Introduction to R

DataVedi

March 25, 2016

# Introduction

With an worldwide base of more than 2 million, R is not just a statistical programming language. R is also

* A powerful tool for all kinds of data processing
* A tool that enables its users to make all kind of graphics and data visualizations
* A versitile toolbox enabling different add on packages which allows its users to demonastrate high quality statistical predictions and data processing

R is indeed easy to learn and as said in the earlier paragraph R is not just a data processing or a statistical tool it is nore than that, and it is perfect to get started with data analytics. To understand the funcationality of any data analysis tool or language it is important for us to understand its 3 major components first

* The Basics, environment, coding syntax
* the way data is handled (Data handling)
* Important functions and the way of Performing analysis

So a question might arise why is R the perfect language to crunch data?

Since its incorporation if 1997, R has always been a free source and a perfect substitute for paid statistical softwares like SAS and Matlab. Did you know companies like Google, Facebook, Bank of America all use R ? That is the commercial spredability of R. Right form data scientist, biologists to wall street traders all have been using R for crunching data because of its simplicity and obvious appeal. R gives you the freedom to shift through complex data sets, manipulate data, create graphics and also to present data findings on dashboards. Another edge which R has over its peers is the constant addition of new packages and enriching the already available function sets. This could be one of the reasons why the nerdy statisticians love R over others.

The motto of this very book is to take you through the journey where your would not only get familliar with R but also would find it easy to apply it in your daily analytical processing without any hassle.

# What is R

R is not just a programming language. It also serves as a software environment for statistical computing and graphics desing. R is majorly used in the fields of:

* Data Manipulation
* Statistical Computing
* Graphics

R comes free, its open source

R is available under an open source license, which literally means that anyone can download it and modify the code according to their will. R is free of charge. It's source code can be accessed, modified and be imporved. This might be the reason why R code is mostly bug free, it is because of this feature of R it is believed ot be as one of the most reliable and stable platform.

R runs on all operating platforms

You can run R on Windows, Linus and also on the Mac. Thats a good example of being versitile isn't it?

R supports extensions

One big advantage of using R is it's extensibility. Developers who use R can write their own softwares and distribute them in the form of add on packages.Because of the relative ease of creating these packages thousands of theme exists and this is a reason why R contains numerous statistical methods.

There are other advantages of R and some of them are:

\* R is supported by a large user network  
  
\* R has excellent graphic capabilities   
  
\* It is supported by a large user network   
  
\* R contains some statistical algorithms that are not yet available in other tools  
  
\* R provides many solutions and is diverse in nature. It is applicable in   
  
 + Data Handling   
  
 + Data Minning   
  
 + Data Visualization   
  
 + Text Mining   
  
 + Big Data and Machine Learning

## Downloading and Installing R

In order to start working on R, one needs to install it first and for the same one needs to follow the following steps:

STEP 1

* *Go to* [*link*](http://cran.r-project.org/) *http"//cran.r-project.org to download R*

STEP 2

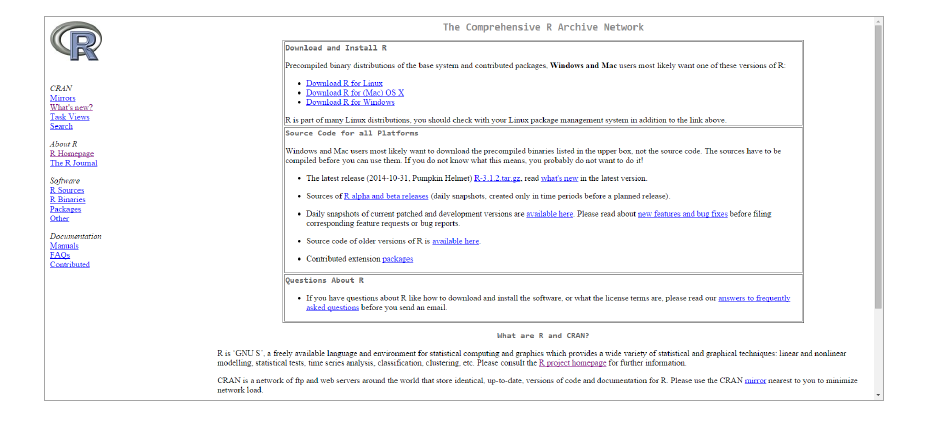
* *Select the relevant version and download the same*

STEP 3

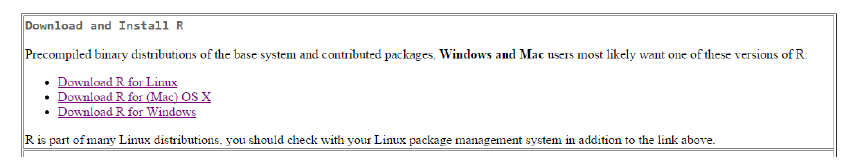
* *Install it by executing the .exe file*

A Step by Step guide to download R into your system is shown as follows:

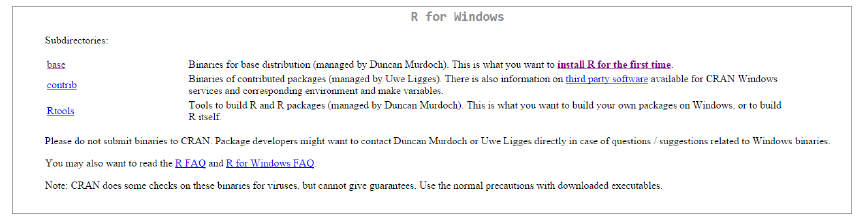
STEP 1



STEP 2



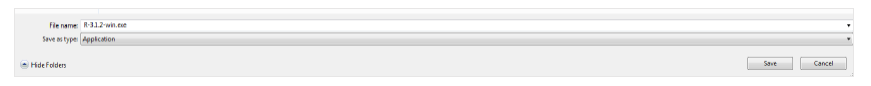
STEP 3



STEP 4



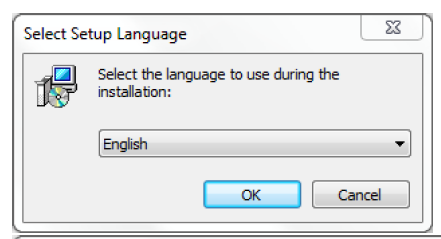
STEP 5



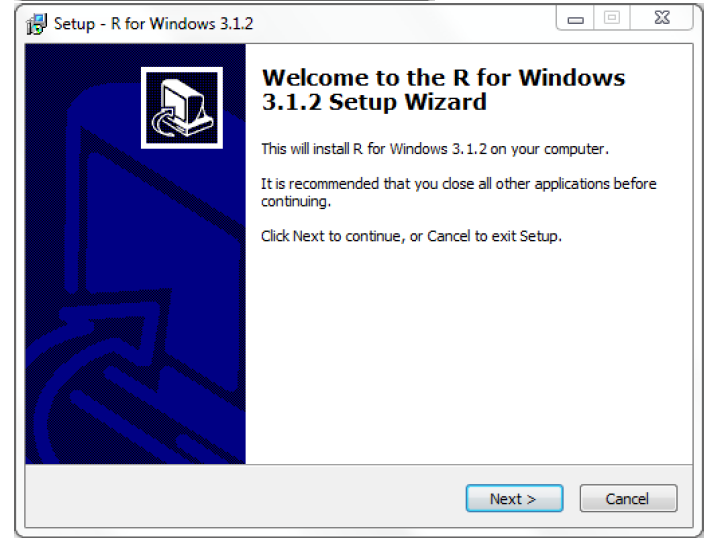
STEP 6



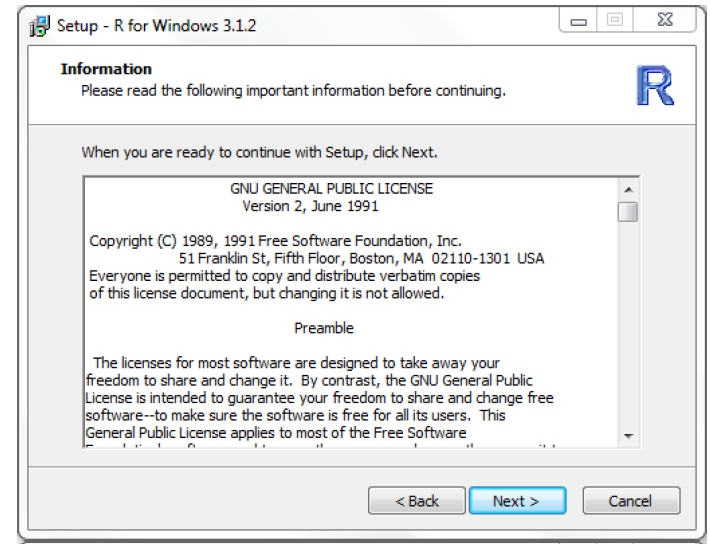
STEP 7



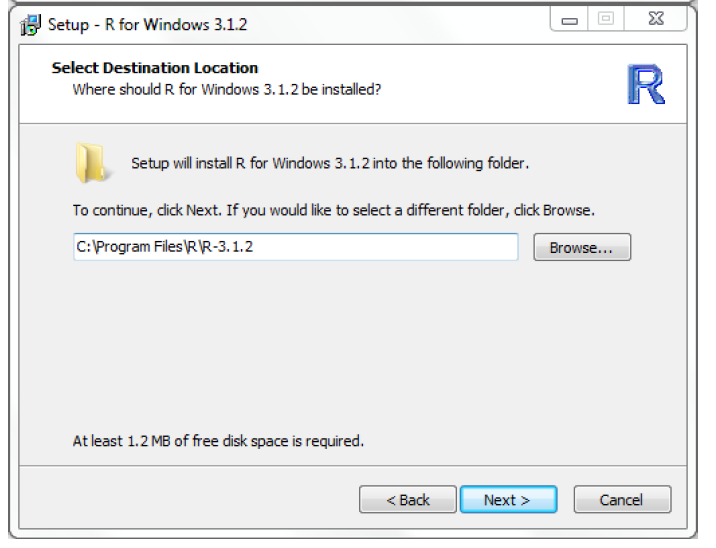
STEP 8



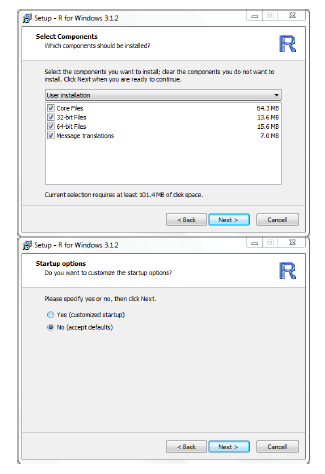
STEP 9



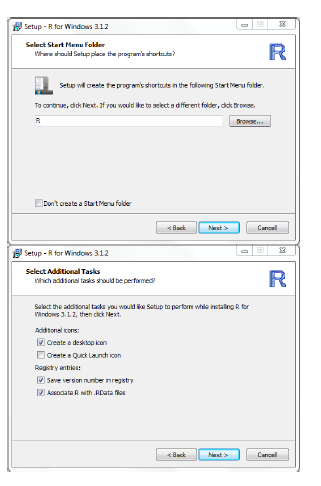
STEP 10



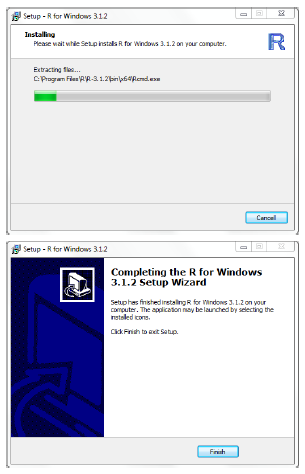
STEP 11



STEP 12



STEP 13



## R Studio

Coding can actually be made fun when one expereiences a good UI interface and R Studio is basically for that.R studio is an user-friendly UI for interacting with R as we all know R is command line interface, coding might be little slow for learners, where as R studio gives us shortcuts for direct clicks It is a free and open-source integrated development environment (IDE) for R. R studio has comprehensive abilities to make the coding on R more efficient. You need R to make R-Studio work. R-Studio is just a skin, the actual core programming language is R. All the commands typed in R-Studio will be submitted to R and the output will be fetched and displayed in R Studio

## Downloading and Installing R Studio

STEP 1

* *Go to www.rstudio.com to donwload R Stuio*

STEP 2

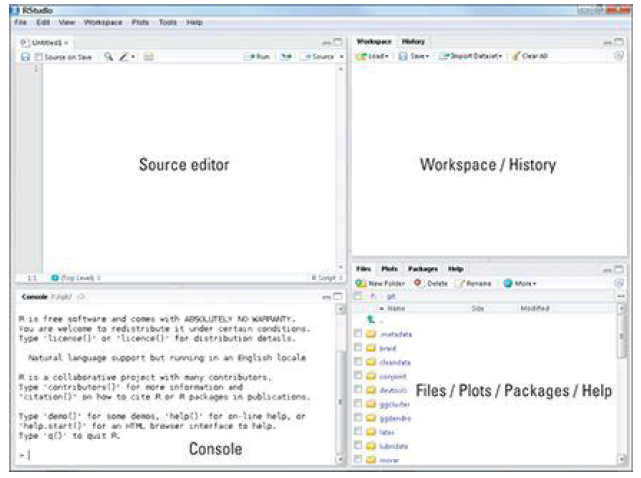
* *Select the relevant version and download the same*

STEP 3

* *Install it by executing the .exe file*

# Three Main Windows

There are three main windows in R and having an overview of them is relatively important before starting to use R. The three windows of R are: \* Console \* Workspace \* Output



**R Console**

This is where we type and submit the commands. Most of the times the output is shown in the console itself Hit Enter key to submit the commands. Up and Down arrows will recall previous command. Type partial command and use 'tab' key for autofill recommendations. The RStudio console includes a variety of features intended to make working with R more productive and straightforward.

## Warmup TIME..!!

Try out these commands on the R console and check the out the results. Write them on the blank box provided on the sides of each command line

|  |  |
| --- | --- |
| Command | Observation |
| 68+28 |  |
| 134\*456 |  |
| sqrt(19) |  |
| log(10) |  |
| exp(5) |  |
| help(log) |  |

## Workspace

During an R session, all user defined objects are stored in a temporary, working memory. Commands are entered interactively at the R user prompt. Up and down arrow keys scroll through your command history.  
Some useful commands while you run your R program are as follows

|  |  |
| --- | --- |
| Code | Description |
| **ls()** | Lists all objects |
| **rm()** | Removes all objects |

The objects in the workspace will last for just for that session, unless we save the workspace

# The Assign Operators

The Assign operator is typically used to assign a value to a name. There are three different assignment operators: two of them have leftwards and rightwards forms.The operators <- and = assign into the environment in which they are evaluated. The operator <- can be used anywhere, whereas the operator = is only allowed at the top level (e.g., in the complete expression typed at the command prompt) or as one of the subexpressions in a braced list of expressions. The operators <<- and ->> are normally only used in functions, and cause a search to made through parent environments for an existing definition of the variable being assigned. If such a variable is found (and its binding is not locked) then its value is redefined, otherwise assignment takes place in the global environment. Note that their semantics differ from that in the S language, but are useful in conjunction with the scoping rules of R. In all the assignment operator expressions, x can be a name or an expression defining a part of an object to be replaced (e.g., z[[1]]).

* Let us demonstrate the usage of an assignment operator with a simple example

Lets suppose I want to assign a value 7 to a variable x, we can do the same with the following

x <- 7

Suppose I want to assign the value of a square root of 119 to k, we can do the same with the following command

k <- sqrt(119)

HANDS ON EXCERSISE

Try out assigning some more values to the following variables

* The sum of 68 and 100 to z
* The product of 134 and 156 to x
* The square root of 430 to y

## Naming Conventions in R

Names should always start with a letter (A-Z or a-z) and can also contain numbers, digits (0-9) and/or periods"". Remember that R is a case sensitive language and a CAPITAL is always a Capital and cannot be replaced with a lower case letter. For example

Mycar <- 1010 assigns the value 1010 to Mycar, and when we ask R to call Mycar we need to use the name Mycar and not mycar. R doesnot recognize the word mycar unless it is defined.

Another example might be Mydatabank is different from mydatabank

## Hands On Learning - 01

Try out these sample commands on R and see what comes out..!

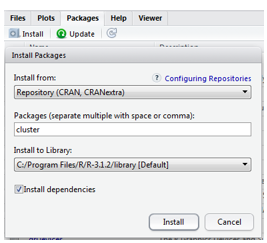
# R Packages

A R package includes a set of functions and datasets. A package consists of functions which are not readily available on R.So these packages are something unique when added to the core bring in extra advantage and would make your crunching easier. There are many packages available and to bring in these packages into the core we need to follow the following steps:

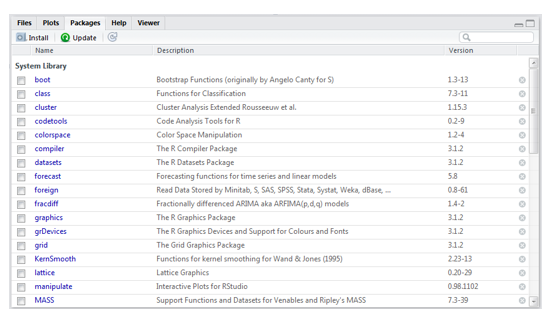
There are two components of R

* Core
* Packages

To download or install a package on R, follow the following steps:



* Select the Packages' menu and selectInstall package...', a list of available packages on your system will be displayed.



* Select one and click `OK', the package is now attached to your current R session. Via the library function Before using a function, we need to install the package that contains it

## Hands on Learning - 02

Try out the following commands on R

* Create three random vectors x, y, z of size 1000.
* Use rnorm() function to create these vectors.
* Draw a 3d scatter plot of these three vectors use the code scatterplot3d(x,y,z)

### Solution to the Hands on Learning Section - 02

x <- rnorm(1000,mean=20,sd=5) y <- rnorm(1000,mean=15,sd=3) z <- rnorm(1000,mean=25,sd=8) scatterplot3d(x,y,z)

install.packages("scatterplot3d") library(scatterplot3d) scatterplot3d(x,y,z)

## Some Useful Packages in R

There are nearly 7000 packages in R. and some useful Data handling Packages are

|  |  |
| --- | --- |
| Package | Description |
| foreign |  |
| xlsx |  |
| XLConnect |  |
| downloader |  |
| RSQLite |  |
| RMySQL |  |

RODBC, RMySQL, RPostgresSQL, RSQLite , downloader, XLConnect, xlsx, foreign, dplyr, tidyr, plyr, reshape2, zoo

Some Data visualization packages:

|  |  |
| --- | --- |
| Package | Description |
| rgl |  |
| htmlwidgets |  |
| dygraphs |  |
| plotly |  |
| shiny |  |
| rcdimple |  |

Some Advanced Analysis Packages:

|  |  |
| --- | --- |
| Package | Description |
| glment |  |
| vcd |  |
| lme4/nlme |  |
| randomForest |  |
| mgcv |  |
| Car |  |

# R Data Types

While programming, one needs to use variable variables to store information. So what is actually a variable? A variable is a reserved memory location to store values.Information of different data types like character, integer, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. There are many types of R Data Types:

* Vectors - Basic R Type.
* Data Frames - Collection of vectors.
* Lists - Collection of R objects
* Other type
* Matrix
* Factor
* Array

## R Vector

A vector is the simplest type of data structure. It can be defined as a *single entity consisting of a collection of things*. A collection of numbers for example is a numeric vector.So, a vector can be defined as a sequence of data elements of the same basic data type. Variables in the vector are called members. Vectors can be of different types namely, numeric, logical or character. An example of a numeric vector is as follows:

An example of a logical vector is as follows:

An example of a character vector is as follows:

As some programmers are aware that in programming language, a function is a peice of code that takes some inputs and does something specific with them. In constructing a vector one has to tell the c() which is a concatenate operator to construct a vector with the five integers as done above in the example of a numeric vector. The entries inside the parentheses are reffered to as arguments.

Vectors can also be constructed using operators. An operator is a symbol like '-' '+', and they have the same meaning they do in mathematics. Another operator which comes handy is the colon (:). Try this out in the R concole by typing the following code

## [1] 1 2 3 4 5

The output is a sequence of numbers between 1 and 5. To check if a variable is a vector or not we can use the command is.vector(name). Try out a simple exercise to broden your understanding

## [1] TRUE

## [1] TRUE

Most mathematical functions and operators can be applied to vectors without writing any loops. Try out of this code in R console

Age+3  
English1<- English+10  
English1<-80  
Total<- English1 + Science   
Total  
Age/Total  
  
x <- rnorm(1000,mean=20,sd=5)   
x-mean(x)

**Use the [] operator to select elements**

To select specific elements: \* Use index or vector of indexes to identify them

To exclude specific elements: \* Negate index or vector of indexes

Age  
Age[3]  
Age[2:5]  
Age[-2]  
Age[3]<-19  
Age[5]<-21  
Age

**Now Check the type of Vectors**

# R Data Frames

How do we actually store data in R? To serve this purpose we use Data Frames. A data frame is a commmon way for storing data in R, and serves the purpose of simplifying data in R. It is a list of vectors of equal length. As an example let us consider three vectors each being numeric, character and a logical vector. If we want to use all the three vectors at once we can call them by using a data frame.

x= c(1,2,3,4,5) # x is a numeric vector having values from 1 to 5  
y= c("H", "E", "L", "L", "O") # y is a character vector having characters HELLO  
z= c(TRUE, FALSE) # z is a logical vector  
a= data.frame(x,y,z) # a is a data frame having vectors x,y,z

a is called a data frame in this case. A data frame can be created using the command *data.frame* followed by the induvidual vectors as arguments.

TIP

Data Frames should have variables of same length and remember that when you use data.frame() function, the character variables are imported into as factors or categorial variables and to find out more about your data frame use the command str(). For example

a<-c(1,2,3)  
str(a) #Tell us the structure of a

TIP

If you are interested to inspect the first and the last lines of your data frame use the functions head() and tail() respectively.

## Accessing a row or a Column or an element in the data frame

What if you want to access a particular row or a column in the data frame in specific? Well, R has functions even for that. Let us decode this problem using a simple example

ALTERNATE COMMAND

An alternate to the [,] command is using $. An example displaying the usage is as follows:

## Built-in Data Frames

As discussed millions of programmers use R for their data analysis i.e, crunching data and the millions of programmers tend to also upload their data sets on R and this facilitates us to utilize these data sets to use as a part of our analysis. These built in data sets come as a primary package in R. So one can typically list these data sets using their names and then load a data set into the memory for furthur analysis. Use the library() function to load a particular package and command data() will typically list all the data sets in the loaded package. An example is illustrated below for clear understanding

data()  
AirPassengers # A data set named Air Passengers is called

## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec  
## 1949 112 118 132 129 121 135 148 148 136 119 104 118  
## 1950 115 126 141 135 125 149 170 170 158 133 114 140  
## 1951 145 150 178 163 172 178 199 199 184 162 146 166  
## 1952 171 180 193 181 183 218 230 242 209 191 172 194  
## 1953 196 196 236 235 229 243 264 272 237 211 180 201  
## 1954 204 188 235 227 234 264 302 293 259 229 203 229  
## 1955 242 233 267 269 270 315 364 347 312 274 237 278  
## 1956 284 277 317 313 318 374 413 405 355 306 271 306  
## 1957 315 301 356 348 355 422 465 467 404 347 305 336  
## 1958 340 318 362 348 363 435 491 505 404 359 310 337  
## 1959 360 342 406 396 420 472 548 559 463 407 362 405  
## 1960 417 391 419 461 472 535 622 606 508 461 390 432

cars # A subset of the data set is called

## speed dist  
## 1 4 2  
## 2 4 10  
## 3 7 4  
## 4 7 22  
## 5 8 16  
## 6 9 10  
## 7 10 18  
## 8 10 26  
## 9 10 34  
## 10 11 17  
## 11 11 28  
## 12 12 14  
## 13 12 20  
## 14 12 24  
## 15 12 28  
## 16 13 26  
## 17 13 34  
## 18 13 34  
## 19 13 46  
## 20 14 26  
## 21 14 36  
## 22 14 60  
## 23 14 80  
## 24 15 20  
## 25 15 26  
## 26 15 54  
## 27 16 32  
## 28 16 40  
## 29 17 32  
## 30 17 40  
## 31 17 50  
## 32 18 42  
## 33 18 56  
## 34 18 76  
## 35 18 84  
## 36 19 36  
## 37 19 46  
## 38 19 68  
## 39 20 32  
## 40 20 48  
## 41 20 52  
## 42 20 56  
## 43 20 64  
## 44 22 66  
## 45 23 54  
## 46 24 70  
## 47 24 92  
## 48 24 93  
## 49 24 120  
## 50 25 85

?cars

## starting httpd help server ...

## done

# Lists

A list is a vector containing objects. Technically, lists are indeed vectors, but the primary difference between a vector and a list is that a list can contain elements of different types. A vector can be furthur explained with the help of an example

## Constructing a List

So, as defined earlier a list is a datastructure that can hold any types of data structures. Let it be a data structure, vector, a character or an object a list might contain any kind of data structures.

TIP

Did you know we can turn an object into a list by using the function as.list(). Try out the following on the R console

## Manipulating a List

Supposed we have already passed a list, and we would like to name the components of a list we can do the same using the function names(). This is useful for extracting a particular component of the list, so basically this makes our life simpler.

x<- c(1,2,3,4,5,6,7)  
y<- c("Rick", "Nick", "Tom", "Harry", "Rashik", "Pappi", "Ritu")  
z<- c(34,32,35,35,34,32,31)  
  
list<- list(x,y,z)  
list

## [[1]]  
## [1] 1 2 3 4 5 6 7  
##   
## [[2]]  
## [1] "Rick" "Nick" "Tom" "Harry" "Rashik" "Pappi" "Ritu"   
##   
## [[3]]  
## [1] 34 32 35 35 34 32 31

names(list)<-c("Numbers", "Names", "Marks") # name the components of the list  
list

## $Numbers  
## [1] 1 2 3 4 5 6 7  
##   
## $Names  
## [1] "Rick" "Nick" "Tom" "Harry" "Rashik" "Pappi" "Ritu"   
##   
## $Marks  
## [1] 34 32 35 35 34 32 31

## Extracting Components from a List

We can actually extract a particular component from a list by using the [] operator. Suppose I would like to extract the 3rd component from the list we created in the previous example. Then I would be writing the following code

## Subsetting a List

We can use the operator [] and c() to choose a particular component. An illustrated code can help you understand the working better

TIP

You can also delete a component of a list by setting it equal to NULL. You can also add a new component to the list or replace a particular component using the $ or [] operators.

## Describing a List

Suppose we would like to know the charactersitics of a list which include:

* The class of the list
* Number of components in the list
* A short kind of a summary of each list

To find the class of the list use the function class()

## Converting a List into a Matrix or a Data Frame

We can actually convert a list into a matrix or a data frame and for doing the same we need to unlist the already defined list and for the same we use the function unlist()

## Lists Examples

# Some other Data Types

**Factor**

Factors are categorical variables in R which take a limited number of different values, such varibales are referred to as categorical variables. Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed. The factor() function is used to create a factor. Factors represent a very efficient way to store character values, because each unique character value is stored only once, and the data itself is stored as a vector of integers.

## Matrix

A matrix is a multidimensional array. Like a vector makes looping operations very easy, matrix makes some multidimensional calculations very easy. A matrix Works perfectly for a lot of optimization problems which involve intense calculations. There are various ways to construct a matrix. When we construct a matrix directly with data elements, the matrix content is filled along the column orientation by default.

\*\* CONSTRUCTING A MATRIX IN R \*\*

To construct a matrix in R, we can use the command matrix(). A code is illustrated below which we can use as an aid for contruction. The number of rows and columns can be defined with the command nrow and ncol.

TRANSPOSE OF A MATRIX

The transpose of a matrix is where the rows of the original matrix becomes the column of the other. To perform the transpose function use the function t()

Let's illustrate transpose of a matrix with the following example

Combining Matrices

Combining matrices is possible with R. For combining matrices we can use the function cbind(). Lets illustrate this with an example

# R History

Helps in accessing previously executed commands. User can send the selected history to either console or to source

# R Source file and Scripts

R script or code file. Can be used to re execute the stored codes. Hit Ctrl+enter to execute the commands. Save R script files for future use.

# How to add comments

Use # for commenting

# Your First R Program

Now, its time you put in your knowledge to test. Try out writing your first full fleged R program according to the data given below

Create income data(vector) for 4 employees with the values 5500, 6700, 8970, 5634 Create a new variable tax and save 0.2 in it Create a new variable year and save 2015 in it Create a new variable company and save "DataVedi" in it Derive net\_income by deducting tax from the income Create Employee name(vector) for 4 employees with the values Redd, Kenn, Finn, Scott Create a data frame with Employee name and Net income Create a new list with all the above information on company, year, tax, Employee name and Salary

**CLUE**

You can write your program accordingly

# R - Function

This section is mainly to describe the availability of some more functions available in R. We are already familliar with functions like c(), is.vector() and str(), there are some more vectors with numeric functions like abs(x), sqrt(x), ceiling(x), floor(x), trunc(x), round(x, digits=n) ,signif(x, digits=n), cos(x), sin(x), tan(x) ,log(x), log10(x), exp(x) ans some impotant String Functions are substr(x) start=n1, stop=n2), toupper(x), grep(pattern, x , ignore.case=FALSE, fixed=FALSE)

The use of functions can be illustrated usng the following code, try it out in R console

# Most Common Errors in R

Errors..! These are something which every programmers would like to avoid and it is the same in the case of programmers using R as well. First time programmers using R are generally faced by errors as these :

\*\* Could not find the object : Error: object 'XXXXX' not found \*\*

and don't really worry about this..! Understanding why the error has come out, nulifies half of your problem. Some common errors are related to syntax and missing packages. Some common reasons why errors pop up in R is because of the fact that the declared variable is not written in the proper syntax. Remember R is case sensitive and myvar is different from Myvar. Most of the first time programmers commonly indulge in this mistake. A typical example is illustrated below:

myvar <- c(15, 17, 16, 15, 16) Myvar

Error: object 'Myvar' not found

* Another common message we can encounter is

*Error: could not find function "qplot"*

This occurs when the relevant package is missing and solving is simple, just fix it by simply installing/ attaching the package that contains the function.

qplot(mpg, wt, data=mtcars) Error: could not find function "qplot" library(ggplot2) qplot(mpg, wt, data=mtcars)

* Coming to the next typical kind of errors we encounter in R is when numeric functions are applied to non numeric variables

Name <- c("John", "Bob", "Kevin", "Smith", "Rick") Name+1 Error in Name + 1 : non-numeric argument to binary operator

* Another error which might pop up is No such file or directory exists. The common reason for this might be because of the fact that the path might be spelled incorrectly or wrong and this can be rectified by updating the right path
* You might also experience an error poping out showing 'The package is incompatible or built for old version of R' don't worry just sit back and update R

# Getting Help in R

R has a built in Help feature to assist its users and using some commands you can invoke help. Some common R help menue commands are:

|  |  |
| --- | --- |
| Code | Description |
| **help.start()** | General Help |
| **?(function)** | Help about a function |
| **help(function)** | Help about a function |