1. Introduction to R and R studio

- R is an open-source programming language that is widely used as a statistical software and data analysis tool. R is an important tool for Data Science.
- R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.
- The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.
- R is free software distributed under a GNU-style copy left, and an official part of the GNU project called GNU S.

Features of R – Data Science

- R provides extensive support for statistical modelling.
- R is a suitable tool for various data science applications because it provides aesthetic visualization tools.
- R is heavily utilized in data science applications for ETL (Extract, Transform, Load). It provides an interface for many databases like SQL and even spreadsheets.
- R also provides various important packages for data wrangling.
- With R, data scientists can apply machine learning algorithms to gain insights about future events.
- One of the important feature of R is to interface with NoSQL databases and analyze unstructured data.

R Libraries

Dplyr: For performing data wrangling and data analysis, we use the dplyr package. We use this package for facilitating various functions for the Data frame in R. Dplyr is actually built around these 5 functions. You can work with local data frames as well as with remote database tables. You might need to:

Select certain columns of data.

Filter your data to select specific rows.

Arrange the rows of your data into order.

Mutate your data frame to contain new columns.

Summarize chunks of your data in some way.

Ggplot2: R is most famous for its visualization library ggplot2. It provides an aesthetic set of graphics that are also interactive. The ggplot2 library implements a “grammar of graphics”.

This approach gives us a coherent way to produce visualizations by expressing relationships between the attributes of data and their graphical representation.

Esquisse: This package has brought the most important feature of Tableau to R. Just drag and drop, and get your visualization done in minutes. This is actually an enhancement to ggplot2.It allows us to draw bar graphs, curves, scatter plots, histograms, then export the graph or retrieve the code generating the graph.

Tidyr: Tidyr is a package that we use for tidying or cleaning the data. We consider this data to be tidy when each variable represents a column and each row represents an observation.

Shiny: This is a very well known package in R. Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions. It’s a Data Scientist’s best friend.

Caret: Caret stands for classification and regression training. Using this function, you can model complex regression and classification problems.

E1071: This package has wide use for implementing clustering, Fourier Transform, Naive Bayes, SVM and other types of miscellaneous functions.

Mlr: This package is absolutely incredible in performing machine learning tasks. It almost has all the important and useful algorithms for performing machine learning tasks. It can also be termed as the extensible framework for classification, regression, clustering, multi-classification and survival analysis.

Applications of R for Data Science

Top Companies that use R for Data Science:

Google: At Google, R is a popular choice for performing many analytical operations. The Google Flu Trends project makes use of R to analyze trends and patterns in searches associated with flu.

Facebook: Facebook makes heavy use of R for social network analytics. It uses R for gaining insights about the behavior of the users and establishes relationships between them.

IBM: IBM is one of the major investors in R. It recently joined the R consortium. IBM also utilizes R for developing various analytical solutions. It has used R in IBM Watson – an open computing platform.

Uber: Uber makes use of the R package shiny for accessing its charting components. Shiny is an interactive web application that's built with R for embedding interactive visual graphics.

Installation of R and R studio

Windows Installation

You can download the Windows installer version of R from R-3.2.2 for Windows (32/64 bit) and save it in a local directory.

As it is a Windows installer (.exe) with a name "R-version-win.exe". You can just double click and run the installer accepting the default settings. If your Windows is 32-bit version, it installs the 32-bit version. But if your windows is 64-bit, then it installs both the 32-bit and 64-bit versions.

After installation you can locate the icon to run the Program in a directory structure "R\R-3.2.2\bin\i386\Rgui.exe" under the Windows Program Files. Clicking this icon brings up the R-GUI which is the R console to do R Programming.

First install R

https://cran.r-project.org/

R studio

https://posit.co/download/rstudio-desktop/

Basics of R

R Command prompt

Once you have R environment setup, then it’s easy to start your R command prompt by just typing the following command at your command prompt:

```
$ R
```

This will launch R interpreter and you will get a prompt > where you can start typing your program as follows:

```
> myString <- "Hello, World!"
> print ( myString)
[1] "Hello, World!"
```

Here first statement defines a string variable myString, where we assign a string "Hello, World!" and then next statement print() is being used to print the value stored in variable myString.

RScriptFile

Usually, you will do your programming by writing your programs in script files and then you execute those scripts at your command prompt with the help of R interpreter called Rscript. So let's start with writing following code in a text file called test.R as under:

```
# My first program in R Programming
myString <- "Hello, World!"

print ( myString)
```

Save the above code in a file test.R and execute it at Linux command prompt as given below. Even if you are using Windows or other system, syntax will remain same.

```
$ Rscript test.R
```

When we run the above program, it produces the following result.

```
[1] "Hello, World!"
```

Comments

Comments are like helping text in your R program and they are ignored by the interpreter while executing your actual program. Single comment is written using # in the beginning of the statement as follows:

```
# My first program in R Programming
```

R does not support multi-line comments

R- Data Types

- Generally, while doing programming in any programming language, you need to use various variables to store various information.

- Variables are nothing but reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.
- You may like to store information of various data types like character, wide character, integer, floating point, double floating point, 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.
- In contrast to other programming languages like C and java in R, the variables are not declared as some data type.
- The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are:
    - Vectors
    - Lists
    - Matrices
    - Arrays
    - Factors
    - Data Frames

The simplest of these objects is the vector object and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Data Type	Example	Verify

Logical	TRUE , FALSE	`v <- TRUE`
		`print(class(v))`
		it produces the following result:
		`[1] "logical"`

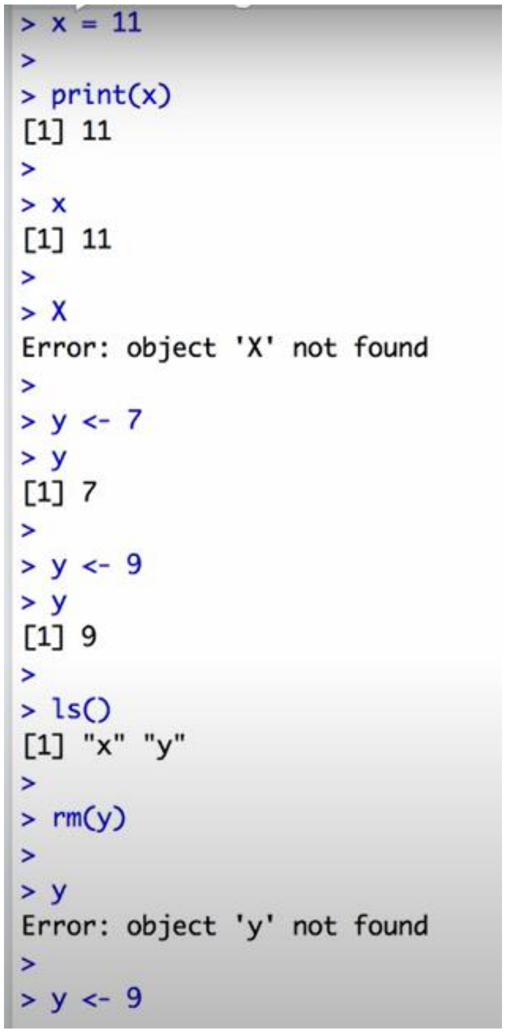
Numeric	12.3, 5, 999	`v <- 23.5`
		`print(class(v))`
		it produces the following result:

Integer	2L, 34L, 0L	`v <- 2L`
		`print(class(v))`
		it produces the following result:
		`[1] "integer"`

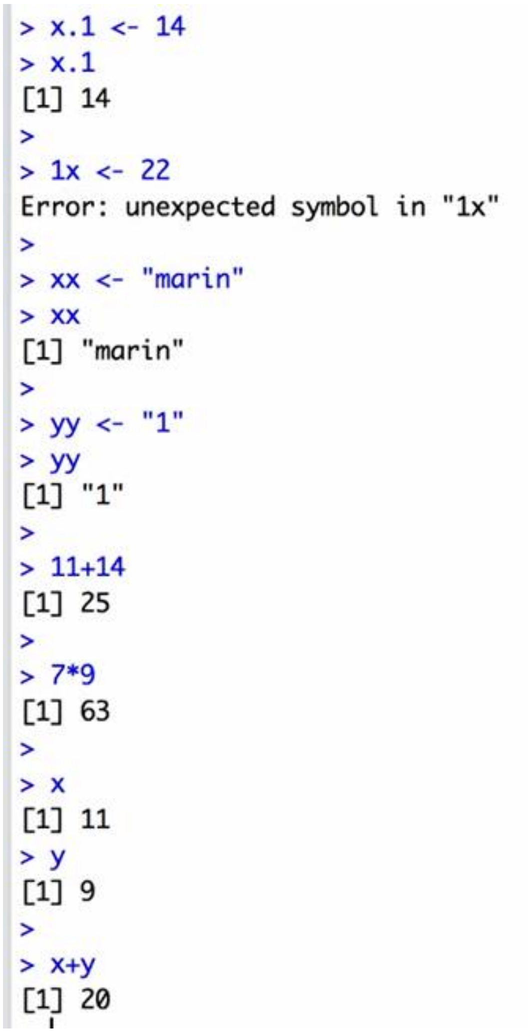
Complex	3 + 2i	`v <- 2+5i`
		`print(class(v))`
		it produces the following result:
		`[1] "complex"`

Character	'a' , '"good", "TRUE", '23.4'	`v <- "TRUE"`
		`print(class(v))`
		it produces the following result:
		`[1] "character"`

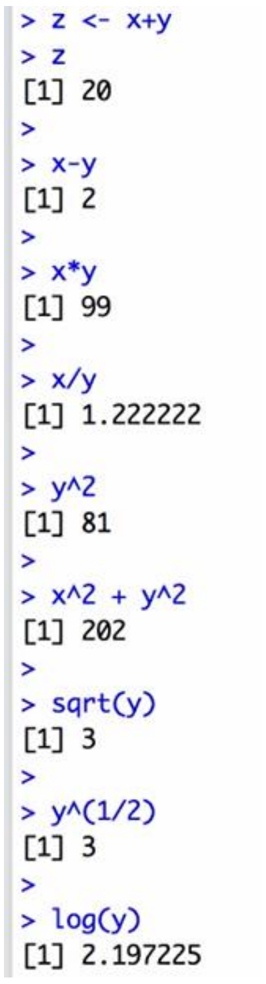
Raw	"Hello" is stored as 48 65 6c 6c 6f	`v <- charToRaw("Hello")`
		`print(class(v))`
		it produces the following result:
		`[1] "raw"`



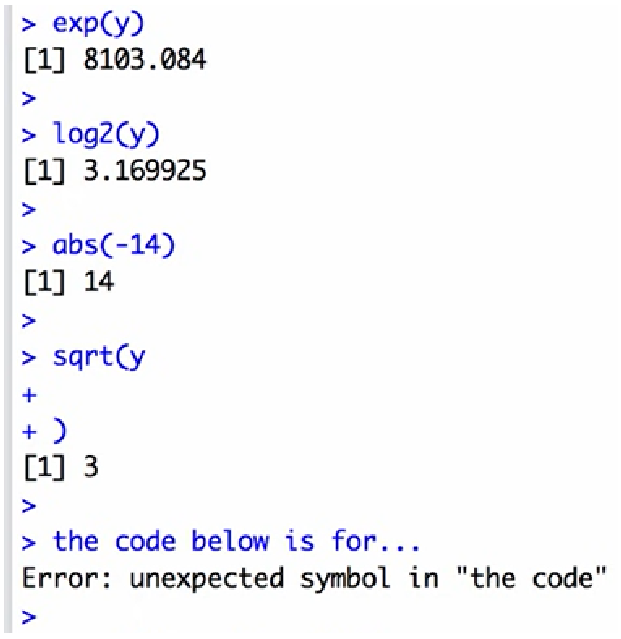
```
> x = 11
>
> print(x)
[1] 11
>
> x
[1] 11
>
> X
Error: object 'X' not found
>
> y <- 7
> y
[1] 7
>
> y <- 9
> y
[1] 9
>
> ls()
[1] "x" "y"
>
> rm(y)
>
> y
Error: object 'y' not found
>
> y <- 9
```



```
> x.1 <- 14
> x.1
[1] 14
>
> 1x <- 22
Error: unexpected symbol in "1x"
>
> xx <- "marin"
> xx
[1] "marin"
>
> yy <- "1"
> yy
[1] "1"
>
> 11+14
[1] 25
>
> 7*9
[1] 63
>
> x
[1] 11
> y
[1] 9
>
> x+y
[1] 20
```



```
> z <- x+y
> z
[1] 20
>
> x-y
[1] 2
>
> x*y
[1] 99
>
> x/y
[1] 1.222222
>
> y^2
[1] 81
>
> x^2 + y^2
[1] 202
>
> sqrt(y)
[1] 3
>
> y^(1/2)
[1] 3
>
> log(y)
[1] 2.197225
```



```
> exp(y)
[1] 8103.084
>
> log2(y)
[1] 3.169925
>
> abs(-14)
[1] 14
>
> sqrt(y
+
+ )
[1] 3
>
> the code below is for...
Error: unexpected symbol in "the code"
>
```

R-Variables

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many R-objects.

A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Variable Name	Validity	Reason
var_name2.	Valid	Has letters, numbers, dot and underscore
var_name%		

Invalid	Has the character '%'. Only dot(.) and underscore allowed.	
2var_name	Invalid	Starts with a number
.var_name ,var.name	valid	Can start with a dot(.) but the dot(.)should not befollowed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid
_var_name	invalid	Starts with _ which is not valid

Variable Assignment

- The variables can be assigned values using leftward, rightward and equal to operator. The values of the variables can be printed using print() or cat()function. The cat() function combines multiple items into a continuous print output.

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)

# Assignment using leftward operator.
var.2 <- c("learn","R")
```

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators

We have the following types of operators in R programming:

☐ Arithmetic Operators

☐ Relational Operators

☐ Logical Operators

☐ Assignment Operators

☐ Miscellaneous Operators

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example

+	Adds two vectors	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v+t)`
		it produces the following result:
		`[1] 10.0 8.5 10.0`

−	Subtracts second vector from the first	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v-t)`
		it produces the following result:
		`[1] -6.0 2.5 2.0`

*	Multiplies both vectors	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v*t)`
		it produces the following result:
		`[1] 16.0 16.5 24.0`

/	Divide the first vector with the second	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v/t)`
		When we execute the above code, it produces the following result:
		`[1] 0.250000 1.833333 1.500000`

%%	Give the remainder of the first vector with the second	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v%%t)`
		it produces the following result:
		`[1] 2.0 2.5 2.0`

%/%	The result of division of first vector with second (quotient)	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v%/%t)`
		it produces the following result:
		[1] 0 1 1

^	The first vector raised to the exponent of second vector	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v^t)`
		it produces the following result:
		`[1] 256.000 166.375 1296.000`

Relational Operators

Following table shows the relational operators supported by R language. Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v>t)`
		it produces the following result:
		`[1] FALSE  TRUE FALSE FALSE`

<	Checks if each element of the first vector is less than the corresponding element of the second vector.	`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v < t)`
		it produces the following result:
		`[1]  TRUE FALSE  TRUE FALSE`

==	Checks if each element of the first vector is equal to the corresponding element of the second vector.	`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v==t)`
		it produces the following result:
		`[1] FALSE FALSE FALSE  TRUE`

<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v<=t)`
		it produces the following result:
		`[1]  TRUE FALSE  TRUE  TRUE`

>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v>=t)`
		it produces the following result:
		`[1] FALSE  TRUE FALSE  TRUE`

Logical Operators

Following table shows the logical operators supported by R language. It is applicable only to vectors of type logical, numeric or complex. All numbers greater than 1 are considered as logical value TRUE.
Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.	`v <- c(3,1,TRUE,2+3i)`
		`t <- c(4,1,FALSE,2+3i)`
		`print(v&t)`
		it produces the following result:
		`[1]  TRUE  TRUE FALSE  TRUE`

|	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.	`v <- c(3,0,TRUE,2+2i)`
		`t <- c(4,0,FALSE,2+3i)`
		`print(v|t)`
		it produces the following result:
		`[1]  TRUE FALSE  TRUE  TRUE`

!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.	`v <- c(3,0,TRUE,2+2i)`
		`print(!v)`
		it produces the following result:
		`[1] FALSE  TRUE FALSE FALSE`

The logical operator && and || considers only the first element of the vectors and give a vector of single element as output.

&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	`v <- c(3,0,TRUE,2+2i)`
		`t <- c(1,3,TRUE,2+3i)`
		`print(v&&t)`
		it produces the following result:
		`[1] TRUE`

||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	`v <- c(0,0,TRUE,2+2i)`
		`t <- c(0,3,TRUE,2+3i)`
		`print(v||t)`
		it produces the following result:
		`[1] FALSE`

Assignment Operators

These operators are used to assign values to vectors.

Operator	Description	Example

<-
or
=
or
<<-
	Called Left Assignment	`v1 <- c(3,1,TRUE,2+3i)`
		`v2 <<- c(3,1,TRUE,2+3i)`
		`v3 = c(3,1,TRUE,2+3i)`
		`print(v1)`
		`print(v2)`
		`print(v3)`
		`it produces the following result:`
		`[1] 3+0i 1+0i 1+0i 2+3i`
		`[1] 3+0i 1+0i 1+0i 2+3i`
		`[1] 3+0i 1+0i 1+0i 2+3i`

->
 	
or
 	
->>
	Called Right Assignment	`c(3,1,TRUE,2+3i) -> v1`
		`c(3,1,TRUE,2+3i) ->> v2`
		`print(v1)`
		`print(v2)`
		`it produces the following result:`
		`[1] 3+0i 1+0i 1+0i 2+3i`
		`[1] 3+0i 1+0i 1+0i 2+3i`

Miscellaneous Operators

These operators are used to for specific purpose and not general mathematical or logic computation.

Operator	Description	Example

:	Colon operator. It creates the series of numbers in sequence for a vector.	`v <- 2:8`
		`print(v)`
		it produces the following result:
		`[1] 2 3 4 5 6 7 8`

%in%	This operator is used to identify if an element belongs to a vector.	`v1 <- 8`
		`v2 <- 12`
		`t <- 1:10`
		`print(v1 %in% t)`
		`print(v2 %in% t)`
		it produces the following result:
		`[1] TRUE`
		`[1] FALSE`

%*%	This operator is used to multiply a matrix with its transpose.	`M = matrix( c(2,6,5,1,10,4), nrow=2,ncol=3,byrow = TRUE)`
		`t = M %*% t(M)`
		`print(t)`
		it produces the following result:
		`     [,1] [,2]`
		`[1,]   65   82`
		`[2,]   82  117`