [1:2, 1] 0 1  
## .. .. ..- attr(\*, "dimnames")=List of 2  
## .. .. .. ..$ : chr [1:2] "v2" "v3"  
## .. .. .. ..$ : chr "v3"  
## .. ..- attr(\*, "term.labels")= chr "v3"  
## .. ..- attr(\*, "order")= int 1  
## .. ..- attr(\*, "intercept")= int 1  
## .. ..- attr(\*, "response")= int 1  
## .. ..- attr(\*, ".Environment")=<environment: R\_GlobalEnv>   
## .. ..- attr(\*, "predvars")= language list(v2, v3)  
## .. ..- attr(\*, "dataClasses")= Named chr [1:2] "numeric" "numeric"  
## .. .. ..- attr(\*, "names")= chr [1:2] "v2" "v3"  
## $ model :'data.frame': 3 obs. of 2 variables:  
## ..$ v2: num [1:3] 1 7.5 2  
## ..$ v3: num [1:3] 1 2 3  
## ..- attr(\*, "terms")=Classes 'terms', 'formula' language v2 ~ v3  
## .. .. ..- attr(\*, "variables")= language list(v2, v3)  
## .. .. ..- attr(\*, "factors")= int [1:2, 1] 0 1  
## .. .. .. ..- attr(\*, "dimnames")=List of 2  
## .. .. .. .. ..$ : chr [1:2] "v2" "v3"  
## .. .. .. .. ..$ : chr "v3"  
## .. .. ..- attr(\*, "term.labels")= chr "v3"  
## .. .. ..- attr(\*, "order")= int 1  
## .. .. ..- attr(\*, "intercept")= int 1  
## .. .. ..- attr(\*, "response")= int 1  
## .. .. ..- attr(\*, ".Environment")=<environment: R\_GlobalEnv>   
## .. .. ..- attr(\*, "predvars")= language list(v2, v3)  
## .. .. ..- attr(\*, "dataClasses")= Named chr [1:2] "numeric" "numeric"  
## .. .. .. ..- attr(\*, "names")= chr [1:2] "v2" "v3"  
## - attr(\*, "class")= chr "lm"

#Summary for basic statistics: means, percentiles for numeric values, and counts for factors:  
summary(v4)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 11.0 11.5 12.0 12.0 12.5 13.0 1

summary(df1)

## X1 X2 X3 f3   
## Min. : 1.000 Min. :2.000 Min. : 2.0 1:1   
## 1st Qu.: 1.000 1st Qu.:3.375 1st Qu.: 2.5 2:1   
## Median : 1.000 Median :4.750 Median : 3.0 3:1   
## Mean : 4.333 Mean :4.750 Mean : 6.0   
## 3rd Qu.: 6.000 3rd Qu.:6.125 3rd Qu.: 8.0   
## Max. :11.000 Max. :7.500 Max. :13.0   
## NA's :1

#More information on the objects:   
str(v4)

## num [1:3] 11 NA 13

str(df1)

## 'data.frame': 3 obs. of 4 variables:  
## $ X1: num 1 1 11  
## $ X2: num 7.5 2 NA  
## $ X3: num 2 3 13  
## $ f3: Factor w/ 3 levels "1","2","3": 1 2 3

# Knitting your .Rmd file

In this course, we will only be knitting our .rmd file to a Word or HTML document - preferably Word. Knitting your .Rmd file makes your codes and workflow shareable because not everyone have RStudio installed on their device!