Basics of R

Anjana K

2023-10-11

Data-Analytics-using-R

Course Repository for add-on programme at VJCET, Kannur

Introduction

R is a programming language created and developed in 1991 by two statisticians at the University of Auckland, in New Zealand. It officially became free and open-source only in 1995. For its origins, it provides statistical and graphical techniques, linear and non-linear models, techniques for time series, and many other functionalities. Even if Python is the most common in the Data Science field, R is still widely used for specialized purposes, like in financial companies, research, and healthcare.

Requirements to Learn R Programming

If you want to start programming in R, you need to install the last versions of R and R studio or use the cloud version - poitcloud. You are surely asking yourself why you need to install both. If you prefer, you can install only R and you will have a basic tool to write the code. In addition, R studio provides an intuitive and efficient graphical interface to write code in R. It allows to divide the interface into subwindows to visualize separately the code, the output of the variables, the plots, the environment, and many other features.

Link to register with posit cloud- https://login.posit.cloud/register?redirect=%2F

Basics of R programming

> Assignment operation
>
> When we program in R, the entities we work with are called objects. They can be numbers, strings, vectors, matrices, arrays, or functions. So, any generic data structure is an object. The assignment operator is <- (or =), which combines the characters < and -. We can visualize the output of the object by calling it:

```
x <- 23
x
```

```
## [1] 23
```

> **Task 1:** Create two variables a,b , assign values to them and find sum, difference, product and quotient using R code.

```
a=34
b=45
sum=a+b
prod=a*b
qu=a/b
sum
```

```
## [1] 79
```

```
prod
```

```
## [1] 1530
```

```
qu
```

```
## [1] 0.7555556
```

Vectors in R Programming

In R, the vectors constitute the simplest data structure. The elements within the vector are all of the same types. To create a vector, we only need the function `c()` :

```
v1 <- c(2,4,6,8,9)
v1
```

```
## [1] 2 4 6 8 9
```

This function simply concatenates different entities into a vector. There are other ways to create a vector, depending on the purpose. For example, we can be interested in creating a list of consecutive numbers and we don't want to specify them manually. In this case, the syntax is`a:b` , where a and b correspond to the lower and upper extremes of this succession. The same result can be obtained using the function `seq()`.

```
v2 <- 1:7
v2
```

```
## [1] 1 2 3 4 5 6 7
```

```
#[1] 1 2 3 4 5 6 7
v3 <- seq(from=1,to=7)
v3
```

```
## [1] 1 2 3 4 5 6 7
```

```
#[1] 1 2 3 4 5 6 7
```

The function `seq()` can also be applied to create more complex sequences. For example, we can add the argument by the step size and the length of the sequence:

```
v4 <- seq(0,1,by=0.1)
v4
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
#[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
v5 <- seq(0,2,len=11)
v5
```

```
##  [1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
```

```
#[1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
```

To repeat the same number more times into a vector, the function `rep()` can be used:

```
v6 <- rep(2,3)
v6
```

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

```
v7 <-c(1,rep(2,3),3)
v7
```

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

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

There are not only numerical vectors. There are also logical vectors and character vectors:

```
x <- 1:10
y <- 1:5
l <- x==y
l
```

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

```
c <- c('a','b','c')
c
```

```
## [1] "a" "b" "c"
```

Factors in R Programming

Factors are specialized vectors used to group elements into categories. There are two types of factors: `ordered` and `unordered`. For example, we have the countries of five friends. We can create a factor using the function `factor()`

```
states <- c('italy','france','germany','germany','germany')
statesf<-factor(states)
statesf
```

```
## [1] italy   france  germany germany germany
## Levels: france germany italy
```

To check the levels of the factor, the function `levels()` can be applied.

```
levels(statesf)
```

```
## [1] "france"  "germany" "italy"
```

Matrices in R Programming

As you probably know, the matrix is a 2-dimensional array of numbers. It can be built using the function `matrix()`

```
Synatx: var=matrix(data,nrow=m,ncol=m,byrow=TRUE)
```

```
m1 <- matrix(1:6,nrow=3)
m1
```

```
m2 <- matrix(1:6,ncol=3)
m2
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

It can also be interesting combine different vectors into a matrix row-wise or column-wise. This is possible with rbind() and cbind() :

```
countries <- c('italy','france','germany')
age <- 25:27
rbind(countries,age)
```

```
##           [,1]    [,2]     [,3]
## countries "italy" "france" "germany"
## age       "25"    "26"     "27"
```

or

```
countries <- c('italy','france','germany')
age <- 25:27
cbind(countries,age)
```

```
##      countries age 
## [1,] "italy"   "25"
## [2,] "france"  "26"
## [3,] "germany" "27"
```

Arrays in R Programming

Arrays are objects that can have one, two, or more dimensions. When the array is one-dimensional, it coincides with the vector. In the case it's 2D, it's like to use the matrix function. In other words, arrays are useful to build a data structure with more than 2 dimensions.

```
a <- array(1:16,dim=c(6,3,2))
a
```

```
## , , 1
## 
##      [,1] [,2] [,3]
## [1,]    1    7   13
## [2,]    2    8   14
## [3,]    3    9   15
## [4,]    4   10   16
## [5,]    5   11    1
## [6,]    6   12    2
## 
## , , 2
## 
##      [,1] [,2] [,3]
## [1,]    3    9   15
## [2,]    4   10   16
## [3,]    5   11    1
## [4,]    6   12    2
## [5,]    7   13    3
## [6,]    8   14    4
```

lists

The list is a `ordered collection` of objects. For example, it can a collection of vectors, matrices. Differently from vectors, the lists can contain values of different type. They can be build using the function `list()` :

```
x <- 1:3
y <- c('a','b','c')
l <- list(x,y)
l
```

```
## [[1]]
## [1] 1 2 3
## 
## [[2]]
## [1] "a" "b" "c"
```

Data frames in R Programming

A data frame is very similar to a matrix. It's composed of rows and columns, where the columns are considered vectors. The most relevant difference is that it's easier to filter and select elements. We can build manually the dataframe using the function `data.frame()` :

```
countries <- c('italy','france','germany')
age <- 25:27
df <- data.frame(countries,age)
df
```

```
##   countries age
## 1     italy  25
## 2    france  26
## 3   germany  27
```

Reading data from en external file or data source

An alternative is to read the content of a file and assign it to a data frame with the function `read.table()` :

```
df <- read.csv('sentiment.csv')
df
```

```
##
## 1
## 2
## 3
## 4
## 5
## 6  Rama, an informant: Most dwellings in Adiya and Paniya communities were built by the government ar
## 7
## 8
## 9
## 10
## 11
## 12
## 13
## 14
## 15
## 16
## 17
## 18
## 19
## 20
## 21
## 22
## 23
## 24
## 25
## 26
## 27
## 28
## 29
## 30
## 31
## 32
## 33
## 34
## 35
## 36
## 37
## 38
## 39
## 40
## 41
## 42
## 43
## 44
## 45
## 46
## 47
## 48
## 49
## 50
## 51
```

Built-in datsets in R

R provides pre-loaded data using the function `data()`. A simple example is shown bellow:

```
data(mtcars)
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

The function `head()` allows visualizing the first 6 rows of the mtcars dataset, which provides the data regarding fuel consumption and ten characteristics of 32 automobiles.

All information about the `mtcars` datset can be extracted using the `help(mtcars)` function.

```
help(mtcars)
```

- the function `dim()` to look at the dimensions of the data frame
- the function `names()` to see the names of the variables

```
dim(mtcars)
```

```
## [1] 32 11
```

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Five point summary of a features/ numerical columns of a dataframe

The summary statistics of the variables can be obtained through the function `summary()`

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

We can access specific columns using the expression `dataframe$namevariable`. If we want to avoid specifying every time the name of the dataset, we need the function `attach()`.

For example:

```
attach(mtcars)
mpg
```

In this way, we attach the data frame to the search path, allowing to refer to the columns with only their names. Once we attached the data frame and we aren't interested anymore to use it, we can do the inverse operation using the function `detach()`.

Accessing values in a dataframe We can select the first row in the data frame using this syntax:

```
mtcars[1,]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

First column of `mtcars`:

```
mtcars[,1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Basic filtering operation We can filter the rows using a logical expression:

```
mtcars[mpg>20,]
```

Showing a particular column satisfying a row condition as:

```
mtcars[mpg>20, mpg]
```

Conditionals in R programming

`If` statement in R Programming

The syntax of the `if` statement is similar to the one in `Python`. As before, the difference is the addition of the parenthesis and curly brackets.

Syntax- if:

```
if (cond1) {statement1} else {statement2}
```

Syntax if-else

```
if (cond1) {statement1} else if {statement2} else {statement3}
```

Example

```
# code to check whether a number is even or not
x= as.integer(readline(prompt="Enter the number:"))
if (i%%2==0) print('even') else print('odd')
```

Task-1: Assign two values to `a` and `b`. Find the largest.

```r
a <- 10
b <- 2
if (b > a){
  print('b is greater than a')
}else if (a == b){
  print('a and b are equal')
}else {
  print('a is greater than b')
  }
```

```
## [1] "a is greater than b"
```

There is also a vectorized version of the if statement, the function `ifelse(condition,a,b)` . It's the equivalent of writing:

```r
ifelse(a>b,'a is greater than b','b is greater than a')
```

```
## [1] "a is greater than b"
```

Iteratives in R programming

The `for` loop is used to iterate elements over the sequence like in `Pandas`. The difference is the addition of the parenthesis and curly brackets. It has slightly different syntax:

Example:

```r
for (i in 1:4)
{print(i)}
```

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

`while` loop

`while` executes a statement or more statements as long as the condition is `TRUE`

Syntax `while (cond) statement`