Data Science with R

A continuous training session

Ogundepo Ezekiel Adebayo

29 March, 2019

Relax, programming is cool!

If you doubt me, please kindly ask Hadley Wickham, the Chief data scientist at R studio.

R for Data Science

Data science is an exciting discipline that allows you to turn raw data into understanding, insight, and knowledge [1]. The goal of R for Data Science is to help you learn the most important tools in R that will allow you to do data science. Data science is a huge field, and there's no way you can master it by reading a single book [1].

What you will learn

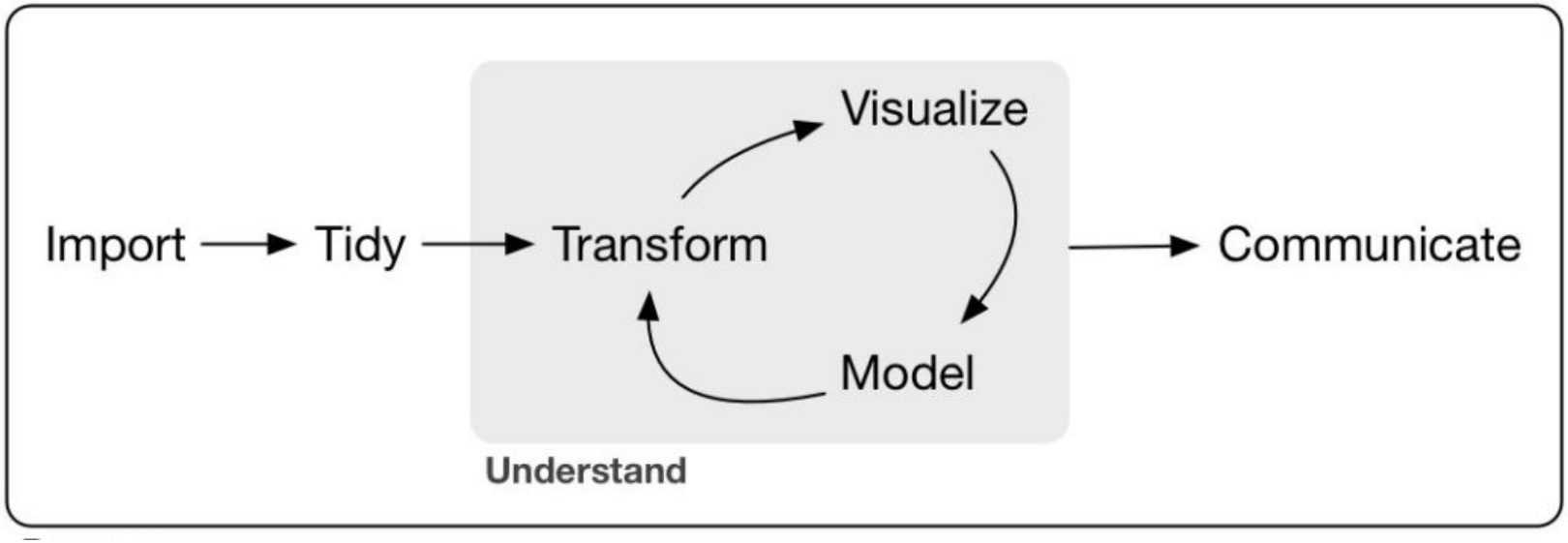

Figure 1: Program ~ Inspired by Hadley Wickham [1]

R and R Studio

R is a statistical programming language for data analysis and visualization while R Studio is an integrated development environment (IDE) for R programming. R Studio makes programming easier in R.

R worth it

Figure 2: R worth it

Introduction to R

In this chapter, you will take your first steps with R. You will learn how to use the console as a calculator and how to assign variables. You will also get to know the basic data types in R. Let's get started!

Welcome to R programming

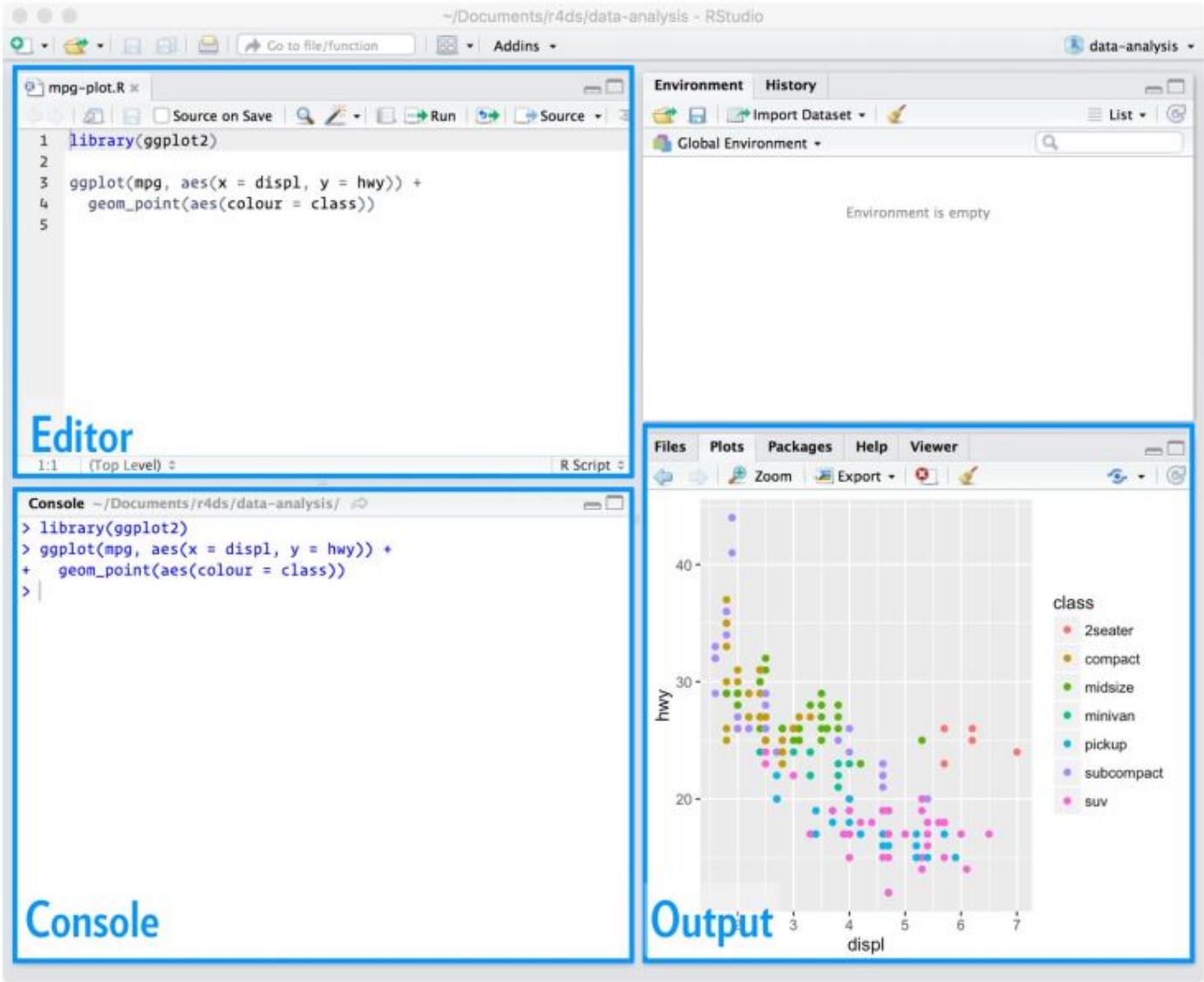

Figure 3: R interface

R as a calculator

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operations:

- Addition :
- Subtraction :
- Multiplication :
- Division :
- Exponentiation:
- Modulo :

Calculate $6 + 12$

```
6+12
```

```
## [1] 18
```

Calculate $800 - 900$

```
800-900
```

R as a calculator

Calculate 4×5

```
4*5
```

```
## [1] 20
```

Calculate $\frac{2018}{2}$

```
2018/2
```

```
## [1] 1009
```

Calculate 2^3

```
2^3
```

```
## [1] 8
```

R as a calculator

Calculate 20%%3

```
20%%3
```

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

Calculate the square root of $\sqrt{4}$

```
sqrt(4)
```

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

Calculate $(\sqrt{4})^2$

```
(sqrt(4))^2
```

```
## [1] 4
```

Comment in R

R makes use of the # sign to add comments, so that you and others can understand what the R code is about. Just like Twitter! Comments are not run as R code, so they will not influence your result. For example, any code like $\#3 + 4$ at the console is a comment. R ignores any code in #, this means that the code will not run.

```
#3+4
```

Variable assignment

A basic concept in **statistical** programming is called a variable. A variable allows you to store a value (e.g. 5) or an object (e.g. a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored with this variable.

Example

Store the value of 4 as your first name

```
ezekiel=4
```

To know what is stored in memory as your first name, type your first name in the console and press return key from the keyboard

```
ezekiel
```

```
## [1] 4
```

Variable assignment and data types in R

```
x=3;y=4;z=10
```

```
x+y
```

```
## [1] 7
```

```
z-x-y
```

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

```
x*y
```

```
## [1] 12
```

```
z^x
```

```
## [1] 1000
```

Naming Rules for Variables

The best naming convention is to choose a variable name that will tell the reader of the program what the variable represents

Rules for naming variables

- All variables must begin with a letter of the alphabet.
- After the initial letter, variable names can also contain (_ or .) and numbers. No spaces or special characters, however are allowed.
- Uppercase characters are different from lowercase characters (in R and also in Python)

Example

	Samples of acceptable variable names	Uncceptable variable names
1	Grade	Grade(Test)
2	GradeOnTest	GradeTest#1
3	Gaudence_Uwimana	Gaudence Uwimana
4	sales_annex_2017	2017sales_annex

Basic classes of objects

R works with numerous **atomic** classes of objects. Some of the most basic atomic data types to get started are:

- Decimas values like 4.7 are called **numeric**
- Natural numbers like 4 are called **integers**. Integers are also numeric
- Boolean values (**TRUE** or **FALSE**) are called **logical**
- Text (or string) values are called **characters**
- Factors : Categorical variable where each level is a category

Basic data structure or types

1. Vector : A collection of elements of the same class
2. Matrix : All columns must uniformly contain only one variable type
3. data.frame : The columns can contain different classes
4. List : Can hold object of different classes and lenght

Create a vector

Vectors are one-dimensional arrays that can hold numeric data, character data, or logical data. In R, you can create a vector with the combine function **c()**. You place the vector elements separate by a comma between the parenthesis.

For example

```
numeric_vector <- c(1, 2, 3, 6, 7, 10)
character.vector <- c('Bosco', 'Lucie', 'John', 'Agness', 'Marc')
```

Notice

Adding a space behind the commas in the **c()** function improves the readability of your code

Naming a vector

As a data analyst, it is important to have a clear view on the data that you are using. Understanding what each element refers to is essential. You can give a name to the elements of a vector with the **names ()** function

Create a vector

Example

```
sales_tax <- c(140000, 200000, 600000, 180000, 170000)
names(sales_tax) <- c('Monday','Tuesday','Wednessday',
                      'Thursday','Friday')
sales_tax
```

```
##      Monday     Tuesday Wednessday    Thursday      Friday
##      140000      200000      600000      180000      170000
```

Arithmetic with vectors

It is important to know that if you sum two vectors in R, it takes the element-wise sum

Example

```
a <- c(1, 2, 3, 4, 5)
b <- c(6, 7, 8, 9, 10)
c <- a + b
```

```
c
```

```
## [1]  7  9 11 13 15
```

Vector selection

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.

To select the first elements of vector **a**, you type **a**[1].

To select the second element of the vector, you typed **a**[2], etc.

Example

```
a
a[1]
a[2]
```

Short group work

What does it do?

a[a>3]

Create special vectors

```
a= 1: 10 # Create sequence 1 to 10
b= 10:1 # Create sequence 10 to 1
```

To create sequence with increament of 2 from 1 to 16, we can **seq()** function e.g.

```
seq(1, 16, 2)
seq(1, 20, 0.1)
seq(20, 1, -0.1)
```

Create special vectors cont.

If you have a sequence value you don't know the last element, say you just know the start of the sequence and the length of the sequence, e.g.

```
seq(5, by=2, length=50)
length(seq(5, by=2, length=50))
```

Repeating elements for certain number of time

```
rep(5, 10) # Repeat 5 in 10 times
rep(1:4, 5) # Repeat 1 to 4 five times
rep(1:4, each=3) # Each element of 1 to 4 3 times
```

Short group work

Tell what the following line of codes is or are doing

```
rep(1:4, each=3, time=2)
rep(1:4, 1:4)
rep(1:4, c(4,1,8,2))
```

Assignment 1

African Institute for Mathematical Sciences (AIMS), Rwanda

R for Data science training

A short assignment

1. Create a vector of a number from 1 to 50
2. If x is a vector of 2,3,4,2,4,5,3,5,3,5,7. Use R to get its length
3. Find the sum, mean and variance of vector x.
4. Plot a beautiful bar chart for vector x
5. x=2, y=3, z = 6,
    a. what is $(x+y)^3$
    b. what is z/y
    c. what is $x \times y$

Figure 4: Assignment 1

Some fun- Using R to print out the Body Mass Index

```
# A function that calculates BMI!

# Press ctrl+shift+Enter to run the script

# Hmmmm R console is now your robot, please follow the prompt

BMI=function(height, weight,name){
  name= readline('Welcome to Data science with R! \n Please, what name can I call you? :')
  height= readline('What is your height in cm ? :')
  height=as.double(height)/100
  weight= as.double(readline('What is your weight in kg ? :'))
  bmi=weight/height^2

  if (bmi< 18.5)
    cat('Dear', name,', your BMI is', bmi,'. Therefore, you are underweight')
  else if (bmi>=18.5 & bmi<=24.9)
    cat('Dear ', name,', your BMI is', bmi,'. Therefore, you have healthy weight')
  else if (bmi>=25 & bmi<=29.9)
    cat('Dear', name,', your BMI is', bmi,'. Therefore, you are overweight')
  else
    cat('Dear', name,', your BMI is', bmi,'.Sorry, do more exercise because of your obesity')
  }

BMI()
```

Figure 5: BMI Function

Matrices

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 we are only working with rows and columns, a matrix is called two dimensional array.

You can construct a matrix in R with the **matrix ()** function.

Example

```
A <- matrix(1:9, nrow = 3, byrow = T); A
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

Matrices

- The first argument is the collection of elements that #Rstats will arrange into the rows and columns of the matrix. Here, we use 1:9 which is a shortcut for c(1, 2, . . . , 9).
- The arguement **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=F**
- The argument **nrow** indicates that the matrix should have 3 **rows**

Short group work

Construct a matrix with 3 rows containing the numbers 1 up to 9 filled **column-wise**

Progressing from vector to matrix

```
fiscal_year2016_17 <- c(140, 134)
fiscal_year2017_18 <- c(160, 158)
performance_analysis <- matrix(c(fiscal_year2016_17,
                                 fiscal_year2017_18),nrow = 2,
                                 ncol = 2,byrow = T)
performance_analysis
```

```
##      [,1] [,2]
## [1,]  140  134
## [2,]  160  158
```

Naming a matrix

To help you understand what is stored in the performance analysis matrix, it is good to add the names of the rows and columns respectively. Not only does this help you to read the data, but it also useful to select certain elements from the matrix.

```
rownames(performance_analysis) <-
  c('Fiscal year July-June 2016/17',
   'Fiscal year July-June 2017/18')
colnames(performance_analysis) <- c('Actual', 'Target')
performance_analysis
```

```
##                                    Actual Target
## Fiscal year July-June 2016/17     140    134
## Fiscal year July-June 2017/18     160    158
```

Other examples

```
A <- matrix(c(1, 3, 5, 7, 9, 11, 13, 15, 17), ncol = 3,
            byrow = F); A
```

```
##      [,1] [,2] [,3]
## [1,]    1    7   13
## [2,]    3    9   15
## [3,]    5   11   17
```

```
B <- matrix(c(2,4,6,8,10,12,14,16,18), ncol = 3,
            byrow = F); B
```

```
##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
## [3,]    6   12   18
```

Matrices selection

To select elements in a matrix we can use square brackets [,], between the square brackets, you indicate the position of the row and column in which the elements to select are.

To select the element in the first row and second column of matrix **A**, you type **A**[1,2].

To select the element in the third row and second column of matrix **A**, you type **A**[3,2], etc.

Example

```
A
A[1,2]
A[3,2]
```

Arithmetic Operation

We can perform all the arithmetic operations on matrices

- Addition

```
C <- A + B; C
```

```
##      [,1] [,2] [,3]
## [1,]    3   15   27
## [2,]    7   19   31
## [3,]   11   23   35
```

- Subtraction

```
D <- B - A; D
```

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

Arithmetic Operation

Multiplication

```
F <- A  %*%  B; F
```

```
##      [,1] [,2] [,3]
## [1,]  108  234  360
## [2,]  132  294  456
## [3,]  156  354  552
```

Arithmetic Operation

- Transpose

$$G = t(A) = \begin{pmatrix} 1 & 7 & 13 \\ 3 & 9 & 15 \\ 5 & 11 & 17 \end{pmatrix}$$

```
G <- t(A); G
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    7    9   11
## [3,]   13   15   17
```

Arithmetic Operation

- Determinant

$$G = det(A) = \begin{vmatrix} 1 & 7 & 13 \\ 3 & 9 & 15 \\ 5 & 11 & 17 \end{vmatrix}$$

```
G <-det(A); G
```

```
## [1] 4.263256e-14
```

Arithmetic Operation

Inverse

For inverse, we use **solve()** a base function in R

```
H <- solve(B)
H
```

Did you encounter a problem?

Be of good cheer; for I have overcome the world!- Jesus Christ in John 16:33

Inverse function to tackle the problem

```
inverse <- function(A){
  if (det(A)<0.01)
   cat("Since the given matrix is singular.
       Sorry, I can't find inverse")
  else
    solve(A)
  }
```

```
inverse(A)
```

```
## Since the given matrix is singular.
##       Sorry, I can't find inverse
```

Short group work

Use the function that you wrote to find the inverse of matrix J, where J is:

$$J = \begin{pmatrix} 5 & 1 & 0 \\ 3 & -1 & 2 \\ 4 & 0 & -1 \end{pmatrix}$$

Note

Assign the matrix to J and call inverse(J) in **R**

Short group work

Can you also confirm it with the base function **solve(J)**?

```
## solve(J)
```

Are they the same? Try it with this **R-code**

```
## inverse(J)==solve(J)
```

System of linear equation

We can use matrix skills to solve any system of linear equations

Solve the following system of equations

$$x - y = 3$$
$$2x + 3y = -4$$

Matrices preparation

$$A = \begin{pmatrix} 1 & -1 \\ 2 & 3 \end{pmatrix} \quad B = \begin{pmatrix} x \\ y \end{pmatrix} \quad C = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

$$B = A^{-1} \times C$$

Coding in R

```
A=matrix(c(1,-1,2,3),nrow = 2,byrow = T)
A
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    2    3
```

```
C=matrix(c(3,-4),nrow = 2, byrow = T)
C
```

```
##      [,1]
## [1,]    3
## [2,]   -4
```

Coding in R

```
B=solve(A)%*%C
B
```

```
##      [,1]
## [1,]    1
## [2,]   -2
```

```
x = B[1,1]; x
```

```
## [1] 1
```

```
y = B[2,1]; y
```

```
## [1] -2
```

Eigenvalues and Eigenvectors

Consider the following matrix

$$B = \begin{pmatrix} 1 & -6 \\ 3 & -8 \end{pmatrix}$$

1. Determine the eigenvalues of B
2. Determine the eigenvectors corresponding to each eigenvalue of B

Solution

```
B <- matrix(c(1,-6, 3, -8), nrow = 2, ncol = 2, byrow = TRU
print(B) # To see the matrix
```

```
##      [,1] [,2]
## [1,]    1   -6
## [2,]    3   -8
```

Eigenvalues and Eigenvectors

The function for calculating `eigenvalues` is `eigen()`. Note the function `eigen()` will produce a list as results. You will soon know what a `list()` is in the next chapter of this book.

```
eigen(B)
```

```
## eigen() decomposition
## $values
## [1] -5 -2
##
## $vectors
##           [,1]      [,2]
## [1,] 0.7071068 0.8944272
## [2,] 0.7071068 0.4472136
```

Short group work

Consider the following matrix

$$B = \begin{pmatrix} 4 & 5 & -5 \\ 0 & 4 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

1. Determine the eigenvalues of B
2. Determine the eigenvectors corresponding to each eigenvalue of B

Dataframe

Dataframes are another way to put data in tables! Unlike matrices, dataframes can have different types of data!

A dataframe has the variables of a data set as `columns` and the observations as `rows`. This will be a familiar concept for those coming from different statistical software packages such as `Excel`, `SPSS`, or `STATA`

The function for dataframe is `data.frame()`.

Example

```r
# Make a dataframe with columns named a and b
data.frame(a = 2:4, b = 5:7)
```

a	b
2	5
3	6
4	7

The numbers `1 2 3` at the left on your `console` are `row` labels and are not a `column` of the dataframe

Each column in a dataframe is a vector!

Dataframe

Example

a= c(6, 5, 1)
b= c(1, 1, 3)
data=data.frame(a,b) # The output is
data

Group work

Create a dataframe and call it data for the following vectors:

```
height <- floor(rnorm(n=100,mean = 135, sd=12))
weight <- ceiling(rnorm(n=100, mean = 55, sd=9))
```

Quick, have a look at your dataset

Working with large datasets is common in data science. When you work with (extremely) large datasets and dataframes, your first task as a data analyst is to develop a clear understanding of its structure and main elements. Therefore, it is often useful to show only part of the entire dataset.

1. `head()`: enables you to show the first observations of a dataframe.
2. `tail()`: enables you to print out the last observations in your dataset.

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

Have a look at the structure

Another method that is often used to get a rapid overview of your dataset is the function `str()`.

3. `str()`: Shows you the structure of your dataset

The structure of a dataframe tells you :

1. The total number of observations
2. The total number of variables
3. A full list of the variables names
4. The first observations

Note

Applying the `str()` function will often be the first thing that you do when receiving a new dataset or dataframe. It is a great way to get more insight in your dataset before diving into the real analysis.

Example

Considering these vectors:

```
height <- floor(rnorm(n=120,mean = 135, sd=12))
weight <- ceiling(rnorm(n=120, mean = 55, sd=9))
```

Create a dataframe for it.

```
data <- data.frame(height, weight)
```

```
str(data)
```

```
## 'data.frame':    120 obs. of  2 variables:
##  $ height: num  159 146 147 148 129 136 140 122 140 132
##  $ weight: num  62 73 62 60 57 61 70 61 70 67 ...
```

Example

```
head(data, 5)
```

height	weight
159	62
146	73
147	62
148	60
129	57

```
tail(data,3)
```

	height	weight
118	140	61
119	111	43
120	139	52

Using built-in datasets in R

There are several ways to find the included datasets in R.

Using `data()` will give you a list of the datasets of all loaded packages.

Example

```
# This shows the library in which datasets are stored.
data()
```

Example

```
library(datasets)
data <- airquality
str(data)
```

To get help in the proper description of the dataset `?airquality`

References


Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data. " O'Reilly Media, Inc.".