R-Programming

Dr. Ch. Janaki
C-DAC Bangalore

DAY 1

Session 15:

- The R project for Statistical Computing
- Why R
- Introduction & Installation of R
- R Basics, Finding Help,
- Code Editors for R,
- Exploring RGui
- Exploring RStudio
- Basic Mathematical & Arithmetic operations in R

Session 16:

- Data Objects- Data Types & Data Structures (e.g. lists. Arrays, matrices, data frames)
- Packages in R
- Working with Packages
- Handling Data in R Workspace
- Reading & Importing data from Text files, Excel files, Multiple
- Exporting Data from R

Session 18:

- Functions
- Built in functions in R (numeric, character, statistical)
- Interactive reporting with R markdown

DAY 2

Session 17:

- o Introduction to tidy verse (group of packages)
- o Manipulating and Processing Data in R
- o Creating, Accessing and Sorting data frames
- o Extracting, Combining, Merging, reshaping data frames

- o Introduction to R Shiny

What is R?

- The R statistical programming language is a free open-source package based on the S language (developed by Bell Labs).
- R was created by Ross Ihaka and Robert Gentleman at the university of Auckland, New Zealand
- The language is very powerful for writing programs.
- Many statistical functions are already built in.
- Contributed packages expand the functionality to cutting edge research.

Ross Ihaka

Robert Gentleman

R VS PYTHON

R & Python – Both are powerful languages.

R – Good for Data Analytics, Statistical analysis, Visualization, pre and post-processing. Easy to learn.

Python - For Machine learning and Deep learning, data analytics, non-statistical tasks, handling large datasets.

BASIC FEATURES OF R

- R is for data analysis and data visualization tool.
  - Visualization in the form of charts, plots and graphs

- It is supported with number of graphical, statistical techniques.

- There are several GUI editors of R language, out of which RGUI and Rstudio are commonly used.

- Common characteristics of R
  - Effective and powerful data handling
  - Arrays and Matrices related operations
  - Graphical representations of the analysis

Basic Features of R – Statistical Features

- R provides various statistical and graphical techniques, such as
  - Linear and non-linear modeling
  - Classical statistical tests
  - Time-series analysis
  - Classification, Clustering etc.

- R has various predefined packages. User can also install packages.

- R can generate static graphs. To generate dynamic and interactive graphics, user has to install additional packages

BASIC FEATURES OF R - PROGRAMMING FEATURES

- R supports following
  - Basic Math operations
  - Vector Operations
  - Matrix Operations
  - Some other data structures like data frames and lists.

- It can be used with other programming languages such as Python, Perl, Ruby, Julia and on Hadoop & Spark

Basic Features of R - Packages

- CRAN (Comprehensive R Archive Network) – Collection on R packages.
  https://cran.r-project.org/web/packages/available_packages_by_name.html
- A Package is a collection of functions and datasets.
- R provides 2 types of packages
  - Standard Packages (in-built) part of R source code
  - Contributed Packages (user-defined)
- To access the contents of package you have to first install (if it is not in-built) and load it.

  Eg: install.packages("ggplot2")

- These packages are widely used in Finance, Genetics, HPC, Machine Learning, Medical Imaging, Social Sciences and Spatial Statistics

BASIC FEATURES OF R – GRAPHICAL USER INTERFACE

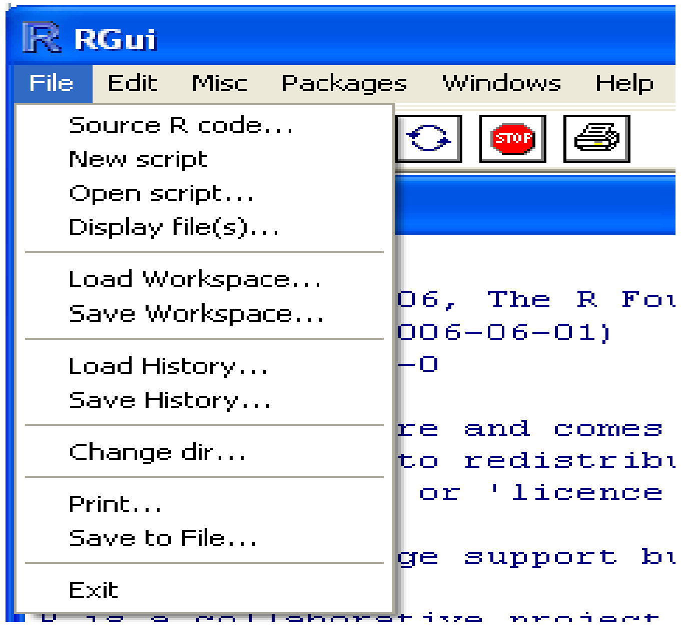
- Some popular text editors and Integrated Development Environments (IDEs) that support R programming are
  - ConTEXT
  - Eclipse
  - Emacs (Emacs Speaks Statistics)
  - Vim editor
  - jEdit
  - Rstudio
  - WinEdit

GETTING STARTED

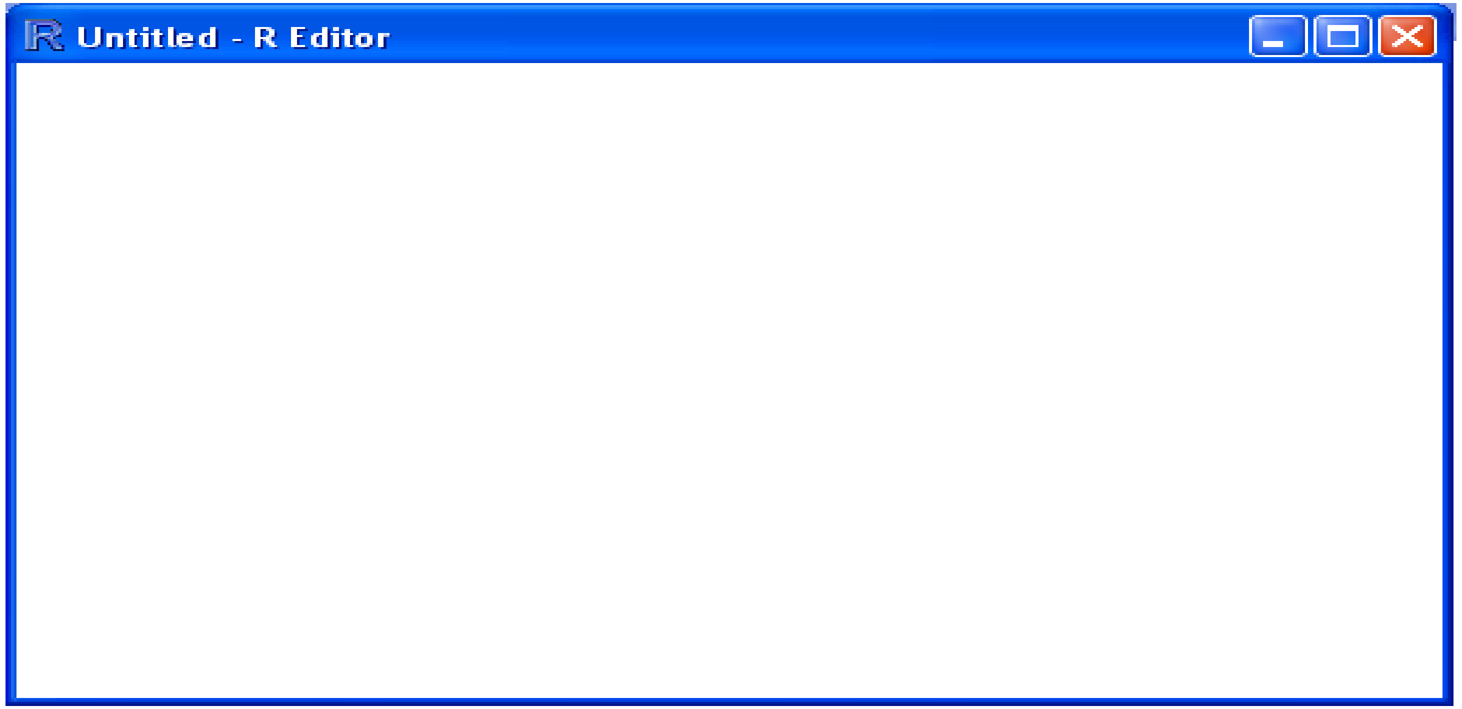
- Where to get R?
- Go to www.r-project.org
- Downloads: CRAN (The Comprehensive R Archive Network)
- Set your Mirror: Any of the mirror site can be selected.

GETTING STARTED

- Opening a script.
- This gives you a script window.



```
06, The R Fou
006-06-01)
-0

re and comes
to redistribu
 or 'licence

ge support bu
```



Getting Started

- Basic assignment and operations.
  - Arithmetic Operations:
    - +, -, *, /, ^ are the standard arithmetic operators
  - Assignment
    - To assign a value to a variable use "<-" or "="
  - Matrix Arithmetic.
    - * is element wise multiplication
    - %*% is matrix multiplication

R Arithmetic operators

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

Getting Started

- How to use help in R?
  - R has a very good built-in help system.
  - If you know which function you want help with simply use ?_______ with the function in the blank.
    - Ex: `?hist.`
  - If you don't know which function to use, then use help.search("_______").
    - Ex: `help.search("histogram")`

Packages

- Packages are collections of
  - R functions,
  - Data sets
  - compiled code in a well-defined format.
  - documentation for the package
  - Test scripts

- The directory where packages are stored is called the library.

- To access and use the package, it has to be loaded first.

Packages

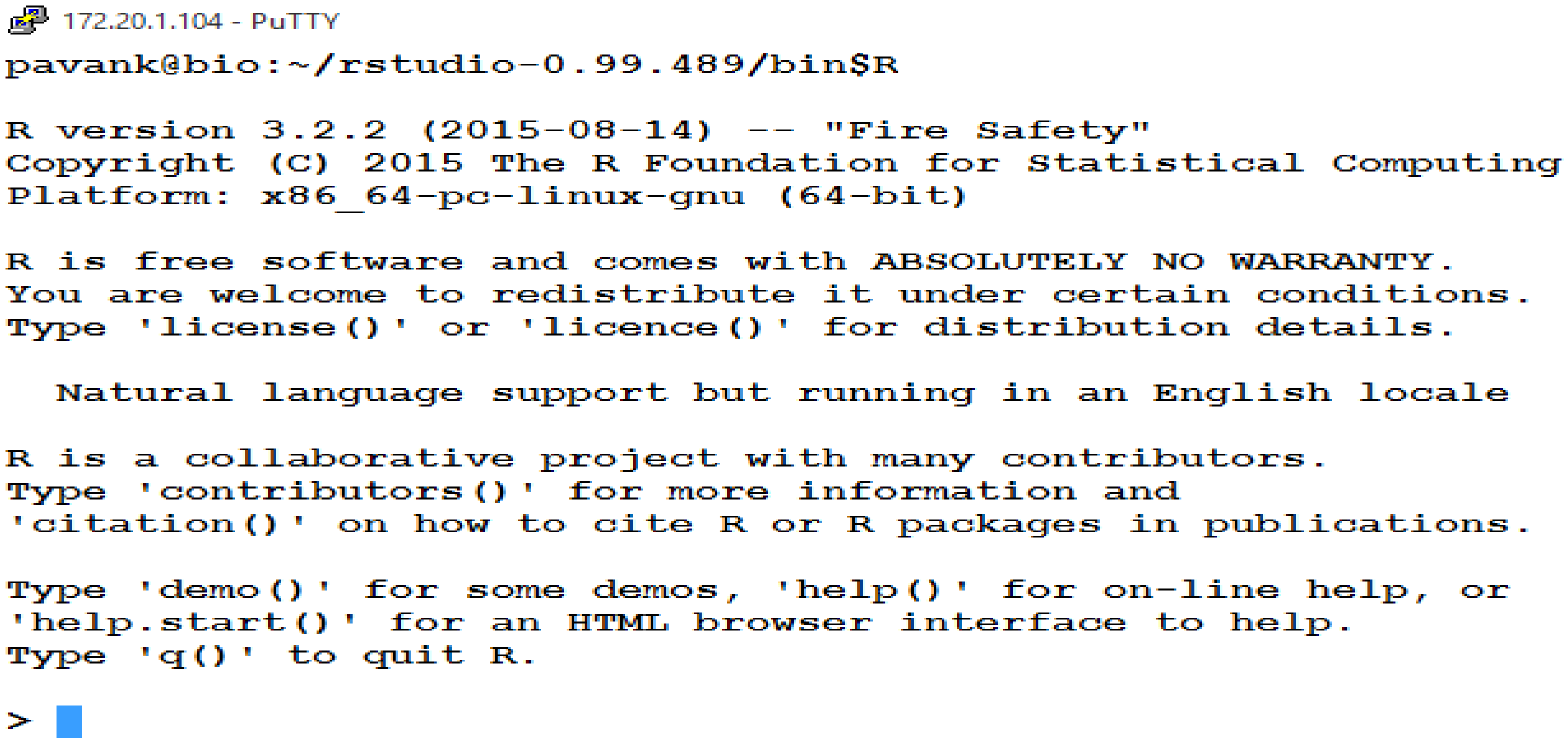
- **R** comes with a standard set of packages. Others are available for download and installation. Once installed, they have to be loaded into the session to be used.
- To install or add new R packages
  - `install.packages(“package_name”)`
- To load the package
  - `library(package_name)`
- To see default packages on R
  - `library()`
- To see installed packages on R
  - `installed.packages()`
  - Remove package: **remove.packages(**"package_name"**)**
- You can create your own package

CRAN

- It is A Comprehensive R Archive Network, contains many packages which can be used in many domains like
  - Genetics, Bioinformatics
  - Finance
  - HPC (High Performance Computing)
  - Machine Learning
  - Medical Imaging
  - Big data

R CONSOLE

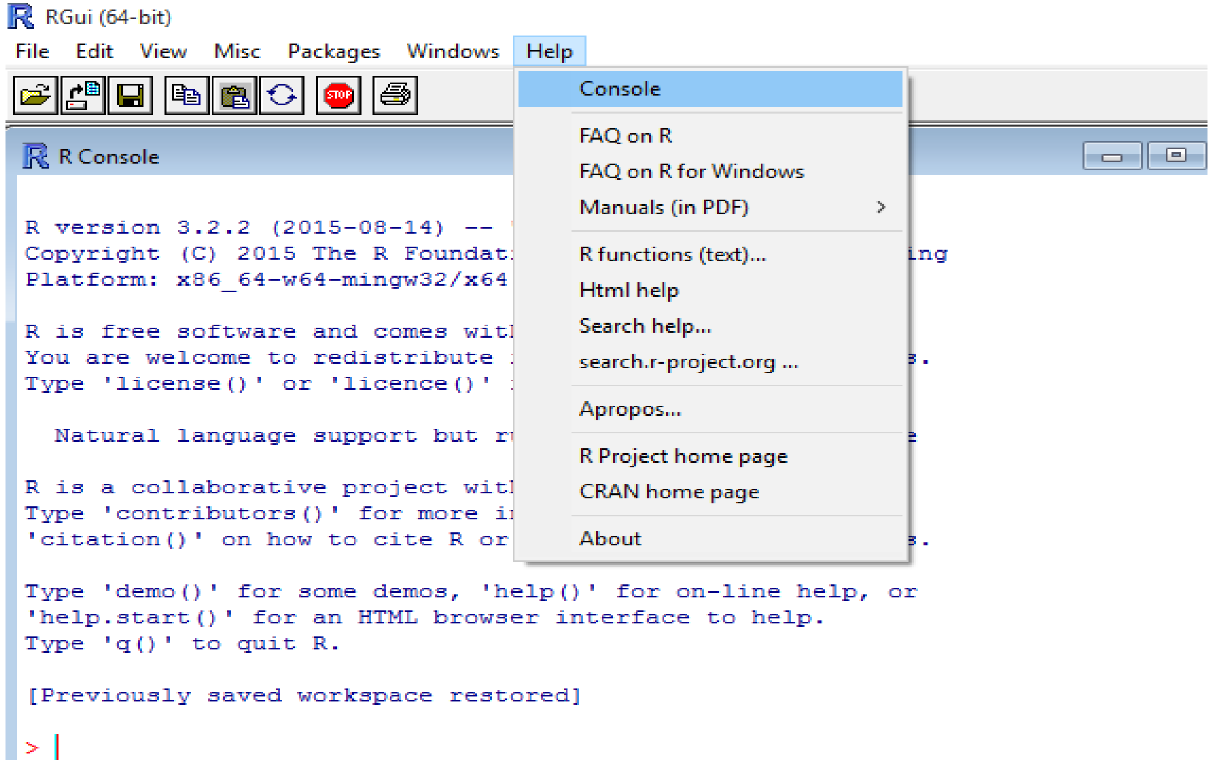
- After installing R on the Linux machine. Just type R on the command line
- After R console is opened, it shows some basic information about R, such as R version, date of release, licensing



```
pavank@bio:~/rstudio-0.99.489/bin$R

R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

R CONSOLE

- In the previous figure, notice “>” symbol.
- This is called R prompt, which allows users to write commands and then press ENTER key to execute the command.
- To get more information about the console, go to Help->Console.



```
R version 3.2.2 (2015-08-14) --
Copyright (C) 2015 The R Foundat                          ing
Platform: x86_64-w64-mingw32/x64

R is free software and comes wit
You are welcome to redistribute                           s.
Type 'license()' or 'licence()'

  Natural language support but r                          e

R is a collaborative project wit
Type 'contributors()' for more i
'citation()' on how to cite R or                          s.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> |
```

DEVELOPING A SIMPLE PROGRAM

- Sample program for printing
  - Here, we are using the `print()` function to display “Hello World” on the R console

```
>print(“Hello World”)
[1] “Hello World”
```

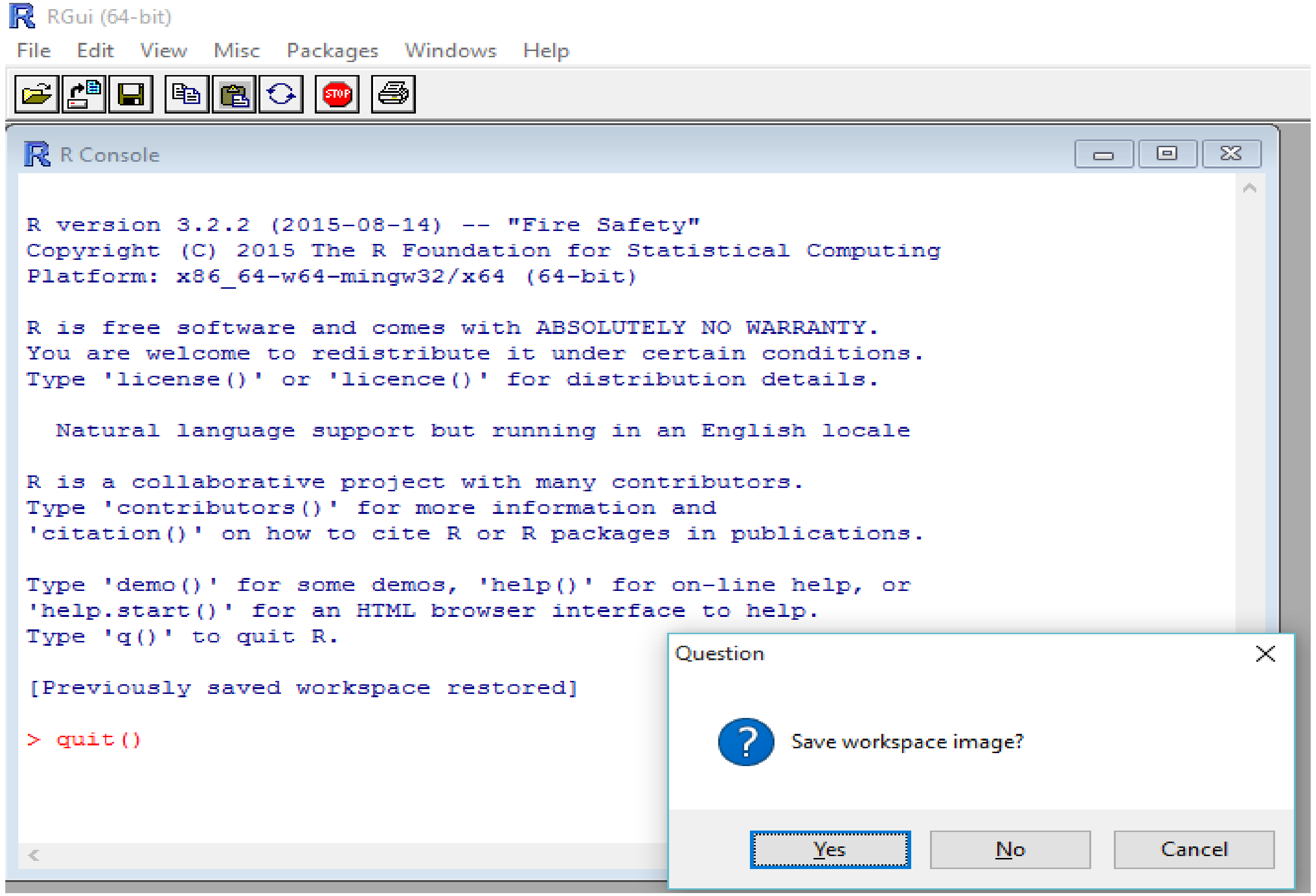
  - Here, we are doing simple math

```
>2+3
[1] 5
```

- Code begins with ‘>’ symbol and output begins with [1]

QUITTING R

- You can quit an active session of R by entering `q()` command
- After executing the `q()` command, the question dialogue box appears asking whether to save the work space.



```
R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> quit()
```

Question

Save workspace image?

HANDLING BASIC EXPRESSIONS

- Anything that you type on R console, it executes immediately on pressing the ENTER key.
- **Basic Arithmetic in R**

```
>12+45+9-7
[1] 59
```

R executes the expression in the following order

12+45+9=66

66-7=59

HANDLING BASIC EXPRESSIONS

Let’s look at complex mathematical operation

```
18+22/2-4/4*3.5
```

To calculate such complex mathematical expressions, R uses BODMAS (Brackets of Division Multiplication Addition Subtraction)

```
>18+(22/2)-((4/4)*3.5)
[1] 25.5

>(18+22/2-4/4)*3.5
[1] 98.875
```

HANDLING BASIC EXPRESSIONS

- **Mathematical Operators in R**
  - +, -, *, () - Simple Mathematical operations
  - pi - Stands for Pie value
  - X^Y - X raised to Y
  - sqrt(x) - square root of x
  - abs(x) - Absolute value of x
  - factorial(x) - Factorial of x
  - log(x) - logarithm of x
  - cos(x), sin(x), tan(x) - Trigonometric functions

DECLARING VARIABLES IN R

- Variables are symbols that are used to contain and store the values.
- Two ways to assign the values
  - Using “=” symbol

```
>MyVar=10
```

  - Using “<-” symbol

```
>MyVar<-10
```

Here, `MyVar` is a object and it is assigned with the value 10.

Any of the above mentioned can used to assign the values.

VARIABLE TYPES IN R

- **Numbers**
  - Real numbers
  - R organizes numbers in 3 formats
    - **Scalar** : Represents a single number (0 dimensional)
    - **Vector** : Represents row of numbers (1 dimensional)
    - **Matrix**: Represents the table like format (2 dimensional)
  - **Working with Vectors**
    - It consists of ordered collection of numbers or strings
    - Numerical Vector
    - String/character vector

VARIABLE TYPES IN R

- **Numeric Vector:**
  - Vector of numeric values.
  - A scalar number is the simplest numeric vector.
  - Example:

```
1.5
## [1] 1.5
```

  - To store it for future use,

```
X<-1.5
```

VARIABLE TYPES IN R - VECTORS

Constructing the numeric and character vectors in R

- `c()` is used to construct the vector (Integer/Character)

  `> c(10,20,20,30,40) - A` Numerical/Integer vector

  `> c('Hello2', 20, 'Hello4', 30) -` A combination of Numerical and Character vector

  `> c('Hello1', 'Hello2', 'Hello3') -` Character vector

INTEGER AND DOUBLE VECTORS

A number by default is considered double in R.

```
> batch<-c(“cdac”,”dbda”, 2022)

> typeof(batch)

> newbatch<-c(03,11,2022)

> typeof(newbatch)

> newbatch<- c(03L,11L,2022L)  # To read as integer
```

VARIABLE TYPES IN R - VECTORS

- We can also combine single-element vectors and multi element vectors and obtain a vector with the same elements as previously created.
- Example

```
> c(1, 2, c(3, 4, 5))
[1] 1 2 3 4 5


 > y=c(1, 2, c(3, 4, 5),c(5,6,7))


 > Z = c(10:20,y, sum(10:20))


 > length(Z)
```

Answer?

VARIABLE TYPES IN R - VECTORS

- Creating the vector using `(:)` operator

```
>1:15 (generates numbers from 1 to 15)
> c(1:15)
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> c(1,15)
     Answer?
> c(1-15)
     Answer?
```

```
> sum(1:15) ## it sums the numbers from 1 to 15
    [1] 120
> mean(1:15)
```

VARIABLE TYPES IN R

- **Strings (characters)**
  - A string should be specified by using quotes. Both single and double quotes will work

```
a <- "hello"        ## Assigning a string to variable a
a           ## Printing variable a
"hello"       ## Output of variable a
b <- c("hello","there")       ## Assigning two strings to variable b
b                           ## Printing variable b
"hello" "there"            ## Output of variable b
b[1]              ## Printing first element of variable b
"hello”        ## Output of variable b[1]
```

VARIABLE TYPES IN R

Logical Vectors

- In contrast to numeric vectors, a logical vector stores a group of TRUE or FALSE values.
- The simplest logical vectors are TRUE and FALSE themselves
- A more usual way to obtain a logical vector is to ask logical questions about R objects.
- For example, we can ask R whether 1 is greater than 2:

```
1>2
## [1] FALSE
c(1, 2) > 2
## [1] FALSE FALSE
```

VARIABLE TYPES IN R – LOGICAL VECTORS

- Examples

```
c(1, 2) > c(2, 1)
## [1] FALSE TRUE
```

Execution `c(1 > 2, 2 > 1)`

```
c(2, 3) > c(1, 2, -1, 3)
## [1] TRUE TRUE TRUE FALSE
```

Execution `c(2 > 1, 3 > 2, 2 > -1, 3 > 3)`

```
y=c(1, 2, c(3, 4, 5),c(5,6,7))
y == "a‟  # what will be the output?
```

VARIABLE TYPES IN R

- **Named Vectors**
  - It is a vector with names corresponding to the elements.
  - We can give names to a vector when we create it

```
> dbda <- c(janaki=1, nanda=2, madhavi=3, raghu=4 )
```

To print the vector

```
> dbda   or > print(dbda)   or   > show(dbda)
```

```
##janaki nanda madhavi raghu
 ##      1      2            3      4
```

NAMED VECTOR

> names(dbda)- print only names without values

> unname(dbda) – Print values without names

> str(dbda) – see the structure of object ‘x’

> dbda[order(dbda, decreasing = TRUE)]

➢ table(is.na(dbda)) - Number of NA’s in object “dbda”

> dbda[c(5)]<-NA – assign NA to 2nd element
> table(is.na(dbda)) – now check again for NAs

EXTRACTING AN ELEMENT

- While `[]` creates a subset of a vector, `[[]]` extracts an element from a vector. (indexing operators used by “R”)

Example:

```
> dbda[c(2)] – to access 2nd element of the object
or
> dbda["nanda”] – access element of object “dbda” based on
  names  of the object.

> dbda[[“nanda"]] - to get value of 2nd element “nanda”
```

EXTRACTING AN ELEMENT BASED ON THE VALUE

- Example: Extract elements which are greater than certain value

```
input <- c(21, 44, 69, 9, 12, 16, 19, 224, 261, 300)

input > 220
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
TRUE

input[input > 220]
[1] 224 261 300
```

VARIABLE TYPES IN R - FACTORS

- Another important way R can store data is in the form of factors to represent **categorical** data
- Example of Factor data Yes/No, Male/Female, Grades - A/B/C/D/E/F, Marital status etc.

```
> data<-c("lion","tiger","fox","wolf","tiger","wolf","lion","tiger","fox")
> data
[1] "lion" "tiger" "fox" "wolf" "tiger" "wolf" "lion" "tiger" "fox"
> is.factor(data)
[1] FALSE
```

```
> factor_data <- factor(data)
> is.factor(factor_data)
[1] TRUE
> factor_data
[1] lion tiger fox wolf tiger wolf lion tiger fox
Levels: fox lion tiger wolf
```

- **Store data in the form of factors using factor function or using data frames**

VARIABLE TYPES IN R – DATA FRAMES

- **Data Frames**
  - It is the collection of many vectors of different types, stores in single variable

```
> a<-c(1,2,3,4)
> b<-c(2,4,6,8)
> levels <- factor(c(‘A’,’B’,’B’,’A’))
> MyDataFrame<-data.frame(a, b, levels)
> MyDataFrame
  a b levels
1 1 2      A
2 2 4      B
3 3 6      A
4 4 8      B
```

TELLING THE CLASS OF VECTORS

- Sometimes we need to tell which kind of vector we are dealing with before taking an action.
- The `class()` function tells us the class of any R object:

```
class(c(1, 2, 3))
## [1] "numeric"
class(c(TRUE, TRUE, FALSE))
## [1] "logical"
class(c('Hello', 'World’))
## [1] "character“
Class(MyDataFrame)
## [1] "data.frame"
```

TELLING THE CLASS OF VECTORS

- If we need to ensure that an object is indeed a vector of a specific class, we can use `is.numeric`, `is.logical`, `is.character`, and some other functions with similar names:

```
is.numeric(c(1, 2, 3))
## [1] TRUE
is.numeric(c(TRUE, TRUE, FALSE))
## [1] FALSE
is.numeric(c("Hello", "World"))
## [1] FALSE
is.character(c('a','b','c’))
## [1] TRUE
```

CONVERTING VECTORS

- Different classes of vectors can be coerced to a specific class of vector.
- For example, some data are **string representation** of numbers, such as 1 and 20.
- We need to convert it to numeric representation in order to apply numeric functions.

```
strings <-c("1", "2", "3")
class(strings)
## [1] "character”
--------------------------
strings + 10
## Error in strings + 10: non-numeric
argument to binary operator
-------------------------
numbers <- as.numeric(strings)
numbers
## [1] 1 2 3
class(numbers)
## [1] "numeric”
----------------------------
numbers + 10
## [1] 11 12 13
```

CONVERTING VECTORS

- Different classes of vectors can be coerced to a specific class of vector.
- For example, some data are **string representation** of numbers, such as 1 and 20.
- We need to convert it to numeric representation in order to apply numeric functions.

```
as.numeric(c("1", "2", "3", "a"))
## Warning: NAs introduced by coercion
## [1] 1 2 3 NA
-------------------------
as.logical(c(-1, 0, 1, 2))
## [1] TRUE FALSE TRUE TRUE
--------------------------
as.character(c(1, 2, 3))
## [1] "1" "2" "3"
--------------------------
as.character(c(TRUE, FALSE))
## [1] "TRUE" "FALSE"
```

CALLING FUNCTIONS IN R

- Many predefined functions are there in R.
- To invoke, user has to type the function names
- For example

```
> sum(10,20,30)
1] 60
# Replicate  lements of Vectors and Lists using rep function
> rep("Hello",3)
[1] "Hello" "Hello" "Hello“
> sqrt(100)
[1] 10
> substr("example",2,4)
[1] "xam"
```

CREATING AND USING OBJECTS

- R uses objects to store the results of a computation

```
> myobj<-25+12/2-16+(7*pi/2)
> myobj
[1] 25.99557
```

Assigns a mathematical expression to an object called **myobj**

Invokes the **myobj** object

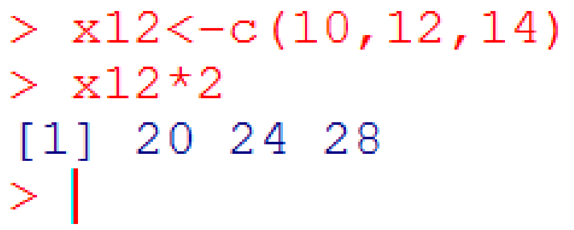
- R is case sensitive – that is, it treats myobj and Myobj as completely different objects.

```
> Myobj
Error: object 'Myobj' not found
```

CREATING AND USING OBJECTS

- An object can be assigned a set of numbers, as for example:

```
> x12 <- c(10,6,8)
> x12
[1] 10  6  8
```



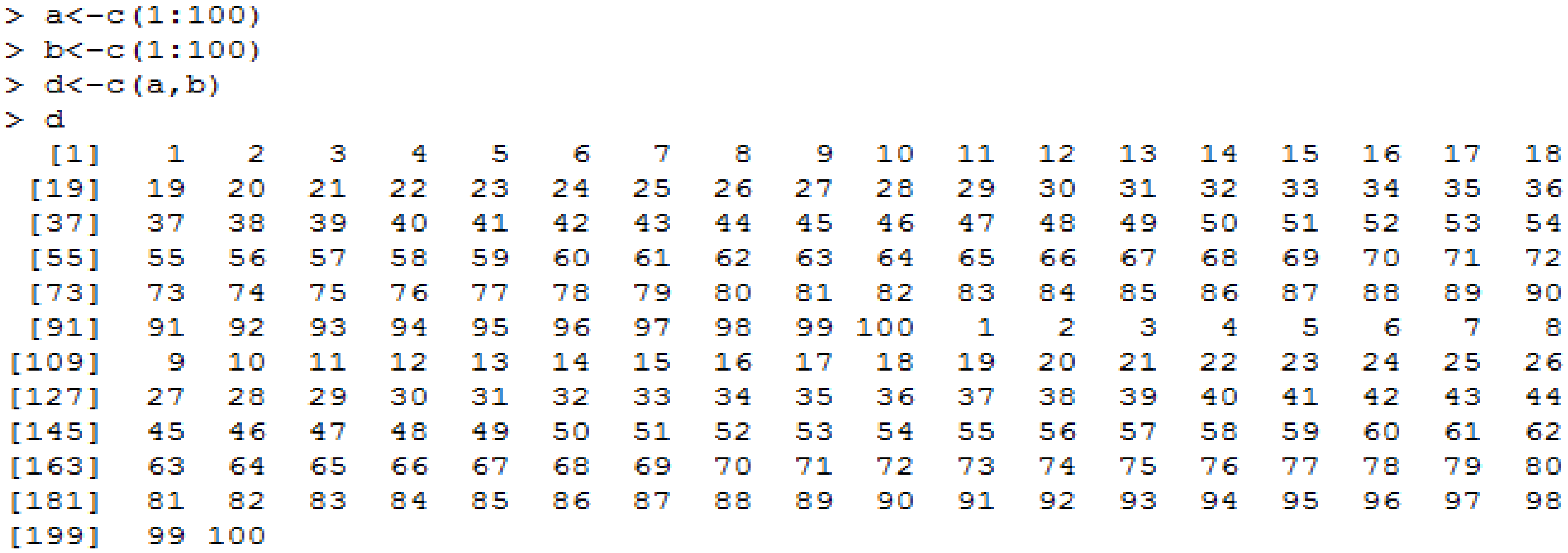
```
> x12<-c(10,12,14)
> x12*2
[1] 20 24 28
> 
```

- Operations can then be performed on the whole set of numbers.
  - For example, for the object **x12** created above, check the results of the following:

```
> x12 * 10
[1] 100  60  80
```

READING DATASETS

- Using the `c()` command:
  - `c()` function is used to combine or concatenate two or more values. Here example shown is concatenating 2 numerical vectors.
  - Syntax for the `c()` command



```
> a<-c(1:100)
> b<-c(1:100)
> d<-c(a,b)
> d
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100   1   2   3   4   5   6   7   8
[109]   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
[127]  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44
[145]  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62
[163]  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
[181]  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98
[199]  99 100
```

HANDLING DATA IN R WORKSPACE

- Handling Workspace includes following
  - Using the working directory
  - Inspecting the working environment
  - Modifying global options
  - Managing the library of packages

HANDLING DATA IN R WORKSPACE

- Handling Workspace includes following
  - Using the working directory
    - The directory in which R is running is called the **working directory** of the R session.
    - When you access other files on your hard drive, you can use absolute paths (for example, `D:\Workspaces\test-project\data\2015.csv`)
    - In an R terminal, you can get the current working directory of the running R session using `getwd()`

INSPECTING THE ENVIRONMENT

- In R, every expression is evaluated within a specific environment.
- An environment is a collection of symbols and their bindings.
- If you type commands in the RStudio console, your commands are evaluated in the **Global Environment**.
- Example:
  - If we run `x <- c(1, 2, 3)`, the numeric vector `c(1, 2, 3)` is bound to symbol `x` in the global environment.
  - Global environment has one binding that maps `x` to integer vector `c(1,2,3)`

HANDLING DATA IN R WORKSPACE

- The ls() or objects() function is used to return the list of objects in the workspace

```
> ls()
[1] "a”  "b”    "bubba"    "fun"      "levels"   "msg“
[7] "myobj"    "n"        "x12"      "yourname“
```

- The rm() function is used to remove the variables that are not required anymore in a session

```
> rm(a)
> ls()
[1] "b“   "bubba”  "fun“ "levels” "msg”  "myobj"    "n"
[8] "x12"      "yourname”
```

HANDLING DATA IN R WORKSPACE

- getwd() function: Function used to display the current working directory of the user

```
> getwd()
 [1] "C:/Users/Janaki/Documents‟
```

- save() function: Function used to save the objects created in the active session.

```
> save(x12, file="Examples.rda")
```

  - It will save in the current working directory with the name "Examples.rda"

  save.image() function

  To save all the objects in the active session

  - save.image(file = "my_stuff.RData")

HANDLING DATA IN R WORKSPACE

- load() function : Function used to retrieve the saved data

```
yourname<-"mary‟
> ls()
[1] "b” "fun” "levels” "msg” "myobj” "n” "x12”  "yourname"
> save(yourname, file="yourname.rda")
> rm(yourname)
> ls()
[1] "b” "fun” "levels" "msg” "myobj” "n” "x12‟
> load("yourname.rda") #.rda stands for R Data File.
> ls()
[1] "b” "fun” "levels” ‟msg” "myobj” "n” "x12” "yourname”
```

Executing R Scripts

- Creating and Executing R script on Windows:
  - Open Notepad, and write R commands
  - Save it has “filename.R”
  - From the Rgui, file->Open script. It opens a window for browsing the Rscript
  - Click Open

EXECUTING R SCRIPTS

- Creating and Executing R script on Linux:
- R script is the series of commands written and saved in .R extension
- To run a script "/home/bioinfo/janaki/R/use1.R"
- You may either use:
  - From R Shell

```
source("/home/bioinfo/janaki/R/use1.R")
```

  - On the Linux Shell

```
R CMD BATCH /home/bioinfo/janaki/R/use1.R
```

(OR)

```
Rscript use1.R
```

ACCESSING HELP AND DOCUMENTATION IN R

- Function used to get help pages of the in-built functions are `help()` and `example()`

  `> help(“ls”)` **or** `?ls()`

  `> example(ls) –` It shows the examples of ls function

  find.package(“packagename”) – shows the path where package has been installed.

- Sample datasets : R has many in-built datasets

  > library(datasets)

  `> data()`

  > data(iris)

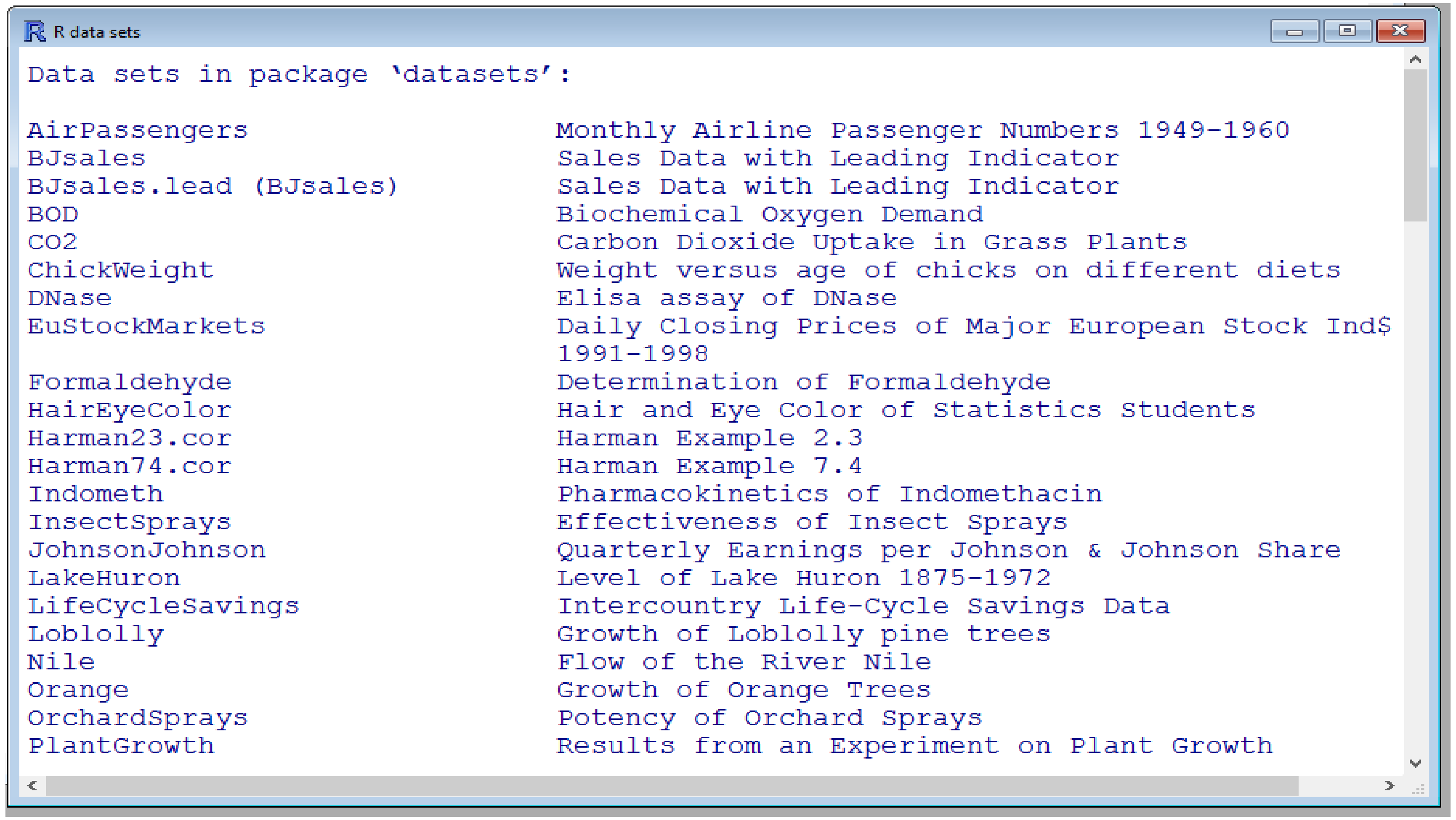
  > summary(iris) – Summary of iris data

  > summary(iris$Sepal.Length) – summary of one variable of iris data

USING BUILT-IN DATASETS IN R

- There many built-in data sets which can be viewed by `data()` command. The output is shown

`>data()` ##Generates the list of built-in datasets

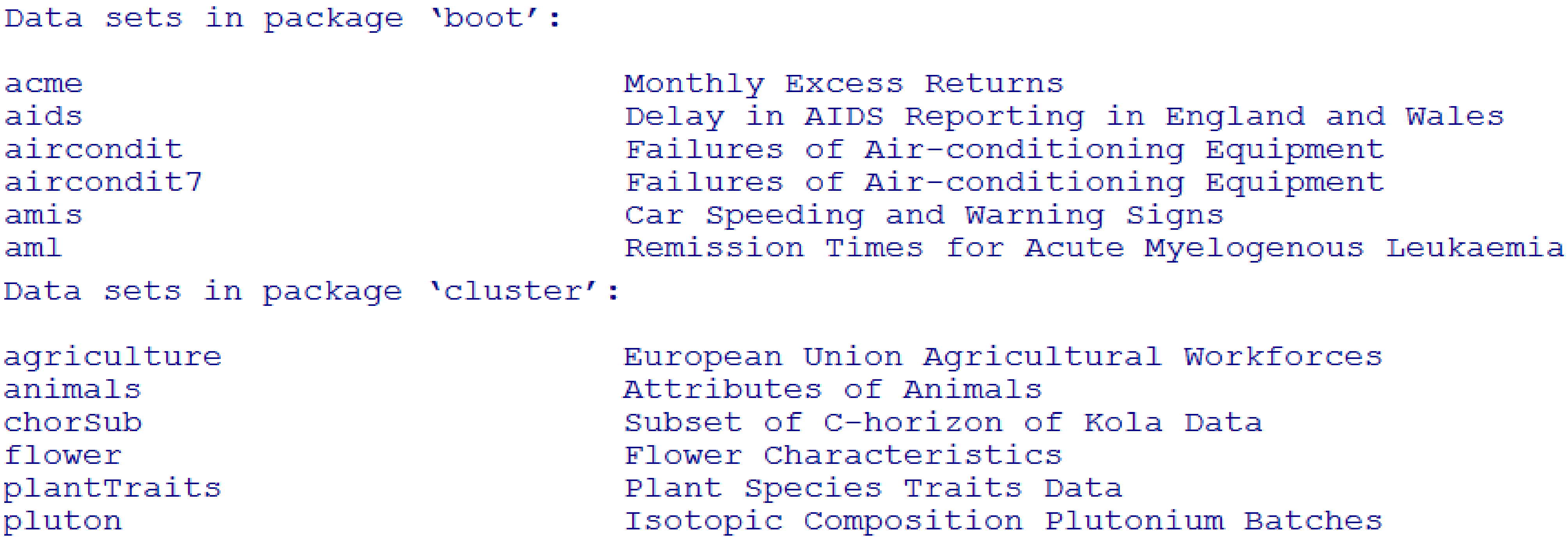


```
Data sets in package 'datasets':

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
DNase                         Elisa assay of DNase
EuStockMarkets                Daily Closing Prices of Major European Stock Ind$
                              1991-1998
Formaldehyde                  Determination of Formaldehyde
HairEyeColor                  Hair and Eye Color of Statistics Students
Harman23.cor                  Harman Example 2.3
Harman74.cor                  Harman Example 7.4
Indometh                      Pharmacokinetics of Indomethacin
InsectSprays                  Effectiveness of Insect Sprays
JohnsonJohnson                Quarterly Earnings per Johnson & Johnson Share
LakeHuron                     Level of Lake Huron 1875-1972
LifeCycleSavings              Intercountry Life-Cycle Savings Data
Loblolly                      Growth of Loblolly pine trees
Nile                          Flow of the River Nile
Orange                        Growth of Orange Trees
OrchardSprays                 Potency of Orchard Sprays
PlantGrowth                   Results from an Experiment on Plant Growth
```

Using Built-in Datasets in R

- There is a command for viewing all the data sets that are user-built or contributed packages.

```
data(package = .packages(all.available = TRUE))
data(package='boot')
```



```
Data sets in package 'boot':

acme                          Monthly Excess Returns
aids                          Delay in AIDS Reporting in England and Wales
aircondit                     Failures of Air-conditioning Equipment
aircondit7                    Failures of Air-conditioning Equipment
amis                          Car Speeding and Warning Signs
aml                           Remission Times for Acute Myelogenous Leukaemia
Data sets in package 'cluster':

agriculture                   European Union Agricultural Workforces
animals                       Attributes of Animals
chorSub                       Subset of C-horizon of Kola Data
flower                        Flower Characteristics
plantTraits                   Plant Species Traits Data
pluton                        Isotopic Composition Plutonium Batches
```

DATA STRUCTURES IN R

DATA STRUCTURES IN R

- Types of data structures in R
  - Vector : It is the structure that can contain one or more values of a single type or mixed (characters, integers)
    - It is represented as one dimensional data
  - Matrices : It is the 2-dimensional representation of data.
  - Arrays : It can be more than 2-dimensional representation of data.
  - Lists: A list is a generic vector that is allowed to include different types of objects.
  - Data Frames: It is the rectangular 2-dimensional representation of data

R VS PYTHON

	R	Python
Datatypes	Character Integer. Numeric Logical Complex Raw	Int float Long Complex and so on

Common datatype	Vector
	a<-c(4,5,1,3,4,5)
	print(a[3])
		List
		a=[4,5,1,3,4,5]
		# print(a[2])

Dataframes	Can be created directly Or use dplyr,reshape2 package for complex dataframes	Pandas package

DATA STRUCTURES IN R- INTEGER VECTORS

- Following functions are used to create the integer vectors
  - c() : Combine (joining items end to end)
  - seq() : Sequence (Generating equidistant series of numbers)
  - rep() : Replicate (used to generate repeated values)

- c() examples

```
> c(42,57,12,39,1,3,4)
 [1] 42 57 12 39 1 3 4
```

- You can also combine vectors of more than one element

```
> x <- c(1, 2, 3)
> y <- c(10, 20)
> Z <- c(x, y)
> Z
```

DATA STRUCTURES IN R- INTEGER VECTORS

- seq(): It is used to generate the series of numbers which is of equidistant
- It accepts three arguments
  - Start element
  - Stop element
  - Jump element

```
> seq(4,9)#It generates the numbers from 4 to 9, only 2 arguments are given
[1] 4 5 6 7 8 9

> seq(4,10,2)   #Three arguments are given, jump by 2 elements
[1] 4 6 8 10
```

DATA STRUCTURES IN R- INTEGER VECTORS

- seq() vector creation is used in plotting the x and y axis in the graphical analysis.
- For example:
  - If x-axis co-ordinates are being created as

```
c(1.65,1.70,1.75,1.80,1.85,1.90)
```

  - Then simply using following command, can create the same

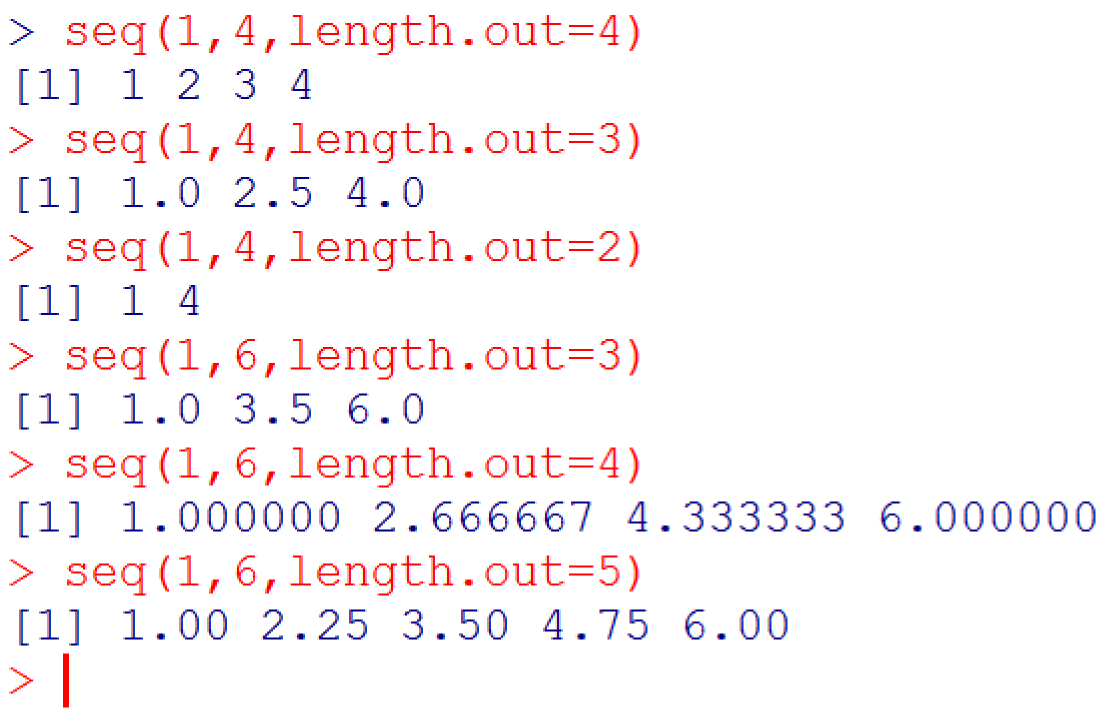
Syntax : seq(from, to, by)

```
seq(1.65,1.90,0.05)
> 4:9                #exactly the same as seq(4,9)
[1] 4 5 6 7 8 9
```

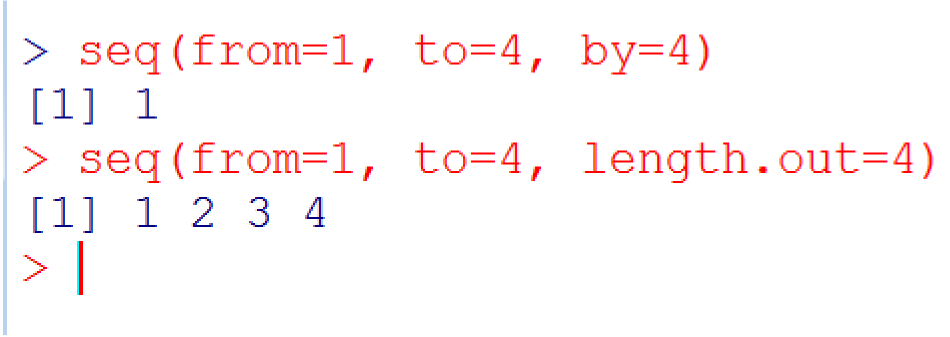
DATA STRUCTURES IN R- INTEGER VECTORS

- Another Example of **seq()** command, Here we are adding **length.out** argument for the **seq()** command

from = “Starting Element”
to = “Ending Element”
by = ((to - from)/(length.out - 1))



```
> seq(1,4,length.out=4)
[1] 1 2 3 4
> seq(1,4,length.out=3)
[1] 1.0 2.5 4.0
> seq(1,4,length.out=2)
[1] 1 4
> seq(1,6,length.out=3)
[1] 1.0 3.5 6.0
> seq(1,6,length.out=4)
[1] 1.000000 2.666667 4.333333 6.000000
> seq(1,6,length.out=5)
[1] 1.00 2.25 3.50 4.75 6.00
> |
```



```
> seq(from=1, to=4, by=4)
[1] 1
> seq(from=1, to=4, length.out=4)
[1] 1 2 3 4
> |
```

From = 1, to = 4
By = 4-1/4-1= 3/3 = 1
Seq(1,4,1)

DATA STRUCTURES IN R- INTEGER VECTORS

- rep(), is used to generate repeated values.

```
> rep(“Janaki", 4)
```

- It is used in two variants, depending on whether the second argument is a vector or a single number

```
> oops <- c(7,9,13)
> rep(oops,3)    # It repeats the entire vector oops 3 times
[1] 7 9 13 7 9 13 7 9 13
> rep(oops,1:3)
[1] 7 9 9 13 13 13
```

Here, oops should be repeated by vector of 1:3 values.

Indicating that 7 should be repeated once, 9 twice, and 13 three times

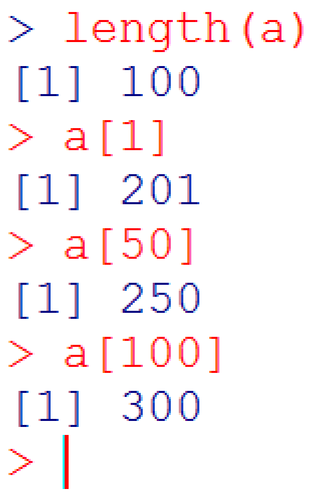
DATA STRUCTURES IN R- INTEGER VECTORS

- Look at following examples

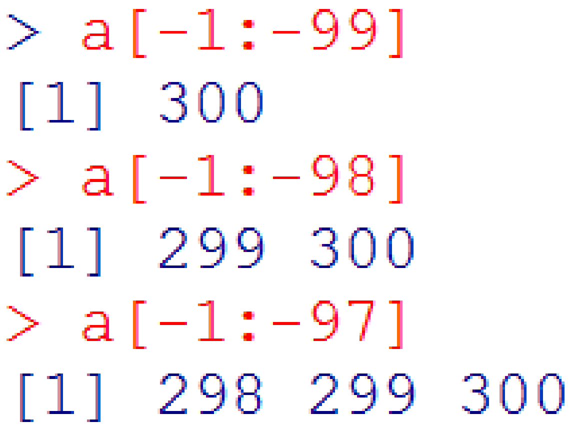
```
> rep(oops,1:4)
Error in rep(oops, 1:4) : invalid 'times' argument
> rep(1:2,c(10,15))
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
> rep(1:2,each=10)
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
> rep(1:2,c(10,10)
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
> rep(1:2,c(10,2)
```

DATA STRUCTURES IN R- INTEGER VECTORS

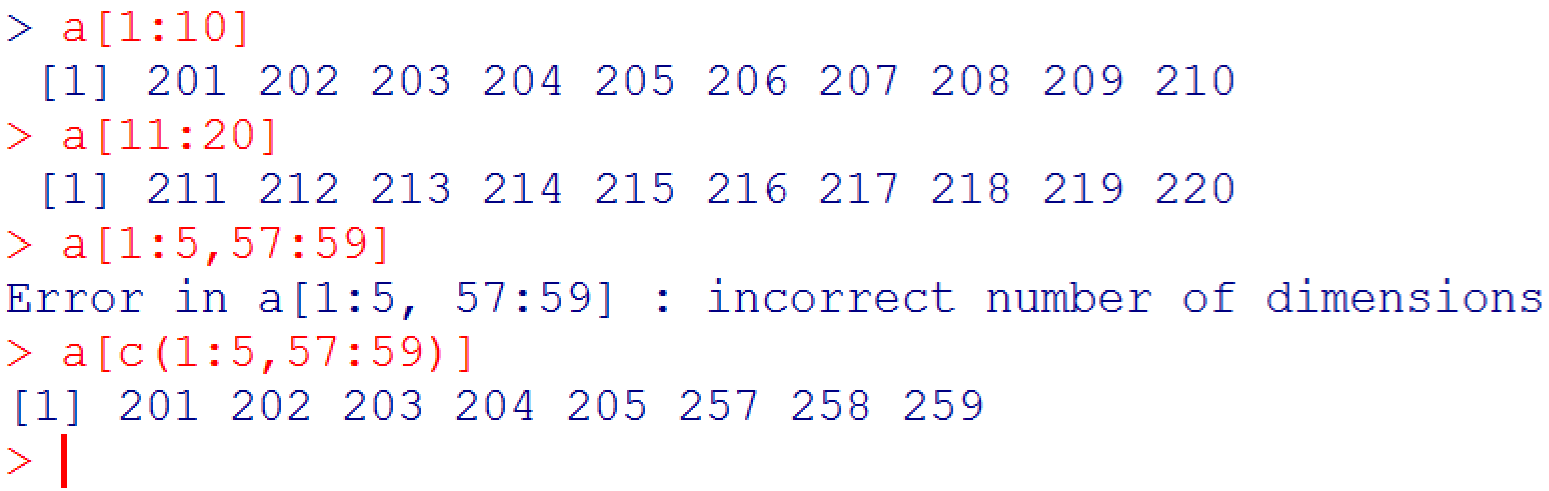
- Integer vectors : Indexing



```
> length(a)
[1] 100
> a[1]
[1] 201
> a[50]
[1] 250
> a[100]
[1] 300
> |
```



```
> a[-1:-99]
[1] 300
> a[-1:-98]
[1] 299 300
> a[-1:-97]
[1] 298 299 300
```



```
> a[1:10]
 [1] 201 202 203 204 205 206 207 208 209 210
> a[11:20]
 [1] 211 212 213 214 215 216 217 218 219 220
> a[1:5,57:59]
Error in a[1:5, 57:59] : incorrect number of dimensions
> a[c(1:5,57:59)]
[1] 201 202 203 204 205 257 258 259
> |
```

DATA STRUCTURES IN R- CHARACTER VECTORS

- Character Vector: A character vector is a vector of text strings, whose elements are specified and printed in quotes

```
> c("Huey","Dewey","Louie")
[1] "Huey" "Dewey" "Louie“
```

- Single quotes or Double quotes can be used for strings

```
> c(’Huey’,’Dewey’,’Louie’)
[1] "Huey" "Dewey" "Louie“
```

- "Huey", it is a string of four characters, not six.
- The quotes are not actually part of the string, they are just there so that the system can tell the difference between a string and a variable name.

DATA STRUCTURES IN R- CHARACTER VECTORS

- If you print a character vector, it usually comes out with **quotes** added to each element. There is a way to avoid this, namely to use the `cat()` function.
- For instance,

```
> cat(c("Huey","Dewey","Louie"))
Huey Dewey Louie
```

ESCAPE SEQUENCES

- **Quoting and escape sequences**
  - If the strings itself contains some quotations, new line characters.
  - This is done using escape sequences

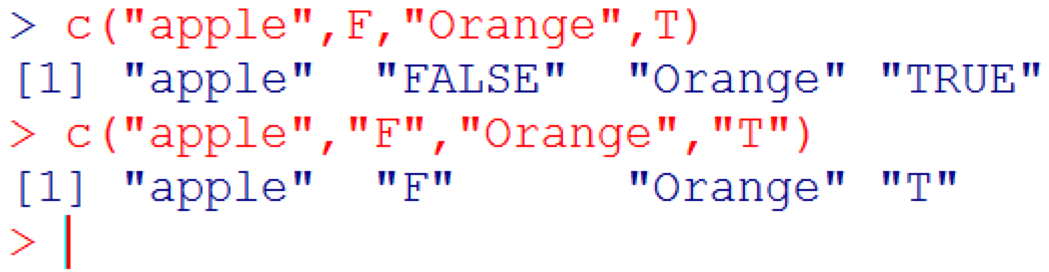
- Here, **\n** is an example of an escape sequence.
- The backslash (\) is known as the escape character
- If you want to insert quotes with in the string, the \” is used. For example

```
> cat("What is \"R\"?\n")
  What is "R"?
```

DATA STRUCTURES IN R- CHARACTER VECTORS

- Logical vectors can take the value TRUE or FALSE
- In input, you may use the convenient abbreviations T and F

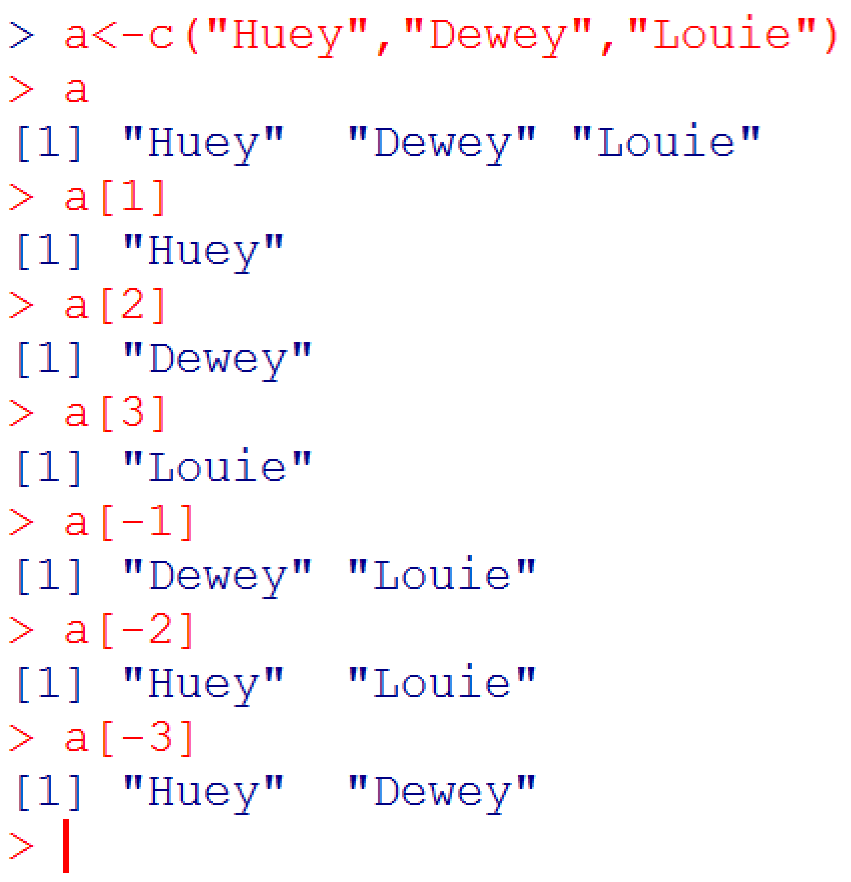
```
>
[1] TRUE TRUE FALSE TRUE
```



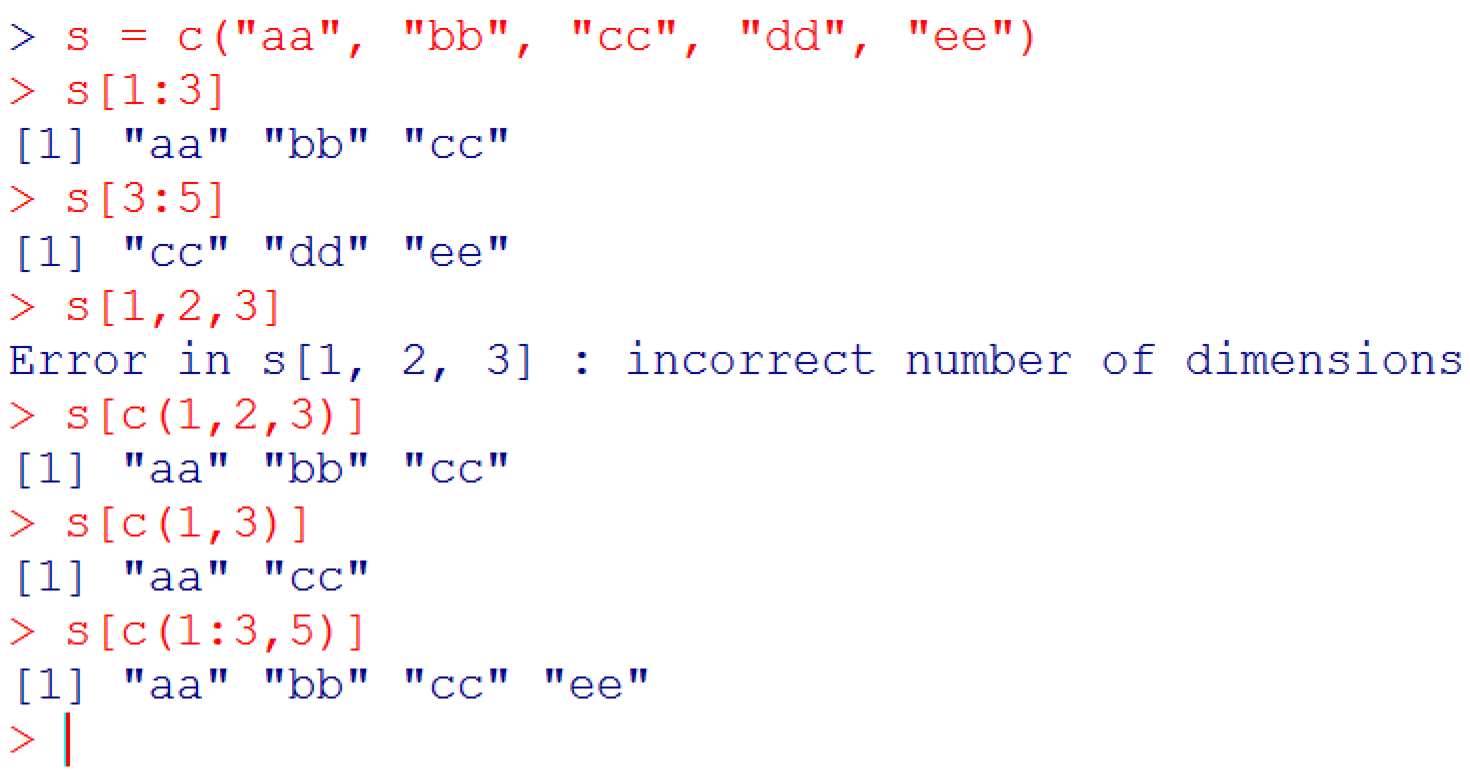
```
> c("apple",F,"Orange",T)
[1] "apple"  "FALSE"  "Orange" "TRUE"
> c("apple","F","Orange","T")
[1] "apple"  "F"      "Orange" "T"
> |
```

DATA STRUCTURES IN R- CHARACTER VECTORS

- Example of Character Vector: **Indexing**



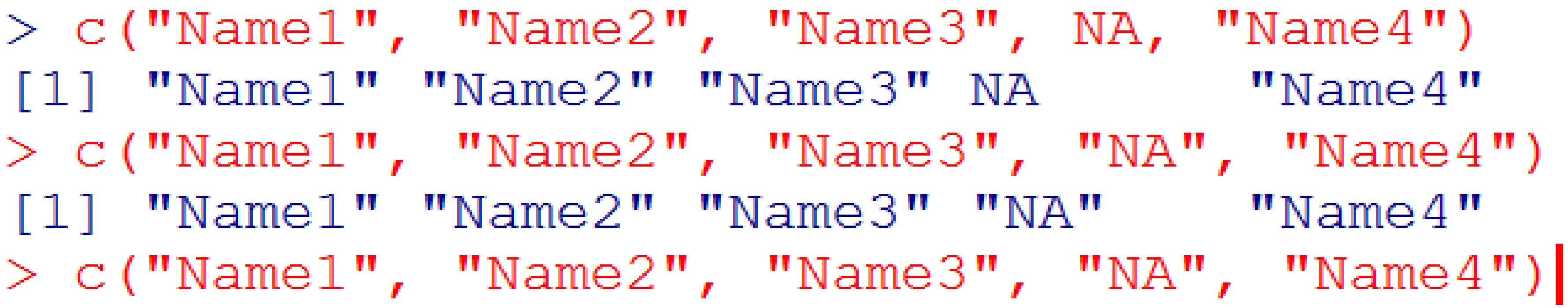
```
> a<-c("Huey","Dewey","Louie")
> a
[1] "Huey"  "Dewey" "Louie"
> a[1]
[1] "Huey"
> a[2]
[1] "Dewey"
> a[3]
[1] "Louie"
> a[-1]
[1] "Dewey" "Louie"
> a[-2]
[1] "Huey"  "Louie"
> a[-3]
[1] "Huey"  "Dewey"
> 
```



```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[1:3]
[1] "aa" "bb" "cc"
> s[3:5]
[1] "cc" "dd" "ee"
> s[1,2,3]
Error in s[1, 2, 3] : incorrect number of dimensions
> s[c(1,2,3)]
[1] "aa" "bb" "cc"
> s[c(1,3)]
[1] "aa" "cc"
> s[c(1:3,5)]
[1] "aa" "bb" "cc" "ee"
> 
```

DATA STRUCTURES IN R- CHARACTER VECTORS

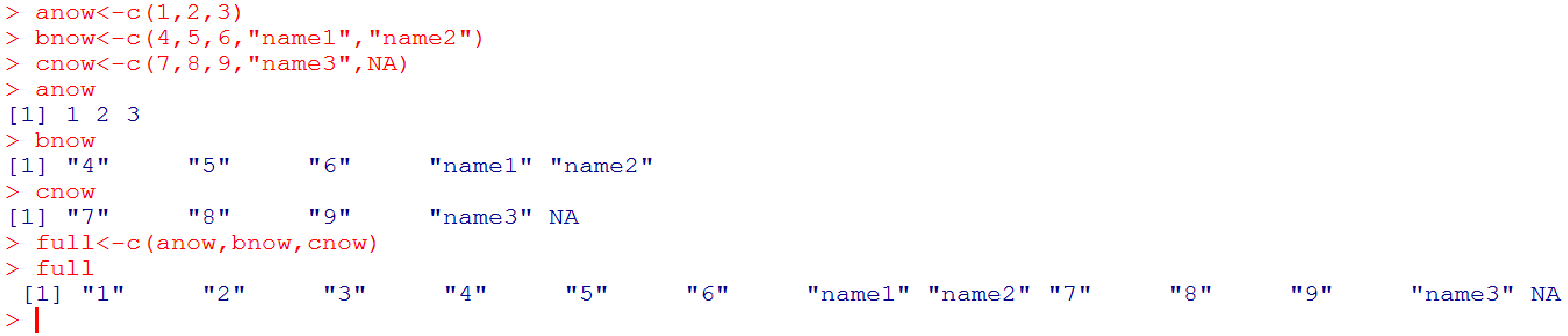
- **Missing values**
  - In many data sets, you may find missing values.
  - We need to have some method to deal with the missing values
- R allows vectors to contain a special NA value.
- Result of computations done on NA will be NA



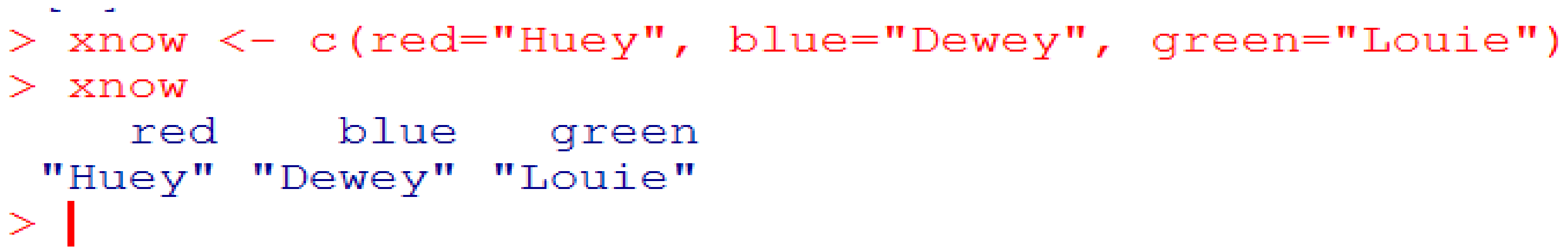
```
> c("Name1", "Name2", "Name3", NA, "Name4")
[1] "Name1" "Name2" "Name3" NA      "Name4"
> c("Name1", "Name2", "Name3", "NA", "Name4")
[1] "Name1" "Name2" "Name3" "NA"    "Name4"
> c("Name1", "Name2", "Name3", "NA", "Name4")
```

DATA STRUCTURES IN R- COMBINATION OF INT AND CHAR

- Example of c()



```
> anow<-c(1,2,3)
> bnow<-c(4,5,6,"name1","name2")
> cnow<-c(7,8,9,"name3",NA)
> anow
[1] 1 2 3
> bnow
[1] "4"     "5"     "6"     "name1" "name2"
> cnow
[1] "7"     "8"     "9"     "name3" NA
> full<-c(anow,bnow,cnow)
> full
 [1] "1"     "2"     "3"     "4"     "5"     "6"     "name1" "name2" "7"     "8"     "9"     "name3" NA
> 
```



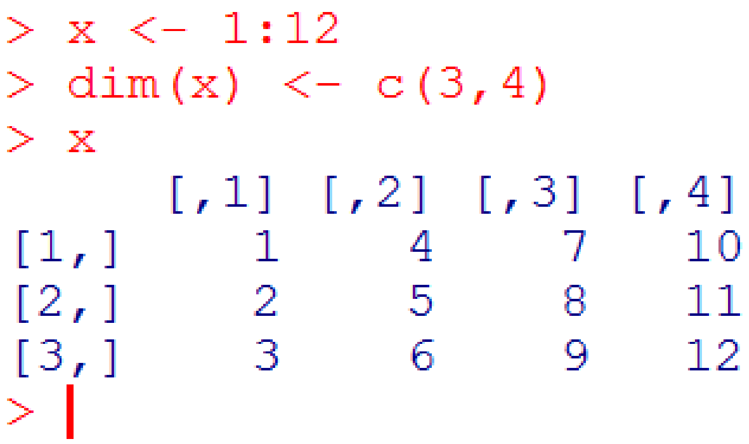
```
> xnow <- c(red="Huey", blue="Dewey", green="Louie")
> xnow
    red    blue   green
 "Huey" "Dewey" "Louie"
> 
```

DATA STRUCTURES IN R- MATRIX

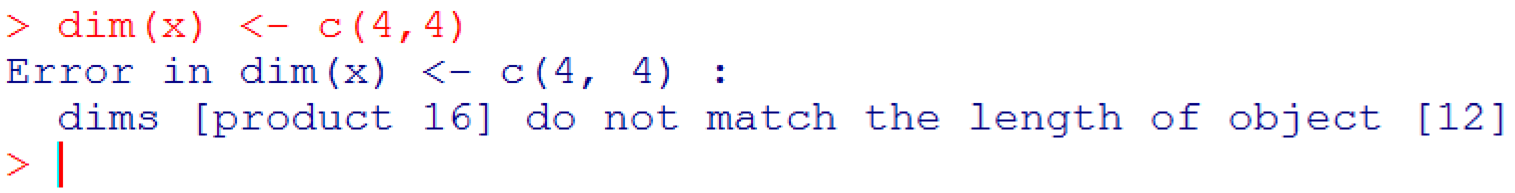
- Matrix: It is two-dimensional representation of numbers.
- Matrices and arrays are represented as vectors with dimensions

`> x <- 1:12`

`> dim(x) <- c(3,4)` *#The dim assignment function sets or changes the dimension attribute of x, causing R to treat the vector of 12 numbers as a 3 × 4 matrix*



```
> x <- 1:12
> dim(x) <- c(3,4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> 
```

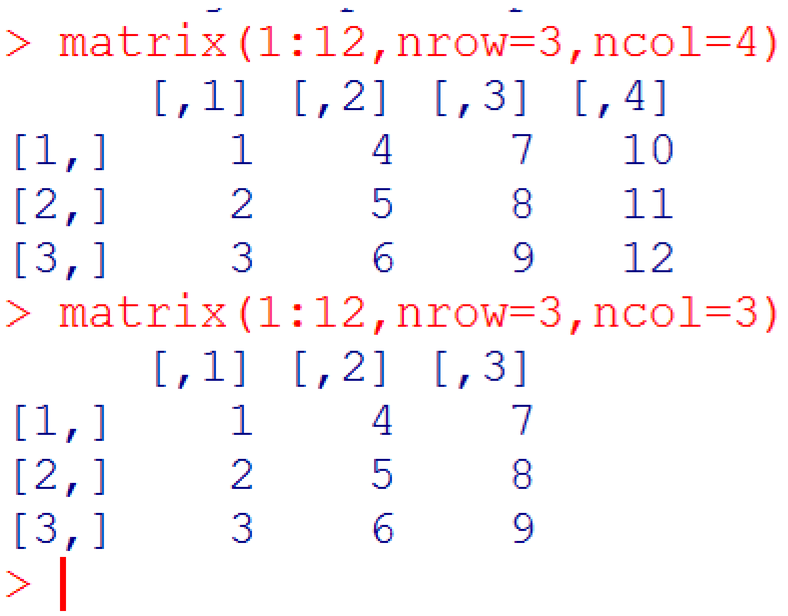


```
> dim(x) <- c(4,4)
Error in dim(x) <- c(4, 4) :
  dims [product 16] do not match the length of object [12]
> 
```

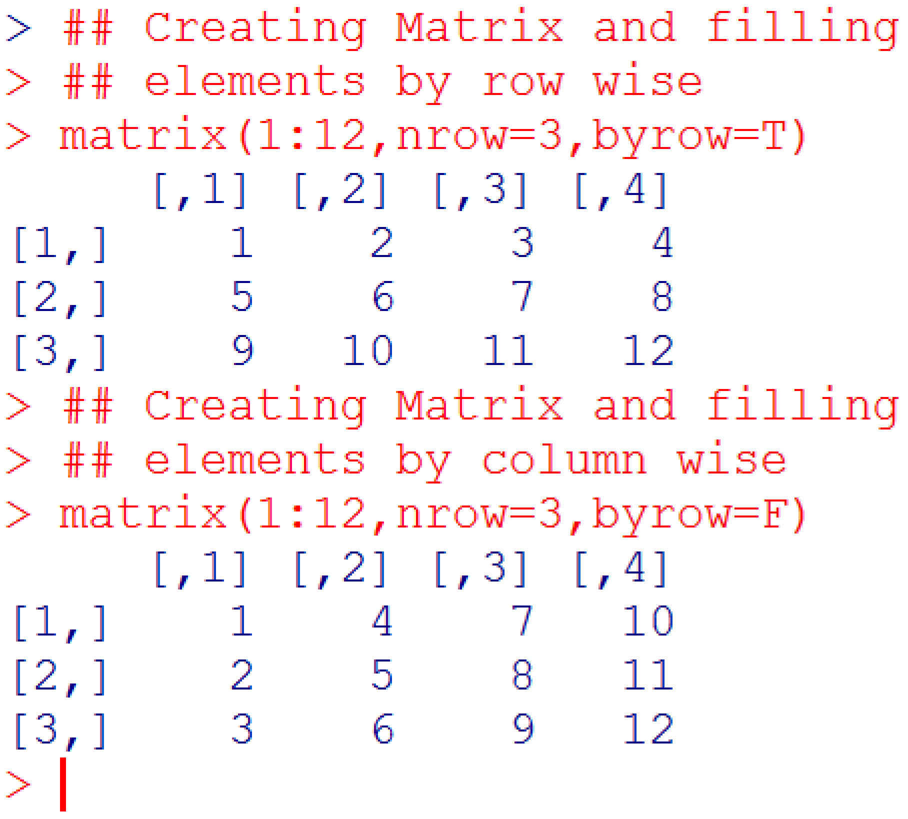
DATA STRUCTURES IN R- MATRIX

- Another way to create Matrix is simply by using `matrix()` function
- **Syntax**

matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)



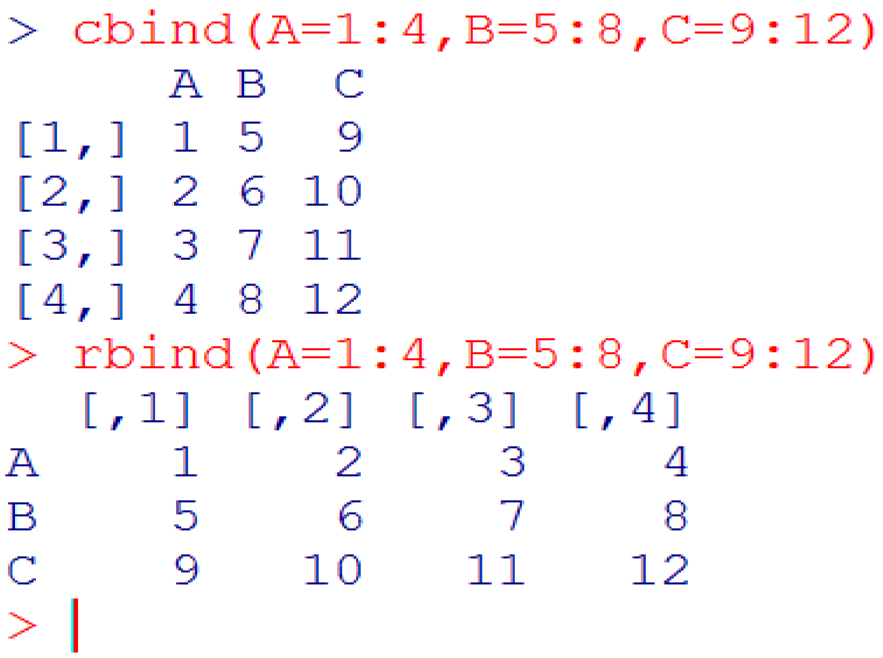
```
> matrix(1:12,nrow=3,ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> matrix(1:12,nrow=3,ncol=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> |
```



```
> ## Creating Matrix and filling
> ## elements by row wise
> matrix(1:12,nrow=3,byrow=T)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> ## Creating Matrix and filling
> ## elements by column wise
> matrix(1:12,nrow=3,byrow=F)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> |
```

DATA STRUCTURES IN R- MATRIX

- You can “glue” vectors together, columnwise or rowwise, using the `cbind` and `rbind` functions.



```
> cbind(A=1:4,B=5:8,C=9:12)
     A B  C
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
> rbind(A=1:4,B=5:8,C=9:12)
  [,1] [,2] [,3] [,4]
A    1    2    3    4
B    5    6    7    8
C    9   10   11   12
> 
```

- The `cbind()` : Column bind
- The `rbind()` : Row bind

- Arrays are similar to matrices but can have more than two dimensions. See **help(array)** for details

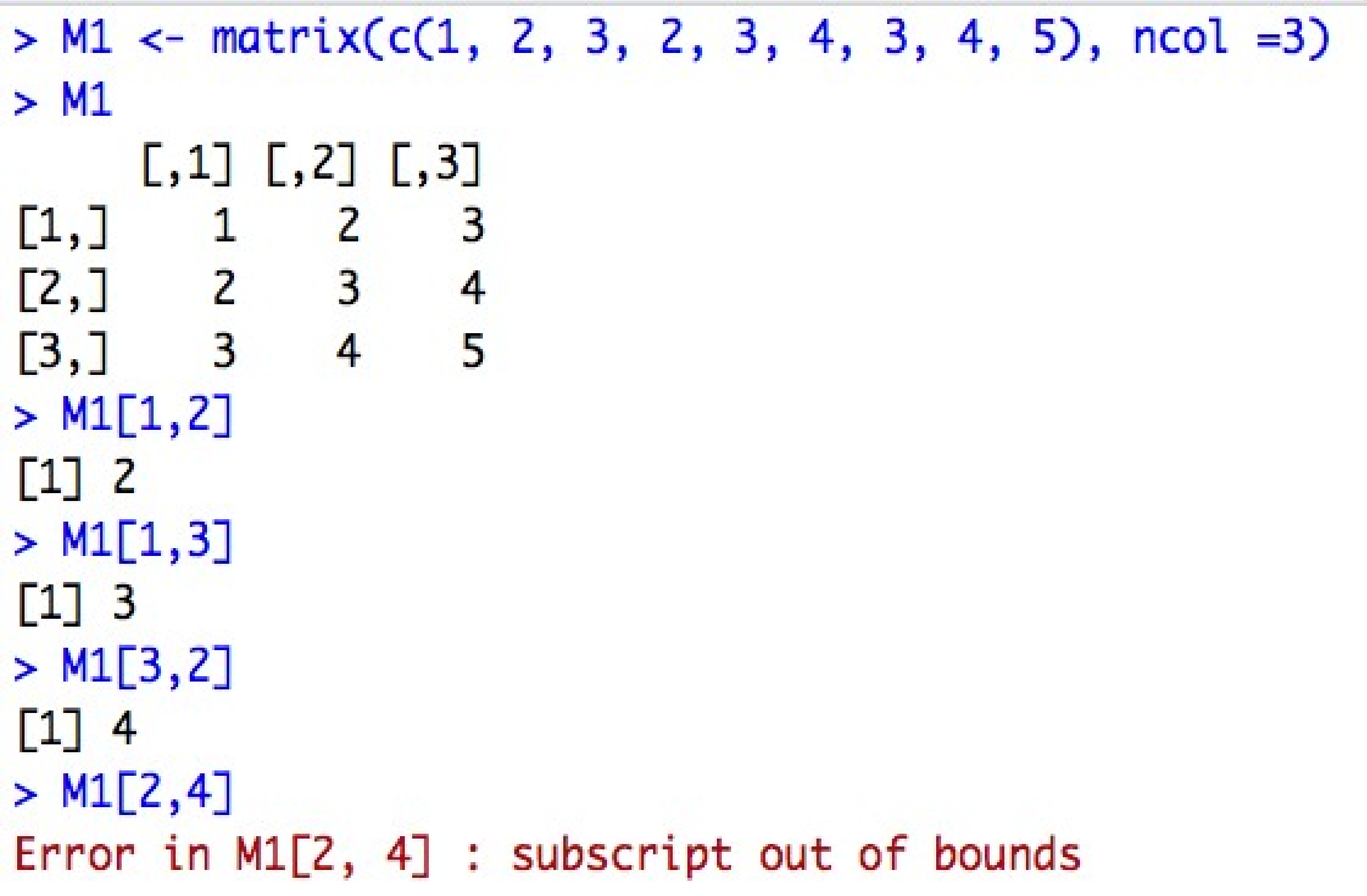
DATA STRUCTURES IN R- MATRIX

Subsetting a matrix

- We can extract the elements from the matrix – Matrix Subsetting.
- Since it is a two-dimensional representation of numbers, we can access it with two-dimensional accessor

```
mat1<- cbind(1:12, 13:24, 25:36)
mat1
[ , ]
mat1[3 , 3]
mat1[8 , 3]
```

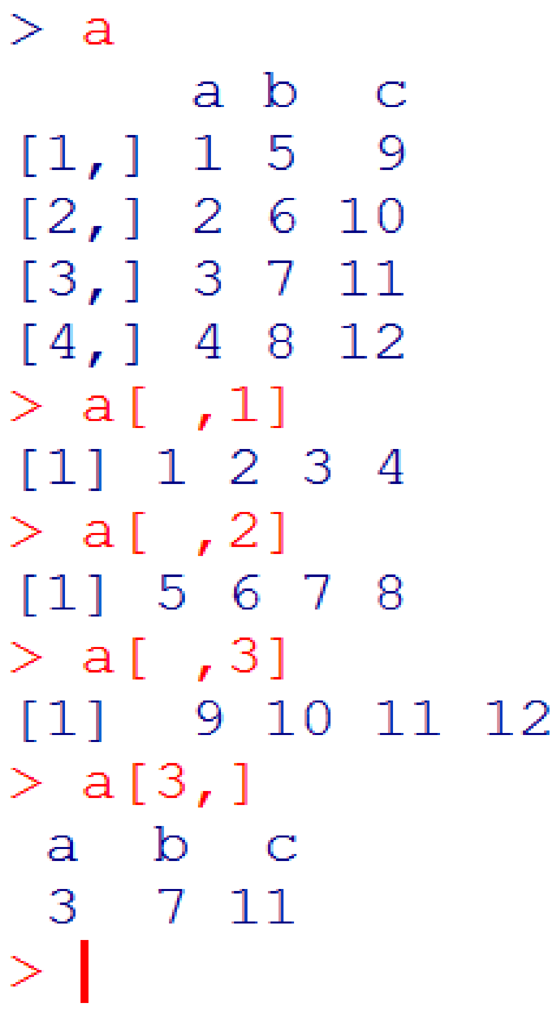
- **Example**



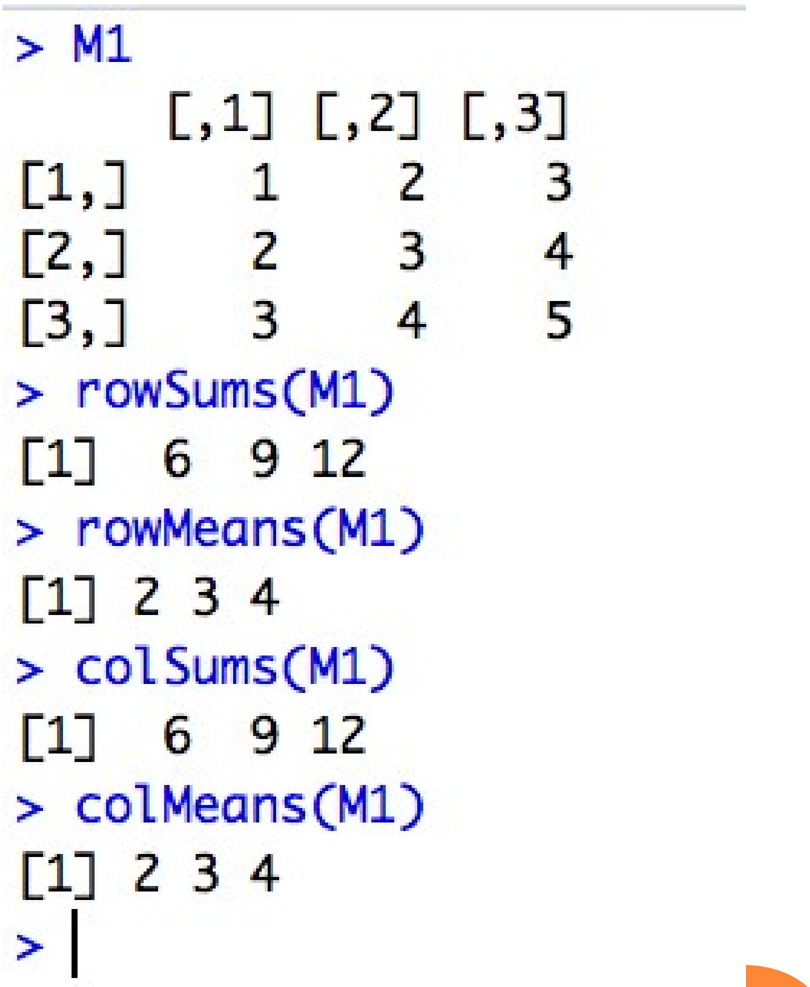
```
> M1 <- matrix(c(1, 2, 3, 2, 3, 4, 3, 4, 5), ncol =3)
> M1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    3    4
[3,]    3    4    5
> M1[1,2]
[1] 2
> M1[1,3]
[1] 3
> M1[3,2]
[1] 4
> M1[2,4]
Error in M1[2, 4] : subscript out of bounds
```

DATA STRUCTURES IN R- MATRIX

- Matrix Operations
  - Addition
  - Substraction
  - Exp
  - Element-wise *
  - Mat Mult %*%
  - rowsums()
  - rowmeans()
  - colsums()
  - colmeans()
  - t()



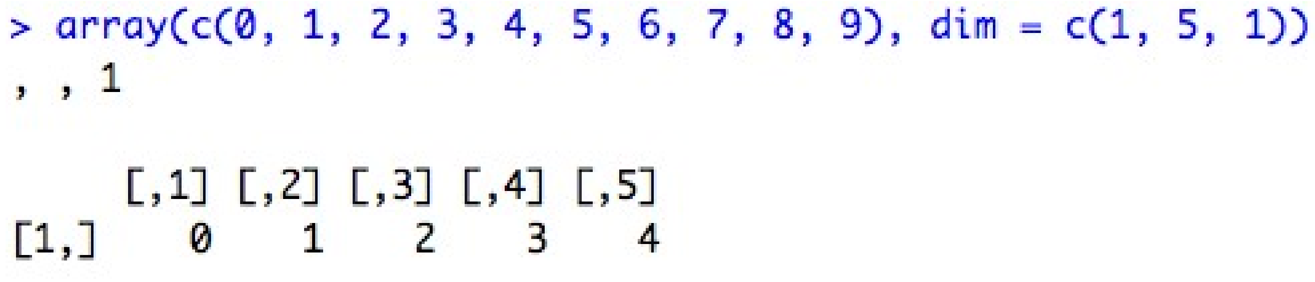
```
> a
        a b  c
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
> a[ ,1]
[1] 1 2 3 4
> a[ ,2]
[1] 5 6 7 8
> a[ ,3]
[1]  9 10 11 12
> a[3,]
 a  b  c
 3  7 11
> 
```



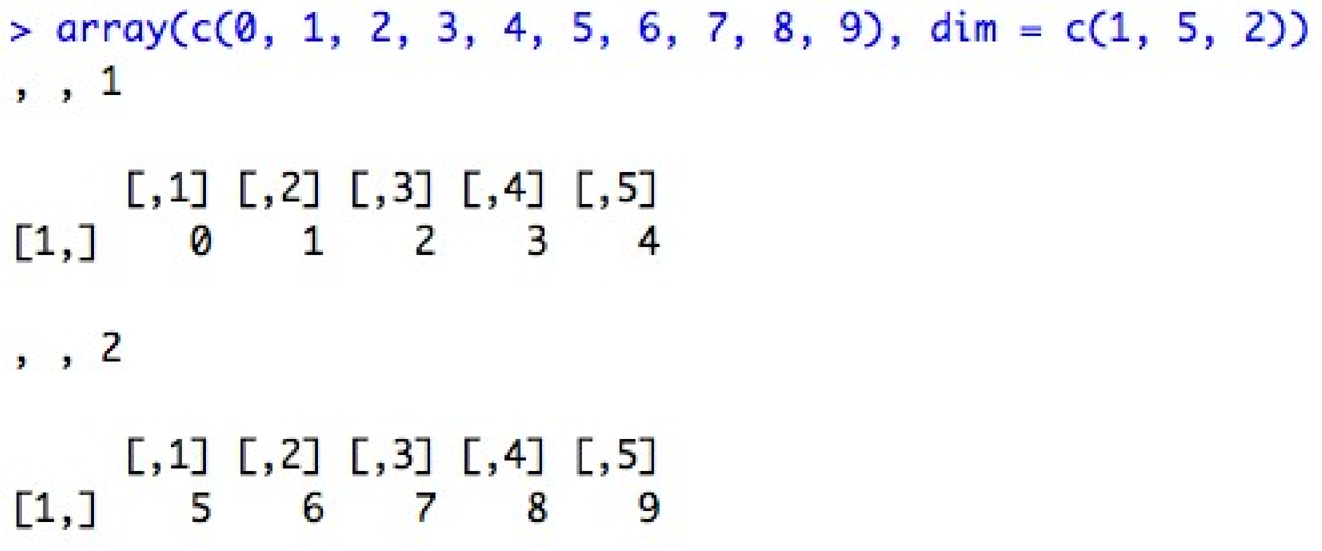
```
> M1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    3    4
[3,]    3    4    5
> rowSums(M1)
[1]  6  9 12
> rowMeans(M1)
[1] 2 3 4
> colSums(M1)
[1]  6  9 12
> colMeans(M1)
[1] 2 3 4
> 
```

DATA STRUCTURES IN R - ARRAYS

- **Arrays**
  - It is a vector that is represented and accessible in a given number of dimensions (**mostly more than two dimensions)**.



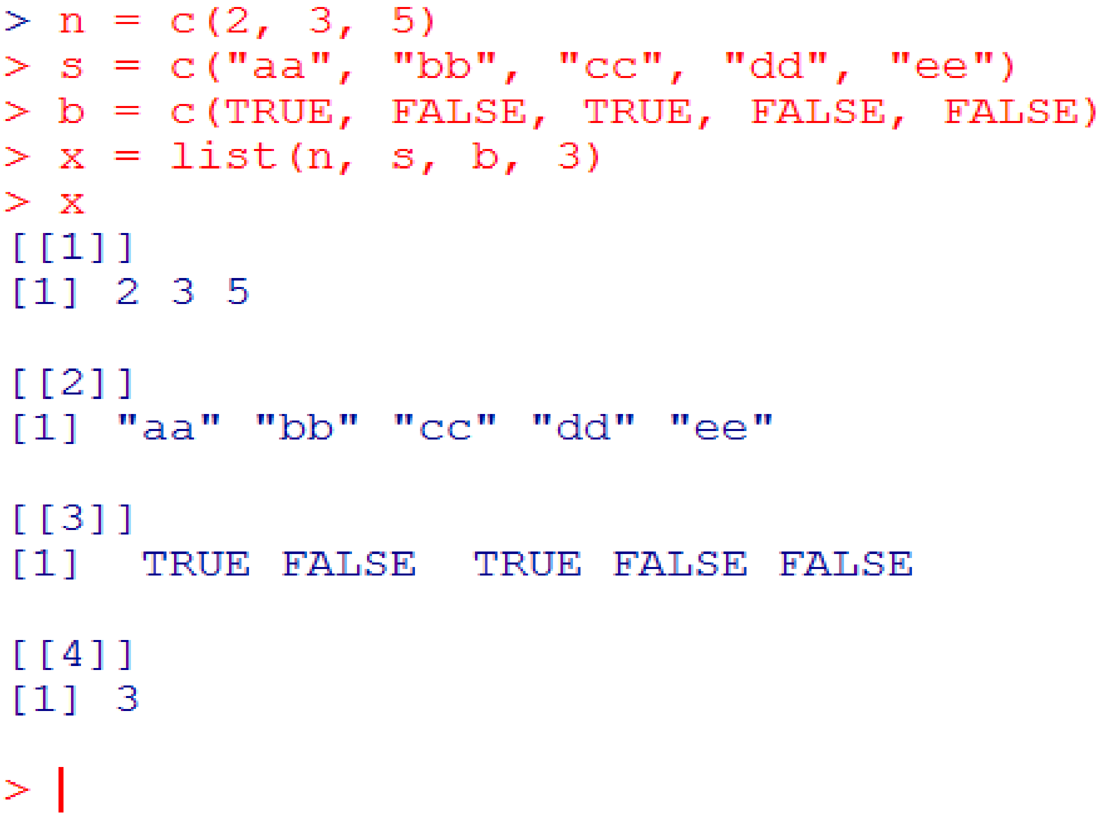
```
> array(c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9), dim = c(1, 5, 1))
, , 1

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    2    3    4
```



```
> array(c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9), dim = c(1, 5, 2))
, , 1

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    2    3    4

, , 2

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

DATA STRUCTURES IN R-LISTS

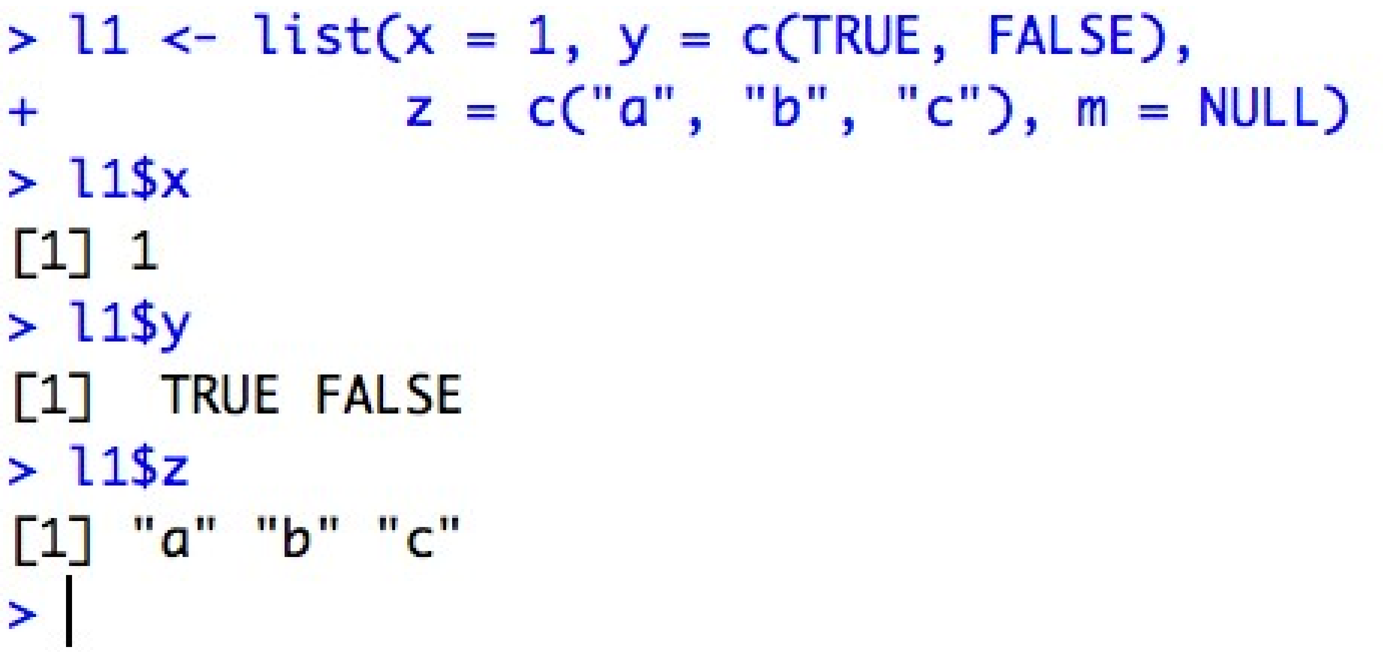
- **Lists**: It is the collection of objects that fall under similar category.
- A list is not fixed in length and can contain other lists.



```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
> x = list(n, s, b, 3)
> x
[[1]]
[1] 2 3 5

[[2]]
[1] "aa" "bb" "cc" "dd" "ee"

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

[[4]]
[1] 3

> 
```

DATA STRUCTURES IN R – ACCESSING LISTS

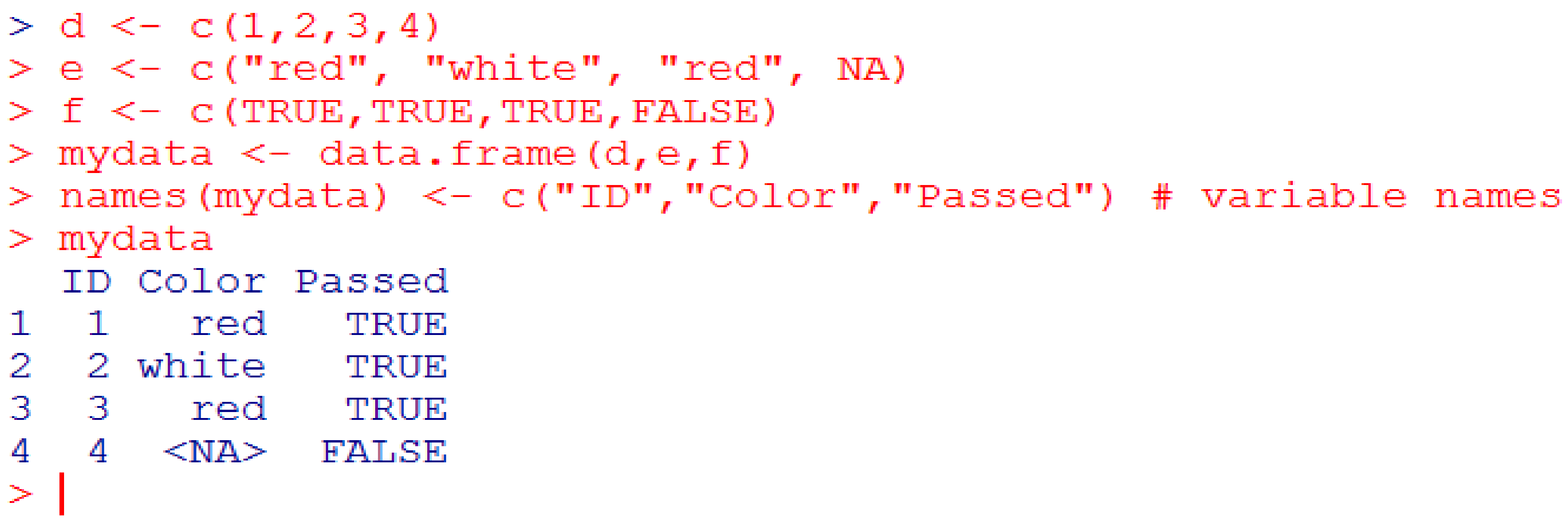
- There are various ways to access the elements of a list.
- The most common way is to use a dollar-sign **$** to extract the value of a list element by name



```
> l1 <- list(x = 1, y = c(TRUE, FALSE),
+            z = c("a", "b", "c"), m = NULL)
> l1$x
[1] 1
> l1$y
[1]  TRUE FALSE
> l1$z
[1] "a" "b" "c"
> |
```

DATA STRUCTURES IN R-DATA FRAMES

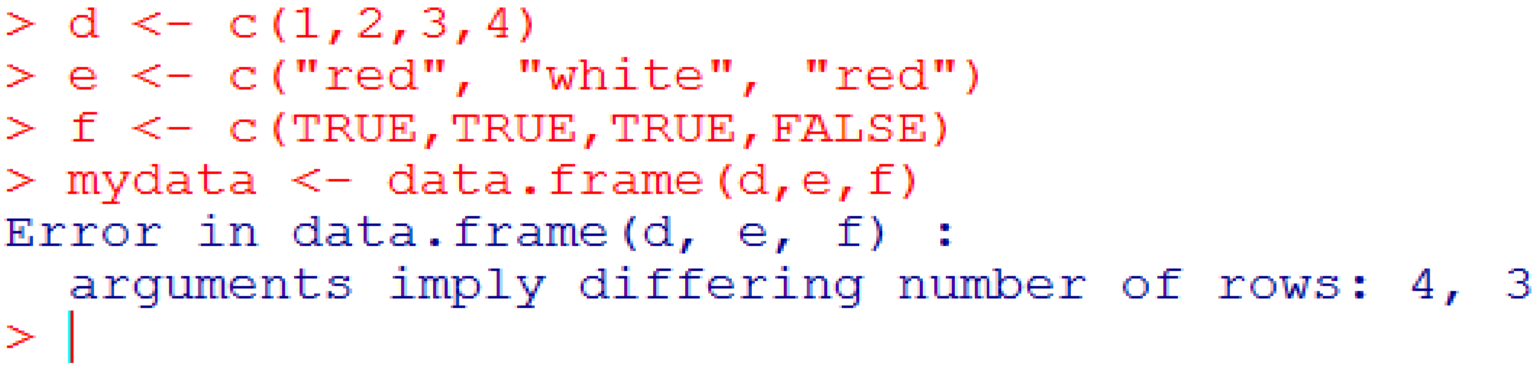
- Data Frame is also 2-dimensional object just like Matrix, for storing data tables.
- Here, different columns can have different modes (numeric, character, factor, etc).
- All data frames are rectangular and R will remove out any ‘short’ object using NA
- **Creating Data Frame**



```
> d <- c(1,2,3,4)
> e <- c("red", "white", "red", NA)
> f <- c(TRUE,TRUE,TRUE,FALSE)
> mydata <- data.frame(d,e,f)
> names(mydata) <- c("ID","Color","Passed") # variable names
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
> |
```

DATA STRUCTURES IN R-DATA FRAMES

- **Error:** Here, in the second vector ‘e’ , is a 3 element vector and ‘d’ and ‘f’ are 4 element vectors.
- **It is a collection of vectors (Integer/Character) of equal lengths**

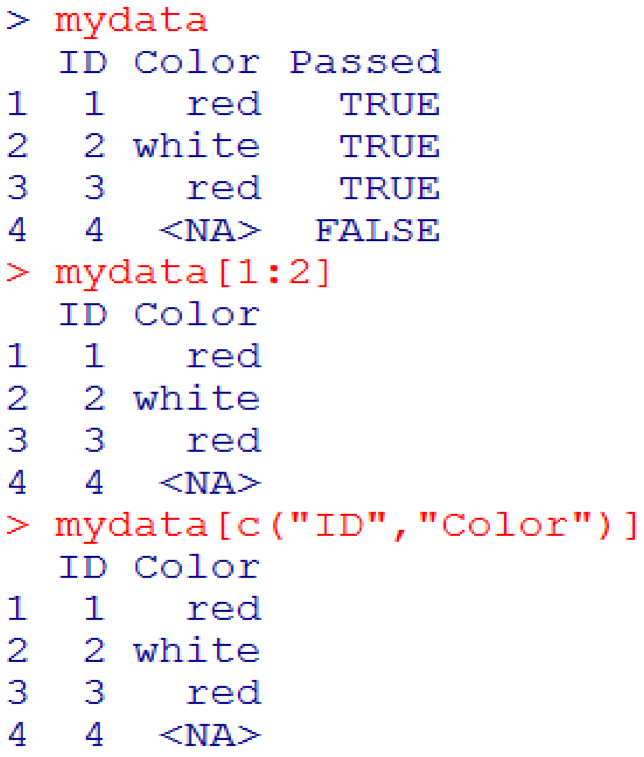


```
> d <- c(1,2,3,4)
> e <- c("red", "white", "red")
> f <- c(TRUE,TRUE,TRUE,FALSE)
> mydata <- data.frame(d,e,f)
Error in data.frame(d, e, f) :
  arguments imply differing number of rows: 4, 3
> |
```

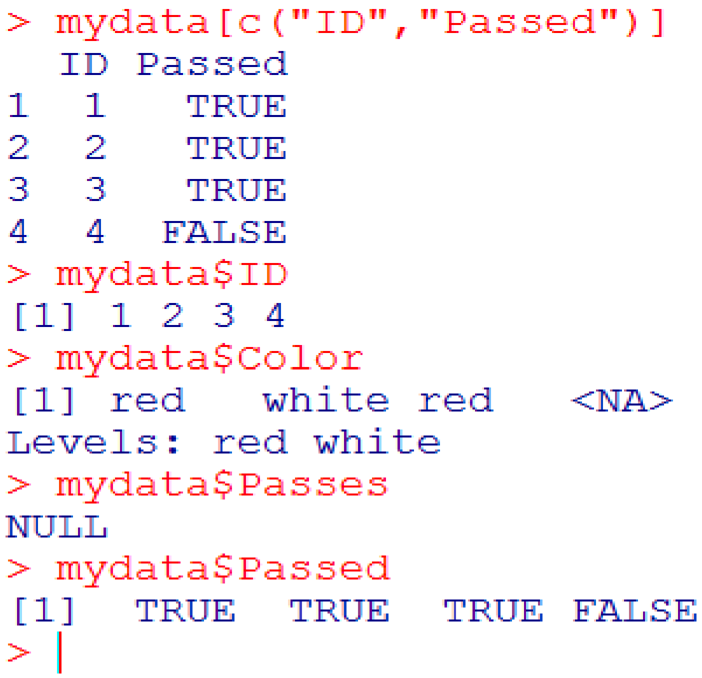
- Each column in the Data Frame can be a separate type of data. In the previous example ‘mydata’ data frame, it is the combination of numerical, character and logical data types.

ACCESSING DATA FRAMES

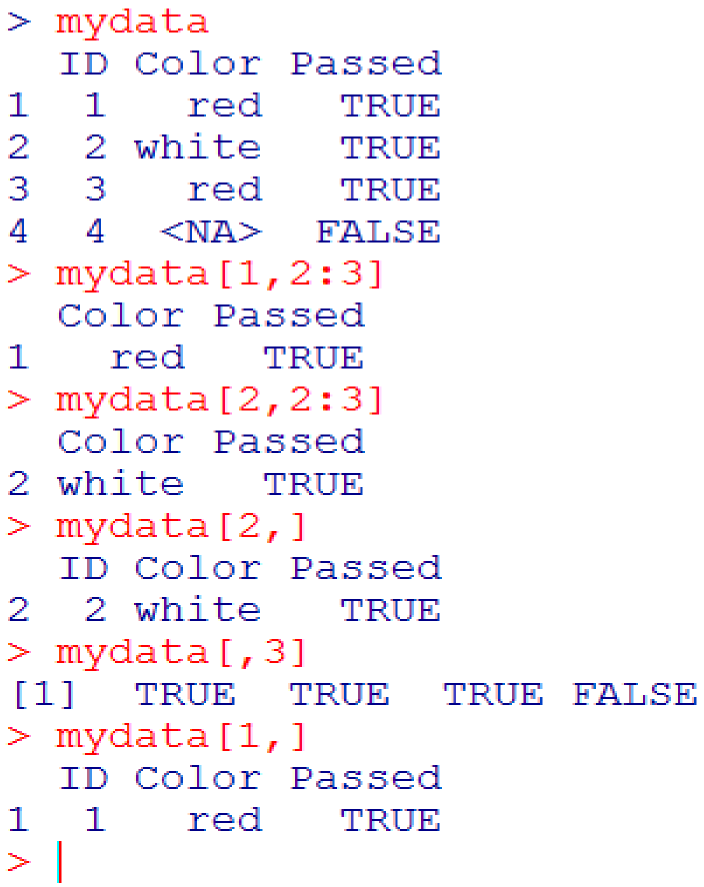
- There are a variety of ways to access the elements of a data frame. Here are few screenshots.



```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
> mydata[1:2]
  ID Color
1  1   red
2  2 white
3  3   red
4  4  <NA>
> mydata[c("ID","Color")]
  ID Color
1  1   red
2  2 white
3  3   red
4  4  <NA>
```



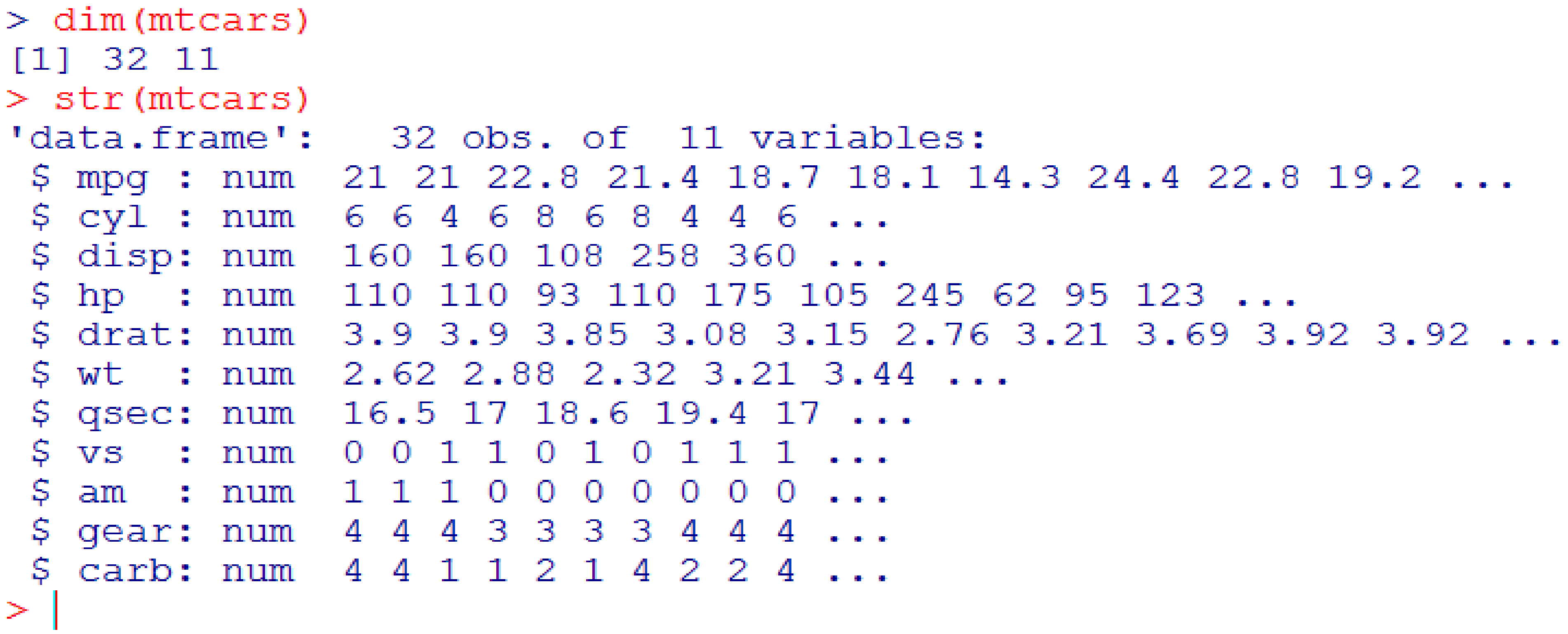
```
> mydata[c("ID","Passed")]
  ID Passed
1  1   TRUE
2  2   TRUE
3  3   TRUE
4  4  FALSE
> mydata$ID
[1] 1 2 3 4
> mydata$Color
[1] red   white red   <NA>
Levels: red white
> mydata$Passes
NULL
> mydata$Passed
[1]  TRUE  TRUE  TRUE FALSE
> 
```



```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
> mydata[1,2:3]
  Color Passed
1   red   TRUE
> mydata[2,2:3]
  Color Passed
2 white   TRUE
> mydata[2,]
  ID Color Passed
2  2 white   TRUE
> mydata[,3]
[1]  TRUE  TRUE  TRUE FALSE
> mydata[1,]
  ID Color Passed
1  1   red   TRUE
> 
```

BUILD-IN DATA FRAMES IN R

- R has some build-in datasets. ‘mtcars’ is one datasets



```
> dim(mtcars)
[1] 32 11
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
> |
```

CREATING DATA SUBSETS

- R deals with huge data, not all of which is useful.
- Therefore, first step is to sort out the data containing the relevant information.
- Extracted data sets are further divided into small subsets of data.
- Function used for extracting the data is subset().
- The following operations are used for subset the data.
  - $ (Dollar) : Used to select the single element of the data.
  - [ ] (Single Square Brackets) : Used to extract multiple elements of data.

CREATING DATA SUBSETS

- We can extract (subset) the part of the data table based on some condition using `subset()` function
- **Syntax**

```
subset(dataset, function)
```

- **Example**

```
## Age.At.Death Age.As.Writer Name Surname Gender Death
## 1 22 16 Jane Doe FEMALE 2015-05-10
## 4 41 36 Jane Austen FEMALE 1817-07-18
```

```
writer_names_df <- subset(writers_df, Age.At.Death <= 40 & Age.As.Writer >= 18)
writer_names_df <- subset(writers_df, Name =="Jane")
writers_df[1,3] <- NULL   #making null value
```

CREATING SUBSETS IN VECTORS

- To create subsets in vectors, subset() or [] can be used

```
## A simple vector
v<-c(1,5,6,4,2,4,2)

#Using subset function
subset(v,v<4)
```

Creates the subset of numbers greater than 4 using subset() function

```
#Using square brackets
v[v<4]
```

Creates the subset of numbers greater than 4 using [] brackets

```
#Another vector
t<-c(“one”, “one”, “two”, “three”, “four”, “two”)

# Remove “one” entries
subset(t, t!=“one”)
```

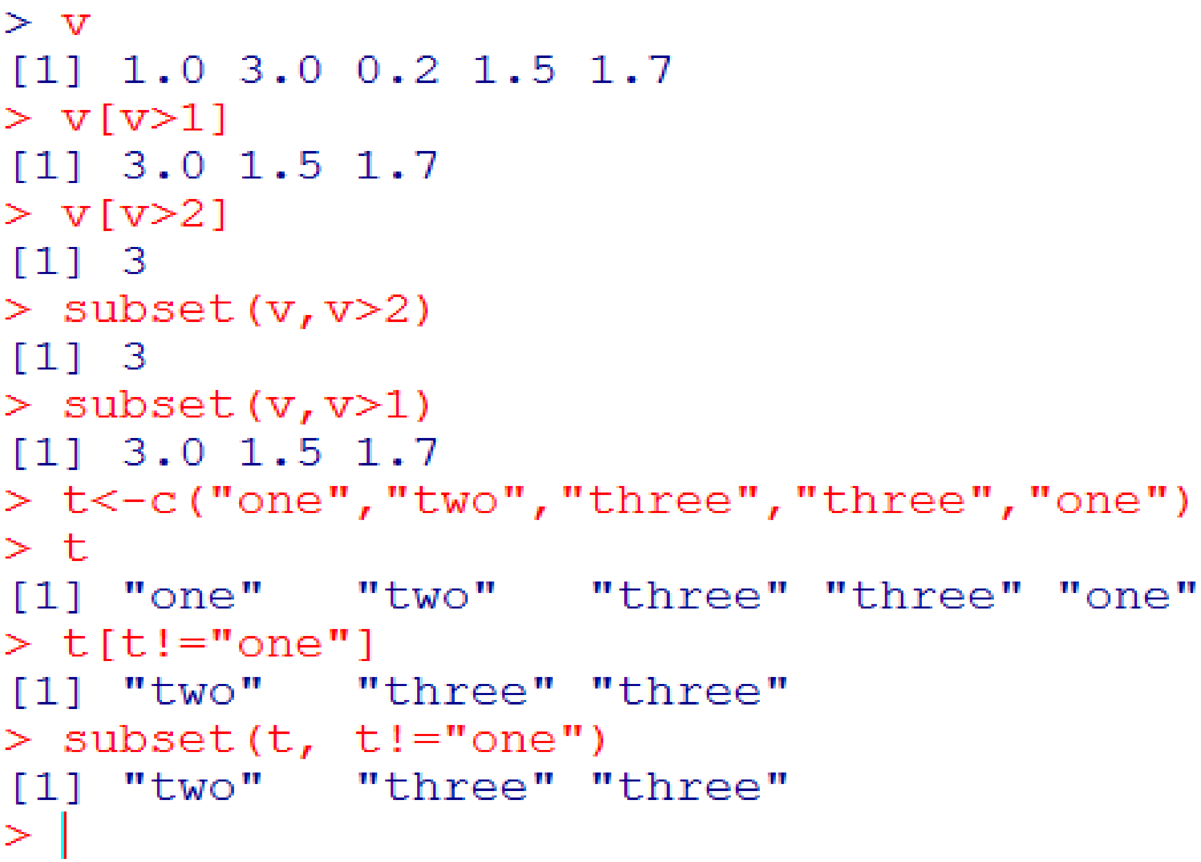
Creates the subset of texts after removing the word, “one” using subset() function

```
t[t!=“one”]
```

Creates the subset of texts after removing the word, “one” using [] function

CREATING SUBSETS IN VECTORS

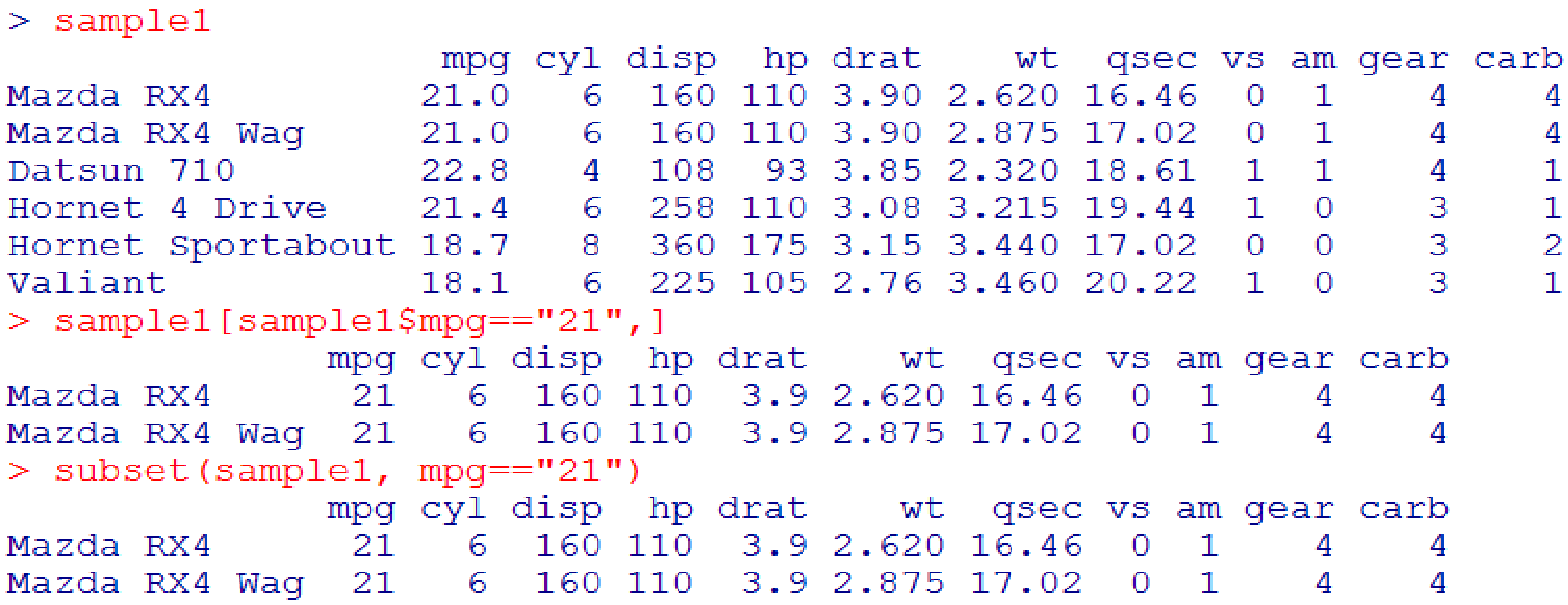
- Execution of code on R console



```
> v
[1] 1.0 3.0 0.2 1.5 1.7
> v[v>1]
[1] 3.0 1.5 1.7
> v[v>2]
[1] 3
> subset(v,v>2)
[1] 3
> subset(v,v>1)
[1] 3.0 1.5 1.7
> t<-c("one","two","three","three","one")
> t
[1] "one"   "two"   "three" "three" "one"  
> t[t!="one"]
[1] "two"   "three" "three"
> subset(t, t!="one")
[1] "two"   "three" "three"
> 
```

CREATING SUBSETS IN DATA FRAMES

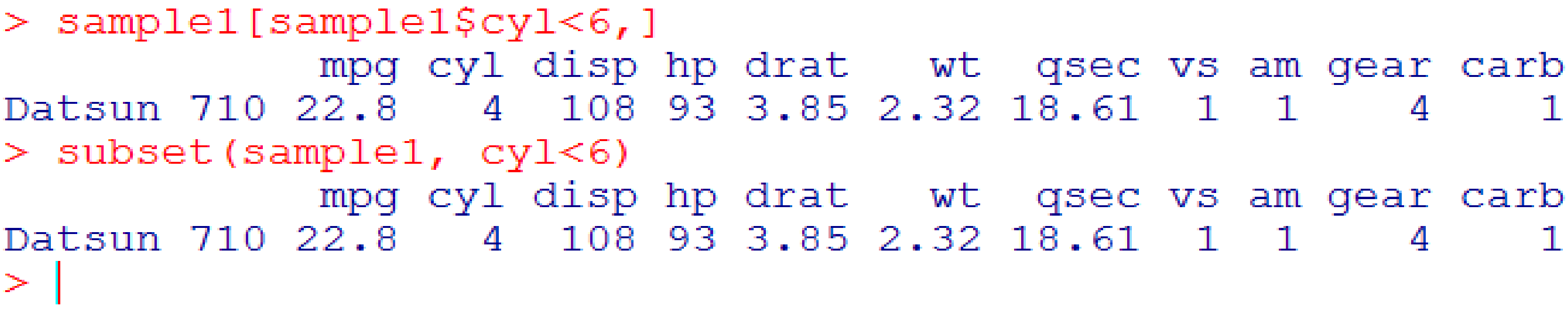
- Data Frames subsets can also be done using subset() and [] function



```
> sample1
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> sample1[sample1$mpg=="21",]
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
> subset(sample1, mpg=="21")
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

CREATING SUBSETS IN DATA FRAMES

- Data Frames subsets can also be done using subset() and [] function



```
> sample1[sample1$cyl<6,]
            mpg cyl disp hp drat   wt  qsec vs am gear carb
Datsun 710 22.8   4  108 93 3.85 2.32 18.61  1  1    4    1
> subset(sample1, cyl<6)
            mpg cyl disp hp drat   wt  qsec vs am gear carb
Datsun 710 22.8   4  108 93 3.85 2.32 18.61  1  1    4    1
> 
```

IMPORT READ AND EXPORT DATA

READING AND GETTING DATA INTO R

- Most often, you will have to deal with large sets of data which are in the form of CSV or TSV formats.

- To perform analysis on such files, you have to import/get that data into R console.

- **Commands to be discussed**
  - c() : Used to combine or concatenate data
  - scan() : Used to read large datasets and retrieve data from CSV files.
  - read.csv(), read.table(), write.csv(), write.table() : Used to read and write from csv files and tables respectively

READING AND COMBINING NUMERICAL DATA

USING THE C() COMMAND

- The c() command is used to concatenate or combine two or more values.
- **Syntax**

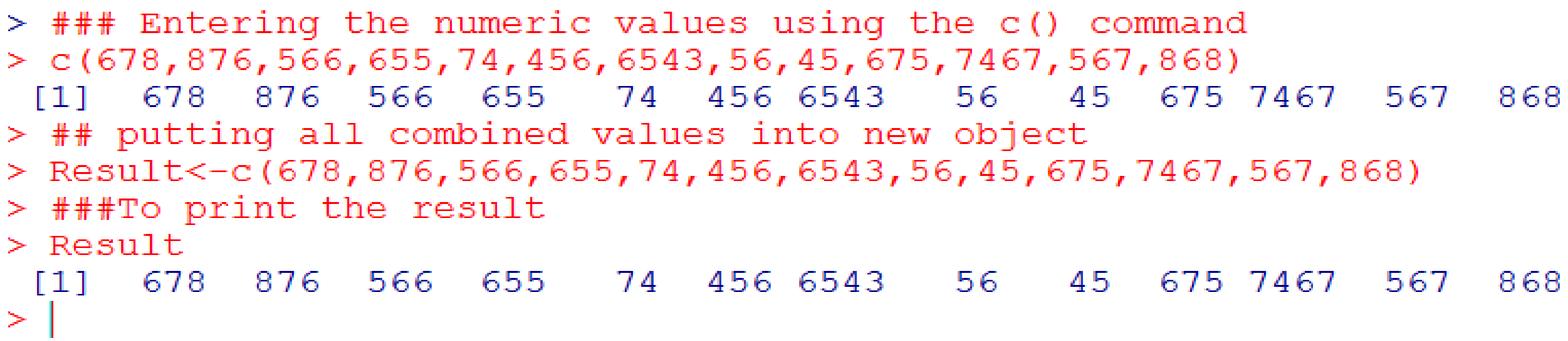
```
### sampleitem1, sampleitem2, sampleitem3 are combined
c(sampleitem1, sampleitem2, sampleitem3)

## putting all combined values into new object
CombinedResult<-c(sampleitem1, sampleitem2, sampleitem3)
```

- **Reading and Combining Numerical Data**

```
### Entering the numeric values using the c() command
Result = c(678,876,566,655,74,456,6543,56,45,675,7467,567,868)

### To print the result
Result
```

USING THE C() COMMAND

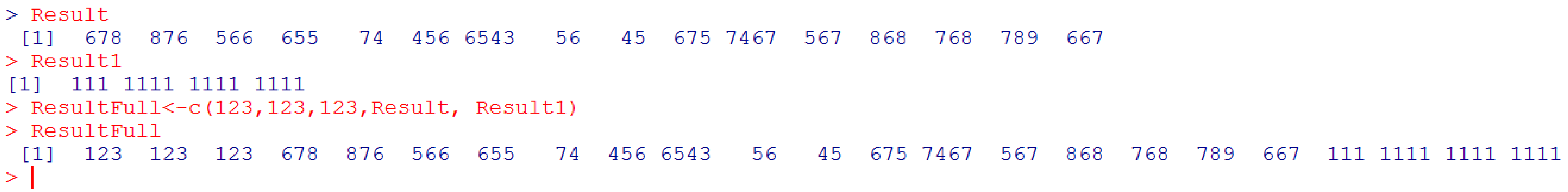
- Executing on R
- Here, we have passed numerical values within the parentheses of c() command with comma separation.
- The values are stored in the new object called “Result” and to print the values on the R, we are entering the name of the object



```
> ### Entering the numeric values using the c() command
> c(678,876,566,655,74,456,6543,56,45,675,7467,567,868)
 [1]  678  876  566  655   74  456 6543   56   45  675 7467  567  868
> ## putting all combined values into new object
> Result<-c(678,876,566,655,74,456,6543,56,45,675,7467,567,868)
> ###To print the result
> Result
 [1]  678  876  566  655   74  456 6543   56   45  675 7467  567  868
> |
```

USING THE C() COMMAND

- Incorporating existing data objects with the new values.



```
> Result
 [1]  678  876  566  655   74  456 6543   56   45  675 7467  567  868  768  789  667
> Result1
[1]  111 1111 1111 1111
> ResultFull<-c(123,123,123,Result, Result1)
> ResultFull
 [1]  123  123  123  678  876  566  655   74  456 6543   56   45  675 7467  567  868  768  789  667  111 1111 1111 1111
> 
```

- Here, we are adding some values (123,123,123) to the existing values that are stored in objects Result and Result1

READING AND COMBINING TEXT DATA

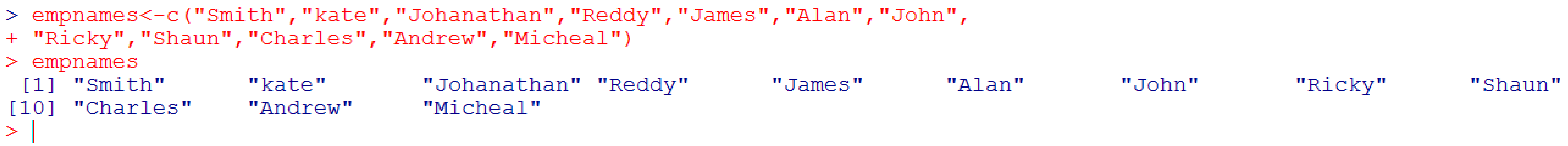
USING THE C() COMMAND

- The text data is entered using quotes.
- There is no difference between the single and double quotes as R converts all the quotes to double quotes.
- You can use either single or double or combination of quotes as shown in the syntax.
- **Syntax**

```
c(‘sampleitem1’, ‘sampleitem2’, ‘sampleitem3’)
c(“sampleitem1”, “sampleitem2”, “sampleitem3”)
c(“sampleitem1”, ‘sampleitem2’, ‘sampleitem3’)
```

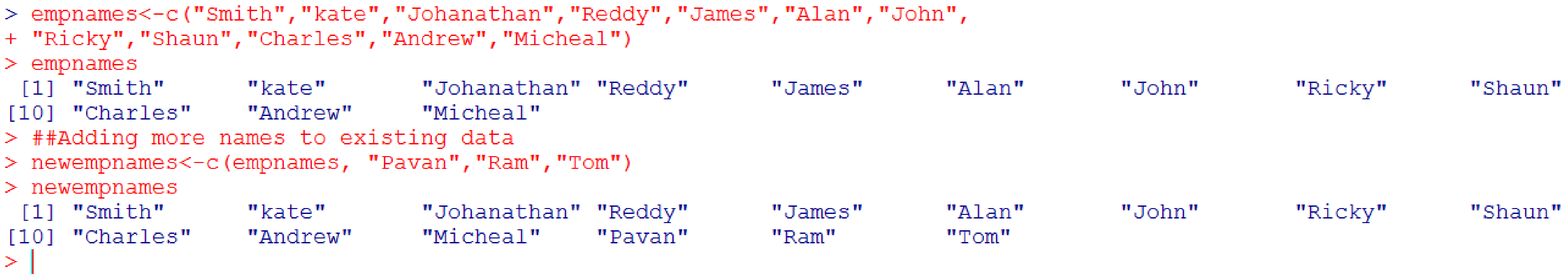
COMBINING AND READING TEXT DATA

- Reading test data on R console



```
> empnames<-c("Smith","kate","Johanathan","Reddy","James","Alan","John",
+ "Ricky","Shaun","Charles","Andrew","Micheal")
> empnames
 [1] "Smith"      "kate"       "Johanathan" "Reddy"      "James"      "Alan"       "John"       "Ricky"      "Shaun"
[10] "Charles"    "Andrew"     "Micheal"
> |
```

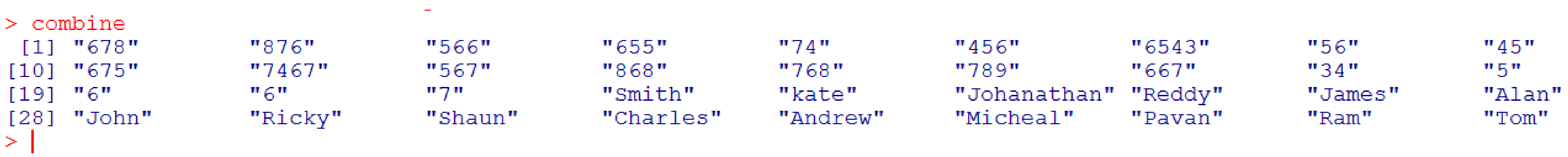
- Adding more data to the existing data



```
> empnames<-c("Smith","kate","Johanathan","Reddy","James","Alan","John",
+ "Ricky","Shaun","Charles","Andrew","Micheal")
> empnames
 [1] "Smith"      "kate"       "Johanathan" "Reddy"      "James"      "Alan"       "John"       "Ricky"      "Shaun"
[10] "Charles"    "Andrew"     "Micheal"
> ##Adding more names to existing data
> newempnames<-c(empnames, "Pavan","Ram","Tom")
> newempnames
 [1] "Smith"      "kate"       "Johanathan" "Reddy"      "James"      "Alan"       "John"       "Ricky"      "Shaun"
[10] "Charles"    "Andrew"     "Micheal"    "Pavan"      "Ram"        "Tom"
> |
```

READING NUMERIC AND TEXT IN R

- When text and numbers are combined, the entire data object becomes a text variable and the numbers are also converted to text.
- Reading both text and numeric data in R
- combine<-c(ResultFull,newempnames)



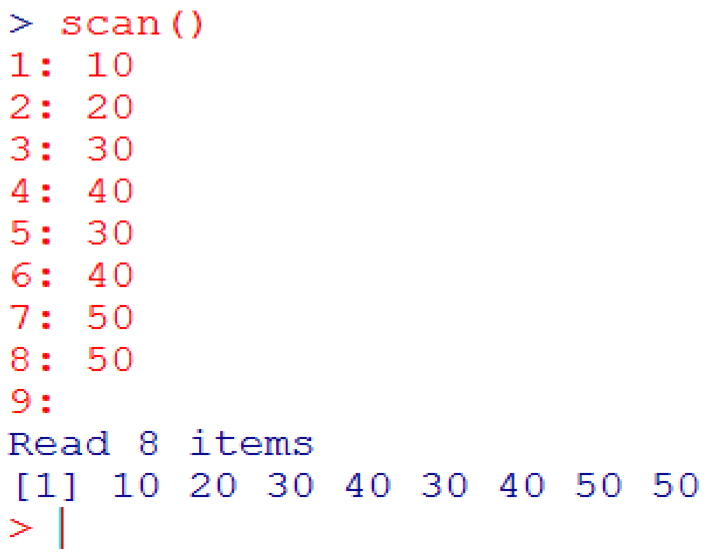
```
> combine
 [1] "678"        "876"        "566"        "655"        "74"         "456"        "6543"       "56"         "45"
[10] "675"        "7467"       "567"        "868"        "768"        "789"        "667"        "34"         "5"
[19] "6"          "6"          "7"          "Smith"      "kate"       "Johanathan" "Reddy"      "James"      "Alan"
[28] "John"       "Ricky"      "Shaun"      "Charles"    "Andrew"     "Micheal"    "Pavan"      "Ram"        "Tom"
> 
```

- Note: Here, numeric data is shown in the double quotes like that of text data.

USING THE SCAN() COMMAND

- The c() command is used only for reading and combining of small data. But this can be tedious when lot of typing is involved.
- In c() command, all the values are separated by , (comma) to make a data object.
- The same can be done with out using commas through the scan() command.

1. After entering the scan() command and press ENTER, console will be waiting for the desired data.
2. User can type the data and DOUBLE press ENTER, your data is shown on the console

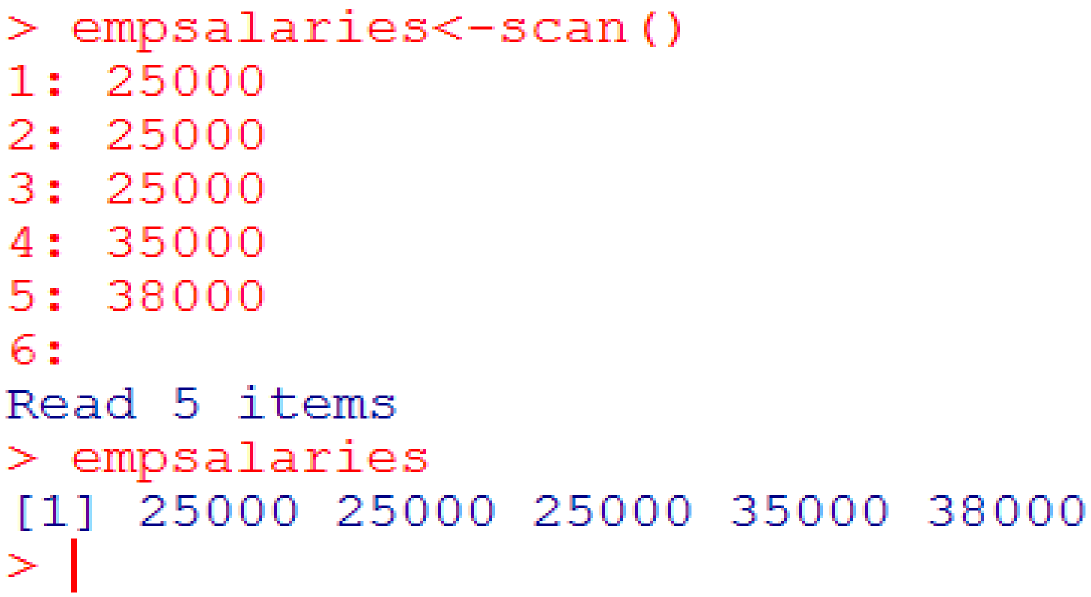


```
> scan()
1: 10
2: 20
3: 30
4: 40
5: 30
6: 40
7: 50
8: 50
9:
Read 8 items
[1] 10 20 30 40 30 40 50 50
> |
```

USING THE SCAN() COMMAND- READING

- Reading the numeric values using the scan() command.

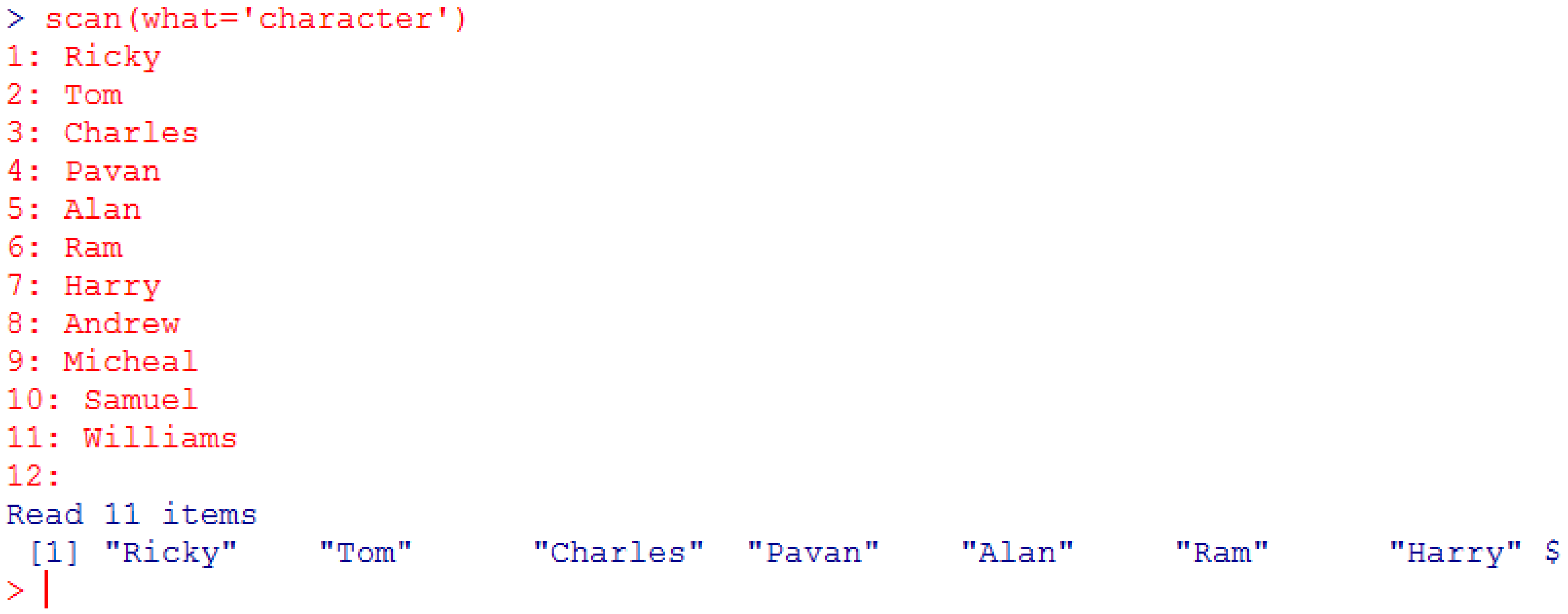
1. After entering the empsalaries<-scan() command and press ENTER, console will be waiting for the desired data.
2. User can type the data and DOUBLE press ENTER, your data is shown on the console
3. To view the stored values, object name "empsalaries" is typed



```
> empsalaries<-scan()
1: 25000
2: 25000
3: 25000
4: 35000
5: 38000
6:
Read 5 items
> empsalaries
[1] 25000 25000 25000 35000 38000
> |
```

USING THE SCAN() COMMAND- READING

- Reading the text data using scan() command
- Syntax here depicts that user is specifying that the data that has to be entered will be characters and not numbers.



```
> scan(what='character')
1: Ricky
2: Tom
3: Charles
4: Pavan
5: Alan
6: Ram
7: Harry
8: Andrew
9: Micheal
10: Samuel
11: Williams
12:
Read 11 items
 [1] "Ricky"    "Tom"       "Charles"  "Pavan"    "Alan"     "Ram"      "Harry"   $
> 
```

READING THE DATA OF A FILE FROM DISK

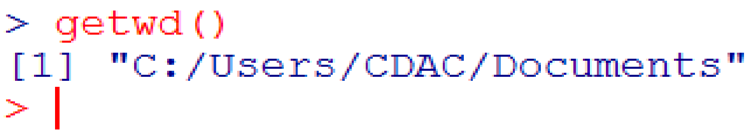
- Using the scan() command, you can also read the data from files.
- The scan() command can read data in a vector or list from the console or file.
- To read a file using scan() command, add `` file=`filename` `` to the command as shown

```
## Reading data from the file called sample.txt
readdata<-scan(file='sample.txt')
```

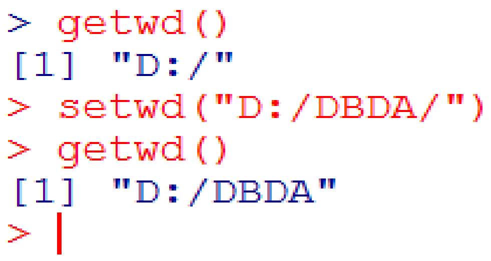
- Now, the contents of `sample.txt` file is stored in `readdata` object.
- File name should be enclosed with in the quotation marks

READING THE DATA OF A FILE FROM DISK

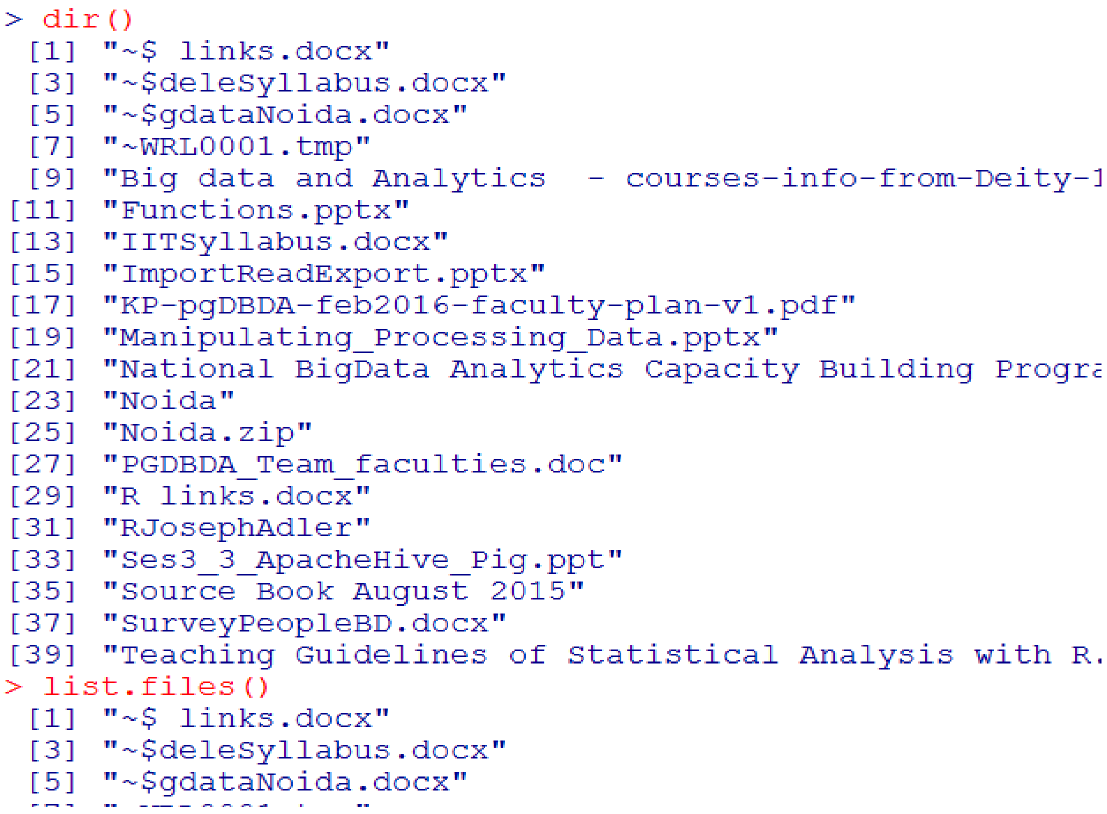
- On execution of the command, R will look for the sample.txt file in the current working directory.
- To know the current working directory and to change the directory, use following commands



```
> getwd()
[1] "C:/Users/CDAC/Documents"
> |
```



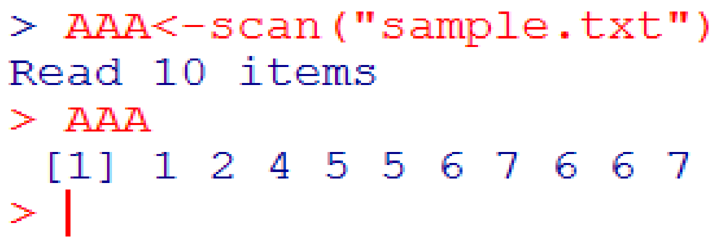
```
> getwd()
[1] "D:/"
> setwd("D:/DBDA/")
> getwd()
[1] "D:/DBDA"
> |
```



```
> dir()
 [1] "~$ links.docx"
 [3] "~$deleSyllabus.docx"
 [5] "~$gdataNoida.docx"
 [7] "~WRL0001.tmp"
 [9] "Big data and Analytics  - courses-info-from-Deity-1
[11] "Functions.pptx"
[13] "IITSyllabus.docx"
[15] "ImportReadExport.pptx"
[17] "KP-pgDBDA-feb2016-faculty-plan-v1.pdf"
[19] "Manipulating_Processing_Data.pptx"
[21] "National BigData Analytics Capacity Building Progra
[23] "Noida"
[25] "Noida.zip"
[27] "PGDBDA_Team_faculties.doc"
[29] "R links.docx"
[31] "RJosephAdler"
[33] "Ses3_3_ApacheHive_Pig.ppt"
[35] "Source Book August 2015"
[37] "SurveyPeopleBD.docx"
[39] "Teaching Guidelines of Statistical Analysis with R.
> list.files()
 [1] "~$ links.docx"
 [3] "~$deleSyllabus.docx"
 [5] "~$gdataNoida.docx"
```

READING THE DATA OF A FILE FROM DISK

- Using scan() command for reading from file



```
> AAA<-scan("sample.txt")
Read 10 items
> AAA
 [1] 1 2 4 5 5 6 7 6 6 7
> |
```

- The scan() command has an option of choosing the file by browsing the file system

```
scan(file.choose())
```

Note: scan(file.choose()) function will not work in Linux OS

USING THE READ.CSV() COMMAND

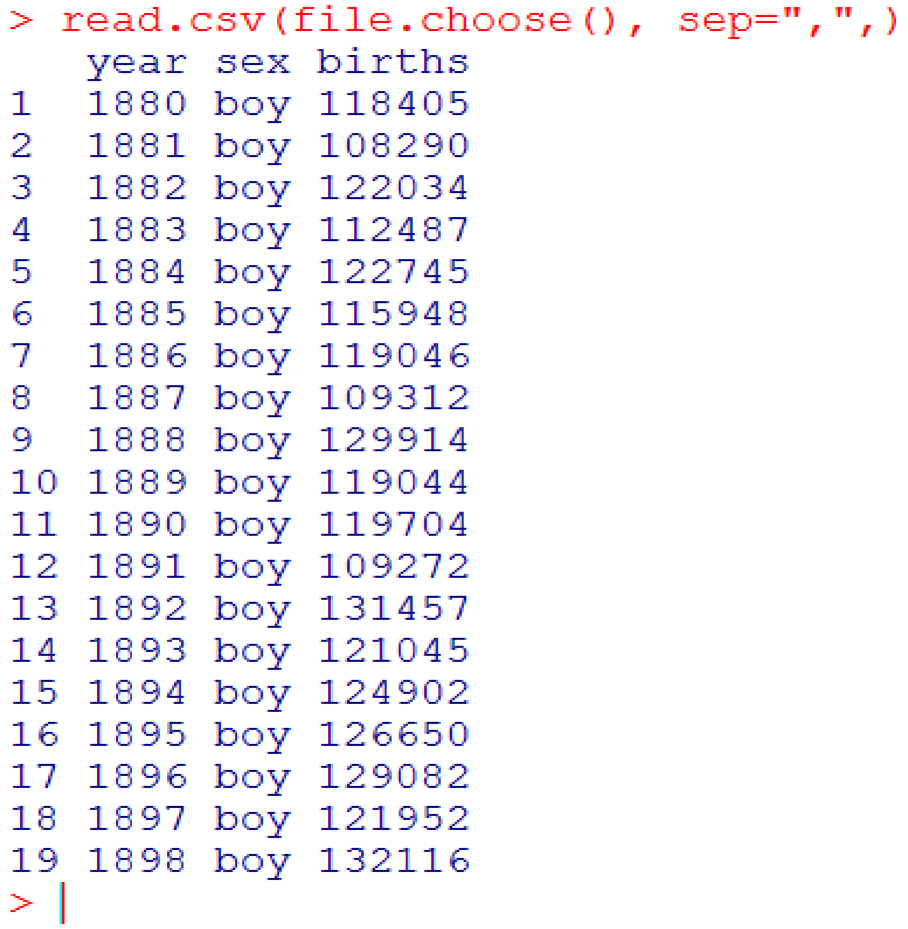
- Reading from CSV files, read.csv() command is used.
- The command read.csv() reads entire CSV file and display the contents on the R console.
- **Syntax**

```
read.csv(file, header = TRUE, sep = “,”)
```

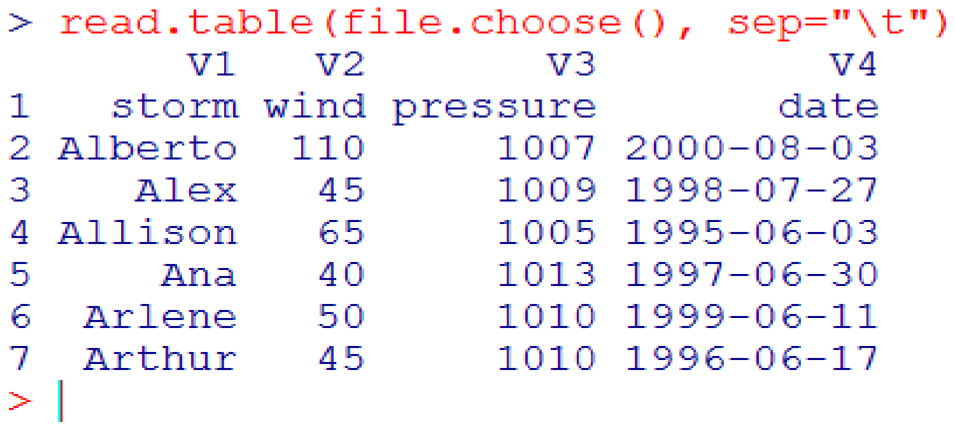
  - **file**: to specify the file name
  - **sep**: to provide the separator
  - **header**: to specify whether or not the first row of CSV file should be set as column names. Default is TRUE

USING THE READ.CSV() COMMAND

- Before executing the read.csv() command, file is read and saved in appropriate format CSV/XLS or TSV format



```
> read.csv(file.choose(), sep=",",)
   year sex births
1  1880 boy 118405
2  1881 boy 108290
3  1882 boy 122034
4  1883 boy 112487
5  1884 boy 122745
6  1885 boy 115948
7  1886 boy 119046
8  1887 boy 109312
9  1888 boy 129914
10 1889 boy 119044
11 1890 boy 119704
12 1891 boy 109272
13 1892 boy 131457
14 1893 boy 121045
15 1894 boy 124902
16 1895 boy 126650
17 1896 boy 129082
18 1897 boy 121952
19 1898 boy 132116
> |
```



```
> read.table(file.choose(), sep="\t")
        V1   V2       V3         V4
1    storm wind pressure       date
2 Alberto   110     1007 2000-08-03
3     Alex   45     1009 1998-07-27
4 Allison    65     1005 1995-06-03
5      Ana   40     1013 1997-06-30
6   Arlene   50     1010 1999-06-11
7   Arthur   45     1010 1996-06-17
> |
```

IMPORTING DATA FROM FWF

- Reading data from FWF (fixed width format) in to a dataframe.
- To read data from fwf, we have **read.fwf()** function in R
- You use this function when your data file has columns containing spaces, or columns with no spaces to separate them.
- **Syntax :**

```
read.fwf(file, width=“”,col.names=“”)
```

- **Example**

```
read.fw(“fwf.txt", widths=c(4,-13,1,-2,2),col.names=c("Subject","Gender","Marks"))
```

IMPORTING EXCEL SPREADSHEETS INTO R

- From the base R, you will not able to import Excel file directly.
- Package to be installed is **xlsx**, **openxlsx** package.

- **Reading Excel Spreadsheets into R From The Clipboard**
  - Functions used in R are **read.table(file="File_Name")**

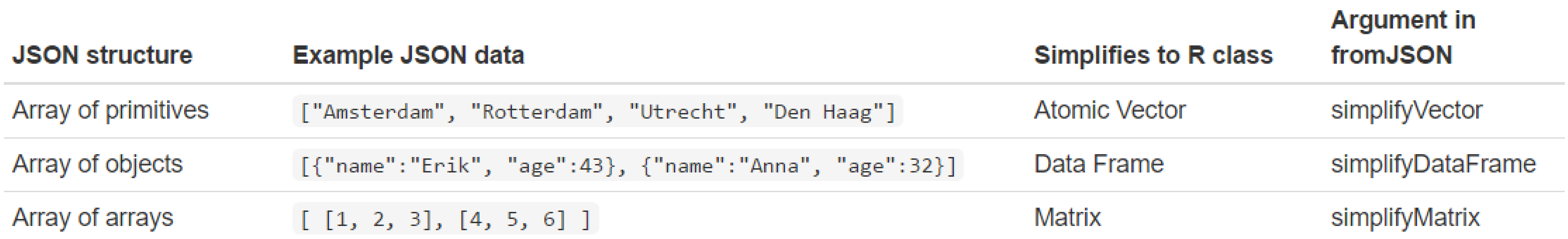
- **You can convert Excel file to CSV file and import in R using read.csv()**

IMPORTING JSON (IN JAVASCRIPT OBJECT NOTATION) FILES INTO R

- Package used for importing json files into R is rjson.
- Library need to load is jsonlite
- Function used is **fromJSON**
- Three procedures under fromJSON() :

  simplifyVector, simplifyDataFrame and simplifyMatrix

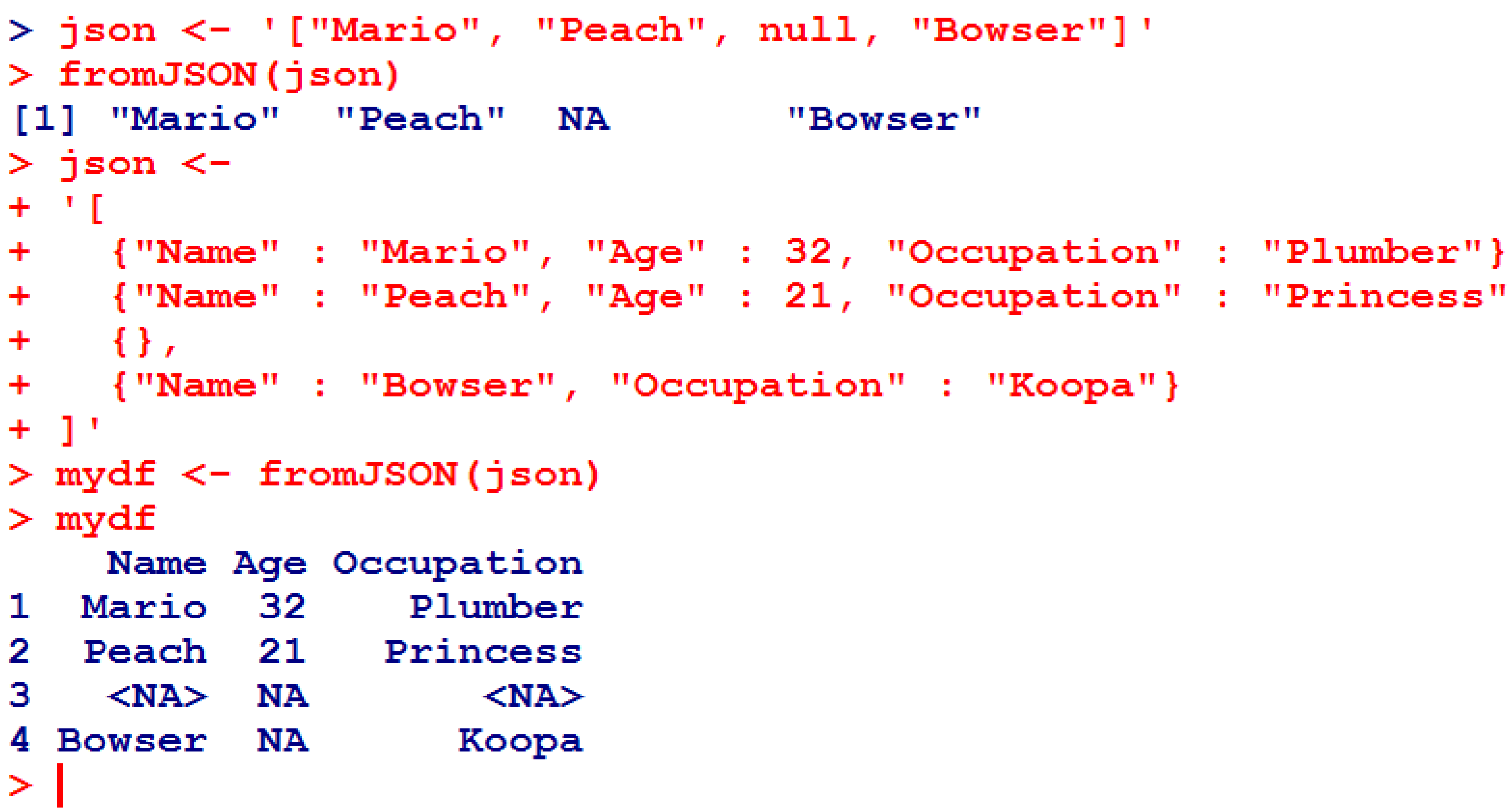
install.packages(“rjson”)
library(rjson)



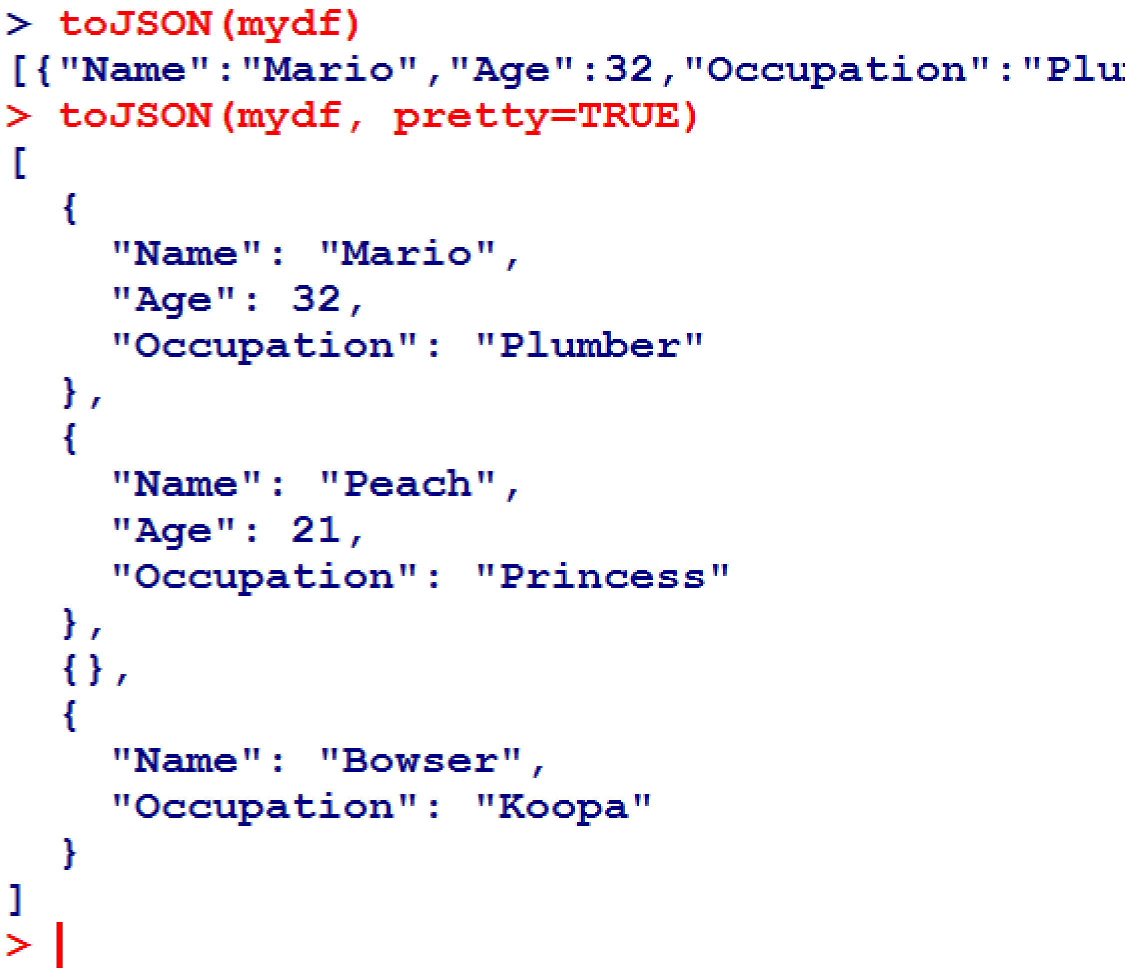
JSON structure	Example JSON data	Simplifies to R class	Argument in fromJSON
Array of primitives	`["Amsterdam", "Rotterdam", "Utrecht", "Den Haag"]`	Atomic Vector	simplifyVector
Array of objects	`[{"name":"Erik", "age":43}, {"name":"Anna", "age":32}]`	Data Frame	simplifyDataFrame
Array of arrays	`[ [1, 2, 3], [4, 5, 6] ]`	Matrix	simplifyMatrix

IMPORTING JSON FILES INTO R

- Simple commands



```
> json <- '["Mario", "Peach", null, "Bowser"]'
> fromJSON(json)
[1] "Mario"  "Peach"  NA       "Bowser"
> json <-
+ '[
+   {"Name" : "Mario", "Age" : 32, "Occupation" : "Plumber"}
+   {"Name" : "Peach", "Age" : 21, "Occupation" : "Princess"
+   {},
+   {"Name" : "Bowser", "Occupation" : "Koopa"}
+ ]'
> mydf <- fromJSON(json)
> mydf
    Name Age Occupation
1  Mario  32    Plumber
2  Peach  21   Princess
3   <NA>  NA       <NA>
4 Bowser  NA      Koopa
>
```



```
> toJSON(mydf)
[{"Name":"Mario","Age":32,"Occupation":"Plu
> toJSON(mydf, pretty=TRUE)
[
  {
    "Name": "Mario",
    "Age": 32,
    "Occupation": "Plumber"
  },
  {
    "Name": "Peach",
    "Age": 21,
    "Occupation": "Princess"
  },
  {},
  {
    "Name": "Bowser",
    "Occupation": "Koopa"
  }
]
>
```

IMPORTING DATA FROM DATABASES INTO R

- Packages used for importing from various databases
  - MonetDB.R
  - Rmongodb
  - RMySQL,
  - Mongolite
  - Rmongo
  - RODBC
  - Roracle
  - RPostgreSQL
  - RSQLite
  - RJDBC

IMPORTING DATA FROM MYSQL INTO R

- Packages and library needed

```
install.packages("RMySQL")
 library(RMySQL)
```

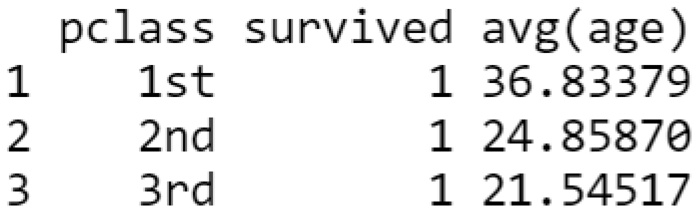
- MySQL Connection

```
con = dbConnect(MySQL(),user="training", password="training123",
dbname="trainingDB", host="localhost")
```

- Retrieving data

```
myQuery <- "select pclass, survived, avg(age) from titanic where survived=1 group by
pclass;“
 dbGetQuery(con, myQuery)
```

http://www.unomaha.edu/mahbubulmajumder/data-science/fall-2014/lectures/20-database-mysql/20-database-mysql.html#/1

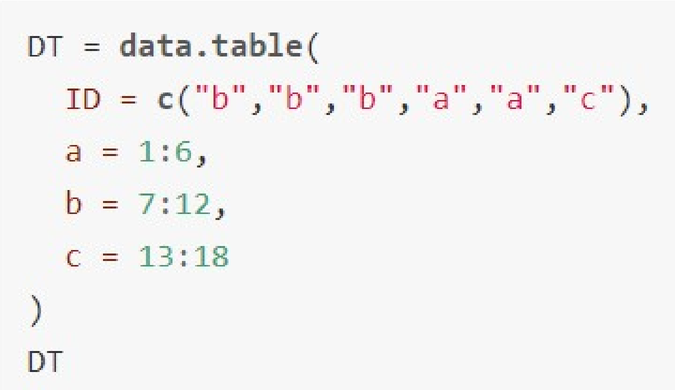


```
  pclass survived avg(age)
1    1st        1 36.83379
2    2nd        1 24.85870
3    3rd        1 21.54517
```

IMPORTING LARGE DATA SETS INTO R

- Package used in data.table
- Function is fread()
- Example :

```
library(data.table)
data <- fread(“textfile.txt”)
```



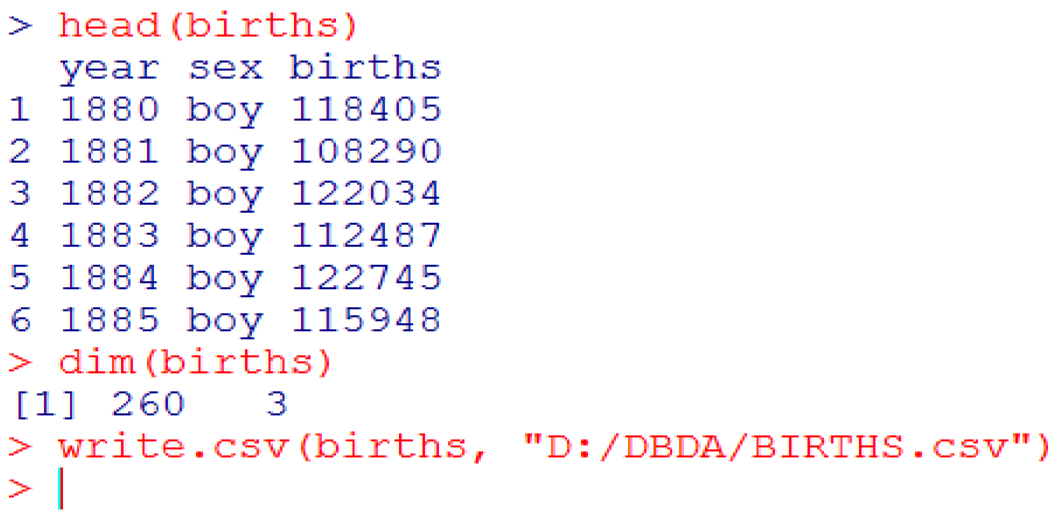
```
DT = data.table(
  ID = c("b","b","b","a","a","c"),
  a = 1:6,
  b = 7:12,
  c = 13:18
)
DT
```

EXPORTING DATA FROM R

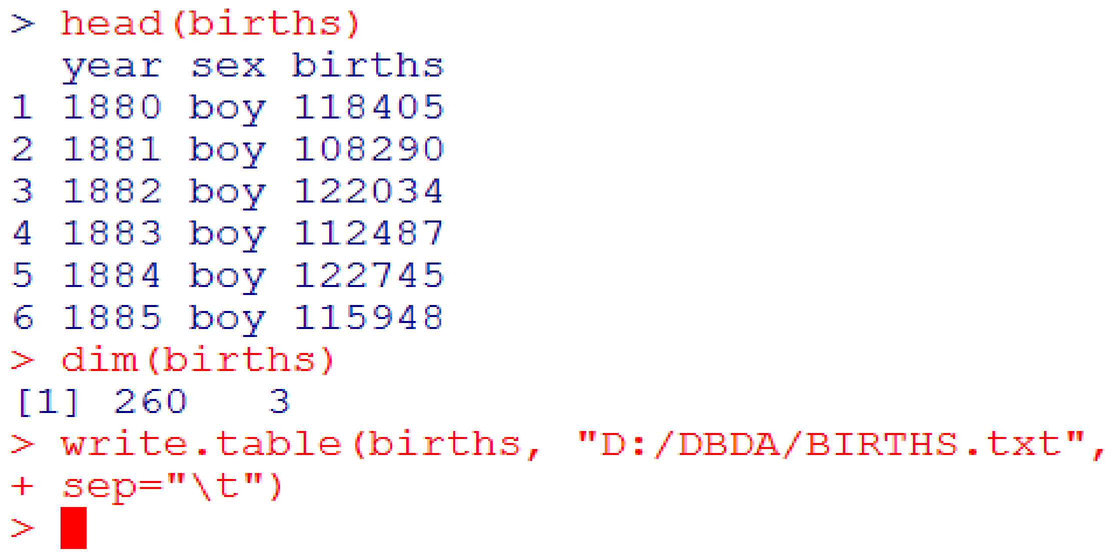
- After undergoing any computations in R, the data now needs to be used in reports or various other sources.
- Therefore, you need to extract data from R.
- To export data from R, we use write.csv() and write.table() functions.

USING THE WRITE.TABLE() AND WRITE.CSV() COMMAND

- The write.table() command is used to write the data stored in a vector to a file.
- The data is saved using the delimiters such as spaces or tabs as shown.
- Here, in the below screenshots, we are saving the 'births' object to the BIRTHS.csv and BIRTHS.txt in the D drive local system



```
> head(births)
  year sex births
1 1880 boy 118405
2 1881 boy 108290
3 1882 boy 122034
4 1883 boy 112487
5 1884 boy 122745
6 1885 boy 115948
> dim(births)
[1] 260   3
> write.csv(births, "D:/DBDA/BIRTHS.csv")
> 
```



```
> head(births)
  year sex births
1 1880 boy 118405
2 1881 boy 108290
3 1882 boy 122034
4 1883 boy 112487
5 1884 boy 122745
6 1885 boy 115948
> dim(births)
[1] 260   3
> write.table(births, "D:/DBDA/BIRTHS.txt",
+ sep="\t")
> 
```

THANK YOU!!!!!