TMC 204

Statistical Data Analysis with R

Introduction and Data Structures in R

Presented By : Aditya Joshi

Asst. Professor

Department of Computer Application

Graphic Era Deemed to be University

Introduction to R

- R is an Open Source (GPL) Statistical Environment modeled after S and S-Plus. The S language was developed in the late 1980’s at AT&T Labs.
- The R Project was Started by Robert Gentleman and Ross lhaka(Hence the name R) of the statistical department of the University of Auckland in 1995.

R is a

- Programming Language
- A statistical package
- An interpreter
- Open Source
- Object Oriented Language.

- R is a programming language and software environment for statistical Computing and Graphics Supported by the R Foundation for Statistical computing.
- The R Language is widely used among Statisticians and data miners for developing statistical software and data analysis.
- R is powerful statistical program but it is first and foremost a programming language. Many routines have been written for R by people all over the world and made freely available from the R project website as “packages” however the basic installation (for windows, Linux or Mac) contains a powerful set of tools for most purposes.

Features of R Programming Language

R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R:

1. Well developed, simple and effective Programming language which includes conditional loops, user defined recursive functions and input and output facilities
2. R has an effective data handling and storage facility.
3. R provides a suite for operators for calculations on arrays, lists, vectors and matrices.
4. R provides a large, coherent and integrated collection of tools for data analysis

5. R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers

6. As a conclusion, R is the most widely used statistical programming language.

Downloading and installing R

To Install R:

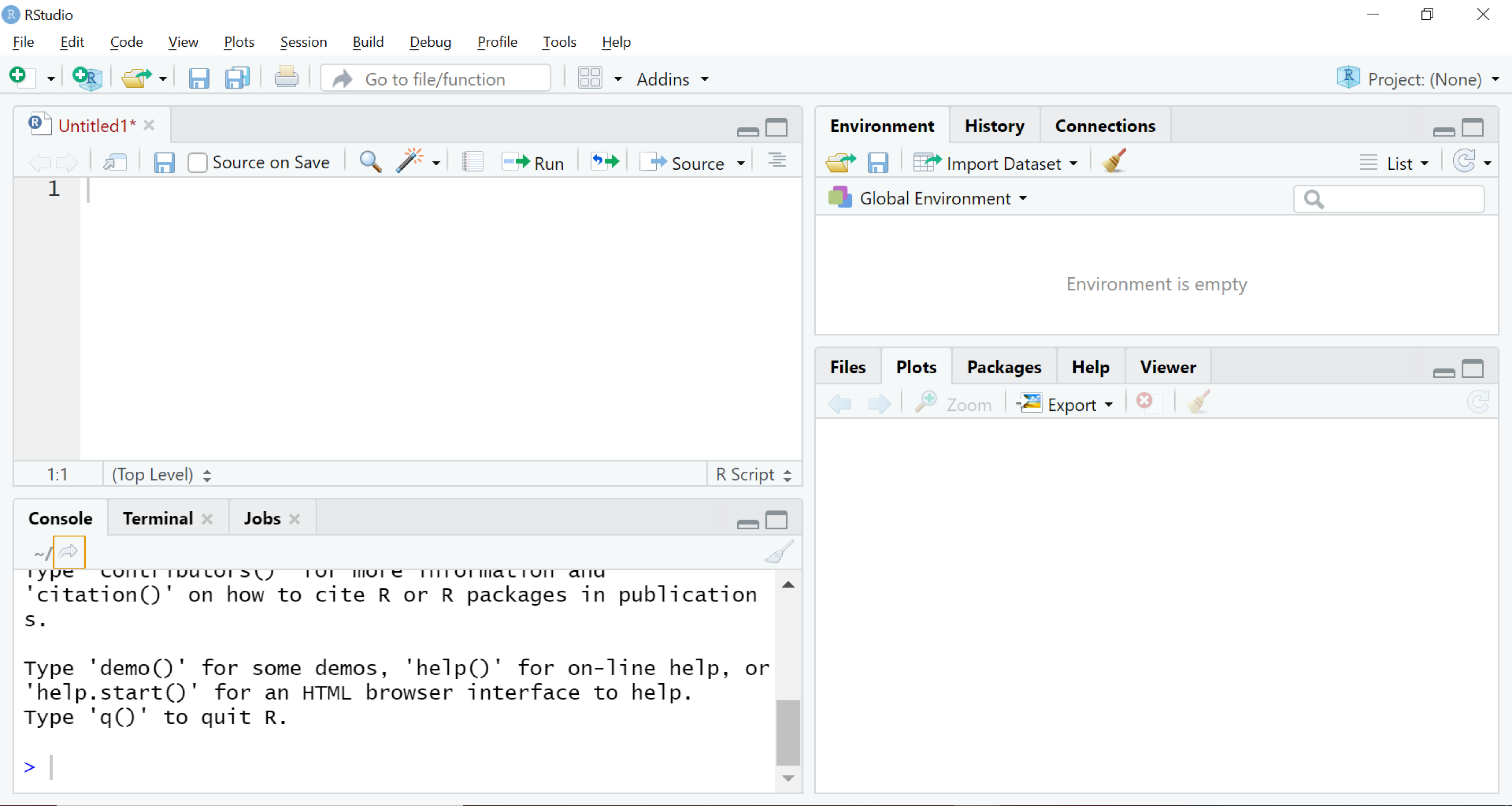
- Open an internet browser and go to www.r-project.org.
- Click the "download R" link in the middle of the page under "Getting Started."
- Select a CRAN Comprehensive R Archive Network location (a mirror site) and click the corresponding link.
- Click on the "Download R for Windows" link at the top of the page.
- Click on the "install R for the first time" link at the top of the page.
- Click "Download R for Windows" and save the executable file somewhere on your computer. Run the .exe file and follow the installation instructions.
- Now that R is installed, you need to download and install RStudio.

To Install RStudio

- Go to www.rstudio.com and click on the "Download RStudio" button.
- Click on "Download RStudio Desktop."
- Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

RStudio: An Integrated Development Environment (IDE) for R

- RStudio is an integrated development environment (IDE) that allows you to interact with R more readily. RStudio is similar to the standard RGui, but is considerably more user friendly. It has more drop-down menus, windows with multiple tabs, and many customization options. The first time you open RStudio, you will see three windows. A forth window is hidden by default, but can be opened by clicking the **File** drop-down menu, then **New File,** and then **R Script.**



```
'citation()' on how to cite R or R packages in publication
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.

>
```

R Programming IDE and editors

R is a programming language meant for statistical computing and data science.

R can be run in the command line for terminal nerds and graphical user interfaces in integrated development environments.

- **1.RStudio**
- **2.R Tools for Visual Studio**
- **3.Rattle**
- **4.StatET for R (Eclipse StatET)**
- **5.ESS (Emacs Speaks Statistics)**
- **6.Tinn-R**
- **7. R AnalyticalFlow**
- **8. Radiant**
- **9.RBox**
- **10. NVim-R**
- **11. r4intelliJ**

Getting Help with R

Helping Yourself

- Before asking others for help, it’s generally a good idea for you to try to help yourself.
- R includes extensive facilities for accessing documentation and searching for help.
- There are also specialized search engines for accessing information about R on the internet, and general internet search engines can also prove useful.
- Once R is installed, there is a comprehensive built-in help system. At the program's command prompt you can use any of the following:

R Help

You can get help from CRAN website and Internet

The Help Command In R

R contains a lot of built in help the basic command to bring up help is

help(topic)

For example help(mean)

You can use this also

?topic

For example ?mean

- You can also access the help system via your web browser by typing

help.start()

- You can use following command if you don't know the exact keyword

apropos('partword')

For example apropos('plot')

This method uses internet search engine for searching information

RSiteSearch("{generalized linear model}")

get vignettes on using installed packages

long form documentation of your package it contains HTML pdf and source

vignette()

browseVignettes(package="packagename")

demo(package="packagename")

Command Packages   2600 packages in CRAN

- The R Program is built from a series of modules, called **packages.**
- **Packages are bundles of code that add new functions to R**
- R packages are a collection of R functions, complied code and sample data.
- They are stored under a directory called **"library"** in the R environment.
- By default, R installs a set of packages during installation. More packages are added later, when they are needed for some specific purpose.

Check Available R Packages

- **search()** this command is used to what package is loaded and ready to use it will Get all packages currently loaded in the R environment
- **Installed.packages()** this command is to see what package are available
- **.libPaths()** Get library locations containing R packages
- **library()** Get the list of all the packages installed

Install a New Package

There are two ways for installing package One is installing directly from the CRAN directory and another is downloading the package to your local system and installing it manually.

1. Install directly from CRAN

By command **install.packages(“packagename”)** The following command gets the packages directly from CRAN webpage and installs the package in the R environment. You may be prompted to choose a nearest mirror. Choose the one appropriate to your location.

#install the package name XML

Install.packages(“XML”)

2. Install package manually

Go to this link https://cran.r-project.org/web/packages/available_packages_by_name.html to download the package needed. Save the package as a **.zip** file in a suitable location in the local system.

Now you can run the following command to install this package in the R environment.

```
install.packages(file_name_with_path, repos = NULL, type = "source")
# Install the package named "XML"
install.packages("E:/XML_3.98-1.3.zip", repos = NULL, type = "source")
```

The other way is from package menu in r environment

Loading Package

- Before a package can be used in the code, it must be loaded to the current R environment. You also need to load a package that is already installed previously but not available in the current environment.
- A package is loaded using the following command −

library(packagename)

Removing and unloading packages

- If you have loaded some packages and want to remove them command used is

detach(package:name)

Use R Like Calc

- R can be used for doing simple mathematical calculations

By typing

```
 > 2+3+4
[1]9
```

You can do bit more complicated calculations

Like

```
> 15+8/2*100
[1] 415
```

Some of the mathematical operations Available in R

Command / Operation	Explanation
+ - / * ( )	Standard mathematical characters as well as parentheses
pi	Value of pi 3.142
x^y	X raised to the power y
sqrt(x)	Square root of x
abs(x)	Absolute value of x
factorial(x)	Factorial of x
cos(x),sin(x), acosx() etc	Trigonometric functions
exp(x)	Exponent of x
log10(x) log2(x)	Log of x to the base 2 and 10

Storing results for calculations

- To make a result object simply type the name followed by equal sign and any thing after = will be evaluated and stored as your result.

```
> a=23+2
> b=22-1
➢sum=a+b
```

= sign or <- sign

Older version of R uses = sign

```
> a<-29/2
> b<-22/2
> mul<-a*b
> a+b->sum
```

Reading and Getting data into R

- For creating more complex series of numbers to work on so that you will be able to use these complex series for further analysis

Using combine command for making data

For creating sample we have to use **c()** command

It is short for combine or concatenate

Every thing in parentheses joined up to make a single item

c(item.1, item.2, item.3,item.n)

You will assign joined up items to a named object

sample.name=c(item.1, item.2, item.3,item.n)

Entering Numerical items as data

```
data1=c(3,4,5,6,7,8,2,3,5)
```

If you want to see the result you must type its name

```
> data1
[1] 3 4 5 6 7 8 2 3 5
```

How long we can take data

```
> data1=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18)
> data1
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
[18] 18
```

You can add data in at any point from previous data item stored in new data object

```
> data1=c(1,2,3,4,5)
> data1
[1] 1 2 3 4 5
> data2=c(data1,6,7,8,9)
> data2
[1] 1 2 3 4 5 6 7 8 9
> data3=c(10,20,data2)
> data3
 [1] 10 20  1  2  3  4  5  6  7  8  9
```

Entering text items as data

You can use single or double quotes between data items for entering them as text

our.text=c(“item1”, “item2”,’item3’)

```
> day1=c("Mon", "Tue","Wed",'Thu','Fri',"Sat","Sun")
> day1
[1] "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun"
> mix=c(data1,day1)
> mix
 [1] "1"   "2"   "3"   "4"   "5"   "Mon" "Tue" "Wed"
 [9] "Thu" "Fri" "Sat" "Sun"
```

Using Scan Command for Making Data

our.data=scan() this will prompt for enter your data

```
 > data1=scan()                                Numerical Data
1: 1
2: 2
3: 3
4:
Read 3 items
> data1
[1] 1 2 3
```

Entering Text as data

```
> day=scan(what='character')
1: Mon Tue Wed
4: Thu
5: fri
6: Sat
7: sun
8:
Read 7 items
> day
[1] "Mon" "Tue" "Wed" "Thu" "fri" "Sat" "sun"
```

Using the clipboard to make data

- You can make scan() command easier to use than c() command because it does not require commas so we can easily copy from spreadsheet and paste data to R steps are:

1. If the data are in numbers type the command in R before switching to spreadsheet containing data
2. Highlight the cells and copy the data into clipboard
3. Return to R and paste the data from clipboard to R
4. Once you are finished enter black line to complete the data

If data is in character the add the what=‘character in the scan() command ’

- If the data is separated with simple spaces , you can simply copy and paste.
- If the data is separated with some other character then you need to tell R which character is used as the separator.
- A common CSV file which uses commas to separate the data items

You have to use

scan(sep=‘,’)

If characters are there then

scan(sep=‘,’, what=‘char’)

```
> data=scan(sep=',',what='char')
1: Andhra Pradesh	2014	1262	5
2:
Read 1 item
> data
[1] "Andhra Pradesh\t2014\t1262\t5"
> data=scan(sep='\t',what='char')
1: Andhra Pradesh	2014	1262	5
5:
Read 4 items
> data
[1] "Andhra Pradesh" "2014"	"1262"
[4] "5"
```

Reading a file of data from a disk

- > **data=scan(file='sample.txt', what='char')**
- Read 11 items
- > **data**
- [1] "aditya" "joshi" "this"   "is"   "a"
- [6] "very"   "good"   "topic" "1"   "2"
- [11] "3"

- R looks for data file in default directory so command is

> **getwd()**

[1] "C:/Users/Aditya/Documents"

If your file is in another location

```
> data=scan(file='C:/Users/Aditya/Desktop/sample.txt', what='char')
Read 11 items
> data
 [1] "aditya" "joshi"  "this"   "is"     "a"
 [6] "very"   "good"   "topic"  "1"      "2"
[11] "3"
>
```

Setting working directory

You can alter working directory with setwd() command

```
> setwd('C:/Users/Aditya/Desktop')
> getwd()
[1] "C:/Users/Aditya/Desktop"
```

To set working directory one level up you can use

```
> setwd('..')
> getwd()
[1] "C:/Users/Aditya"
>
```

Looking the contents of directory

For checking the the contents of directory with files and folder are there

dir()

list.files() are used

```
> dir('Desktop')
 [1] "AdityaJoshi_AssignmentML.pdf"
 [2] "Client Authentication Agent.lnk"
 [3] "CodeBlocks.lnk"
 [4] "CPP"
 [5] "CPPPrograms"
 [6] "dengue"
 [7] "desktop.ini"
 [8] "Microsoft Edge.lnk"
 [9] "MinGW Installer.lnk"
[10] "Paperstostudy"
[11] "R"
[12] "sample.txt"
[13] "Suspended.docx"
```

```
> dir('Desktop/CPPPrograms')
 [1] "ascii.cpp"              "ascii.exe"
 [3] "ascii.o"                "defaultargument.cpp"
 [5] "defaultargument.exe" "defaultargument.o"
 [7] "enum.cpp"               "enum.exe"
 [9] "enum.o"                 "factorial.cpp"
[11] "factorial.exe"          "factorial.o"
[13] "funcpro1.cpp"           "funcpro1.exe"
[15] "funcpro1.o"             "Hello.cpp"
[17] "Hello.exe"              "Hello.o"
[19] "inlinefunc.cpp"         "inlinefunc.exe"
[21] "inlinefunc.o"           "inlinefunc2.cpp"
[23] "inlinefunc2.exe"        "inlinefunc2.o"
[25] "libfunction.cpp"        "libfunction.exe"
[27] "libfunction.o"          "manip.cpp"
[29] "manip.exe"              "manip.o"
[31] "memory.cpp"             "memory.exe"
[33] "memory.o"               "pointer1.cpp"
[35] "pointer1.exe"           "pointer1.o"
[37] "refrence.cpp"           "refrence.exe"
[39] "refrence.o"             "scope.cpp"
[41] "scope.exe"              "scope.o"
[43] "sizeof.cpp"             "sizeof.exe"
[45] "sizeof.o"               "struandunsize.cpp"
```

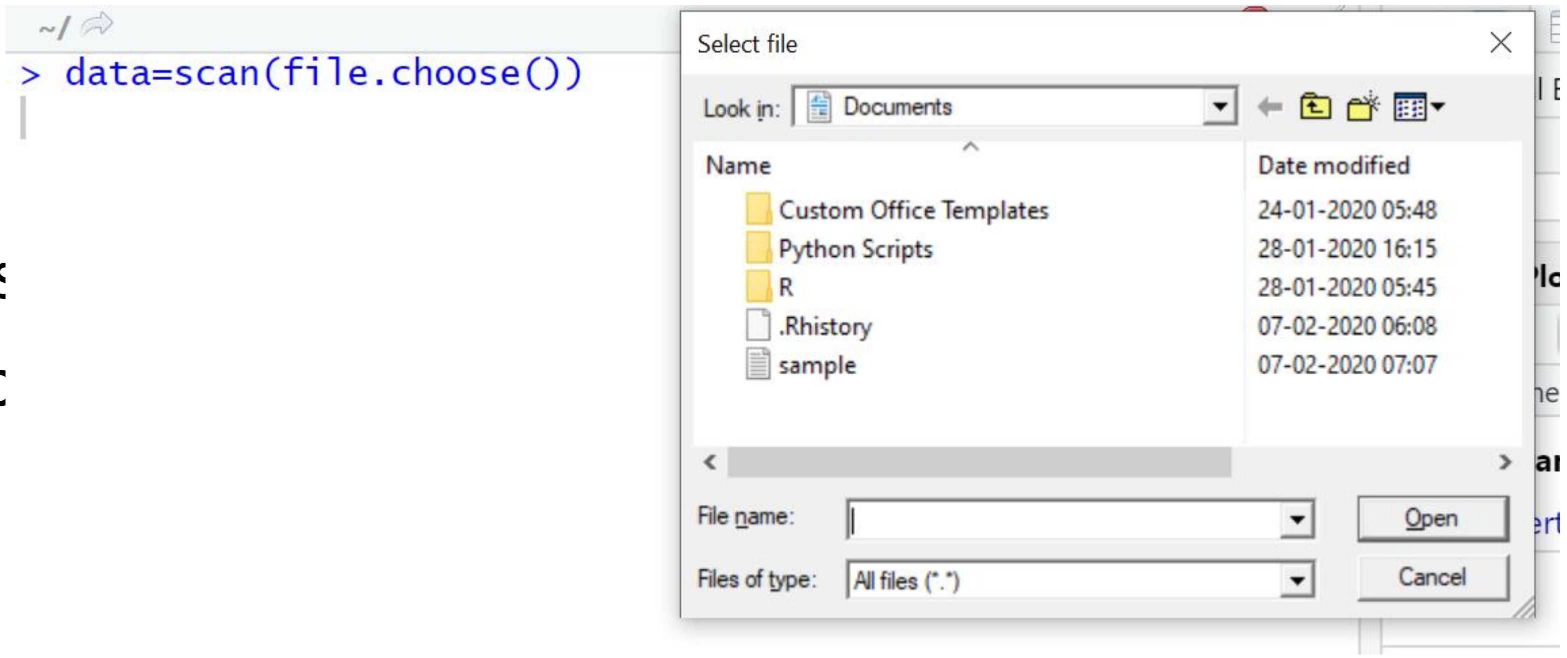
```
> list.files('DEsktop')
 [1] "AdityaJoshi_AssignmentML.pdf"
 [2] "Client Authentication Agent.lnk"
 [3] "CodeBlocks.lnk"
 [4] "CPP"
 [5] "CPPPrograms"
 [6] "dengue"
 [7] "desktop.ini"
 [8] "Microsoft Edge.lnk"
 [9] "MinGW Installer.lnk"
[10] "Paperstostudy"
[11] "R"
[12] "sample.txt"
[13] "Suspended.docx"
>
```

For invisible files

```
> dir(all.files = TRUE)
 [1] "."                      ".."
 [3] ".Rhistory"              "Custom Office Templates"
 [5] "desktop.ini"            "My Music"
 [7] "My Pictures"            "My Videos"
 [9] "Python Scripts"         "R"
[11] "sample.txt"
>
```

Selects the file from browser type or from popup window

```
>data=scan(file.choose())
> data=scan(file.choose(),what='char')
Read 11 items
> data
 [1] "aditya" "joshi"  "thi
 [6] "very"   "good"   "to
[11] "3"
>
```



```
~/
> data=scan(file.choose())
```

Select file

Look in: Documents

Name	Date modified
Custom Office Templates	24-01-2020 05:48
Python Scripts	28-01-2020 16:15
R	28-01-2020 05:45
.Rhistory	07-02-2020 06:08
sample	07-02-2020 07:07

File name:

Files of type: All files (*.*)

Open

Cancel

Reading Bigger Files

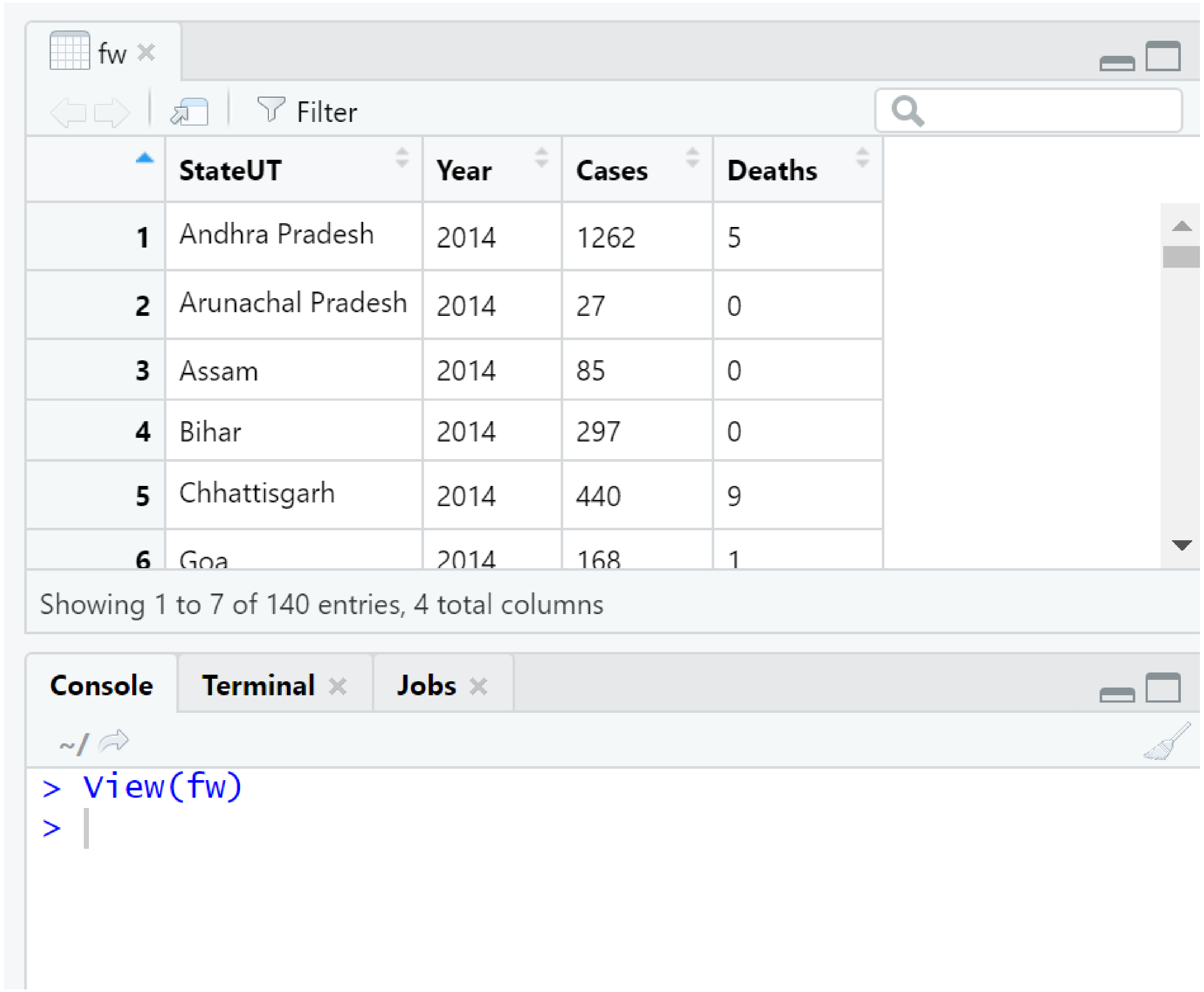
- Scan command is helpful to read simple vector for reading the more complicated files having multiple items i.e two dimensional in nature containing rows and columns
- In most cases you have to prepare data in spreadsheet for reading data from spreadsheet command **read.csv()** is used it is under **utils package**

```
> read.csv("Denguecases.csv")
              StateUT Year Cases Deaths
1      Andhra Pradesh 2014  1262      5
2   Arunachal Pradesh 2014    27      0
3               Assam 2014    85      0
4               Bihar 2014   297      0

> read.csv("Denguecases.csv", header=TRUE)
              StateUT Year Cases Deaths
1      Andhra Pradesh 2014  1262      5
2   Arunachal Pradesh 2014    27      0
3               Assam 2014    85      0
4               Bihar 2014   297      0
```

```
> read.csv("Denguecases.csv", header=FALSE)
                 V1   V2    V3     V4
1            StateUT Year Cases Deaths
2      Andhra Pradesh 2014  1262      5
3   Arunachal Pradesh 2014    27      0


> read.csv("Denguecases.csv", header=TRUE,
row.names=NULL,col.names=c("a","b","c","d"))
                  a    b     c  d
1      Andhra Pradesh 2014  1262  5
2   Arunachal Pradesh 2014    27  0
3              Assam 2014    85  0
```

- If you have file saved in .XLS or .xlsx format save the data as CSV format. For making the columns together in R for making it single entity.
- You can read the file with the help of searching it by the browser window.

```
 > fw=read.csv(file.choose())
> fw
             StateUT Year Cases Deaths
1     Andhra Pradesh 2014  1262      5
2  Arunachal Pradesh 2014    27      0
3              Assam 2014    85      0
4              Bihar 2014   297      0
5       Chhattisgarh 2014   440      9
```



	StateUT	Year	Cases	Deaths
1	Andhra Pradesh	2014	1262	5
2	Arunachal Pradesh	2014	27	0
3	Assam	2014	85	0
4	Bihar	2014	297	0
5	Chhattisgarh	2014	440	9
6	Goa	2014	168	1

Showing 1 to 7 of 140 entries, 4 total columns

```
> View(fw)
> 
```

Alternative command for Reading Data in R

```
> read.table(file.choose(),sep=',',header = TRUE)
            StateUT Year Cases Deaths
1      Andhra Pradesh 2014  1262      5
2    Arunachal Pradesh 2014    27      0
```

Defaults: header=FALSE, sep=“ “ single space, dec=“.”
dec the character used in the file for decimal points.

```
> read.delim(file.choose())             for tab sep values
       StateUT.Year.Cases.Deaths
1      Andhra Pradesh,2014,1262,5
2    Arunachal Pradesh,2014,27,0
```

```
> read.csv2(file.choose())
       StateUT.Year.Cases.Deaths
1      Andhra Pradesh,2014,1262,5
2    Arunachal Pradesh,2014,27,0
```

Defaults: sep=“;”, header=TRUE, and dec=“,”

Missing Values in data files

- Missing Values in data file usually denoted by NA

```
> fw=read.csv(file.choose())
> fw
              StateUT Year Cases Deaths
1      Andhra Pradesh 2014  1262      5
2   Arunachal Pradesh 2014    27      0
3               Assam 2014    NA      0
4               Bihar 2014   297      0
5        Chhattisgarh 2014   440      9
```

Viewing named object

- > data2=c(2,3,4,6,7)
- > data1=2+3+4
- > fw=read.csv(file.choose())

data2, data1, fw are named object

Viewing previously loaded named object

```
> ls()
[1] "data1" "data2" "fw"
```

ls() command is used to list all the named items available

Viewing only Matching name by search pattern

```
> ls(pattern = 'd')
[1] "data1" "data2"
```

Removing object from R

- You can remove objects from memory and therefore permanently delete them using the rm() or remove() command

```
> rm(data1,data2,fw)
> data1
Error: object 'data1' not found
> data1=2+3+4
> data2=c(2,3,4,6,7)
> fw=read.csv(file.choose())
> rm(list=ls(pattern='d'))
> ls()
[1] "fw"
```

Types of data Items

- **Number Data**

Plain Values that are whole numbers are **integers** values. Values that contain decimals are **numeric**.

In R integers and decimals entire sample are treated as **numeric**

```
> data2
[1] 2 3 4 6 7
> data3
[1]  2.1 12.5 11.0 22.1
```

- **Text Items**

R is having two sorts of text data items

1. Plain Text labels these are called **character** values Plain Text and the quote marks to remind you .

```
> data
[1] "Mon"  "Tue"  "Wed"  "Thur" "Fri"  "Sat"
```

1. Plain Text without quote marks is called **factor** .

```
> data=factor(c("single","married","single","single","married"))
> data
[1] single  married single  single  married
Levels: married single
```

Converting between number and Text Data

as.character(): convert to plain text

```
> data1=as.character(data)
> data1
[1] "single"  "married" "single"  "single"  "married“
```

as.factor(): plain text to factor

```
> data2=as.factor(data1)
> data2
[1] single  married single  single  married
Levels: married single
```

as.integer(): decimal(numeric) to integers

```
> data3
[1]  2.1 12.5 11.0 22.1
> data1=as.integer(data3)
> data1
[1]  2 12 11 22
```

as.numeric(): integers to numeric

```
> data=c(1,2,4,5,7,8)
> data
[1] 1 2 4 5 7 8
> data1=as.numeric(data)
> data1
[1] 1 2 4 5 7 8
```

The Structure of Data Items

There are various data structures associated with R

1. Vector
2. Matrix
3. Data Frame
4. List
5. Factor

Vector

- Vector is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw.
- A vector's type can be checked with the **typeof()** function.
- Another important property of a vector is its length. This is the number of elements in the vector and can be checked with the function **length().**
- **It can be thought of a One Dimensional Object**

How to Create Vector

- Vectors are generally created using the c() function.
- Since, a vector must have elements of the same type, this function will try and coerce elements to the same type, if they are different.
- Coercion is from lower to higher types from logical to integer to double to character.

```
> x <- c(1, 5, 4, 9, 0)
> typeof(x)
[1] "double"
> length(x)
[1] 5
> x1 <- c(1, 5.4, TRUE, "hello")
> x1
[1] "1"     "5.4"   "TRUE"  "hello"
> typeof(x1)
[1] "character"
```

Creating a vector using : operator

- If we want to create a vector of consecutive numbers, the : operator is very helpful.

```
> x <- 1:7; x
[1] 1 2 3 4 5 6 7
```

Creating a vector using seq() function

- More complex sequences can be created using the seq() function, like defining number of points in an interval, or the step size.

```
> seq(1, 3, by=0.2
+ )
 [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
> seq(1, 5, length.out=4)
[1] 1.000000 2.333333 3.666667 5.000000
```

How to access Elements of a Vector?

- Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

Using integer vector as index

- Vector index in R starts from 1, unlike most programming languages where index start from 0.
- We can use a vector of integers as index to access specific elements.
- We can also use negative integers to return all elements except that those specified.
- But we cannot mix positive and negative integers while indexing and real numbers, if used, are truncated to integers.

```
> x
[1] 1 2 3 4 5 6 7
> x[3]              access 3rd  element
[1] 3
> x[c(2, 4)]        access 2nd and 4th element
[1] 2 4
> x[-1]             access all element except 1
[1] 2 3 4 5 6 7
> x[c(2, -4)]
Error in x[c(2, -4)] : only 0's may be mixed with negative subscripts
> x[c(2.4, 3.54)]       real converted to integers
[1] 2 3
```

Using logical vector as index

When we use a logical vector for indexing, the position where the logical vector is TRUE is returned.

```
> y[c(TRUE,FALSE,FALSE)]

[1] 1

> y

[1] 1 2 3
```

Using character vector as index

This type of indexing is useful when dealing with named vectors. We can name each elements of a vector.

```
> x <- c("first"=3, "second"=0, "third"=9)
> x
 first second  third
     3      0      9
> x["second"]
second
     0
> x[c("first", "third")]
first third
    3     9
```

How to modify a vector in R?

- We can modify a vector using the assignment operator.
- We can use the techniques discussed above to access specific elements and modify them.
- If we want to truncate the elements, we can use reassignments.

```
> x=c(1,2,3,4,5,6)
> x
[1] 1 2 3 4 5 6
> x[2] <- 0; x
[1] 1 0 3 4 5 6
> x[x<0] <- 5; x
[1] 1 0 3 4 5 6
> x[x<4] <- 5; x
[1] 5 5 5 4 5 6
> x <- x[1:4]; x      // truncate x to first 4 elements
[1] 5 5 5 4
```

How to delete a Vector?

We can delete a vector by simply assigning a NULL to it.

```
[1] 5 5 5 4
> x
[1] 5 5 5 4
> x <- NULL
> x[4]
NULL
```

R Lists

- List is a data structure having components of mixed data types.
- A vector having all elements of the same type is called atomic vector but a vector having elements of different type is called list.
- We can check if it's a list with typeof() function and find its length using length(). Here is an example of a list having three components each of different data type.

```
> x
$a
[1] 2.5
$b
[1] TRUE
$c
[1] 1 2 3
> typeof(x)
[1] "list"
> length(x)
[1] 3
```

How to create a list in R programming?

List can be created using the list() function.

```
> x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)
```

Here, we create a list x, of three components with data types double, logical and integer vector respectively.

```
> x
$a
[1] 2.5
$b
[1] TRUE
$c
[1] 1 2 3
```

Its structure can be examined with the str() function.

```
> str(x)
List of 3
 $ a: num 2.5
 $ b: logi TRUE
 $ c: int [1:3] 1 2 3
```

In this example, a, b and c are called tags which makes it easier to reference the components of the list.

However, tags are optional. We can create the same list without the tags as follows. In such scenario, numeric indices are used by default.

```
> x <- list(2.5,TRUE,1:3)

> x

[[1]]

[1] 2.5

[[2]]

[1] TRUE

[[3]]

[1] 1 2 3
```

How to access components of a list?

Lists can be accessed in similar fashion to vectors. Integer, logical or character vectors can be used for indexing.

```
> x[c(1:2)]                         # index using integer vector
[[1]]
[1] 2.5
[[2]]
[1] TRUE
> x[-2]                             # exclude second component
[[1]]
[1] 2.5
[[2]]
[1] 1 2 3
> x[c(T,F,F)]                       #using logical operator
[[1]]
[1] 2.5
```

```
> x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)
> x[c("a","b")]
$a
[1] 2.5
$b
[1] TRUE

> x["a"]
$a
[1] 2.5
> typeof(x["a"])
[1] "list"
> x[["a"]]
[1] 2.5
> typeof(x[["a"]])
[1] "double"
```

How to modify a list in R?

- We can change components of a list through reassignment. We can choose any of the component accessing techniques discussed above to modify it.

```
> x<-list("name" = "Aditya", "age" = 35, "speaks" =c("english", "french"))
> x
$name
[1] "Aditya"

$age
[1] 35

$speaks
[1] "english" "french"
```

```
> x[["name"]] <- "ABC"; x
$name
[1] "ABC"

$age
[1] 35

$speaks
[1] "english" "french"
```

How to add components to a list?

Adding new components is easy. We simply assign values using new tags and it will pop into action.

```
> x[["married"]] <- TRUE
> x
$name
[1] "ABC"

$age
[1] 35

$speaks
[1] "english" "french"

$married
[1] TRUE
```

How to delete components from a list?

We can delete a component by assigning NULL to it.

```
>  x[["age"]] <- NULL
> str(x)
List of 3
 $ name   : chr "ABC"
 $ speaks : chr [1:2] "english" "french"
 $ married: logi TRUE
> x$married<-NULL
> str(x)
List of 2
 $ name  : chr "ABC"
 $ speaks: chr [1:2] "english" "french"
```

Data Frames

- Data frame is a two dimensional data structure in R. It is a special case of a list which has each component of equal length.
- Each component form the column and contents of the component form the rows.

Create a Data Frame in R

We can create a data frame using the data.frame() function.

```
> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("Sunil","Santosh"))
> str(x)
'data.frame':  2 obs. of  3 variables:
 $ SN  : int  1 2
 $ Age : num  21 15
 $ Name: Factor w/ 2 levels "Santosh","Sunil": 2 1
```

Notice above that the third column, Name is of type **factor**, instead of a character **vector**.

By default, data.frame() function converts character vector into factor.

To suppress this behavior, we can pass the argument stringsAsFactors=FALSE.

```
> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("Anand",
"Harshit"),stringsAsFactors = FALSE)
> str(x)
'data.frame':2 obs. of  3 variables:
 $ SN  : int  1 2
 $ Age : num  21 15
 $ Name: chr  "Anand" "Harshit"
```

```
> x
  SN Age    Name
1  1  21   Anand
2  2  15 Harshit
> typeof(x)
[1] "list"
> class(x)
[1] "data.frame“
```

Functions of data frame

```
> names(x)
[1] "SN"   "Age"  "Name"
> ncol(x)
[1] 3
> nrow(x)
[1] 2
> length(x)       # returns length of the list, same as ncol()
[1] 3
```

Many data input functions of R like, read.table(), read.csv(), read.delim(), read.fwf() also read data into a data frame.

How to access Components of a Data Frame?

Components of data frame can be accessed like a list or like a matrix.

Accessing like a list

We can use either [, [[or $ operator to access columns of data frame.

Accessing with [[or $ is similar. However, it differs for [in that, indexing with [will return us a data frame but the other two will reduce it into a vector.

```
> x["Name"]
   Name
1  Anand
2 Harshit
> x$Name
[1] "Anand"   "Harshit“
> x[["Name"]]
[1] "Anand"   "Harshit"
> x[[3]]
[1] "Anand"   "Harshit"
```

Accessing like a matrix

Data frames can be accessed like a matrix by providing index for row and column. To illustrate this, we use datasets already available in R. Datasets that are available can be listed with the command library(help = "datasets"). We will use the trees dataset which contains Girth, Height and Volume for Black Cherry Trees.

```
> str(trees)
'data.frame':	31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
> head(trees,n=3)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
> tail(trees,n=3)
   Girth Height Volume
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

We can see that trees is a data frame with 31 rows and 3 columns. We also display the first 3 rows of the data frame.

```
> trees[2:3,]                                    #Selects Second and third row
  Girth Height Volume
2   8.6     65   10.3
3   8.8     63   10.2
> trees[trees$Height > 82,]                      # selects rows with Height greater than 82
   Girth Height Volume
6   10.8     83   19.7
17  12.9     85   33.8
18  13.3     86   27.4
31  20.6     87   77.0
> trees[10:12,2]
[1] 75 79 76
```

We can see in the last case that the returned type is a vector since we extracted data from a single column.

This behavior can be avoided by passing the argument drop=FALSE as follows.

```
> trees[10:12,2, drop = FALSE]
   Height
10     75
11     79
12     76
```

How to modify a Data Frame in R?

Data frames can be modified like we modified matrices through reassignment.

```
> x
  SN Age    Name
1  1  21   Anand
2  2  15 Harshit
> x[1,"Age"] <- 20; x
  SN Age    Name
1  1  20   Anand
2  2  15 Harshit
```

Adding Components

Rows can be added to a data frame using the rbind() function.

```
>  rbind(x,list(1,16,"Riya"))
  SN Age    Name
1  1  20   Anand
2  2  15 Harshit
3  1  16    Riya
```

Similarly, we can add columns using cbind().

```
> cbind(x,City=c("Dehradun","New Delhi"))
  SN Age    Name      City
1  1  20   Anand  Dehradun
2  2  15 Harshit New Delhi
```

Since data frames are implemented as list, we can also add new columns through simple list-like assignments.

```
> x$City <- c("New Delhi","Dehradun"); x
  SN Age    Name      City
1  1  20   Anand New Delhi
2  2  15 Harshit  Dehradun
```

Deleting Component

Data frame columns can be deleted by assigning NULL to it.

```
> x$City <- NULL
> x
  SN Age    Name
1  1  20   Anand
2  2  15 Harshit
```

Similarly, rows can be deleted through reassignments.

```
> x <- x[-1,]
> x
  SN Age    Name
2  2  15 Harshit
```

R Factors

- Factor is a data structure used for fields that takes only predefined, finite number of values (categorical data). For example: a data field such as marital status may contain only values from single, married, separated, divorced, or widowed.

- In such case, we know the possible values beforehand and these predefined, distinct values are called levels.

How to create a factor in R?

We can create a factor using the function factor(). Levels of a factor are inferred from the data if not provided.

```
> x <- factor(c("single", "married", "married", "single"));
> x
[1] single  married married single
Levels: married single
> x <- factor(c("single", "married", "married", "single"), levels = c("single", "married", "divorced"));
> x
[1] single  married married single
Levels: single married divorced
```

evels may be predefined even if not used.

factors are stored as integer vectors.it is seen by checking from its structure

```
> x <- factor(c("single","married","married","single"))
> str(x)
 Factor w/ 2 levels "married","single": 2 1 1 2
```

How to access compoments of a factor?

```
> x[3]                          #Third Element
[1] married
Levels: married single
> x[c(2, 4)]                    #second and fourth Element
[1] married single
Levels: married single
> x[-1]                         #all element except 1
[1] married married single
Levels: married single
> x[c(TRUE, FALSE, FALSE, TRUE)]                    #using logical vector
[1] single single
Levels: married single
```

How to modify a factor?

Components of a factor can be modified using simple assignments. However, we cannot choose values outside of its predefined levels.

```
> x
[1] single  married married single
Levels: single married divorced
> x[2] <- "divorced“                    #Modify Second Element
> x
[1] single   divorced married  single
Levels: single married divorced
> x[3] <- "widowed“                     #Cannot Assign Values outside levels
Warning message:
In `[<-.factor`(`*tmp*`, 3, value = "widowed") :
  invalid factor level, NA generated
```

Adding value to levels first

```
> levels(x) <- c(levels(x), "widowed")
> x
[1] single   divorced <NA>     single
Levels: single married divorced widowed
> x[3] <- "widowed"
> x
[1] single   divorced widowed  single
Levels: single married divorced widowed
```

R Matrix

- Matrix is a two dimensional data structure in R programming.
- Matrix is similar to vector but additionally contains the dimension attribute. All attributes of an object can be checked with the attributes() function (dimension can be checked directly with the dim() function).
- We can check if a variable is a matrix or not with the class() function.

How to create a matrix in R programming?

- Matrix can be created using the matrix() function.

- Dimension of the matrix can be defined by passing appropriate value for arguments nrow and ncol.
- Providing value for both dimension is not necessary. If one of the dimension is provided, the other is inferred from length of the data.

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

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

We can see that the matrix is filled column-wise. This can be reversed to row-wise filling by passing TRUE to the argument byrow.

```
> matrix(1:9,nrow=3,byrow=TRUE)                    # fill matrix row-wise
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

It is possible to name the rows and columns of matrix during creation by passing a 2 element list to the argument dimnames.

```
> matrix(1:9,nrow=3,dimnames=list(c("X","Y","Z"),c("A","B","C")))
  A B C
X 1 4 7
Y 2 5 8
Z 3 6 9
```

These names can be accessed or changed with two helpful functions colnames() and rownames().

```
> X<-matrix(1:9,nrow=3,dimnames=list(c("X","Y","Z"),c("A","B","C")))

> colnames(X)

[1] "A" "B" "C“

> rownames(X)

[1] "X" "Y" "Z“

> colnames(X)<-c("C1","C2","C3")

> rownames(X)<-c("R1","R2","R3")

> X

   C1 C2 C3

R1  1  4  7

R2  2  5  8

R3  3  6  9
```

Another way of creating a matrix is by using functions cbind() and rbind() as in column bind and row bind.

```
> cbind(c(1,2,3),c(4,5,6))
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> rbind(c(1,2,3),c(4,5,6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Finally, you can also create a matrix from a vector by setting its dimension using dim().

```
> x <- c(1,2,3,4,5,6)
> x
[1] 1 2 3 4 5 6
> class(x)
[1] "numeric"
> dim(x)<-c(2,3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> class(x)
[1] "matrix"
```

How to access Elements of a matrix?

We can access elements of a matrix using the square bracket [indexing method. Elements can be accessed as var[row, column]. Here rows and columns are vectors.

Using integer vector as index

We specify the row numbers and column numbers as vectors and use it for indexing.

If any field inside the bracket is left blank, it selects all.

We can use negative integers to specify rows or columns to be excluded.

An element at the *m*th row, *n*th column of A can be accessed by the expression A[m, n].

```
> x
    [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x
    [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x[2,3]                                  #Select 2nd  row and 3rd  col
[1] 8
> x[c(1,2),c(2,3)]                        # select rows 1 & 2 and columns 2 & 3
    [,1] [,2]
[1,]    4    7
[2,]    5    8
> x[c(3,2),]                              # selects 3rd and 2nd row
    [,1] [,2] [,3]
[1,]    3    6    9
[2,]    2    5    8
```

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

if the matrix returned after indexing is a row matrix or column matrix, the result is given as a vector.

```
> x[1,]
[1] 1 4 7
> class(x[1,])
[1] "integer"
```

This behavior can be avoided by using the argument drop = FALSE while indexing.

```
> x[1,,drop=FALSE]
     [,1] [,2] [,3]
[1,]    1    4    7
> class(x[1,,drop=FALSE])
[1] "matrix“
```

It is possible to index a matrix with a single vector.

While indexing in such a way, it acts like a vector formed by stacking columns of the matrix one after another. The result is returned as a vector.

```
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x[1:4]
[1] 1 2 3 4
> x[c(3,5,7)]
[1] 3 5 7
```

Using logical vector as index

- x[c(TRUE,FALSE,TRUE),c(TRUE,TRUE,FALSE)]

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

- x[c(TRUE,FALSE),c(2,3)]
- X[c(1,3),c(2,3)]

```
     [,1] [,2]
[1,]    4    7
[2,]    6    9
```

```
> x[x>5]                    #Select elements greater then 5
[1] 6 7 8 9

> x[x%%2 == 0]              #Select even elements
[1] 2 4 6 8
```

Using character vector as index

Indexing with character vector is possible for matrix with named row or column. This can be mixed with integer or logical indexing.

```
> X
   C1 C2 C3
R1  1  4  7
R2  2  5  8
R3  3  6  9
> X[,"C1"]
R1 R2 R3
 1  2  3
> X[TRUE,c("C1","C3")]
   C1 C3
R1  1  7
R2  2  8
R3  3  9
```

```
> X[2:3,c("C1","C3")]
   C1 C3
R2  2  8
R3  3  9
```

How to modify a matrix in R?

- We can combine assignment operator with the above learned methods for accessing elements of a matrix to modify it.

```
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x[2,2] <- 10; x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2   10    8
[3,]    3    6    9
> x[x<5] <- 0; x
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
[3,]    0    6    9
```

A common operation with matrix is to transpose it. This can be done with the function t().

```
> t(x)
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0   10    6
[3,]    7    8    9
```

We can add row or column using rbind() and cbind() function respectively. Similarly, it can be removed through reassignment.

```
> cbind(x, c(1, 2, 3))          #Add COL
     [,1] [,2] [,3] [,4]
[1,]    0    0    7    1
[2,]    0   10    8    2
[3,]    0    6    9    3
> rbind(x,c(1,2,3))                #ADD Row
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
[3,]    0    6    9
[4,]    1    2    3
> x <- x[1:2,]; x                  #Remove Last Row
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
```

Dimension of matrix can be modified as well, using the dim() function.

```
> x
     [,1] [,2] [,3]
[1,]    0    0    7
[2,]    0   10    8
> dim(x) <- c(3,2); x                          #3*2 Matrix
     [,1] [,2]
[1,]    0   10
[2,]    0    7
[3,]    0    8
> dim(x) <- c(1,6); x                          #1*6 Matrix
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0   10    7    8
> c(x)                                    #deconstruction of matrix
[1] 1 2 3 4 5 6 7 8 9
```