

## **Introduction to R Programming Language**



- **Open source programming language**
- **Modeled after S & S-plus developed at AT&T labs in late 1980s**
- **R project was started by Robert Gentleman and Ross Ihaka Dept.  
of Statistics, University of Auckland – 1995**
- **Currently maintained by R core development team –  
International team of volunteer developers (since 1997)**

## R URLs

- [www.cran.r-project.org/](http://www.cran.r-project.org/)
- [www.r-project.org](http://www.r-project.org)
- CRAN: [The Comprehensive R Archive Network](http://www.cran.r-project.org/)
- Install – R from [www.cran.r-project.org/](http://www.cran.r-project.org/)
- Linux: `sudo apt-get install r-base-core`


The Comprehensive R Archive Network - Mozilla Firefox

File Edit View History Bookmarks Tools Help

The Comprehensive R Archive Network

www.cran.r-project.org

Google



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

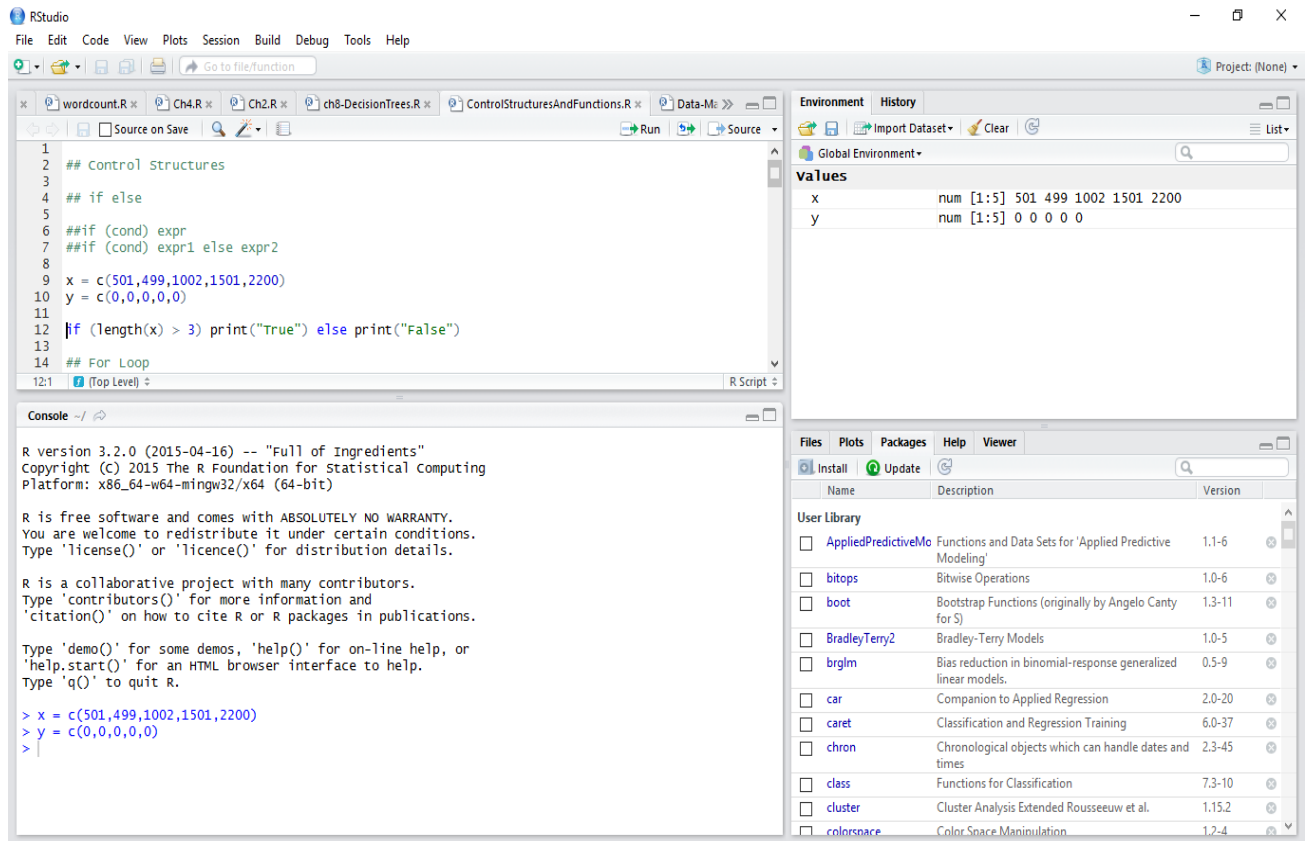
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2015-03-09, Smooth Sidewalk) [R-3.1.3 tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

- Install R ([www.cran.r-project.org](http://www.cran.r-project.org))
- Then install RStudio ([www.rstudio.com](http://www.rstudio.com))
- RStudio interface below



## Getting help from R console

- `help.start()`
- `help(topic)`
- `?topic`
- `??topic`

## How to use R for simple math

```
> 3 + 7
```

```
> 12 + 3 / 4 - 5 + 3*8
```

```
> (12 + 3 / 4 - 5) + 3*8
```

```
> pi * 2^3 - sqrt(4)
```

```
> factorial(4)
```

```
> log(2,10)
```

```
> log(2, base=10)
```

```
> log10(2)
```

```
> log(2)
```

## How to store results of calculations for future use

```
> x = 3+5
```

```
> x
```

```
> y = 12 + 3 / 4 - 5 + 3*8
```

```
> y
```

```
> z = (12 + 3 / 4 - 5) + 3*8
```

```
> z
```

```
> A < - 6 + 8 ## no space between < & -
```

```
> a          ## R is case sensitive
```

```
>A
```

## Identifiers: (Google's R Style Guide)

(<https://google-styleguide.googlecode.com/svn/trunk/Rguide.xml#identifiers>)

- Don't use underscores ( \_ ) or hyphens ( - ) in identifiers.
  - The preferred form for variable names is all lower case letters and words separated with dots (`variable.name`), but `variableName` is also accepted;
  - function names have initial capital letters and no dots (`FunctionName`); constants are named like functions but with an initial k.
- `variable.name` is preferred, `variableName` is accepted  
GOOD: `avg.clicks`  
OK: `avgClicks`  
BAD: `avg_Clicks`



## Using C command (Combine or concatenation) for making data

```
> data1 = c(3,6,9,12,78,34, 5, 7, 7) ## numerical data
```

```
> data1.text = c('Mon', 'Tue', "Wed") ## Text data;
```

## Single or double quote both ok

##copy/paste into R console may not work

```
> data1.text = c(data1.text, 'Thu', 'Fri')
```

## Using scan command for making data

```
> data3 = scan() ## data separated by Space / Press Enter twice to exit
```

```
1: 4 5 7 8
```

```
5: 2 9 4
```

```
8: 3
```

```
9:
```

```
Read 8 items
```

```
> data3
```

```
[1] 4 5 7 8 2 9 4 3
```

```
> d3 = scan(what = 'character')
```

```
1: mon
```

```
2: tue
```

```
3: wed thu
```

```
5:
```

```
Read 4 items
```

```
> d3
```

```
[1] "mon" "tue" "wed" "thu"
```

```
> d3[2]
```

```
[1] "tue"
```

```
> d3[2]='mon'
```

```
> d3
```

```
[1] "mon" "mon" "wed" "thu"
```

```
> d3[6]='sat'
```

```
> d3
```

```
[1] "mon" "mon" "wed" "thu" NA "sat"
```

```
> d3[2]='tue'
```

```
> d3[5] = 'fri'
```

```
> d3
```

```
[1] "mon" "tue" "wed" "thu" "fri" "sat"
```

## Concept of Working Directory

```
>getwd()
```

```
[1] "C:/Users/nkonar/Documents"
```

```
> setwd('D:/R-SW')
```

```
> dir() ## working directory listing
```

```
>ls() ## Workspace listing of objects
```

```
>rm('object')
```

```
> rm(list = ls()) ## Spring Cleaning
```

## Reading data from File

Set the working directory to file location

```
>setwd('D:/R-SW/WD')
```

```
> getwd()
```

```
[1] "D:/R-SW/WD"
```

```
> dir()
```

```
[1] "Arv.txt"      "DiningAtSFO"  "LatentView-DPL" "Negetive-words.txt"  
"Positive-words.txt" "TC-10-Rec.csv" "TC.csv"
```

```
>tcddata = read.csv('TC-10-Rec.csv', header = T, sep = ',')
```

```
(tcddata = read.table('TC-10-Rec.csv', header = T, sep = ','))
```

```
> ls()
```

```
[1] "tcddata"
```

```
> str(tcddata)
```

'data.frame': 10 obs. of 9 variables:

```
$ Id      : int 1 2 3 4 5 6 7 8 9 10
```

Other variables;

## How to access individual data elements / variables

**> tcdata\$Price ## Price is a variable in the data**

```
[1] 13500 13750 13950 14950 13750 12950 16900 18600 21500 12950
```

**> tcdata\$Price[7] = 20000**

**> tcdata\$Price**

```
[1] 13500 13750 13950 14950 13750 12950 20000 18600 21500 12950
```

**This change has happened in workspace only not in the file.**

**How to make it permanent?**

**write.csv / write.table**

**>write.table(tcdata, file ='TC-10-Rec.csv', row.names = FALSE, sep = ',')**

**If row.names is TRUE, R adds one ID column in the beginning of file.**

**So its suggested to use row.names = FALSE option**

**>write.csv(tcdata, file ='TC-10-Rec-new5.csv', row.names = TRUE) ## to test**

## Types of Data Item:

- Vector
- Matrix
- Data Frame
- List

## Vectors:

```
>x=c(1,2,3,4,56)
```

```
>x
```

```
> x[2]
```

```
> x = c(3, 4, NA, 5)
```

```
>mean(x)
```

```
[1] NA
```

```
>mean(x, rm.NA=T)
```

```
[1] 4
```

```
> x = c(3, 4, NULL, 5)
```

```
>mean(x)
```

```
[1] 4
```

## Vectors:

```
>y = c(x,c(-1,5),x)
```

```
>length(x)
```

```
>length(y)
```

There are useful methods to create long vectors whose elements are in arithmetic progression:

```
> x=1:20
```

```
> x
```

Try min(), max(), mean(), range(), sum(), summary(), sd() functions

→ If the common difference is not 1 or -1 then we can use the seq function

```
> y=seq(2,5,0.3)
```

```
> y
```

```
[1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0
```

```
> length(y)
```

```
[1] 11
```

## Vectors:

```
> x=1:5
```

```
> mean(x)
```

```
[1] 3
```

```
> x
```

```
[1] 1 2 3 4 5
```

```
> x^2
```

```
[1] 1 4 9 16 25
```

```
> x+1
```

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

```
> 2*x
```

```
[1] 2 4 6 8 10
```

```
> exp(sqrt(x))
```

```
[1] 2.718282 4.113250 5.652234 7.389056 9.356469
```

It is very easy to add/subtract/multiply/divide two vectors entry by entry.

```
> y=c(0,3,4,0)
```

```
> x+y
```

```
[1] 1 5 7 4 5
```

Warning message:



In  $x + y$  : longer object length is not a multiple of shorter object length

```
> y=c(0,3,4,0,9)
```

```
> x+y
```

```
[1] 1 5 7 4 14
```

```
> x=1:6
```

```
> y=c(9,8)
```

```
> x+y
```

```
[1] 10 10 12 12 14 14
```

## Matrices:

Same data type/mode – number , character, logical

```
a.matrix <- matrix(vector, nrow = r, ncol = c, byrow = FALSE, dimnames = list(char-  
vector-rownames, char-vector-col-names))
```

## dimnames is optional argument, provides labels for rows & columns.

```
> y <- matrix(1:20, nrow = 4, ncol = 5)
```

```
>A = matrix(c(1,2,3,4),nrow=2,byrow=T)
```

```
>A
```

```
>A = matrix(c(1,2,3,4),ncol=2)
```

```
>B = matrix(2:7,nrow=2)
```

```
>C = matrix(5:2,ncol=2)
```

```
>mr <- matrix(1:20, nrow = 5, ncol = 4, byrow = T)
```

```
>mc <- matrix(1:20, nrow = 5, ncol = 4)
```

```
>mr
```

```
>mc
```

```
>dim(B) #dimension
```

```
>nrow(B)
```

**>ncol(B)**

**>A+C**

**>A-C**

**>A%%C   #matrix multiplication**

**>A\*C    #entrywise multiplication**

**>t(A)   ## Transpose**

**>A[1,2]**

**>A[1,]**

**>B[1,c(2,3)]**

**>B[, -1]**

**Matrix Algebra: <http://www.statmethods.net/advstats/matrix.html>**

## Lists

- Vectors and matrices in R are two ways to work with a collection of objects.
- Lists provide a third method. Unlike a vector or a matrix a list can hold different kinds of objects.
- One entry in a list may be a number, while the next is a matrix, while a third is a character string (like "Hello R!").
- statistical functions of R usually return the result in the form of lists. So we must know how to unpack a list using the \$ symbol

```
>x = list(name="Arun Patel", nationality="Indian", height=5.5,  
grades=c(95,45,80))
```

```
>names(x)
```

```
>x$name
```

```
>x$hei #abbrevs are OK
```

```
>x$grades
```

```
>x$g[2]
```

## Data Frame:

- A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).
- This is similar to SAS and SPSS datasets.

```
>d <- c(1,2,3,4)
```

```
>e <- c("red", "white", "red", NA)
```

```
>f <- c(TRUE,TRUE,TRUE,FALSE)
```

```
>myframe <- data.frame(d,e,f)
```

```
>names(myframe) <- c("ID","Color","Passed") # variable names
```

```
>myframe
```

```
>myframe[1:3,] # rows 1 , 2, 3 of data frame
```

```
>myframe[,1:2] # col 1, 2 of data frame
```

```
>myframe[c("ID","Color")] # columns ID and color from data frame
```

```
>myframe$ID # variable ID in the data frame
```

## Data Structures:

### List and Data Frame:

- Lists are by far the most flexible data structure in R.
- They can be seen as a collection of elements without any restriction on the class, length or structure of each element.
- The only thing you need to take care of, is that you don't give two elements the same name. That might cause a lot of confusion, and R doesn't give errors for that:

```
> X <- list(a=1,b=2,a=3)
> X$a
[1] 1
```

Data frames are lists as well, but they have a few restrictions:

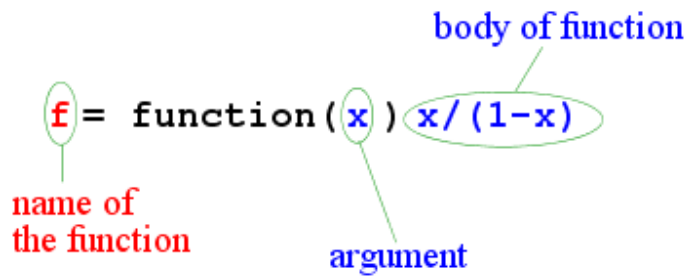
- you can't use the same name for two different variables
- all elements of a data frame are vectors
- all elements of a data frame have an equal length.
- Due to these restrictions and the resulting two-dimensional structure, data frames can mimic some of the behaviour of matrices.
- You can select rows and do operations on rows. You can't do that with lists, as a row is undefined there.
- All this implies that you should use a data frame for any dataset that fits in that two dimensional structure.
- Essentially, you use data frames for any dataset where a column coincides with a variable and a row coincides with a single observation in the broad sense of the word.
- For all other structures, lists are the way to go.
- Note that if you want a nested structure, you have to use lists. As elements of a list can be lists themselves, you can create very flexible structured objects.

## Factor:

- Tell R that a variable is nominal by making it a factor.
- The factor stores the nominal values as a vector of integers in the range [ 1... k ] (where k is the number of unique values in the nominal variable),
- An internal vector of character strings (the original values) mapped to these integers.

```
# variable gender with 20 "male" entries and  
# 30 "female" entries  
>gender <- c(rep("male",20), rep("female", 30))  
>gender <- factor(gender)  
# stores gender as 20 1s and 30 2s and associates  
# 1=female, 2=male internally (alphabetically)  
# R now treats gender as a nominal variable  
>summary(gender)
```

## Function:

  
name of the function      argument      body of function

```
>g = function(x,y) (x+2*y)/3
```

```
>g(1,2)
```

```
>g(2,1)
```