Introduction to R

By,
Sandesh G

CONTENTS

R History

1993-University of Auckland

Developer: R Development Core Team

R is a dialect of **S** language(1984- Bell laboratories)

On February 29, 2000, the software was deemed fully featured enough and stable enough for the 1.0 release to take place

1997- 100 Users!!
2000 -nearly 20 core developers maintaining and extending the language interpreter and its basic functionality

Designed By: **R**obert Gentleman(Left) and **R**oss Ihaka(Right)

Kurt Hornik and Friedrich Leisch established the CRAN archive at TU Vienna as a repository for user contributions.

Few things about R

R is a language and environment for statistical computing and graphics

Rstudio or Rgui can be used to work with R

R is available free of charge and is distributed under the terms of the Free Software Foundation's GNU General Public License

R has very extensive and powerful graphic facilities.

Open Source!!

Currently, the CRAN package repository features 12169 available packages

Case sensitive Functional abstract language

Wide range of Applications. Such as Machine learning, Building Web apps, Sentiment analysis, and much more

Comparative view

R	MATLAB
Designed for Statistics. Hence, provides complete support for Statistics	Incomplete Statistics support
Open Source and Free	Proprietary and Expensive
Massive collection of libraries are available, you can download and install only which are required	It is a complete package. Whether you need it or not it'll always be there for you.
Lightweight IDE	Heavyweight Software
Preferred for Statistics/ Machine learning	Preferred for Engineering Processes

Comparative view

R	PYTHON
" The closer you are to statistics, research and data science, the more you might prefer R "	*" The closer you are to working in an engineering environment, the more you might prefer Python"*
R focuses on better, user friendly data analysis and graphical models	Python emphasizes productivity and code readability
IDE: RStudio	IDE: There are many Python IDEs to choose from. However, Spyder and IPython Notebook are most popular.
Comprehensive R Archive Network(CRAN) is a huge repository of R packages to which users can easily contribute.	PyPi is the Python Package Index: It is a repository of Python software, consisting of libraries. Users can contribute to PyPi.
Cons: R was designed to make data analysis and statistics easier, not to make life easier for your computer.	Cons: Visualization in Python are usually more convoluted, and the results are not nearly as pleasing to the eye or as informative

RGui - Requirements

Windows

- Windows XP and later
- Supports both x86 and x64 architectures
- Created by Robert Gentleman and Guido Masarotto

Mac

- OS X 10.6 (‘Snow Leopard’) and later
- Only 64-bit Intel-based Macs, that is any machine made since mid 2008.
- Created by Stefano Lacus

Linux

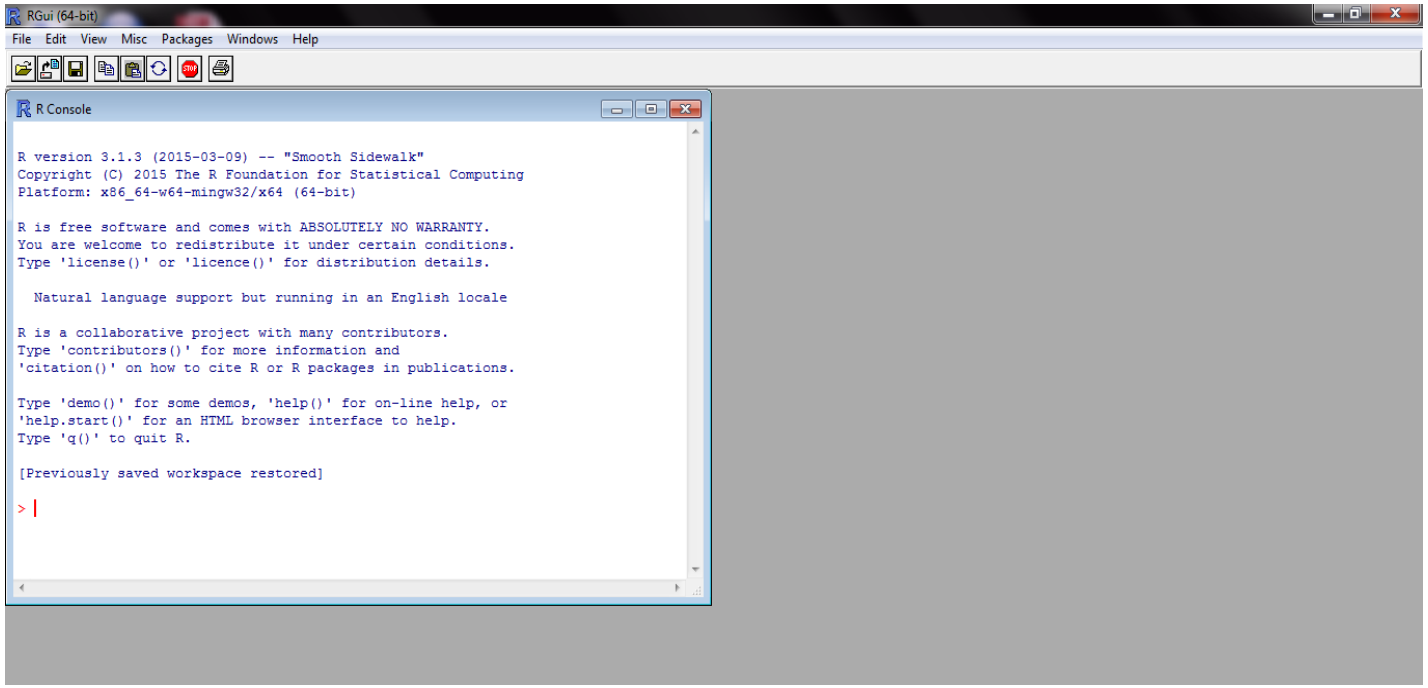
- Ubuntu 10.04.4 (Lucid Lynx) and later
- Intel -i386 and later architecture AMD-amd64
- R(Ubuntu) Provider: Michael Rutter

Unix

- Debian 6.0.10(Squeeze) and later
- Intel -i386 and later architecture AMD-amd64
- R(Debian) provider: Johannes Ranke

RGui

First view of RGui



Very basic R programming IDE by CRAN.

It is a standard Windows GUI executable and provides an R console in its own window

It takes the standard R command-line arguments

Single window or Multiple window mode

Few Commands:

q()- Quits the current session of RGui

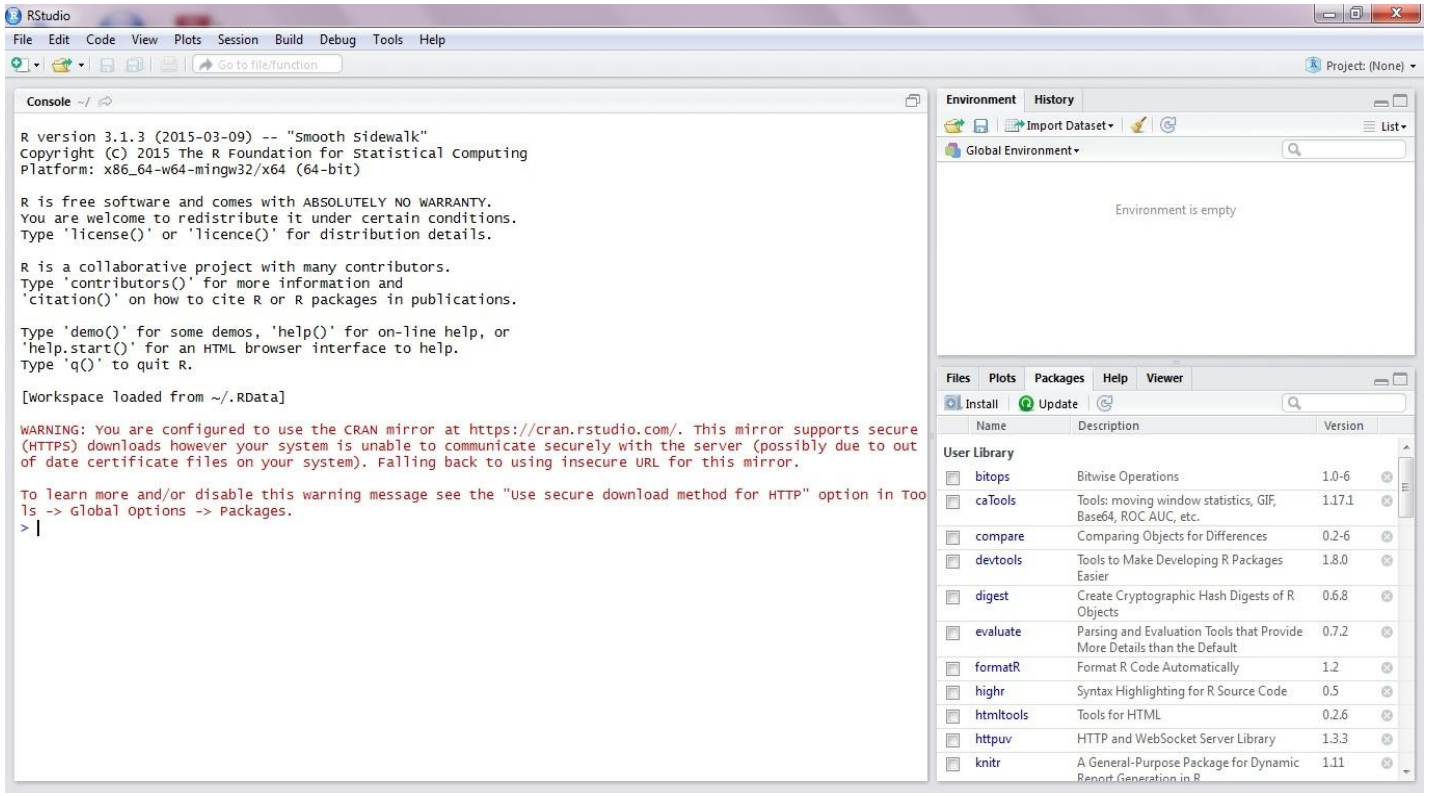
help()- Opens up browser to bring help for particular function or parameters

R.Version() – Check the complete details of current R Gui

objects()/ls() – Prints all the declared variables

RStudio

First view of Rstudio desktop



RStudio is a set of integrated tools designed to work more productive with R

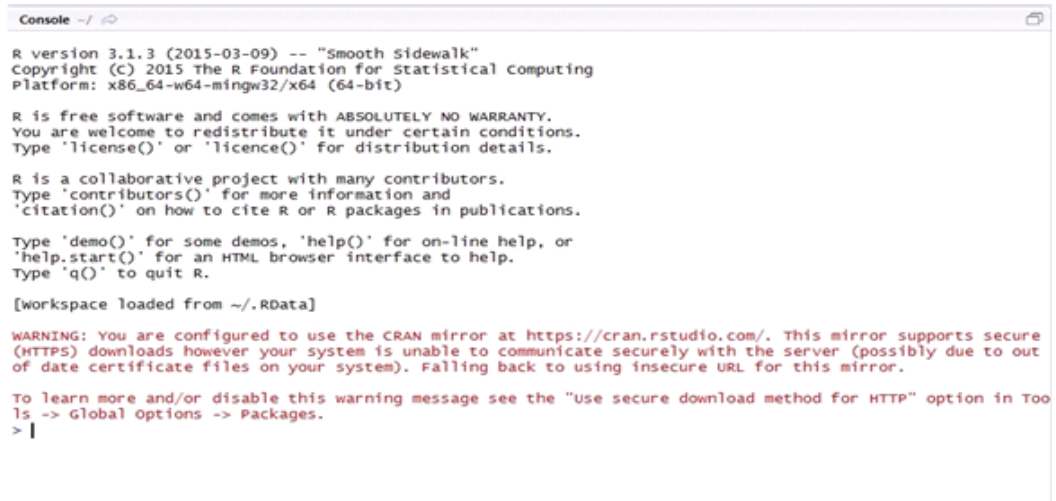
It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

RStudio is available in Desktop and Server editions

Both Open Source and Commercial Versions of Rstudio desktop and Rstudio Server are available

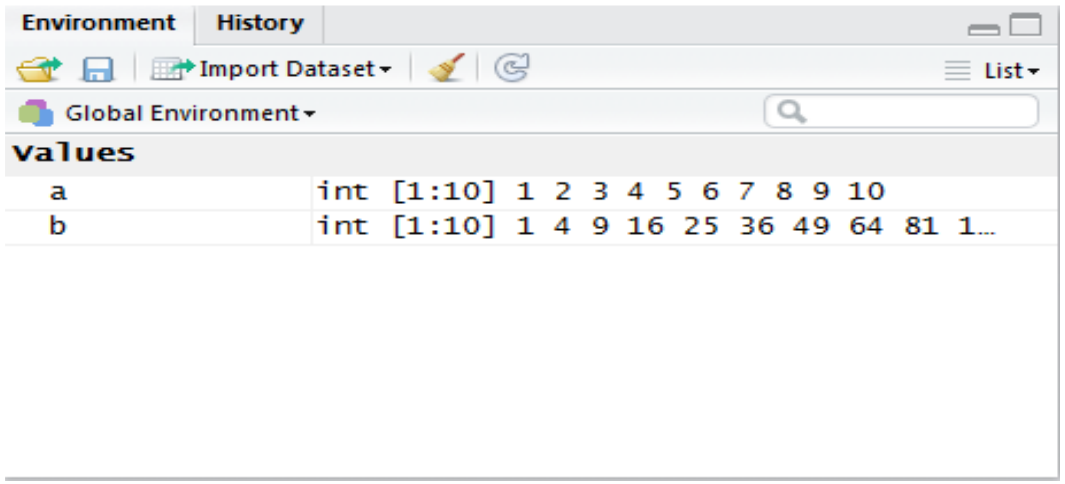
Components of RStudio

Console



Console is a CLI for R language. In console you can execute only one expression/statement at a time. It can be either declaring a variable, function ,calculation etc. . Console in its basic form works as a simple calculator.

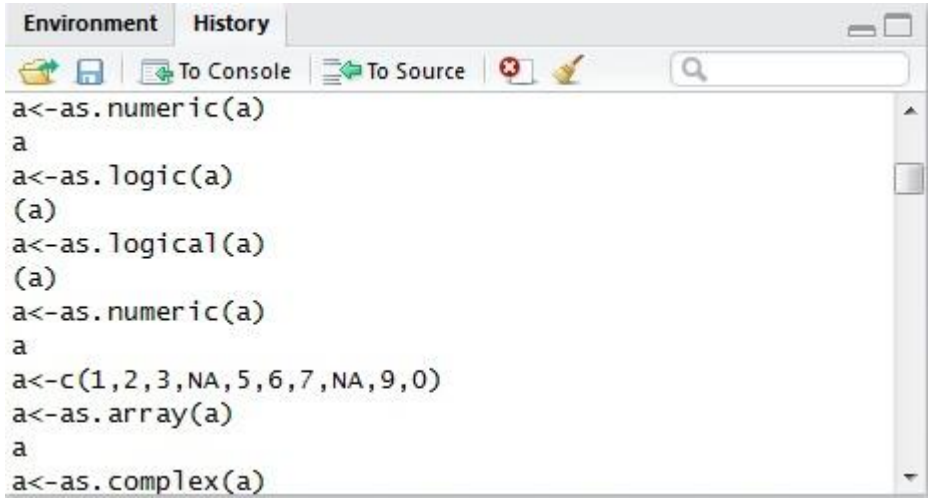
Environment



Environment is a space where you can find all of our variables , functions , datasets etc.. that are declared.

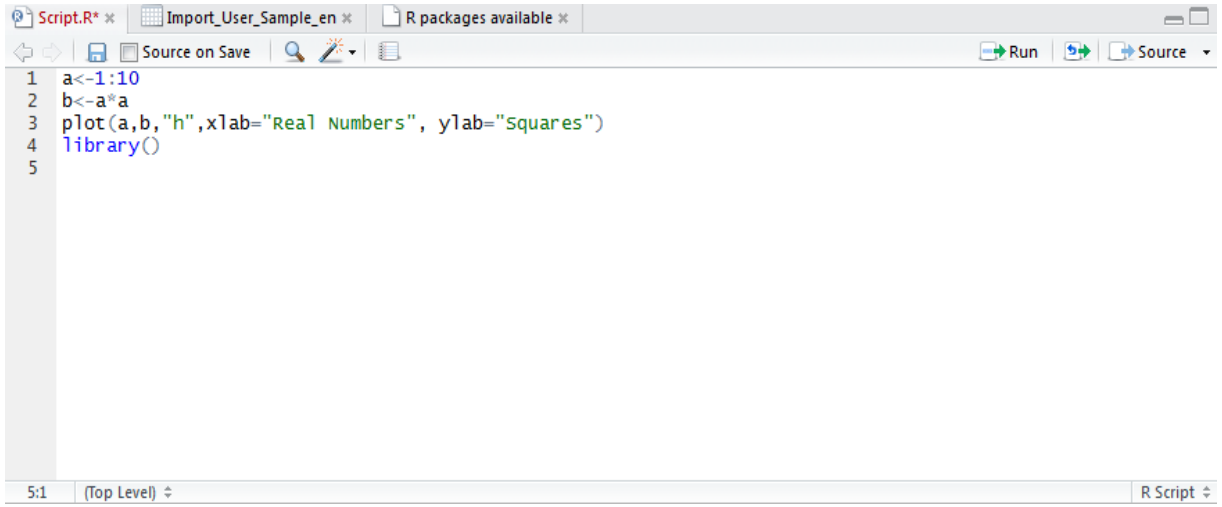
Components of RStudio

History



History contains all the list of previously executed commands. Commands can be directly sent to either script or console from history tab.
History can be saved and retrieved, to and fro the environment. History files are saved in a file format **.Rhistory**

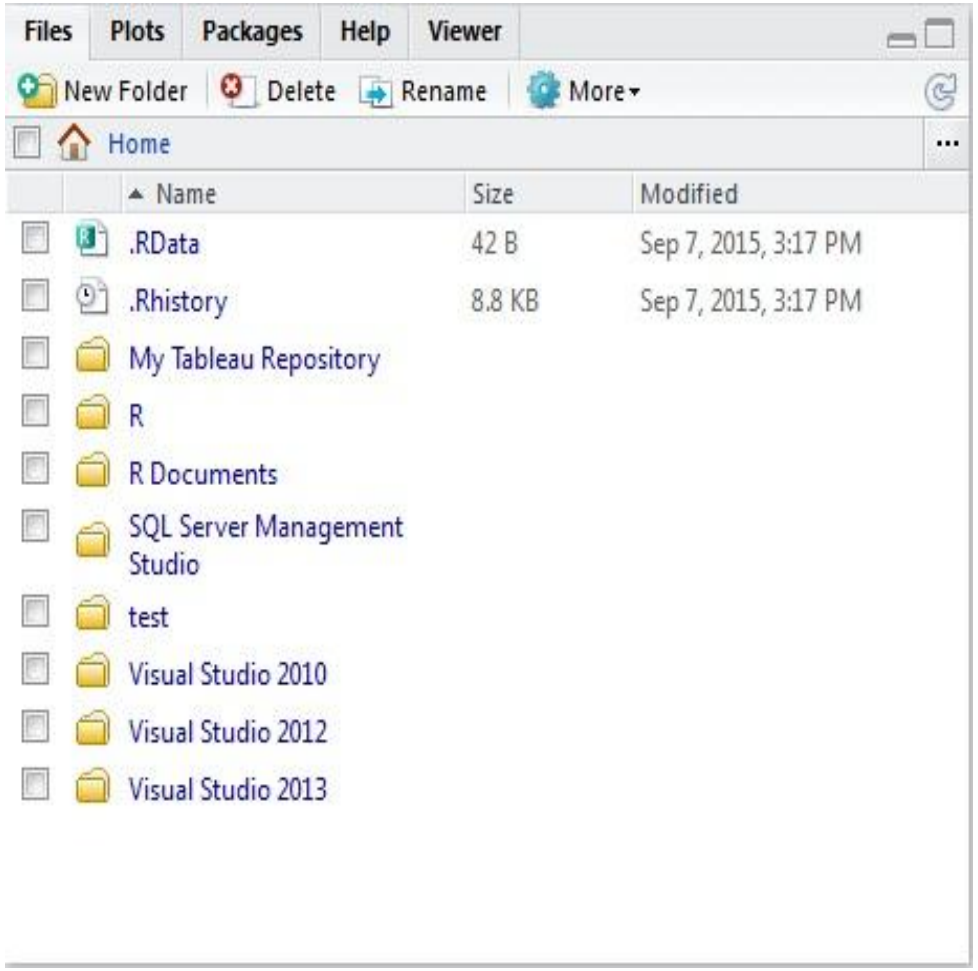
Script Editor



Script window is where multiple lines of commands can be written and executed. Provides text highlighting for user convenience. Script files are saved in **.R** file format.

Components of RStudio

Files



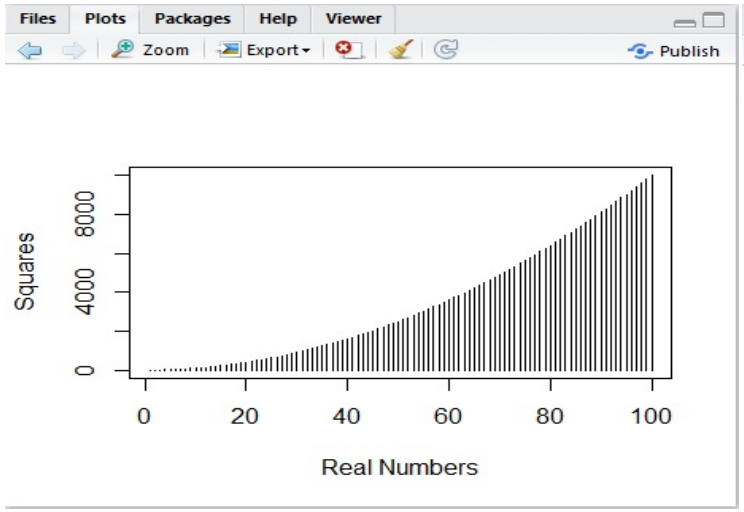
Files tab provides access to local file system from where we can load history , workspace etc..

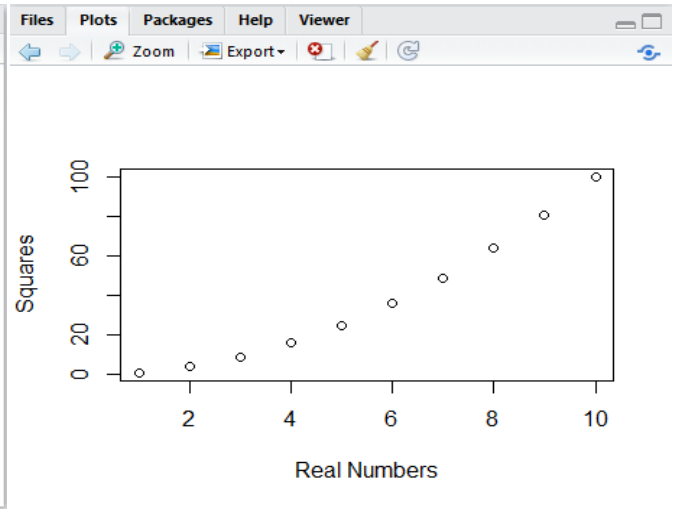
Working directory: Is a folder where all of the data is saved for a particular session.
Set working Directory: setwd("C:/R/Project1/Workspace")
Getting Current working directory: getwd()
Saving Workspace: save.image() **OR**
save(image.file="myfile.RData")
File extension of saved workspace is **.RData**

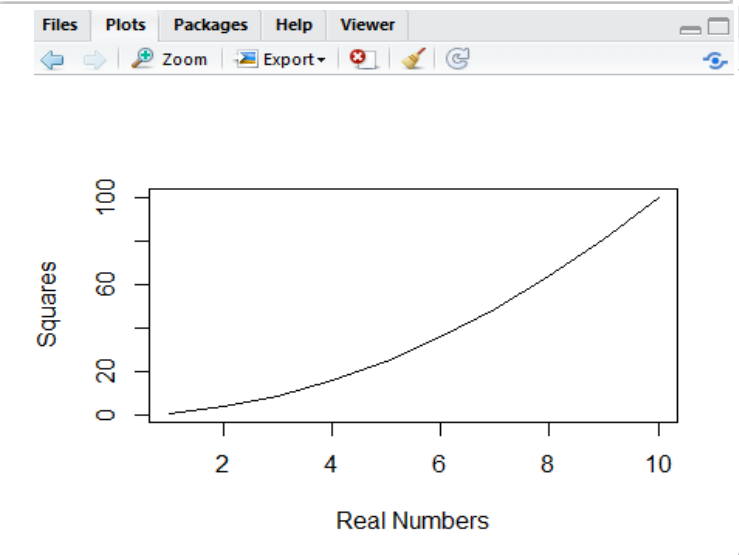
Note: Address path must include forward slash '/' instead of backward slash '\' . Since backward slash is considered as escape character.

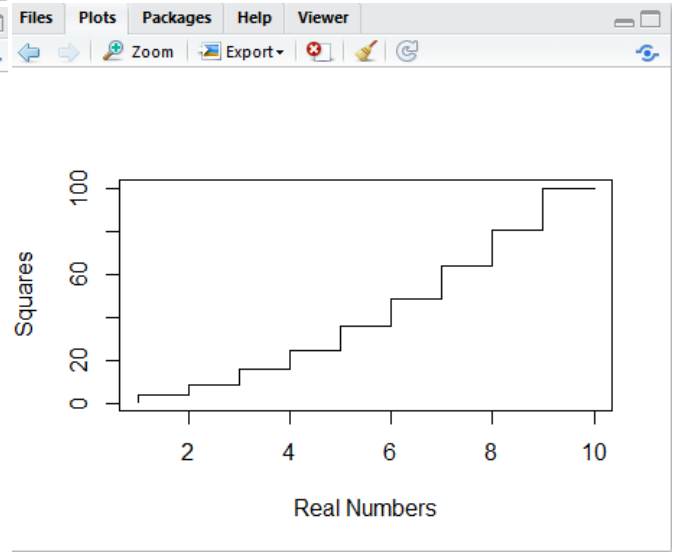
Components of RStudio

Plots









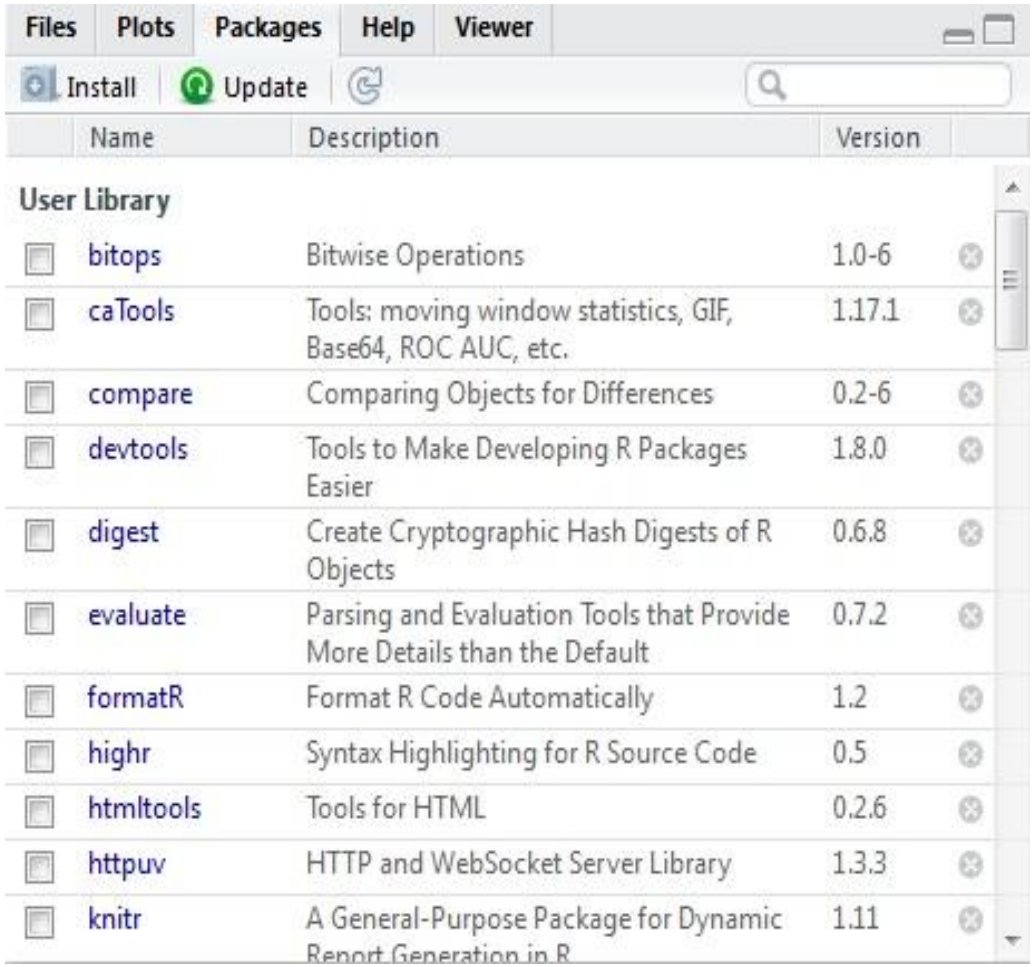
Generic function for plotting of R objects.
Many types of Graphs are available in R. Such as , Histogram ,Dot ,line, staircase ,bar and this list can be even expanded using Graphics libraries.
Plots can also be exported to an image or pdf file.

Plotting Commands:

```
plot(a,b)
plot(a,b,xlab=”A values”,ylab=“B values”)
plot(a,b,”h”,xlab=“A values”,ylab=“B values”)
```

Components of RStudio

Packages



Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library. R comes with a standard set of packages.

Other packages are also available for download and installation. Once installed, they have to be loaded into the session to be used.

Packages can be installed directly from CRAN repository or from zip/tar file available in local memory.

Package related Commands:

Options are also provided for installation of new packages or updating of available packages.

Commands for install/ load/ unload of packages.

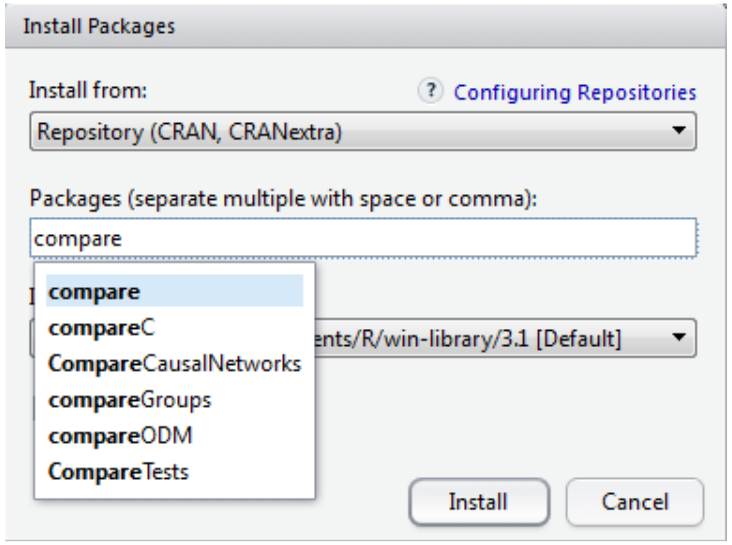
- Install packages: install.packages("compare",getwd(),dependencies=TRUE)
- Loading packages: library("rJython", lib.loc="~/R/win-library/3.1")
- Unloading packages: detach("package:rJython", unload=TRUE)
- Listing installed Packages: library()

Components of RStudio

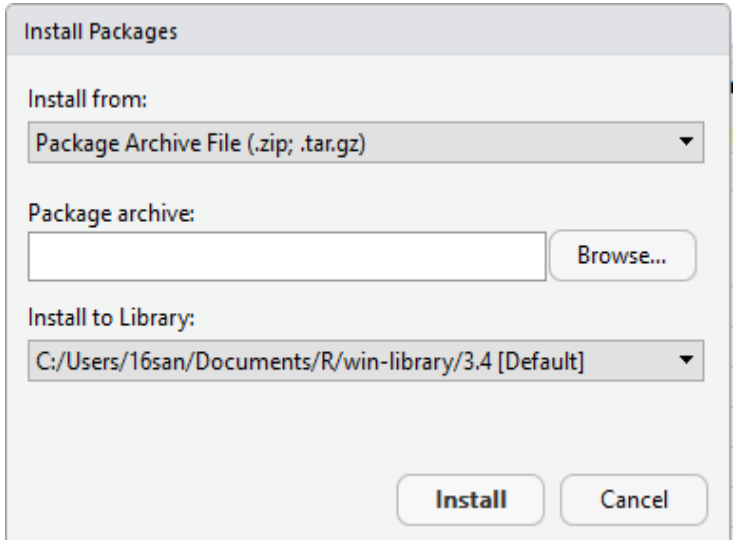
Packages

R packages can be installed either from CRAN Repository or from Local memory(.zip or .tar files).

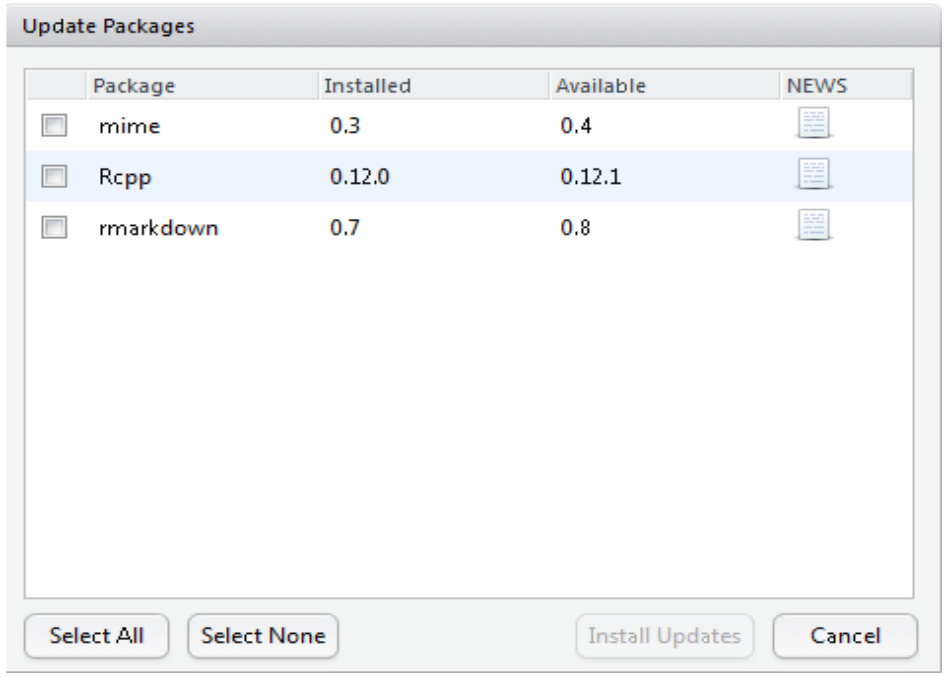
Installing from CRAN Repository



Installing from Local Memory



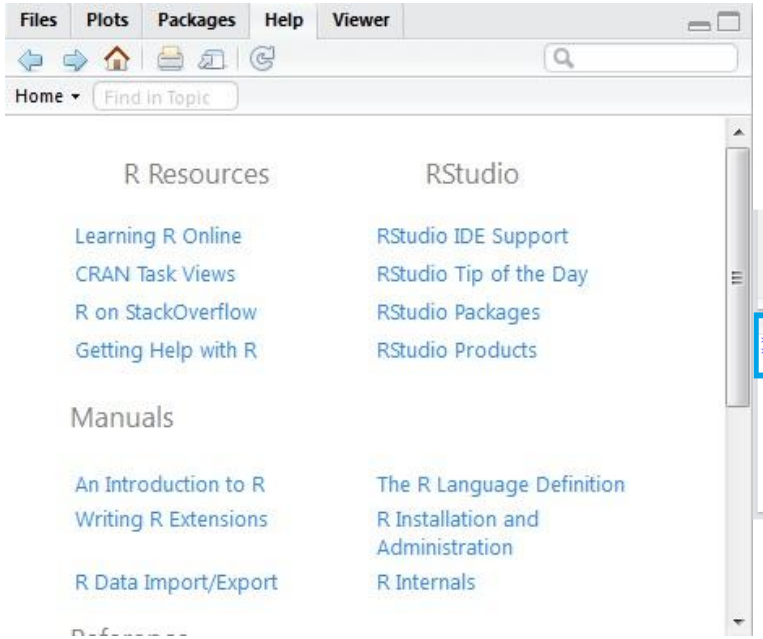
Updating Packages



Components of RStudio

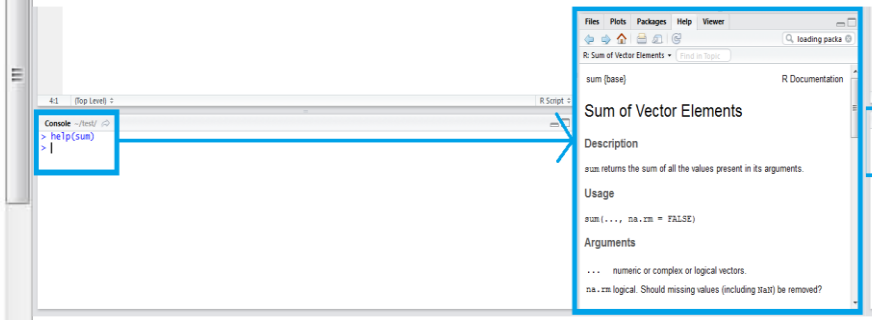
Help

Provides all R related information through RStudio's dedicated help window.

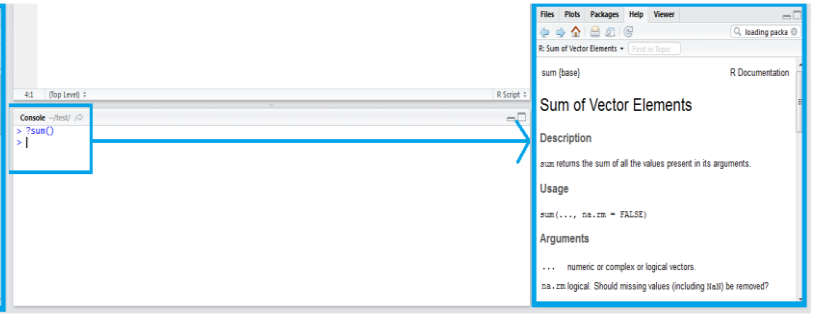


Help Commands:

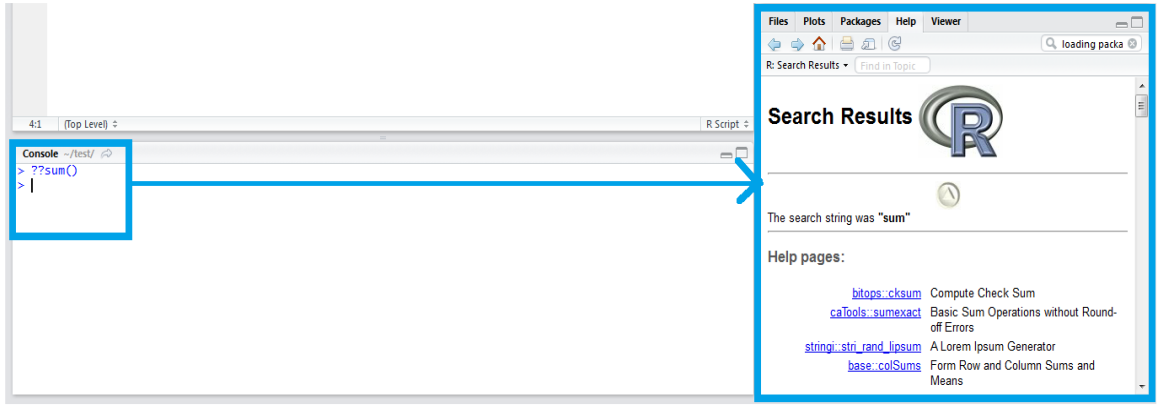
help(name_of_the_function)



?name_of_the_function



??name_of_the_function



Working with R

Assignment

3 ways of assignment

- Using = operator
- Using <- operator
- Using -> operator

Using = operator

```
> x = 1
> y = 2
> z = 3
> a        =        9
> b        =
+ 10
>
```

Using <- operator

```
> x <- 1
> y <- 2
> z <- 3
> a <-     5
> b        <-                  11
>  a < -    3 # Not an assignment
> c <-
+ 12
```

Using -> operator

```
> 1 -> x
> 2 -> y
> 3 -> z
> 9        ->        a
> 10       ->
+ b
> c        - >       12 #Not an
#assignment
```

Working with R

R objects

- Vectors
- Matrices
- Arrays
- Lists
- Tables
- Data Frames

Working with R

R objects - Vectors

Vectors

Sequence of data elements

Atomic data type

Character, Numeric or logical

Creating Vectors

Vectors are generally created using “c()” function.
c()- Combine function
Ex: num<-c(0,1,2,3)
num<- 1:4
num<-NULL # Null vector
num<-c() # Null vector

Naming Vectors

Names can be provided to vector elements using names() function.
Ex: names(num)<-c(‘a’, ’b’, ’c’, ’d’)

Names can also be provided at the time of creating vectors.
Ex: num<-c(one=1,two=2,three=3)
num<-c("one"=1, "two"=2, "three"=3)

Working with R

R objects - Vectors

Coercion of Vectors

Coercion means converting data types of vectors.
There are 2 types of coercion:

- Natural
- Forced

Natural coercion happens while creating a vector.
Ex: num<- c(1,2,'three',4,5)
In this case vector is naturally coerced into string data type.
Forced coercion in made using functions.
Ex: as.numeric(num)
as.logical(num)
as.character(num)

Vector calculus

```
> num1<-c(1,2,3) # num1 and num2 are used for all operations
> num2<-c(4,5,6)
```

Addition:

```
> num1+ num2 # Dimension wise addition
[1] 5 7 9

> num1+1          # Dimension wise addition
[1] 2,3,4

> sum(num1)     #Adds all the elements of num1
[1] 6

> sum(num1,num2)# Adds all the elements of num1 and num2
[1] 21
```

Working with R

R objects - Vectors

Vector calculus

Subtraction:
```
> num2-num1
[1] 3   3   3

> num2-1
[1] 3   4   5
```

Exponentiation:
```
> num1^2
[1] 1   4   9

>num1^num2
[1] 1   32   729
```

Division:
```
> num2/num1
[1] 4.0   2.5   2.0

> num2/2
[1] 2.0   2.5   3.0
```

Multiplication:
```
> num2*num1
[1] 4   10   18

> num2*2
[1] 8   10   12
```

Logical:
```
> num2>num1
[1] TRUE TRUE TRUE
> num2<num1
[1] FALSE FALSE FALSE
>num1==num2
[1] FALSE FALSE FALSE
```

Sub-setting Vector

By Index:
```
num1[1]
a[2]
B[3]
```

By name:
```
num1["a"]
a["one"]
b["1"]
```

By multiple elements:
```
num1[c(1,2)]
a[c(1:100)]
b[c(2,5,7,8)]
```

By multiple elements:
```
num1[c(1,2)]
a[c(1:100)]
b[c(2,5,7,8)]
a[c("one"," two")]
```

By Index - all but some:
```
num1[-1]
a[-c(2,3)]
b[-c(3:10)]
a[-c("one")]
```

By logical:
```
num1[c(T,F,T,F)]
num1[-c(F,T)]
```

Working with R

R objects - Vectors

Vector calculus using functions

Sum of Vectors:
sum(a)

Inner product of vectors:
sum(a*b)

Magnitude of a vector:
sqrt(sum(a*a))

Length of a vector:
length(num1)

Class of vector:
class(a)

Mode of vector:
mode(a)

Working with R

R objects - Matrix

Matrix

2D array of data elements

One atomic data type

Creating a matrix

matrix(values, attributes)

Ex: matA <- matrix(1:9,nrow=3,ncol=3)
matA <-matrix(1:9,nrow=3,ncol=3,byrow=T)
matA <-matrix(c(1,3,2,4),nrow=2,ncol=2)

Recycling in a Matrix

```
> matrix(1:3, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3
```

```
> matrix(1:4, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    1
[2,]    2    4    2

Warning message:
In matrix(1:4, nrow = 2, ncol = 3)
data length [4] is not a sub-multiple or multiple
of the number of columns [3]
```

Working with R

R objects - Matrix

cbind & rbind functions

```
> matA<-cbind(1:3, 1:3)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
> cbind(matA,2:4)
     [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    2    3
[3,]    3    3    4
```

```
> matA<-rbind(1:3, 1:3)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
> rbind(matA,2:4)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
[3,]    2    3    4
```

Working with R

R objects - Matrix

Sub-setting a Matrix

```
> m <- matrix(1:9, nrow = 3)
> m[1,3]                          #particular element
> m[3,2]
> m[,1]
> m[1,]
> m[2]
```

```
> m[c(TRUE,FALSE), ]                    # by logical
> m[c(FALSE,FALSE,TRUE),c(TRUE,TRUE,FALSE)]
> m[FALSE,TRUE]   #Column names
> m[TRUE,FALSE]   #Row names
> m[F, c(T,F)]          #Selected Column names
```

```
> m[2, c(2, 3)]                   # multiple selections
> m[c(1, 2), c(2, 3)]
> m[c(1, 3), c(1, 2)]
```

```
> m[“r1”,”c3” ]                   # by name
> m[“r2”, ]
> m[c(“r2”,”r3”),c(“c2”)]
```

Q & A

Thank you