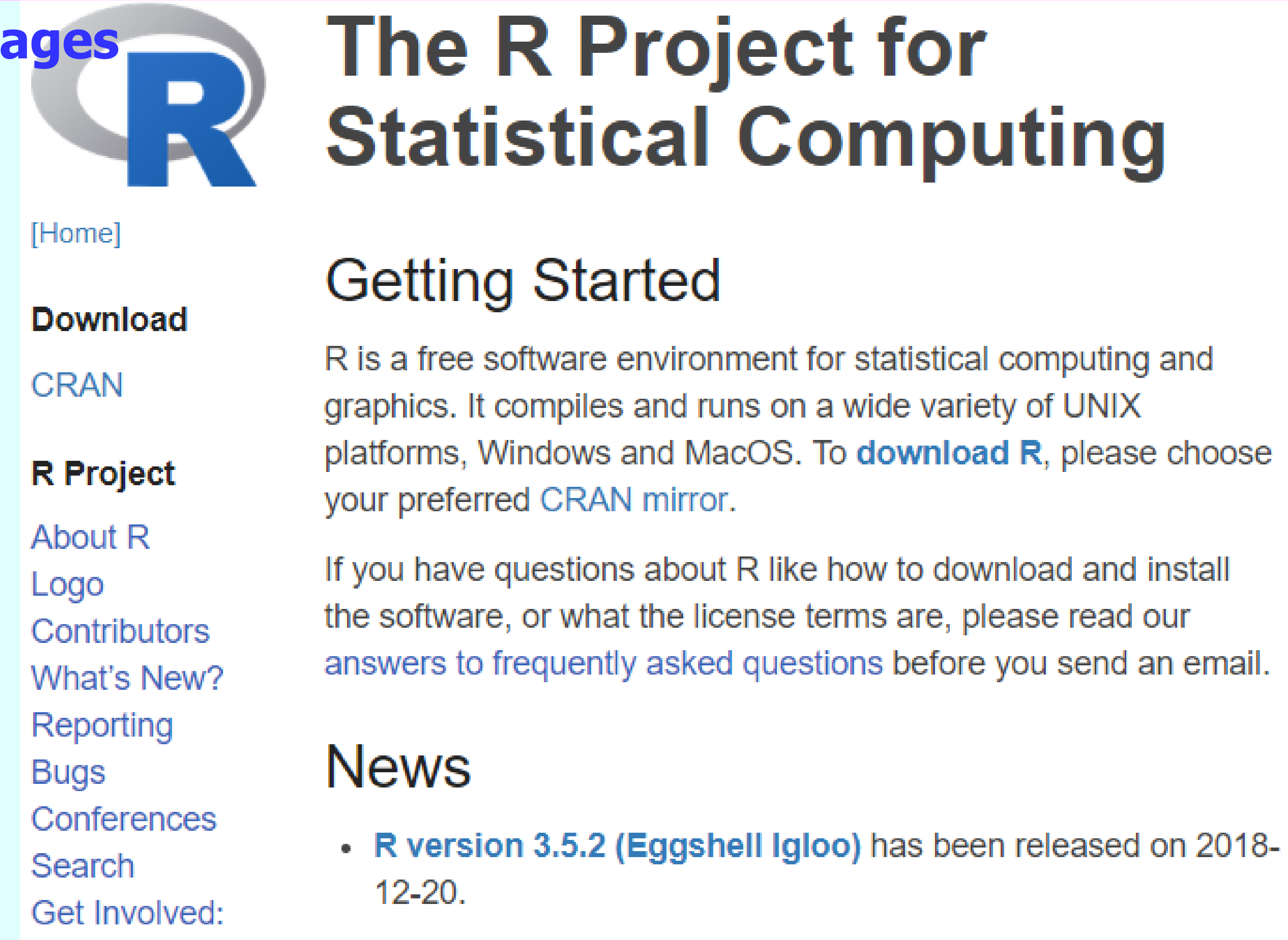
R Basics

1. R and RStudio Packages
2. The Use of R
3. R Objects
4. R Data Structures
5. Factors

1. R and RStudio Packages

- **R Packages**



The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

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.

News

- **R version 3.5.2 (Eggshell Igloo)** has been released on 2018-12-20.

R version 3.5.2 (Eggshell Igloo) has been released on 2018-12-20.

● RStudio

Open source and enterprise-ready professional software for R

https://www.rstudio.com/

● R Resources

1. **An Introduction to R** gives an introduction to the language and how to use R for doing statistical analysis and graphics.

2. **R Data Import/Export** describes the import and export facilities available either in R itself or via packages.

3. **Writing R Extensions** covers how to create your own packages, write R help files, and the foreign language (C, C++, Fortran, ...) interfaces.

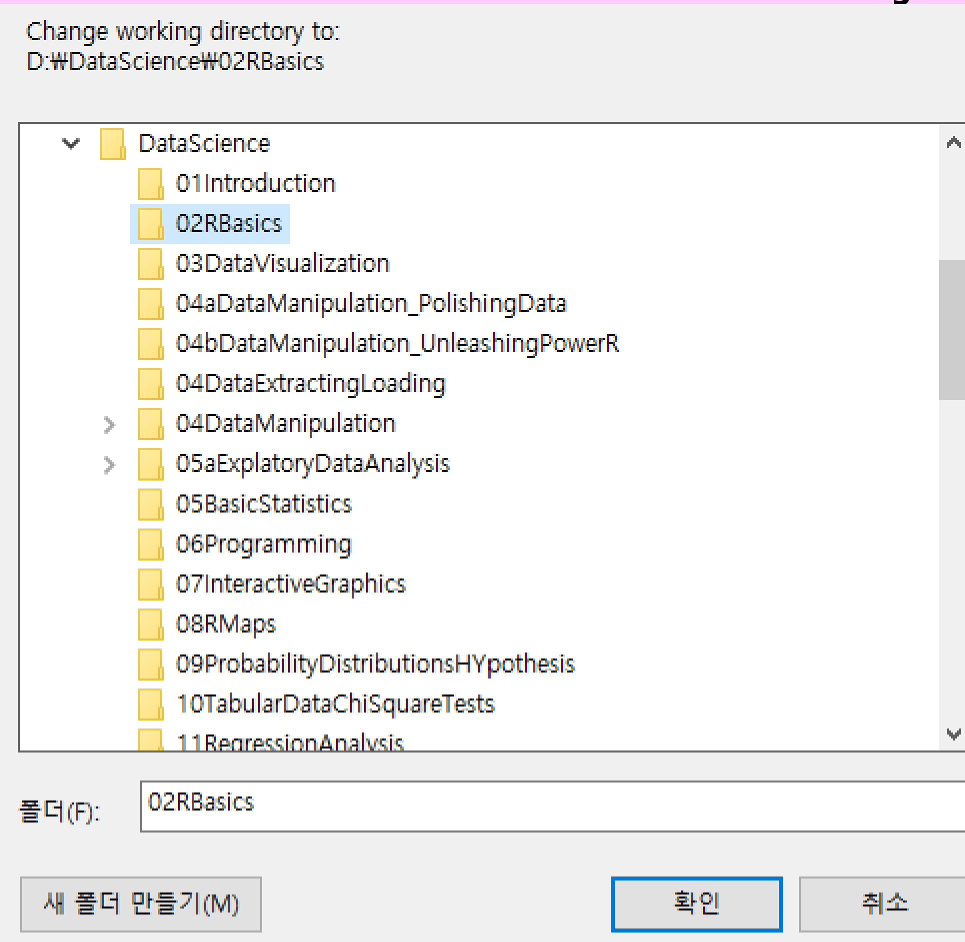
4. **R Internals**: a guide to the internal structures of R and coding standards for the core team working on R itself.

● Using the R console

(1) Set up Working Directory

From the pull-down menu of the R Console, you can set it to your current directory in R:

File > Change dir...
(파일 > 디렉토리 변경…)



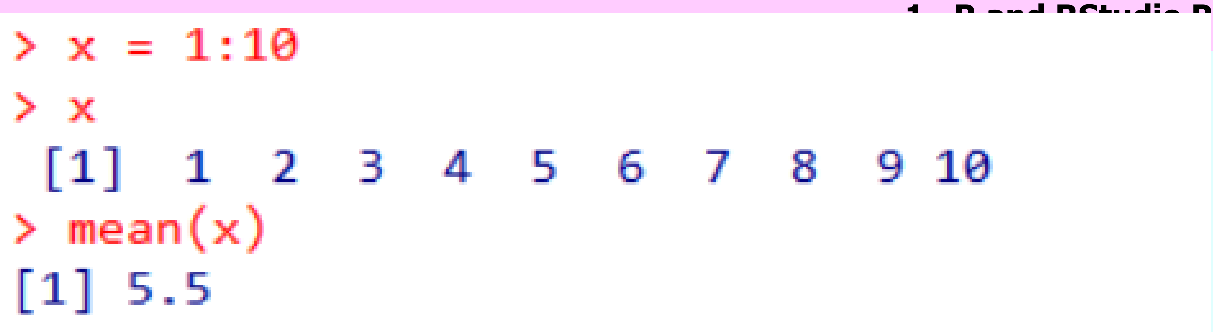
Change working directory to:
D:₩DataScience₩02RBasics

폴더(F): 02RBasics

Or, from the command line of your R console, type

```
> setwd("D:/DataScience/02RBasics")
> # Make sure where you are
> getwd()
[1] "D:/DataScience/02RBasics"
```

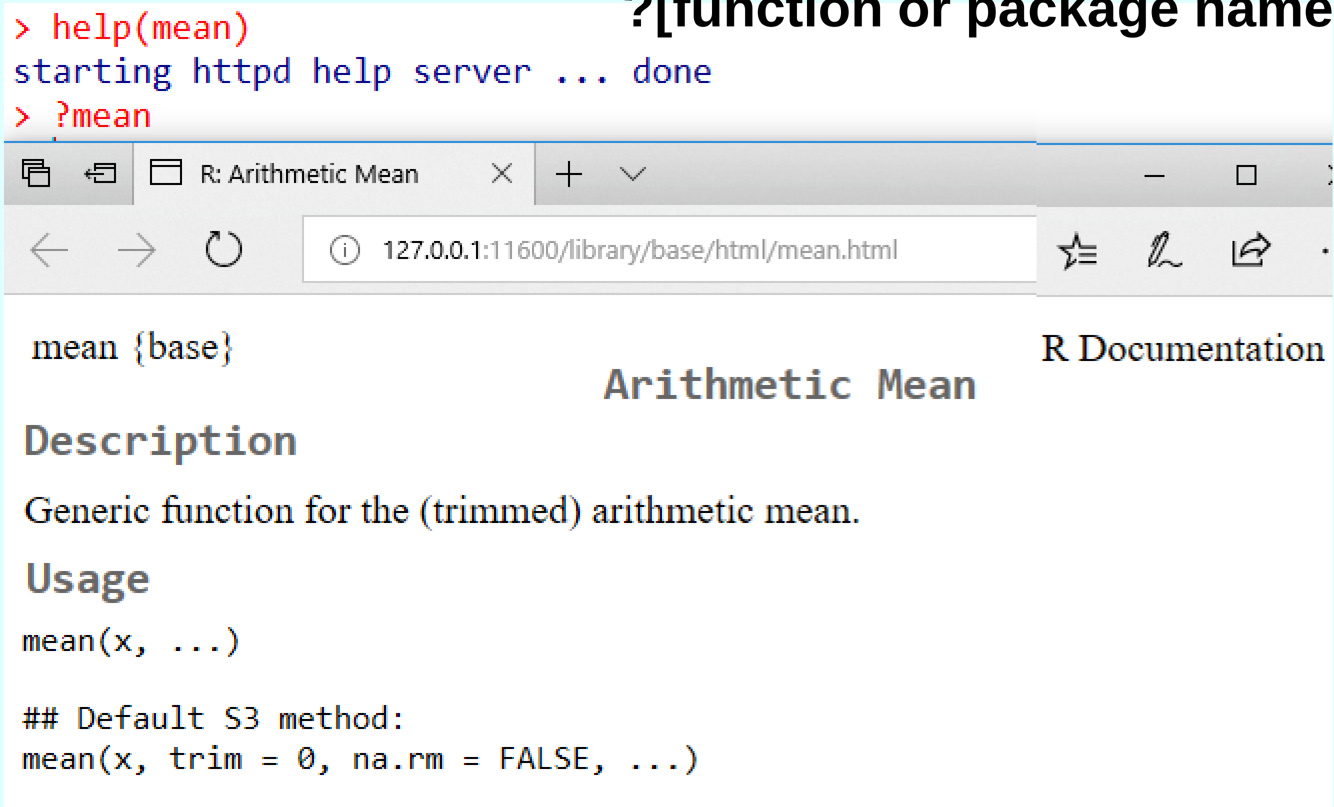
(2) Sample session



```
> x = 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> mean(x)
[1] 5.5
```

(3) Getting help

- You can access some quick documentation for function/package from R with the **help(package="<package name>")**
- You can also access help type **?[function or package name]** in the console.



```
> help(mean)
starting httpd help server ... done
> ?mean
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

● Using RStudio

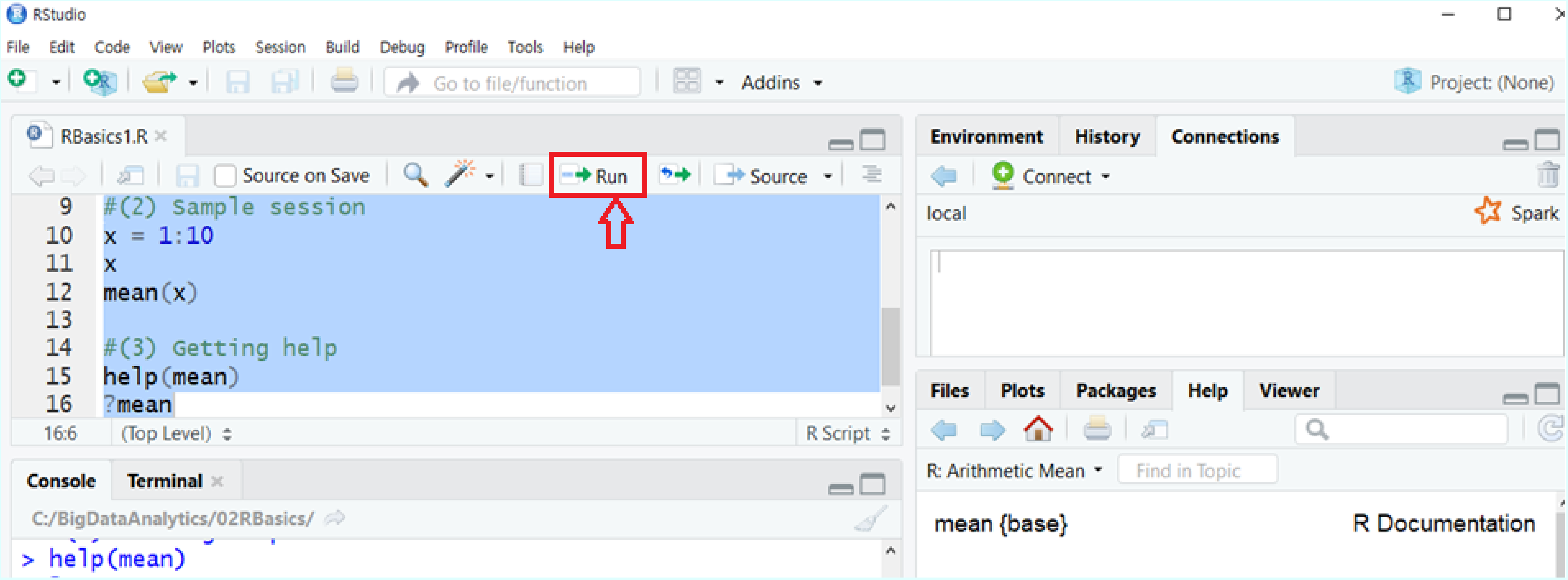
(1) Creating a R script file

File > New File > R Script

(2) Using a R script file

File > Open File >

(3) Running a R script file



```
#(2) Sample session
x = 1:10
x
mean(x)

#(3) Getting help
help(mean)
?mean
```

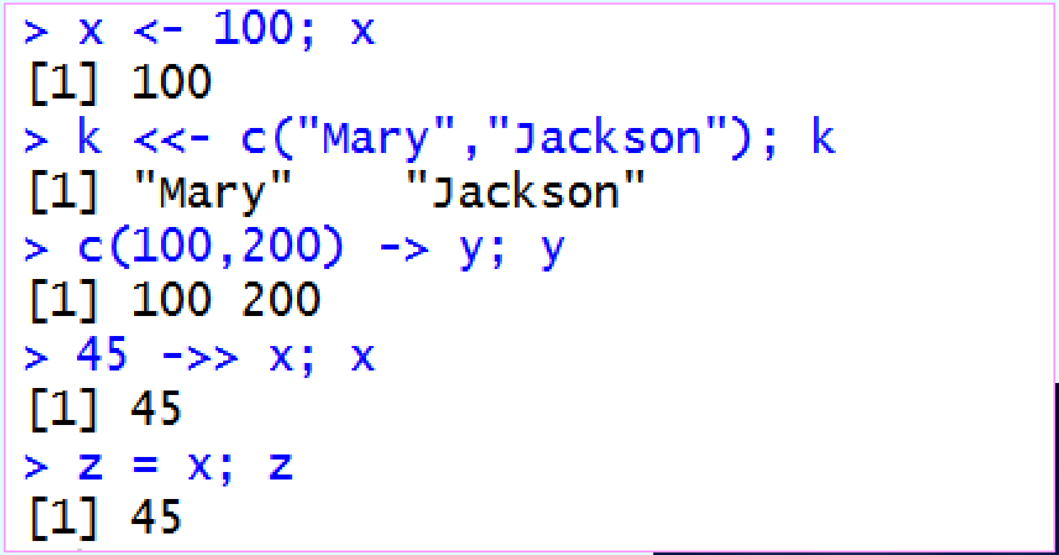
```
C:/BigDataAnalytics/02RBasics/
> help(mean)
```

mean {base}   R Documentation

2. The Use of R

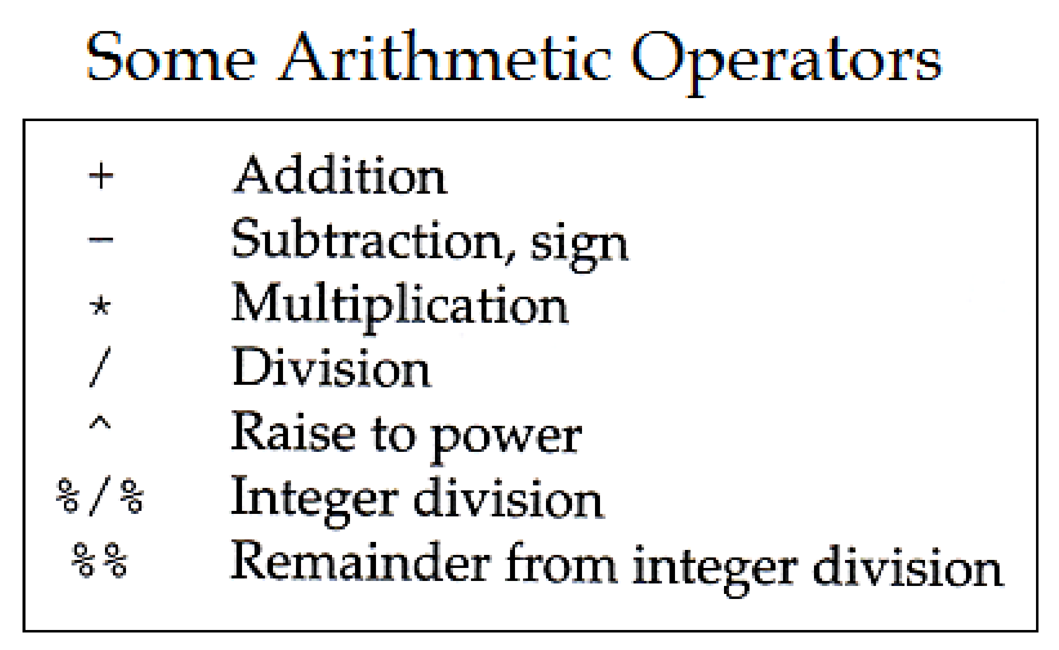
(1) Entering Input

R creates objects to represent values using assignment operators (<-, <<-, ->, ->>, =).



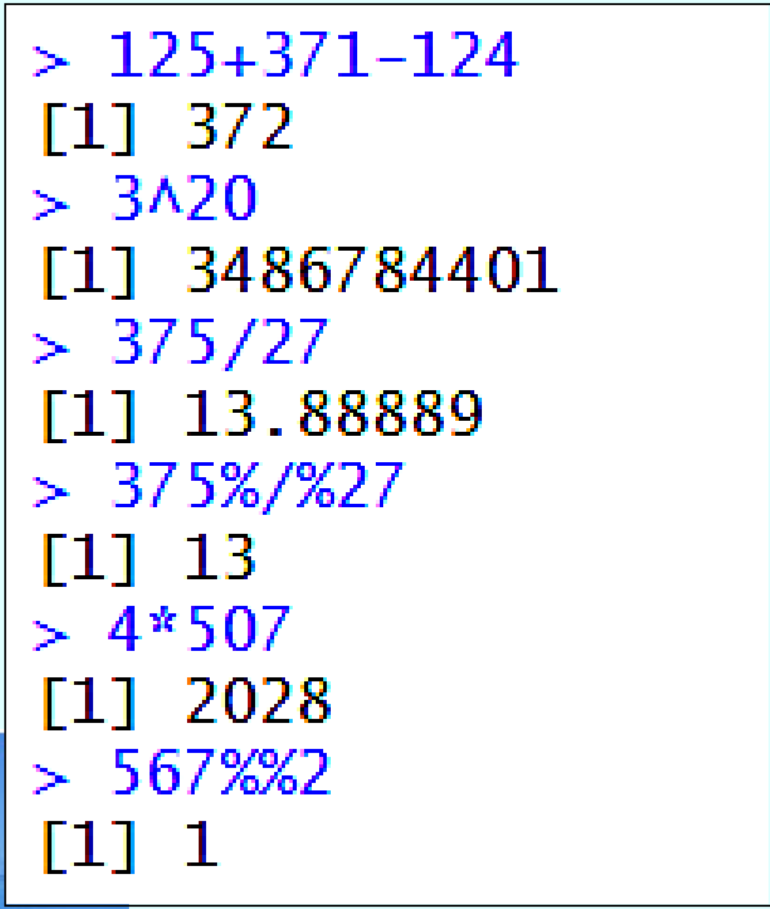
```
> x <- 100; x
[1] 100
> k <<- c("Mary","Jackson"); k
[1] "Mary"    "Jackson"
> c(100,200) -> y; y
[1] 100 200
> 45 ->> x; x
[1] 45
> z = x; z
[1] 45
```

(2) R may be used as a Calculator

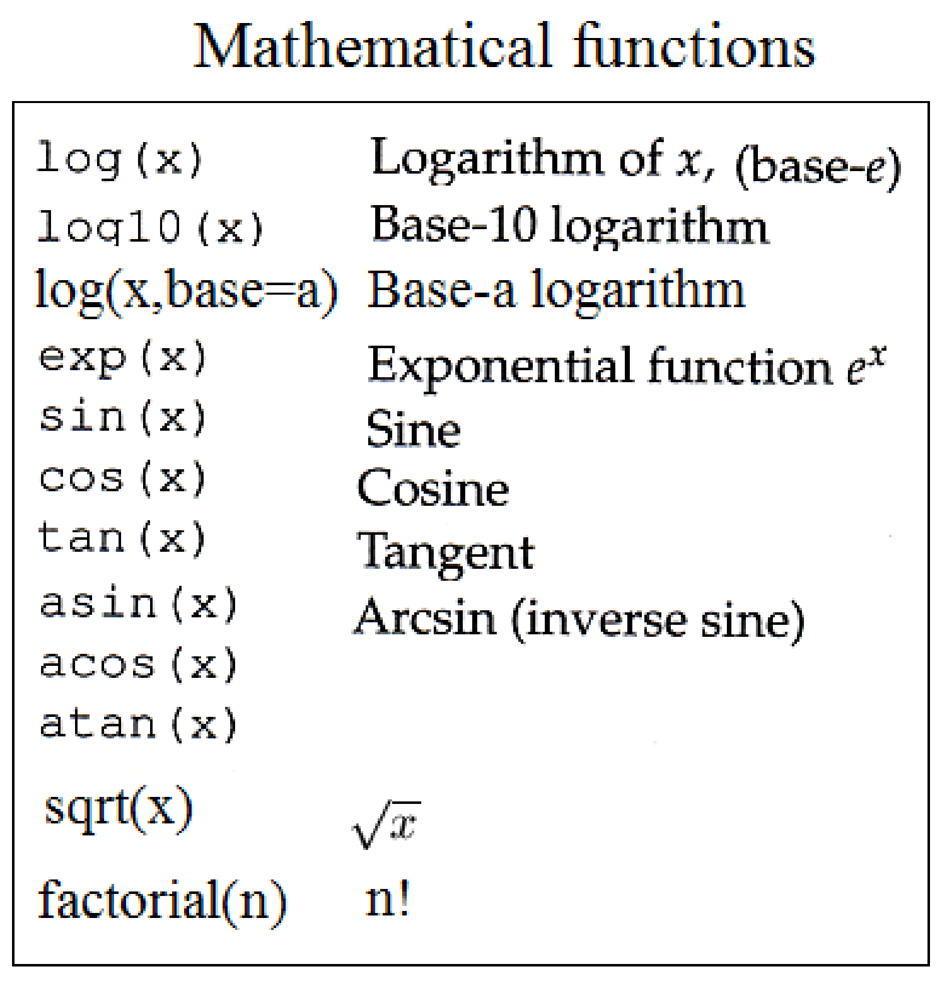


Some Arithmetic Operators

+	Addition
−	Subtraction, sign
*	Multiplication
/	Division
^	Raise to power
%/%	Integer division
%%	Remainder from integer division

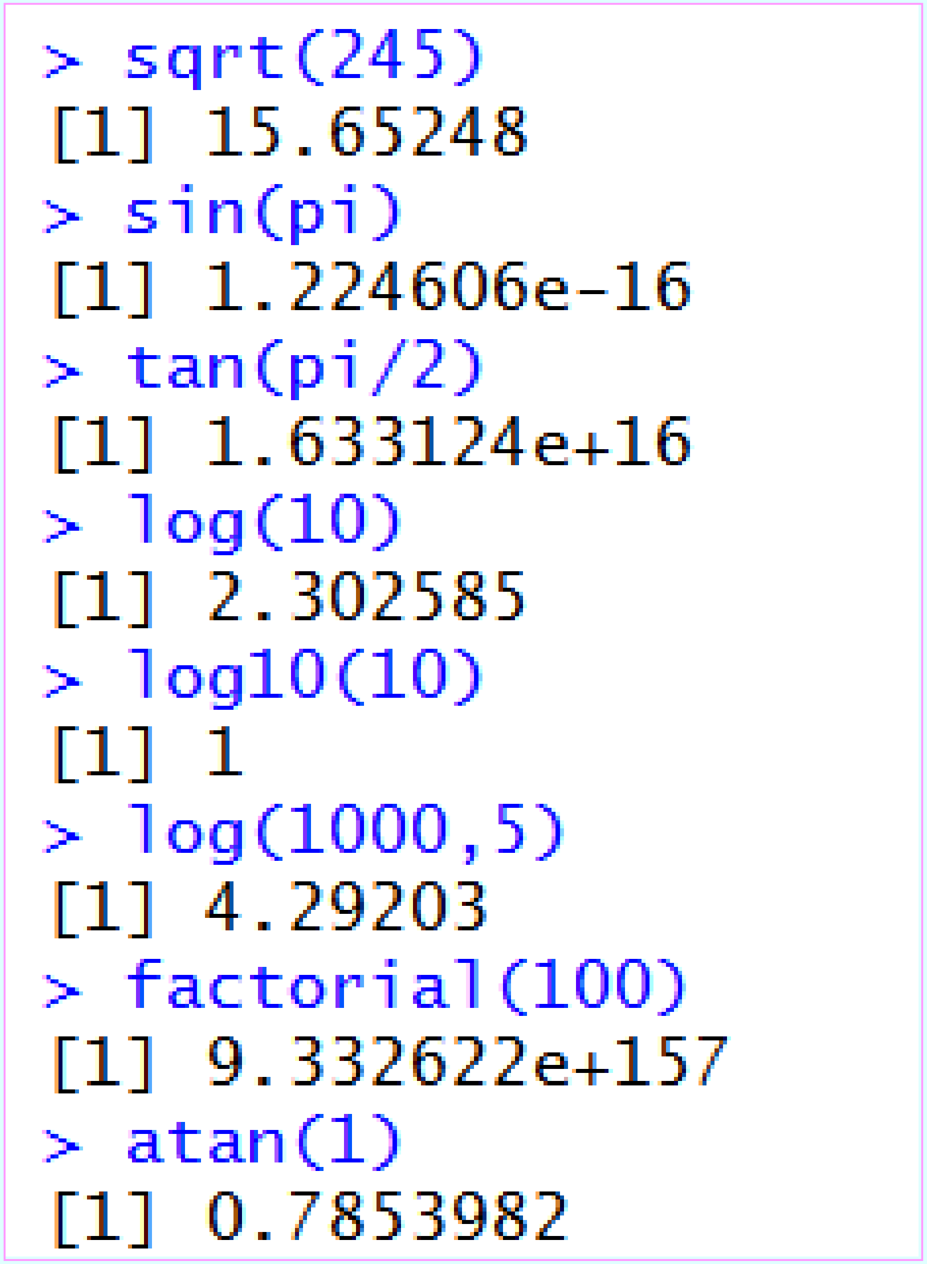


```
> 125+371-124
[1] 372
> 3^20
[1] 3486784401
> 375/27
[1] 13.88889
> 375%/%27
[1] 13
> 4*507
[1] 2028
> 567%%2
[1] 1
```



Mathematical functions

`log(x)`	Logarithm of x, (base-e)
`log10(x)`	Base-10 logarithm
log(x,base=a)	Base-a logarithm
`exp(x)`	Exponential function e^x
`sin(x)`	Sine
`cos(x)`	Cosine
`tan(x)`	Tangent
`asin(x)`	Arcsin (inverse sine)
`acos(x)`	
`atan(x)`	
sqrt(x)	$\sqrt{x}$
factorial(n)	n!

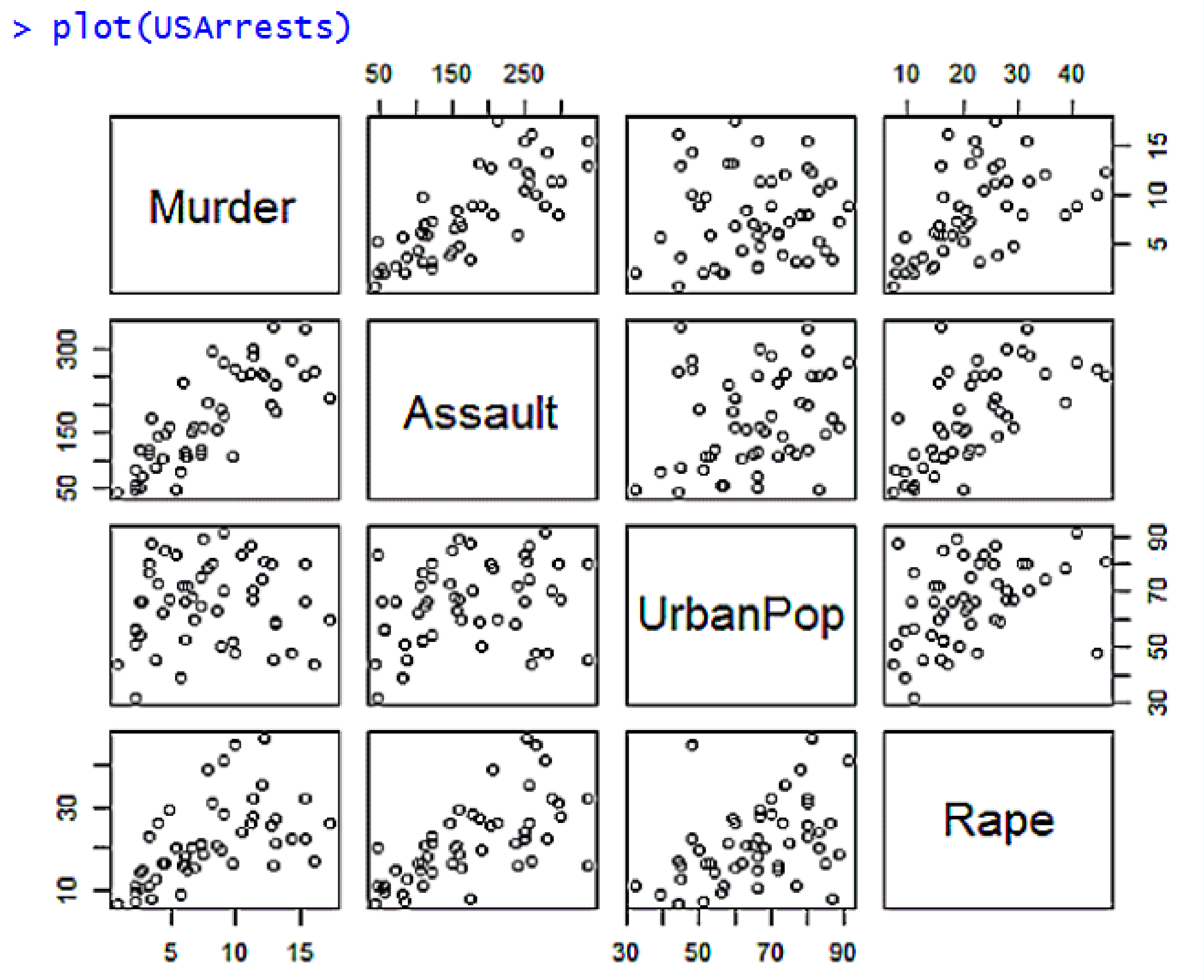


```
> sqrt(245)
[1] 15.65248
> sin(pi)
[1] 1.224606e-16
> tan(pi/2)
[1] 1.633124e+16
> log(10)
[1] 2.302585
> log10(10)
[1] 1
> log(1000,5)
[1] 4.29203
> factorial(100)
[1] 9.332622e+157
> atan(1)
[1] 0.7853982
```

(3) R has Graphical Capabilities

- Line plot
- Scatter plot
- Histogram
- Density
- Boxplot
- Multivariate plot
- Conditioning plot
- Contour plot
- Bubble plot
- Pie chart
- Pyramid plot
- Dot chart

A helpful graphical summary for the USArrests data set is the scatterplot matrix.

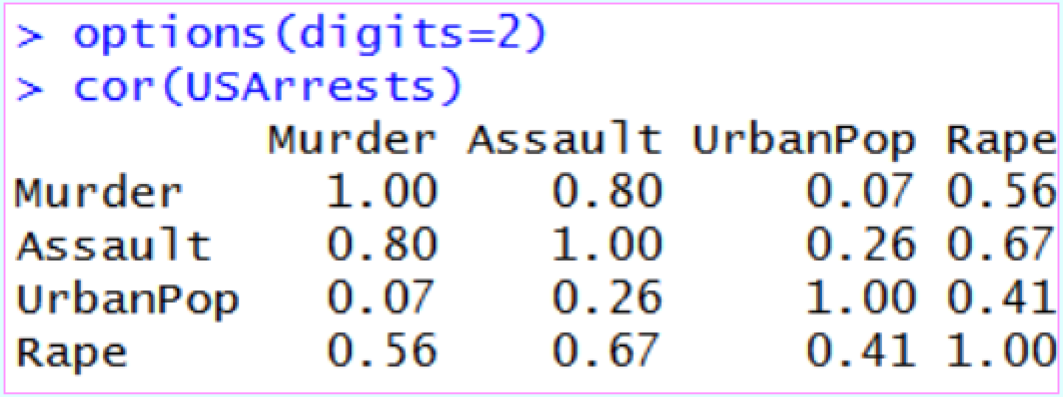


```
> plot(USArrests)
```

(4) R handles a Variety of Specific Analyses

Correlation:

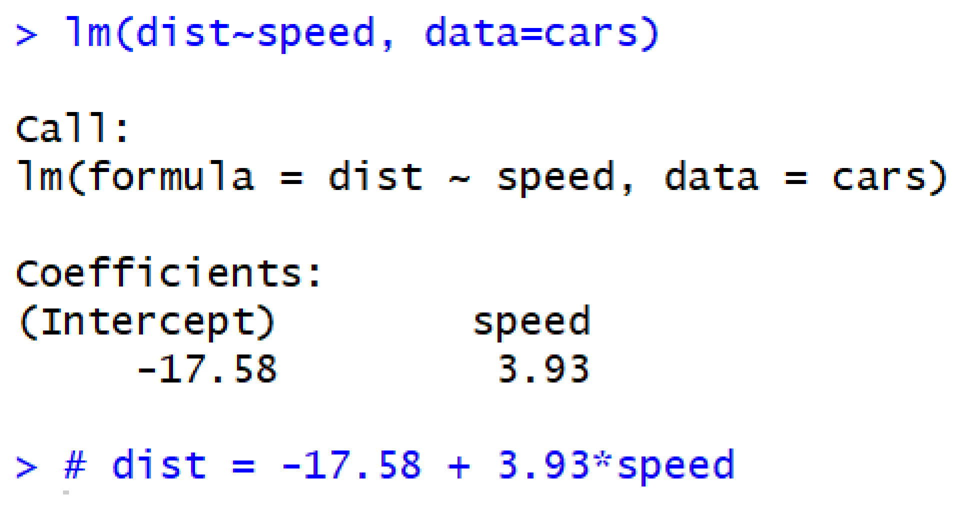
We calculate the correlation matrix for the USArrests data.



```
> options(digits=2)
> cor(USArrests)
         Murder Assault UrbanPop Rape
Murder     1.00    0.80     0.07 0.56
Assault    0.80    1.00     0.26 0.67
UrbanPop   0.07    0.26     1.00 0.41
Rape       0.56    0.67     0.41 1.00
```

Straight Line Regression:

The formula that is supplied to the lm() command asks for the regression of distance travelled by the speed for the cars data.

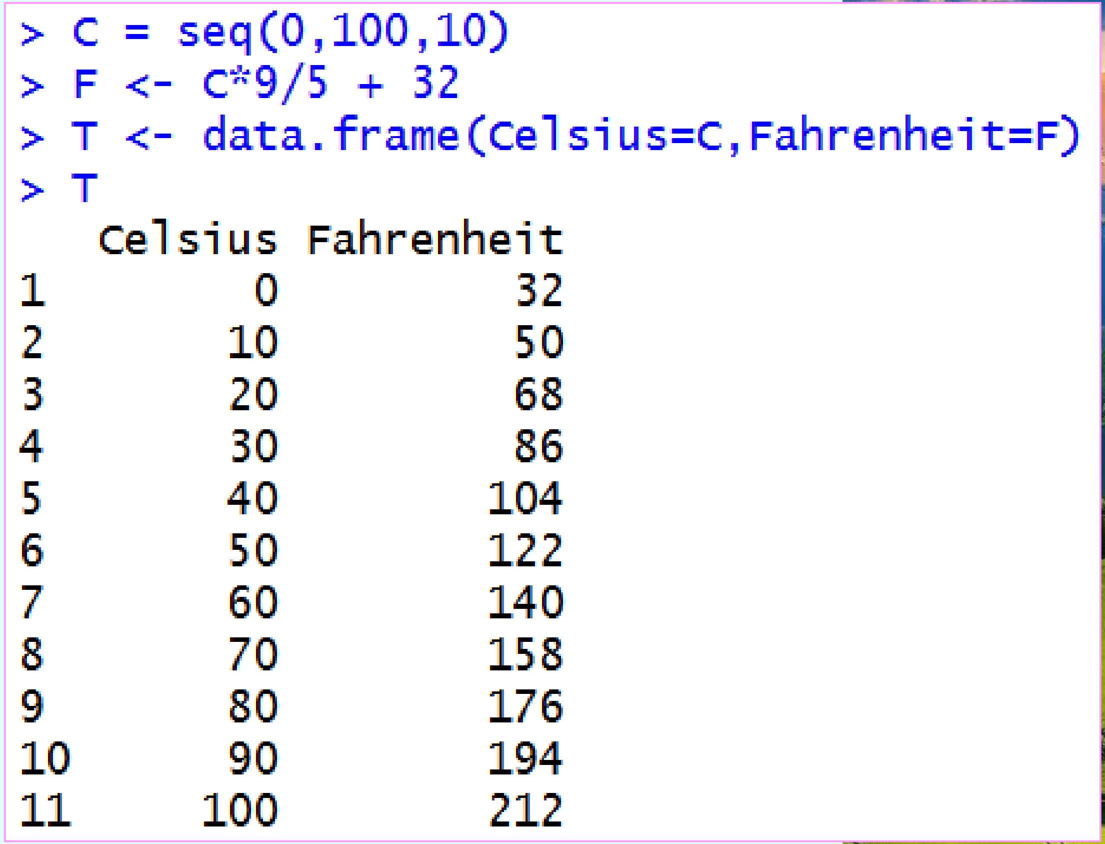


```
> lm(dist~speed, data=cars)

Call:
lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)        speed
     -17.58         3.93

> # dist = -17.58 + 3.93*speed
```

(5) R is an Interactive Programming Language

Celsius to Fahrenheit Formula

$$F=\frac{9}{5}C+32$$

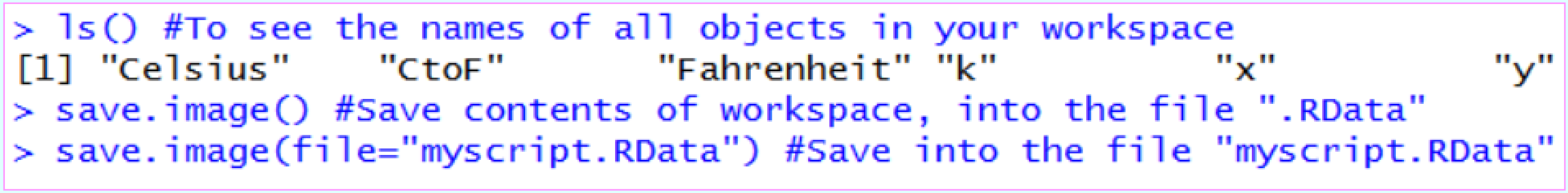


```
> C = seq(0,100,10)
> F <- C*9/5 + 32
> T <- data.frame(Celsius=C,Fahrenheit=F)
> T
   Celsius Fahrenheit
1        0         32
2       10         50
3       20         68
4       30         86
5       40        104
6       50        122
7       60        140
8       70        158
9       80        176
10      90        194
11     100        212
```

3. R Objects

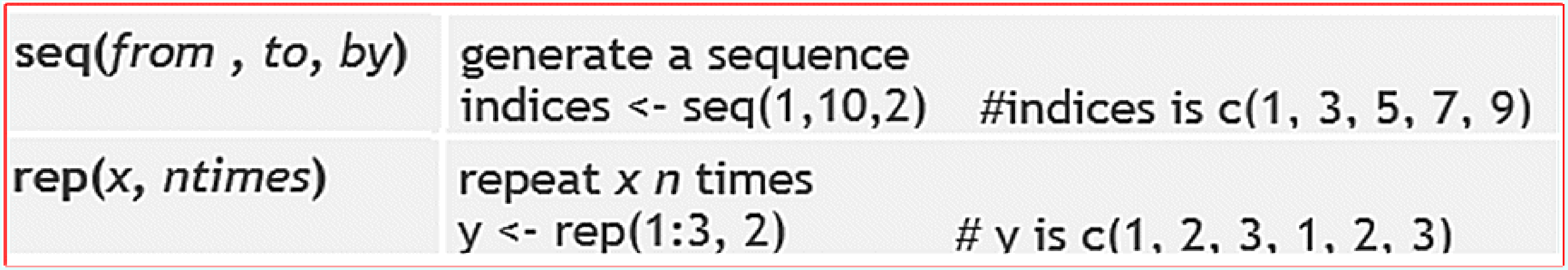
All R entities, including functions and data structures, exist as **objects**.

(1) Command Options



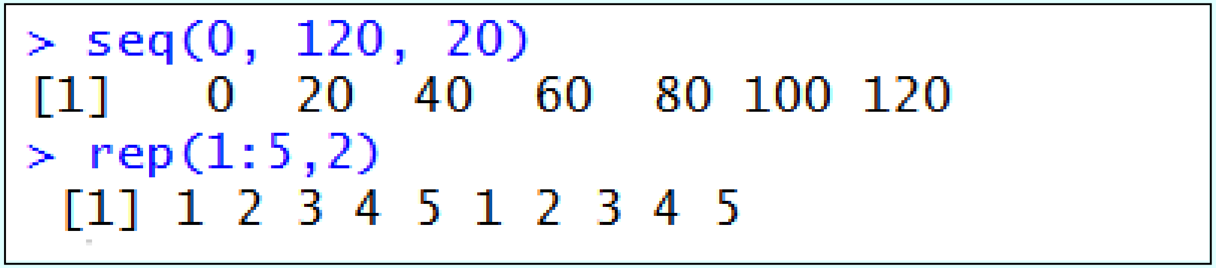
```
> ls() #To see the names of all objects in your workspace
[1] "Celsius"    "CtoF"        "Fahrenheit" "k"             "x"          "y"
> save.image() #Save contents of workspace, into the file ".RData"
> save.image(file="myscript.RData") #Save into the file "myscript.RData"
```

(2) Manipulating R Objects



seq(*from* , *to*, *by*)	generate a sequence indices <- seq(1,10,2) #indices is c(1, 3, 5, 7, 9)

rep(*x*, *ntimes*)	repeat *x n* times
	y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3)



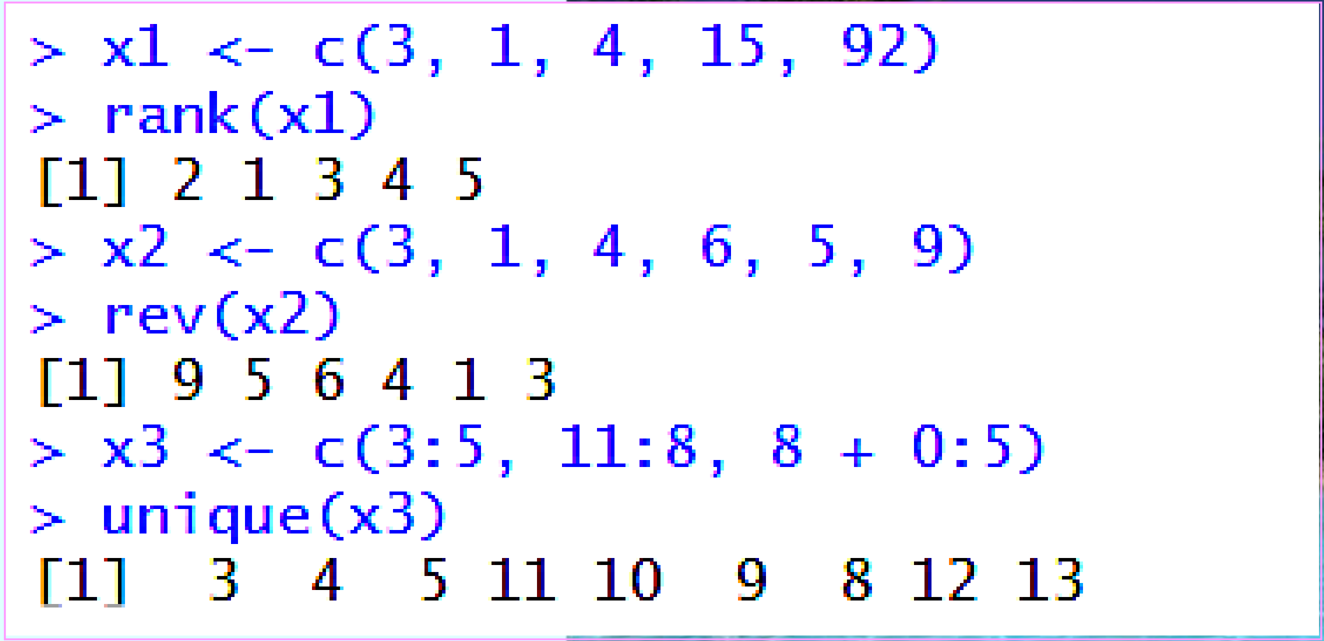
```
> seq(0, 120, 20)
[1]   0  20  40  60  80 100 120
> rep(1:5,2)
 [1] 1 2 3 4 5 1 2 3 4 5
```

(2) Manipulating R Objects

rev(x) reverses an R object, including vector, array etc.

rank(x, ...) returns the sample ranks of the values in a vector.

unique(x, ...) returns a vector, data frame or array like x but with duplicate elements/rows removed.

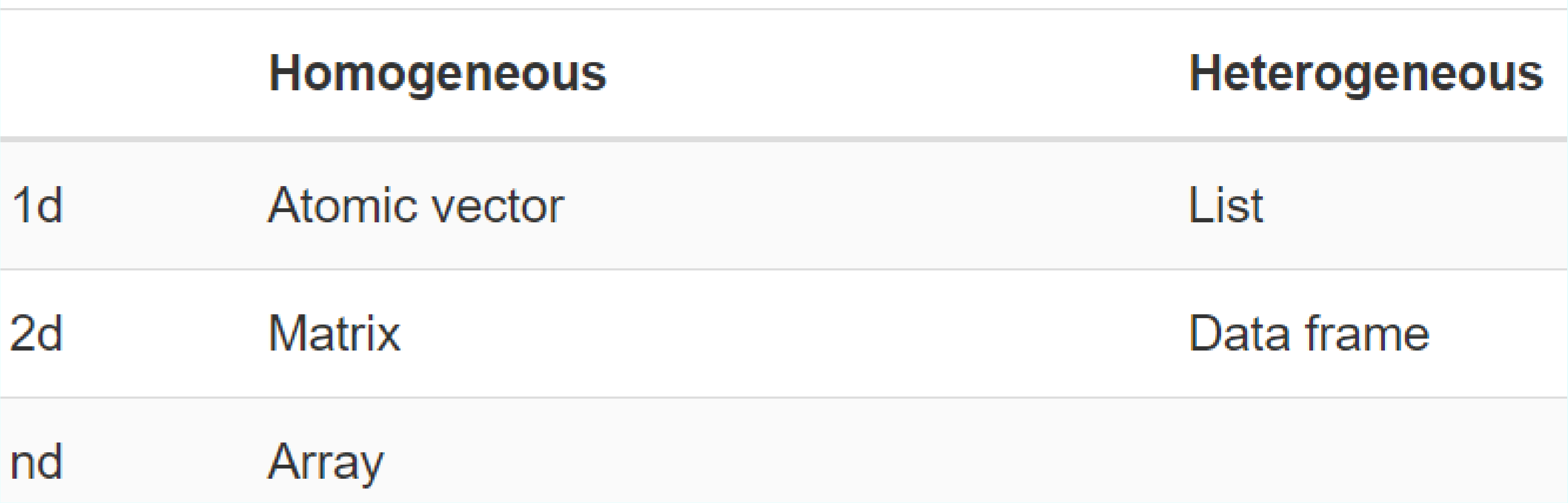


```
> x1 <- c(3, 1, 4, 15, 92)
> rank(x1)
[1] 2 1 3 4 5
> x2 <- c(3, 1, 4, 6, 5, 9)
> rev(x2)
[1] 9 5 6 4 1 3
> x3 <- c(3:5, 11:8, 8 + 0:5)
> unique(x3)
[1]  3  4  5 11 10  9  8 12 13
```

4. R Data Structures

In R, available data structures allow you to hold any type of data and use them for further processing and analysis.

R’s base data structures can be organized by their dimensionality (1d, 2d, nd) and whether they’re homogeneous (all contents must be of the same type) or heterogeneous (the contents can be of different types).



	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

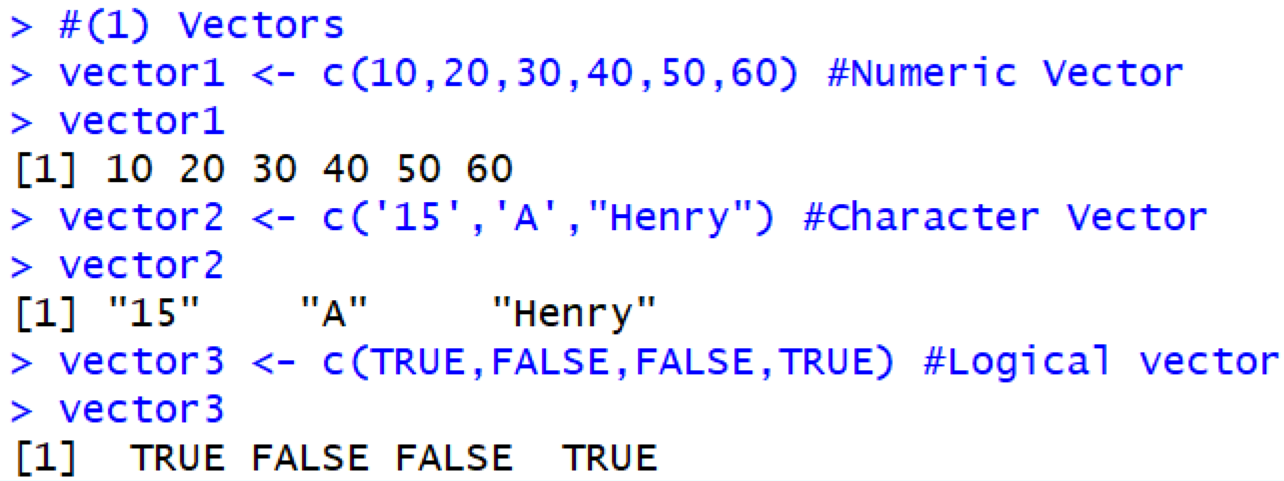
4. R Data Structures

In R, available data structures allow you to hold any type of data and use them for further processing and analysis.

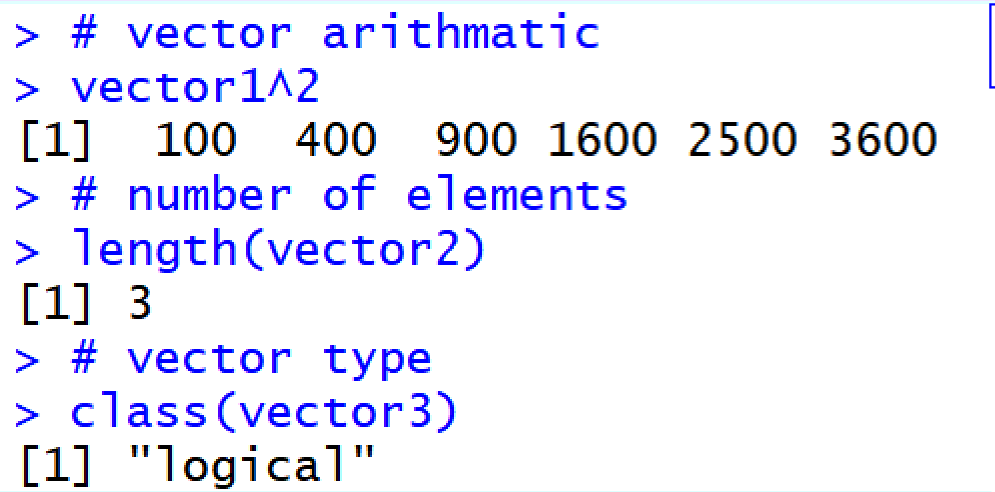
(1) Vectors

Vectors are ordered collection of 'atomic' (same data type) components.

Numeric vector : a vector of numbers
Character vector: a vector of text strings
Logical vector: a vector of the value TRUE or FALSE.



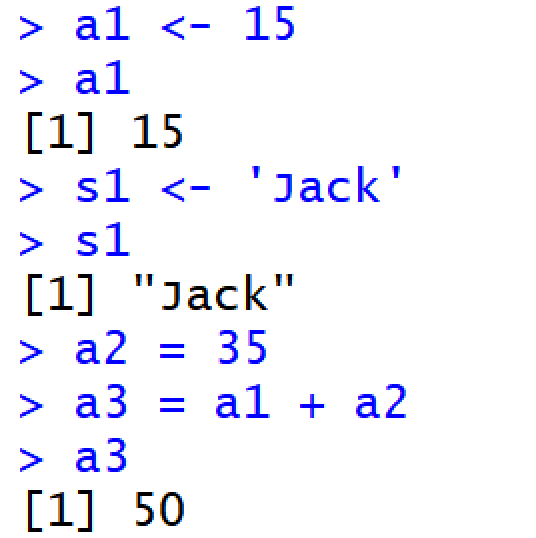
```
> #(1) Vectors
> vector1 <- c(10,20,30,40,50,60) #Numeric Vector
> vector1
[1] 10 20 30 40 50 60
> vector2 <- c('15','A',"Henry") #Character Vector
> vector2
[1] "15"    "A"     "Henry"
> vector3 <- c(TRUE,FALSE,FALSE,TRUE) #Logical vector
> vector3
[1]  TRUE FALSE FALSE  TRUE
```



```
> # vector arithmatic
> vector1^2
[1]  100  400  900 1600 2500 3600
> # number of elements
> length(vector2)
[1] 3
> # vector type
> class(vector3)
[1] "logical"
```

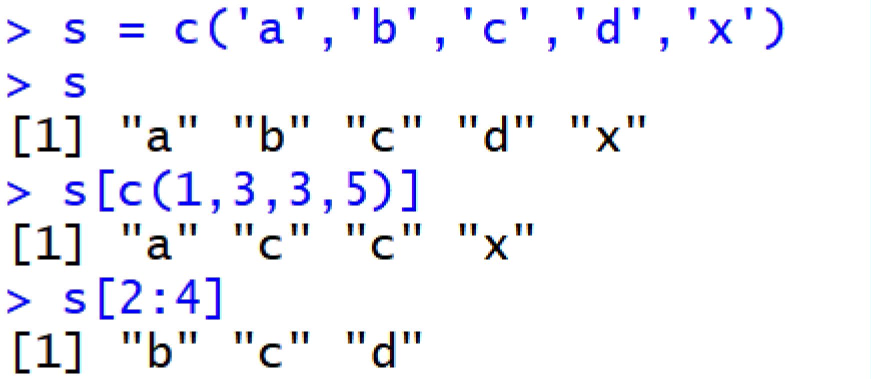
Scalars

Scalars are one-element vectors that are traditionally used to hold some constant values.



```
> a1 <- 15
> a1
[1] 15
> s1 <- 'Jack'
> s1
[1] "Jack"
> a2 = 35
> a3 = a1 + a2
> a3
[1] 50
```

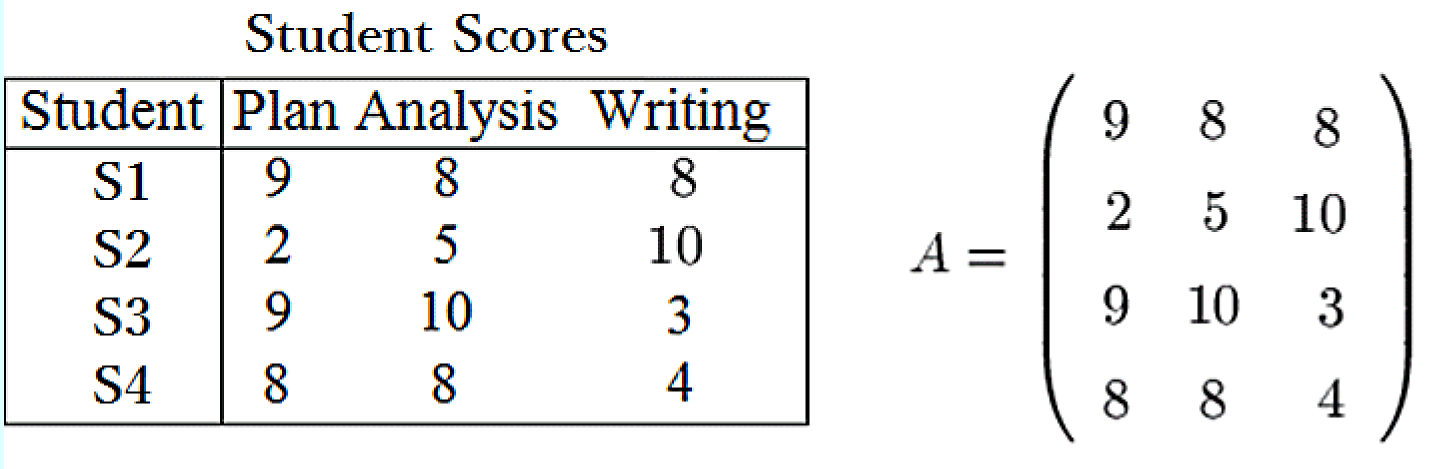
Numeric Index Vector



```
> s = c('a','b','c','d','x')
> s
[1] "a" "b" "c" "d" "x"
> s[c(1,3,3,5)]
[1] "a" "c" "c" "x"
> s[2:4]
[1] "b" "c" "d"
```

(2) Matrices

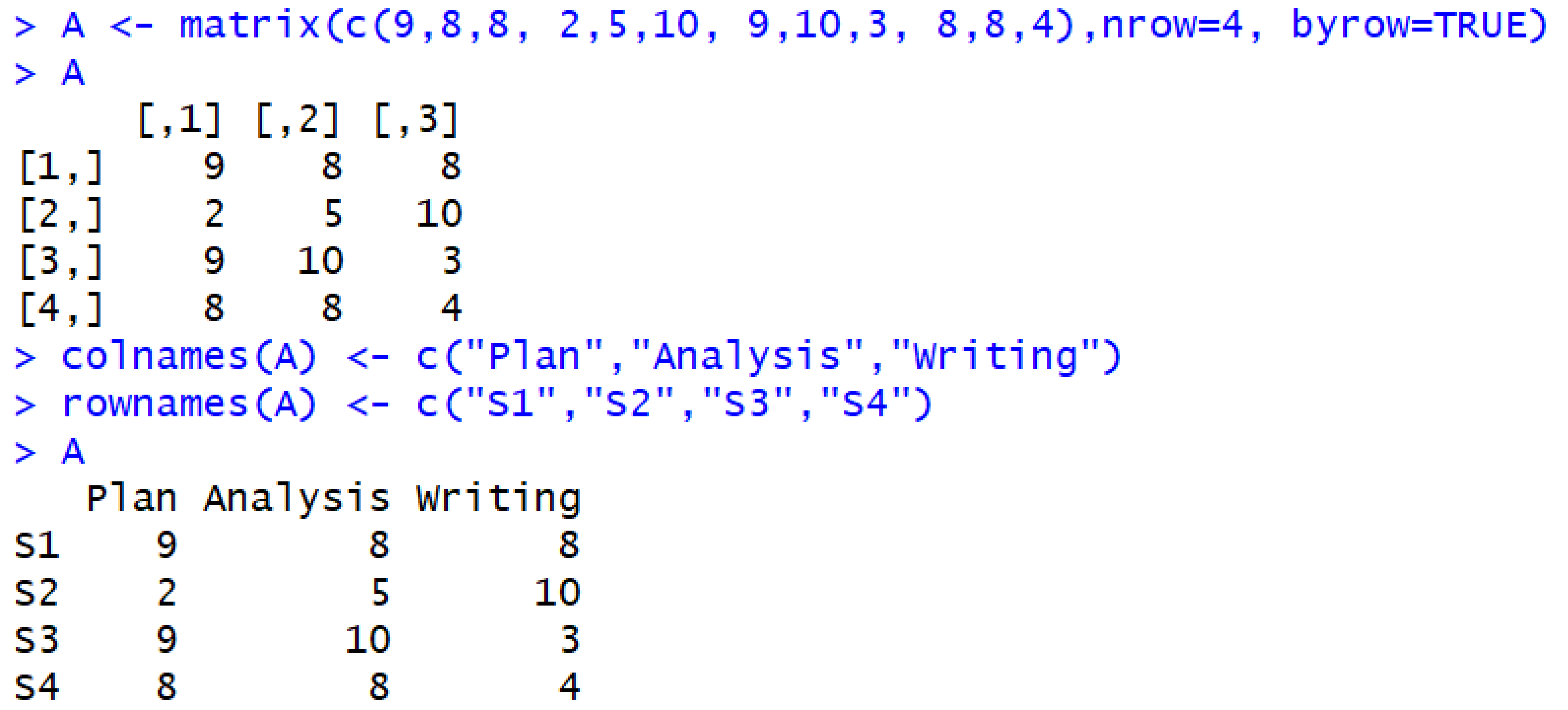
A *matrix* is just a two-dimensional array of data elements.



Student Scores

Student	Plan	Analysis	Writing
S1	9	8	8
S2	2	5	10
S3	9	10	3
S4	8	8	4

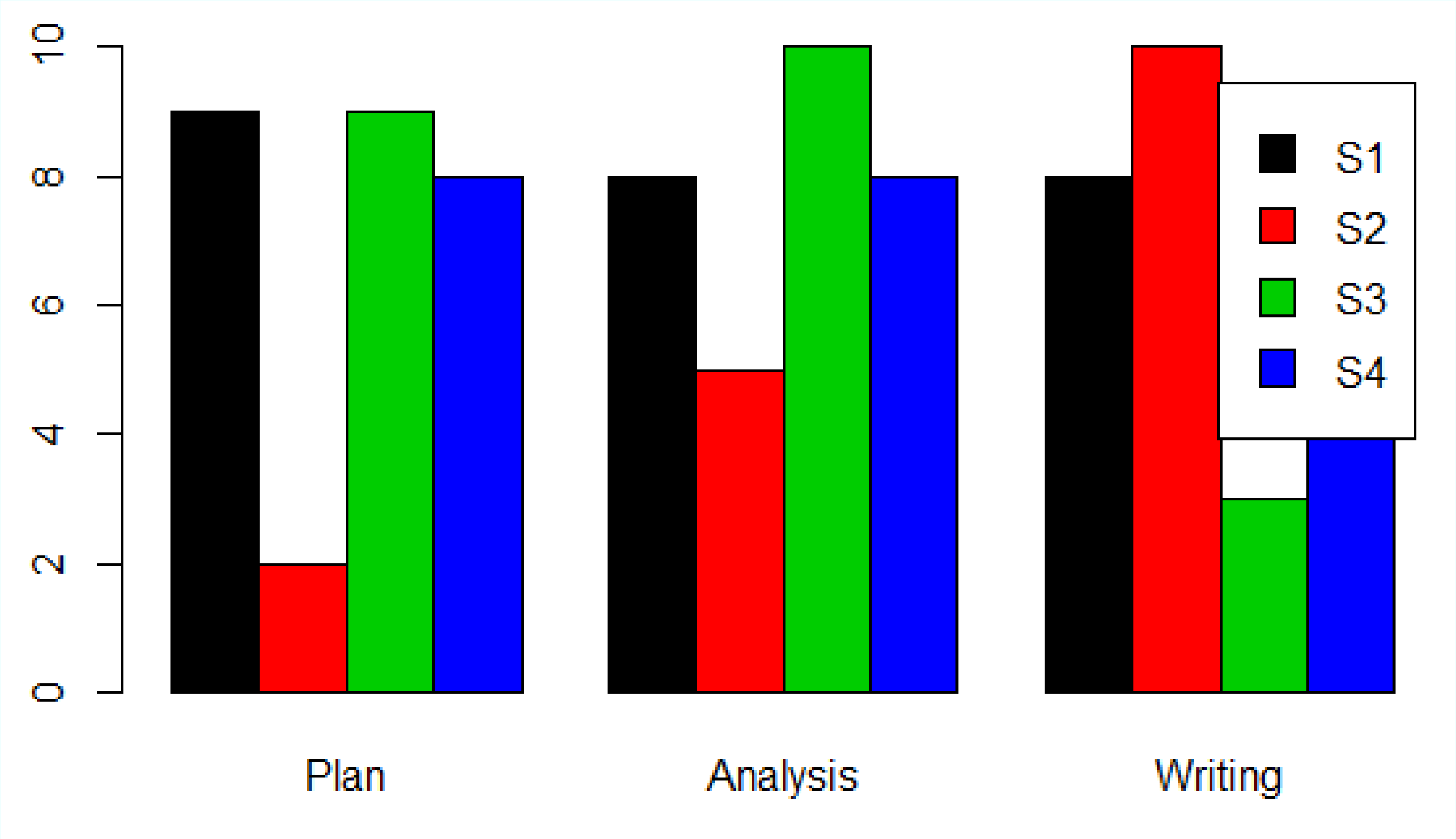
$$A = \begin{pmatrix} 9 & 8 & 8 \\ 2 & 5 & 10 \\ 9 & 10 & 3 \\ 8 & 8 & 4 \end{pmatrix}$$



```
> A <- matrix(c(9,8,8, 2,5,10, 9,10,3, 8,8,4),nrow=4, byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    9    8    8
[2,]    2    5   10
[3,]    9   10    3
[4,]    8    8    4
> colnames(A) <- c("Plan","Analysis","Writing")
> rownames(A) <- c("S1","S2","S3","S4")
> A
   Plan Analysis Writing
S1    9        8       8
S2    2        5      10
S3    9       10       3
S4    8        8       4
```

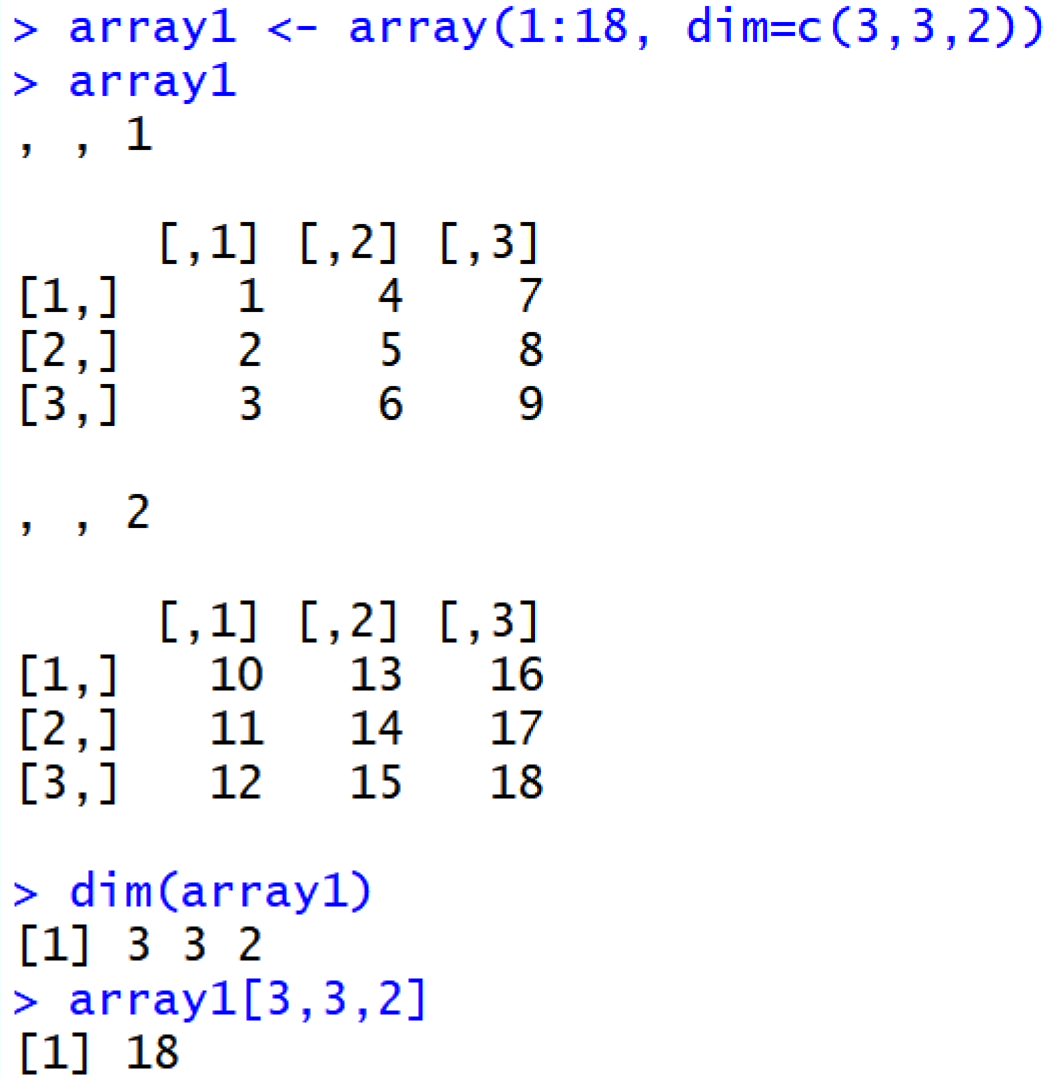
Matrices can be visualized as bar charts.

```
barplot(A,beside=TRUE,legend=TRUE,col=1:4)
```



(3) **Arrays**

Arrays are similar to matrices, but they contain more dimensions. Arrays can be created using the **array()** function.

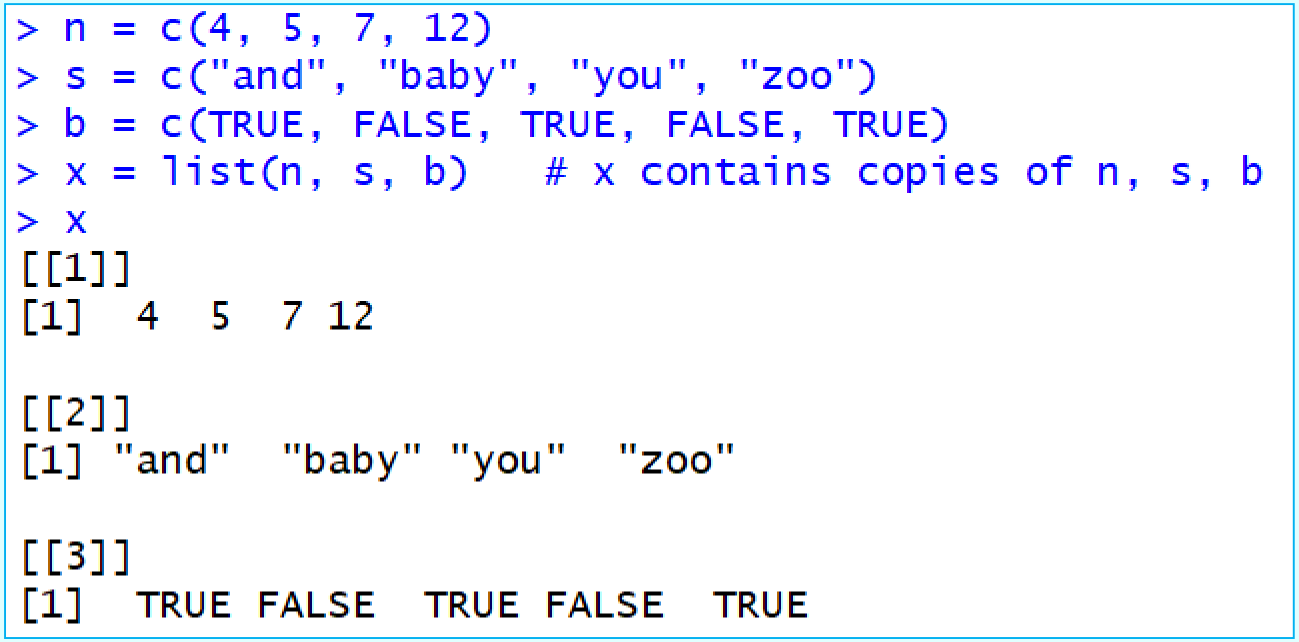


```
> array1 <- array(1:18, dim=c(3,3,2))
> array1
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

> dim(array1)
[1] 3 3 2
> array1[3,3,2]
[1] 18
```

(4) Lists

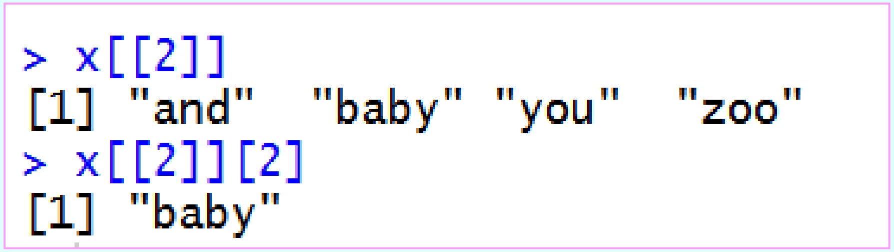
Lists are a special type of vector that can contain elements of different classes.

Lists can be explicitly created using the **list()** function.



```
> n = c(4, 5, 7, 12)
> s = c("and", "baby", "you", "zoo")
> b = c(TRUE, FALSE, TRUE, FALSE, TRUE)
> x = list(n, s, b)   # x contains copies of n, s, b
> x
[[1]]
[1]  4  5  7 12

[[2]]
[1] "and"  "baby" "you"  "zoo"

[[3]]
[1]  TRUE FALSE  TRUE FALSE  TRUE
```

In order to reference a list member directly, we have to use the *double square bracket* "**[[]]**"operator.

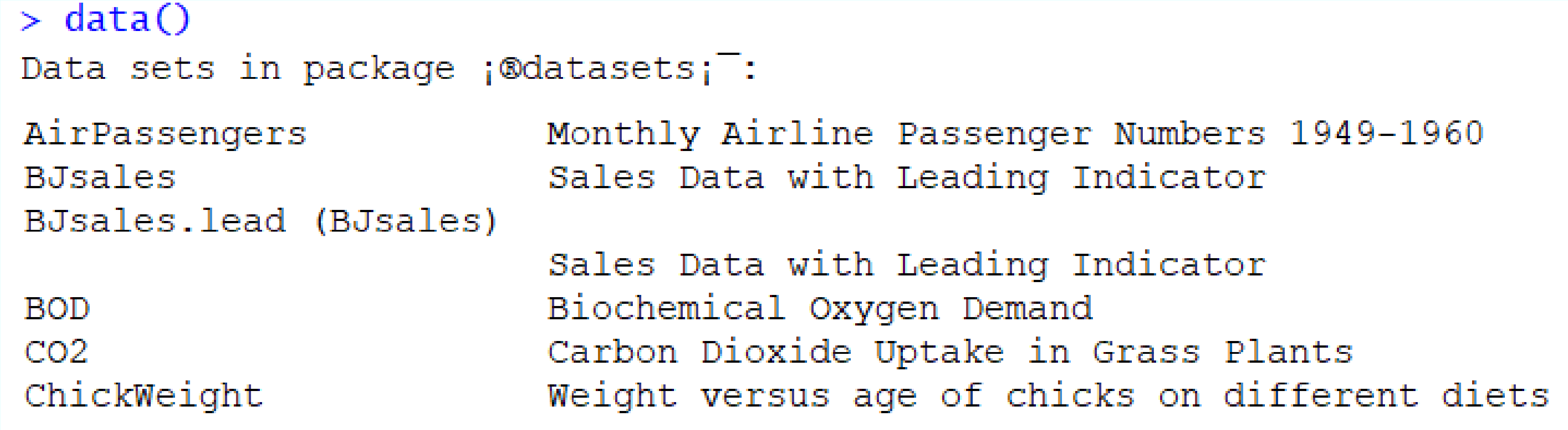


```
> x[[2]]
[1] "and"  "baby" "you"  "zoo"
> x[[2]][2]
[1] "baby"
```

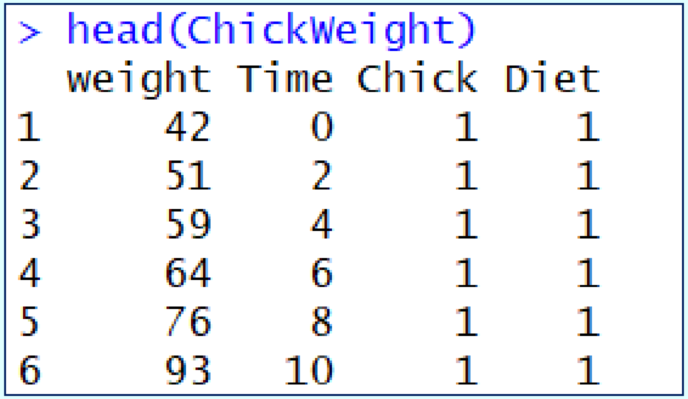
(5) Data Frame

A data frame is a list of vectors of equal length.

Build-in Data Frame



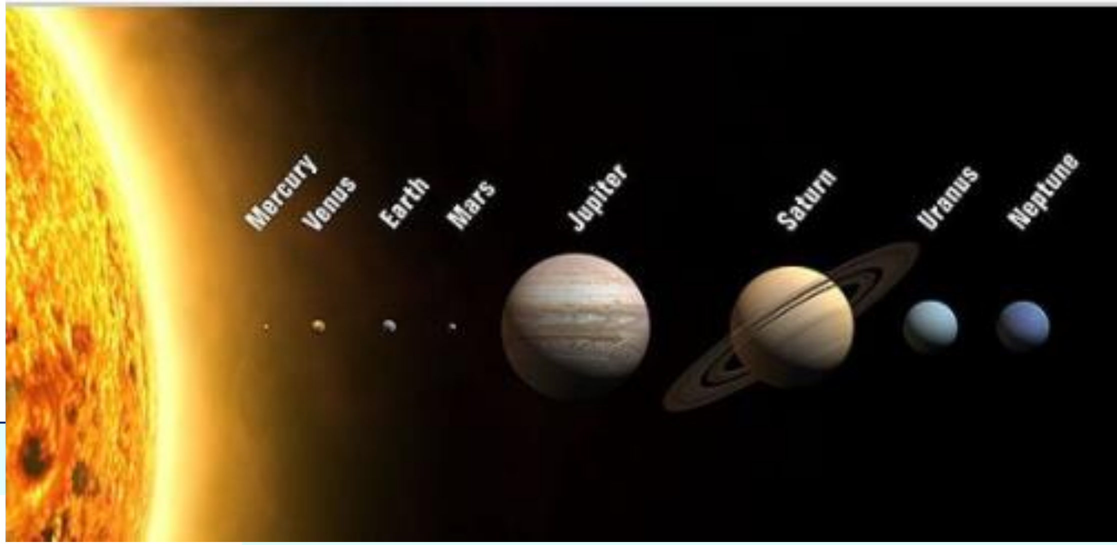
```
> data()
Data sets in package ¡®datasets¡¯:

AirPassengers           Monthly Airline Passenger Numbers 1949-1960
BJsales                 Sales Data with Leading Indicator
BJsales.lead (BJsales)
                        Sales Data with Leading Indicator
BOD                     Biochemical Oxygen Demand
CO2                     Carbon Dioxide Uptake in Grass Plants
ChickWeight             Weight versus age of chicks on different diets
```

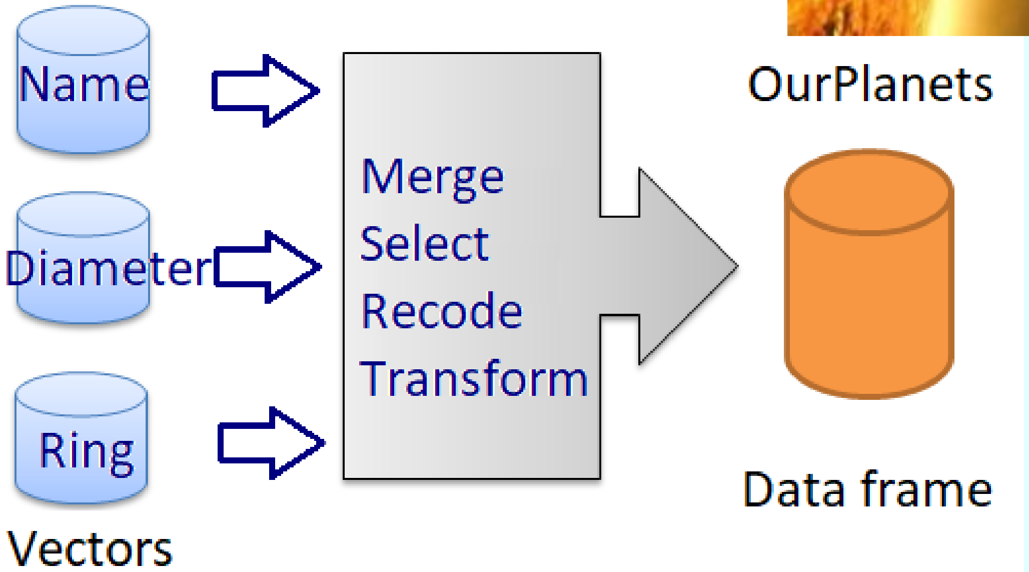


```
> head(ChickWeight)
  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1
```

Creating a data frame

```
> Name <- c("Mercury", "Venus","Earth","Mars","Jupiter","Saturn", "Uranus", "Neptune")
> Diameter <- c(0.382,0.949,1.0,0.532,11.209,9.449,4.007,3.883)
> Ring <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)
> OurPlanets <- data.frame(Name,Diameter,Ring)
> OurPlanets
     Name Diameter  Ring
1 Mercury    0.382 FALSE
2   Venus    0.949 FALSE
3   Earth    1.000 FALSE
4    Mars    0.532 FALSE
5 Jupiter   11.209  TRUE
6  Saturn    9.449  TRUE
7  Uranus    4.007  TRUE
8 Neptune    3.883  TRUE
```





Name

Diameter

Ring

Vectors

Merge
Select
Recode
Transform

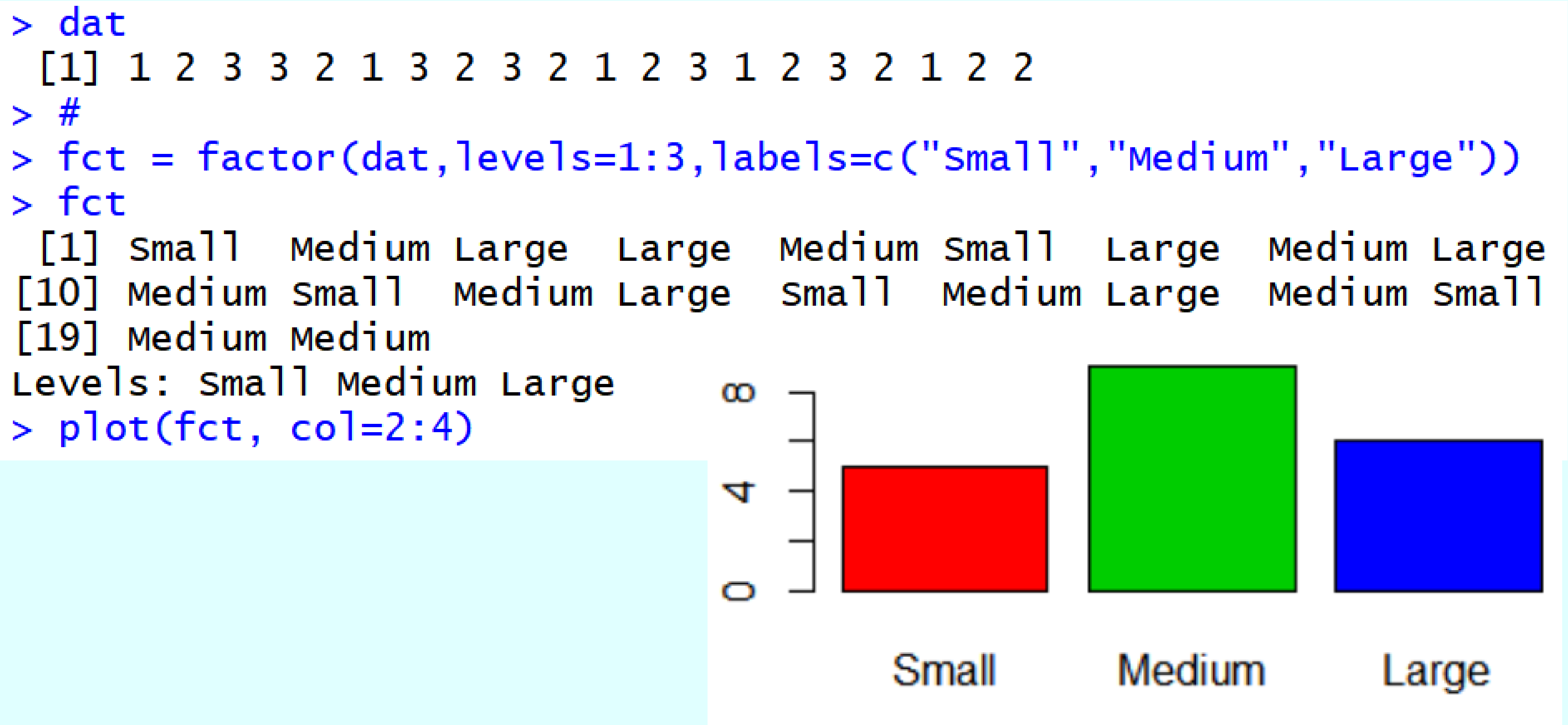
OurPlanets

Data frame

5. Factors

Factors are variables which take on a limited number of different values; such variables are often referred to as **categorical variables**.

factor(x=character(), levels, labels = levels, ...) {base}
The function factor is used to encode a vector as a factor

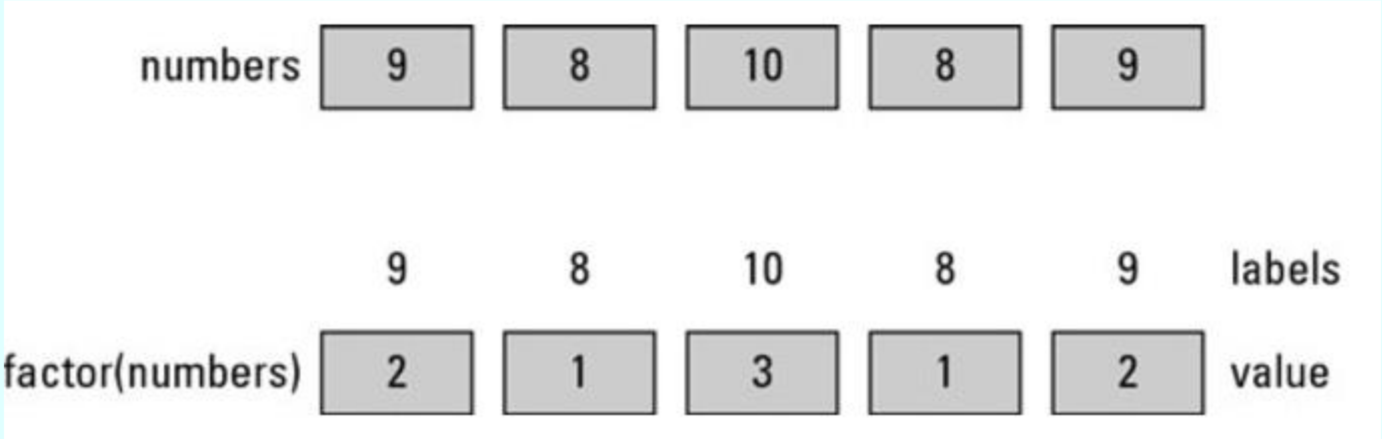


```
> dat
 [1] 1 2 3 3 2 1 3 2 3 2 1 2 3 1 2 3 2 1 2 2
> #
> fct = factor(dat,levels=1:3,labels=c("Small","Medium","Large"))
> fct
 [1] Small  Medium Large  Large  Medium Small  Large  Medium Large
[10] Medium Small  Medium Large  Small  Medium Large  Medium Small
[19] Medium Medium
Levels: Small Medium Large
> plot(fct, col=2:4)
```

To look at the internal representation of numbers, use str():

```
> str(fct)
 Factor w/ 3 levels "Small","Medium",..: 1 2 3 3 2 1 3 2 3 2 ...
> class(fct)
[1] "factor"
> table(fct)
fct
 Small Medium  Large 
     5      9      6 
> mode(fct)
[1] "numeric"
```

```
> numbers <- factor(c(9,8,10,8,9))
> str(numbers)
 Factor w/ 3 levels "8","9","10": 2 1 3 1 2
```


numbers
9
8
10
8
9
9
8
10
8
9
labels
factor(numbers)
2
1
3
1
2
value