UNIT-V FILES

Accessing the keyboard and monitor.

R provides several functions for accessing the keyboard and monitor. So here we saw some function scan(), readline() paste() and cat()

scan() function:-

21.txt	22.txt	23.txt	24.txt
123	123	abc	abc
45	4.25	de f	123 6
6	6	g	y

Eg:- > scan ("21.txt")
Read 4 items
[1] 123 4 5 6
> scan ("22.txt")
Read 4 items
[1] 123.0 4.25 5.0 6.0
> scan ("23.txt")
Error in scan (file, what, nmax, sep, dec, quote, skip, nlines, na.strings, scanel) expected 'areal' got 'abc'
> scan ("23.txt", what = " ")
Read 4 items
[1] "abc" "de" "f" "g"
> scan ("24.txt", what = " ")
Read 4 items
[1] "abc" "123" "6" "y"

Scan() to read in a vector, whether numeric or character from a file or the keyboard with white spaces in the above example we got a vector of four integers that is numeric then the second file one number was non-integral the others were floating point-number too. In third case we got error message because these assigns a character string so the non-numeric content of the file is produced an error. Then we include.

what = " " these assigns a character string to what.
If we assign a variable to a scan function is

```
>v ← scan ("2.txt")
```

scan() assumes that the items of the vector are separated by white space which includes blanks and return lines and horizontal taps for that we use the sep argument for other situations, sep to the new line characters to read in each line as a string

```
Eg:- x1 ← scan ("2s.txt", what = " ")
     Read 4 item
     > x2 ← scan ("23.txt", what = " ", sep = "\n")
     Read 3 items.
     > x1  o/p:- [1] "abc" "de" "f" "g"
     > x2
     [1] "abc" "de f" "g"
     > x1 [2]
     [1] "de"
     > x2 [2] o/p: "def"
```

Scan() to read from the keyboard while specifying an empty string for the file name.

```
>v ← scan ("")
1: 1 2 5 13
4: 3 4 5
7: 8
8:
Red 4 items.
>v o/p: [1] 12 5 13 3 4 5 8
```

Do not wish scan() to announce the no. of item it as read including the argument quiet = TRUE

Readline() function:-

If we want to read in a single line from the keyboard readline() is one of the function

```
Eg:- w ← readline()
     2bc de f
     > w
     [1] "abc" "de" "f"
```

ex:- z

names	ages
John	25
marry	28
Jim	19

> z ← read.table ("z", header = TRUE)

> z

names	ages
John	25
marry	28
Jim	19

In the above example first line contains an optimal header specifing column names.

Note:- scan would not work here, because our file as a mixture of numeric and character data

ex:-
```
1 0 1
1 1 1
1 1 0
1 1 0
0 0 1
```

> m ← matrix (scan ("m"), nrow = 5, by row = TRUE)

o/p:-
```
1 0 1
1 1 1
1 1 0
1 1 0
0 0 1
```

In the above example we use the scan() to read in the matrix row by row and byrow option in the function matrix indicates that we are defining the elements of a matrix rowwise rather than columnwise

we can use read.table() function which returns a data frame and then convert via as.matrix()

ex:- > read.matrix ← function (filename))
{
as.matrix (read.table (filename))
}

Reading txt files:

In computer literature there is often a distinction made b/w text files and binary files in that the term text file is to mean a file that consist mainly of ascii characters are coding for some other human language. For that we can use read lines() function to read in a text file either one line at a time (or) in a single operation.

Eg:- > z1

names	ages
John	25
mary	28
Jim	19

> z1 ← readlines ("z1")

> z1

[1] "John 25" "mary 28" "Jim 19"

Since each line treated as a string the return value here is a vector of strings for this first we need to create a connection.

Introduction to connections:

Connections is a term for a fundamental mechanism used in various kinds of input output operations. It will be used for file access. The connection is created by calling file() , URL() or one of several other R functions

Eg:- > ? connection

> c ← file ("z1", "r")

> readlines (c, n=1)

[1] "John 25"

> readlines (c, n=1)

[1] "mary 28"

> readlines (c, n=1)

[1] "Jim 19"

> readlines (c, n=1)

Character (0)

We opened the connection, assign the result to see and than read the file one line at a time, as specified by argument n=1. When "R" encounter the end of file (eof) it return an empty file. So, we need to set up a connection so that R code keep track of our positions in the file as we read to it through it

Eg:- we can detect eof in our example

```
>c ← file ("z", "r")
>while (TRUE)
{
 r1 ← readlines (c, n=1)
  if (length (r1)==0)
  {
  Print ("reached the end")
   break
  }
  else
   Print (r1)
  }
```

o/p:- [1] "John 25"
[2] "many 28"
[3] "Jim 19"
[1] "reached the end"

To start at the beginning of the file we can use seek()

Seek(): means that we which to position the file pointer zero characters from the start of the file. In other words directly at the beginning.

```
>c ← file ("z1", "r")
>readlines (c, n=2)
[1] "John 25" "many 28"
>seek (con=c, where =0)
   [1] 16
 >readlines (c, n=1)
   [1] "John 25"
```

The call returns 16 means that the file pointer was at position 16 before we make the call, that means the 1st line consist of "John 25" + the end of the line character for the total of 8 characters and the same is true for the 2nd line So, after reading the 1st two line where at position 16.

Accessing files on remote via machine via URL's

Certain input output functions such as read.table and scan accept URL's as arguments

```
> ucl <- "http://archive.ics.uci.edu/ml/machine-learning-
                                        databases/"

> ucl2 <- paste(ucl, "echocardiogram/echocardiogram.data",
                                        sep = " ")

> ecc <- read.csv(ucl2)
> head(ecc)
  x11  x6  x71
1  19   .    .
2  16   .    .
3  57   .    .
4   .   .    .
6   .   .    .
```

Writing to a file:

R file reads are more probably common than write But writes are some times necessary and some methods for writing to files in that we use the function write.table works very much like read.table accept that it writes a data.frame accept that instead of reading one.

```
> kids <- c("Jack", "Jill")
> ages <- c(12, 10)
> d <- data.frame(kids, ages, stringAsFactor = FALSE)
> d
  kids  ages
  Jack   12
  Jill   10
> write.table(d, "kids")
                    <- The file kids will now
>kids
    "kids" "ages"
"1" Jack    12
"2" Jill    10
```

The file %. In the case of writing a matrix of a file just take state that we do not want rows and columns name

> write.table (xc, "xcnew", row.names = FALSE, col.names = FALSE

So Here the "xc" is the file. Is stored in "xcnew" that is table you are not at all mention the colum names, row names that is optional in case you are writing a matrix than you should mention the column names and row names

Eg:- > cat ("abc\n", file = "v")
> cat ("de\n", file = "v", append = TRUE)

abc
de

The 1st call create the file "v" consisting of 1 line which contains abc the second call apends a seacond line call the file is automatically saved after each operation.

we can write multiple fields in a single line

Eg:- cat (file = "v", 1, 2, "xyz\n")
1 2xyz

Here we use the write lines the counter part of readlines if we use a connection must specify "w" to indicates we are writing to the file not reading from it

> c <- file ("wwww", "w")
> writelines (c("abc", "de", "f"), c)
> close(c)

abc
de
f

Getting file and directory Information 1/07/2022

R has vairty of functions for getting information about directories and files, setting file access permissions & so on In that functions a few we will see

- file.info(): It gives the file size, creation time, directory vs ordinary file status and so on
  For Each file whos name is in the argument, a character vector.
- dir(): It returns a character vector listing the names of all the files in the directory specified in the first argument. If the optional argument

recursive = TRUE is specified. the result will show the entire directory tree rooted at the 1st argument

file.exists(): It returns a boolean vector indicating whether the given file exist for each name dirme in the 1st argument, A character vector.

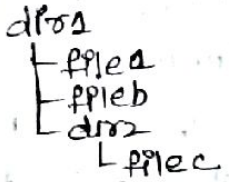
.getwd(): and setwd(): It is used to determine and or change the current working directory

In the above all files and directory related functions you want to know use the following option >? files.

Some the contents of many files:

Here a function to find the some of the contents in all files in a directory tree.

In the below example a director dir1 contain the files "file a" and "file b" as well as a sub directory dir2 contains which holds the "file c" the contains of the files are



file a : 5, 12, 13

file b : 3, 4, 5

file c : 24, 25, 7

If dir1 is in our current directory, the call sumtree ("dir1") will yield the sum of those 9 numbers is 98 other wise we need to specify the full path name of dir1 such as Sumtree ("/home/nm/dir1")

```
Eg: Sumtree ← function (dirtr)
                          ↳ directory name
{
total of terms ← tot ← 0
    fls ← dir (dirtr, recursive = TRUE)
    for (f in fls)
    {
    f ← file.path (dirtr, f)
    if (!file.info (f) $ is dir)
    {
    tot ← tot + sum (scan (f, quiet = TRUE))
    }}
    return (tot)
}
```

In the above example R code has done the recursion for option in dir

Accessing the Internet:

R's socket facilities give the programmer access to the Internet's TCP/IP for readers who are not familiar with this protocal we begin with an overview of TCP/IP

Over view of TCP/IP.

The team network refers to a set of computers Connected together locally, without going through the Internet

the Internet as it name implies connects networks a network in the internet is connected to the one or more other networks via routers, which are special purpose Computers that connected two or more networks together

Every computer on the Internet has an internet Protocal i.e (IP Address) this is numeric, But it Can be stated in charaters as in www.google.com which is than translated into the numeric address by the domain name service.

the IP address is not enough when A sends a message to B they may be several application at computer B. that are receiving internet messages, such as web browsing Image service and so on

So that A specifies a port number in addition to the IP address. the port number indicates which program running at B. is intended as the resepent or reciver. A will also have a port number. so that the responce from B reaches the correct application at A

Sockets in R:

R provides various functions

1. readLines() and writeLines()

this allow we to program as if TCP/IP where Sending message line by line even though this is not actually the case. If our applications is natually viewed in terms of lines, this two functions can be quite handing

serialize() and unserialize() :-

We can these two to send R objects such as a matrix or the complex output of a call to a statistical function. the object is covered to character string form by the sender and then converted back to the original object form at the receiver.

⇒ Readbin() and writebin() :

This are for sending data in binary form R . using serialize and unserialize may be more convenient but far more time consumming . this not only because numbers must be converted to and from there character representations But also because the character representation is much longer which means greater transmition time.

R Socket functions :-

Socket connection() : This establish an R connection via sockets we specify the port number in the argument port and state whether a server or client is to be created by setting the argument server to TRUE or FALSE respectively in the client case we must also supply the servers IP address in the argument host

Server - port no,
Client - host no.

- Socket select() : this is useful when a server is connected to multiple clients.

Sockslist() : is a list of connections and its return value is the sub list of connection that have data ready for the server to read.

Implementing parallel R

Some statistical analysis have very long runtimes so their naturally have been quite a bit of interest in parallel R, in which several R process copperate on a given task. and another reason to go parallel is memory limitation that means if one machine does not have enough memory for the task it may help to pool the memory of several machines in some way. in

In that situations sockets play a key role in many Parallel R packages that are :

- Snow : the serve sends out work task to the clients. the clients performs their task and send the results back to the server, which assembles them into the final results. Communication is done with serialize() and unserialize() and the server uses Socket select() to determine which client results are ready.

task → client
results merge → snow

- Rdsm :- Implements a virtual shared memory and the server is used to store the shared variables. the clients contact the server when where they need to read or write a shared variables. Communication b/w server & clients is done with ReadBin() and writebin() instead of serialize() and unserialize().

* String Manupulations :-

R is a statistical language with numeric vectors and matrees playing a central role, character strings are important as well ranging from Birth dates storing in medical research data files to text mining application, Character data arises quite frequently in R programs

Accordingly R has a no.of string manupulations utilities, In that we will see some manupulations. are.

An Overview of string manipulation functions :-

grep() : The call grep

Syntax: grep(pattern, x)

Searchs for a specified sub string pattern in a vector x of strings. that is x has an 'n' elements it contains n strings.

Eg:- grep("pole", c("Equator", "North pole", "South pole"))

> grep ("pole", c ("Equator", "Northpole", "Southpole"))
integer (0)

In case 2: String pole was not found so an empty vector was return.

nchar():
The call nchar nchar(x) find the length of the string x.

Eg: nchar ("South pole")
[1] 10 (with space)

Paste():
The call paste() concatinate several strings returns the result in one long string.

Eg: > paste ("North", "pole")
[1] North pole
> paste ("North", "pole", sep = ".")
[1] North . pole.

Sprintf(): The call sprintf assemble a string from parts in a formated manner

Eg: sprintf(---)
> i <- 8
> s <- sprintf ("the square of %d is %d", i, i^2)
> s
[1] "the square of 8 is 64"

Substr(): The call Substr.
Substr(x, start, stop) returns the substring in the given character position range start to stop in the given string x

Eg: Substring ("Equator", 3, 5)
1 2 3 4 5 6 7
[1] uat

strsplit(): The strsplit the call.

strsplit(x, split) splits a string x into an R list of substrings based on another string split in x

Eg:- > strsplit("6-16-2011", split="-")

[[1]]

[1] "6" "16" "2011"

regexpr(): The call regexpr(pattern, text) finds the character position of the first instance of pattern with in text.

Eg:- > regexpr("sat", "Equator")

[1] 3

gregexpr(): The call gregexpr(pattern, txt) is same as the regexpr, (regression expression) But it finds all instance of Parttern

Eg:- gregexpr("iss", "mississippi")

[[1]]

[1] 2 5

this finds that iss appears twice in mississippi starting at character position to end file.

Graphics :-

R has a very rich set of graphics facilities facilities.

Creating graphs :

web sites :- http://www.r-project.org/
http://addictedtor.free.fr/graphiques

In the above two galleries provides R's graphical power, few colour full examples browse through above web sites.

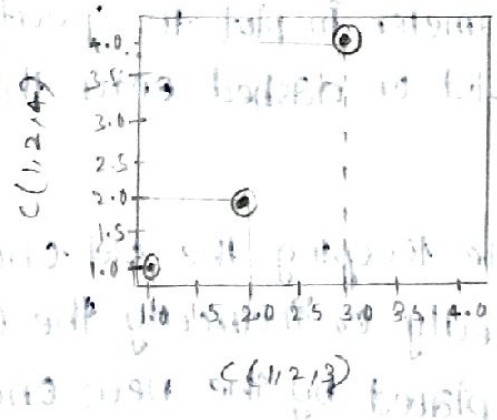
Creating graphs:

The fundamental functions for creating graphs float plotting from adding lines and points to attaching a legend

Plot() function :-

Plot is a generic function (or) a place holder for a family of functions. the function that is actually called depends on the class of the object on which it is called we call plot with an "x" vector and a "y" vector which are interpreted as a set of pairs in the (x,y) plane.

eg:- plot(c(1,2,3), c(1,2,4))

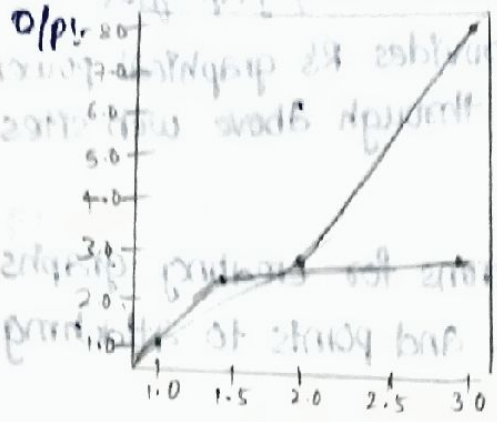


Ablines() function :

we have an empty graph, ready for the next stage which is adding a new line. the call to abline then adds a line to the current graph

eg:- > x <- c(1,2,3)
> y <- c(1,3,8)
> plot(x,y)
> lmout <- lm(y~x)
> abline(lmout)

o/p:-



Call to add a line from (1.5, 3) to (2,5,3) to the present graph

: lines (c(1.5, 2.5), c(3,3)) (line in same graph)

If we want the lines to connect the dots. But don't want the dots themselves. Include type = "l" in our call to lines() (or) plot()

plot(x, y, type = "l") # gives dots in prev graph

If we want know the parameter in plot to specify the type of the line such as solid or dashed enter this command

> help(par) — # gives solid lines or dot lines

Starting a new graph while keeping the old one.
Here we call the plot directly or indirectly the current graph window will be replaced by the new one.
use the commands for our operating systems

- on linux system, call x11()
- on mac, call macintosh()
- on windows, call windows()

Suppose we wish to plot two histograms of vectors x and y and view them side by size on linux operating system use this commonds.

```
> hist(x)
> x11()
> hist(y)
```

Points() function :-

The points() function add a set of x, y points with lebals for each to the currently displayed graph

Suppose we enter that command. the result would be ~~points(tests~~ Super impose on to the current graph the points of the exam scores from that example using '+' sign to map them

```
Points(testsScores$exam, testscores$exam3, pch = "+")
```

Point color. and background colour option :-

```
Par(bg = "Yellow")
```

legend() :-

The legend function is used not super signly to add a legend sap. to multi curve graph that means the green curve is for men and the red curve displays the data for the women.

So use this commond :-

```
examp(legend)
```

```
legend(x, y = NULL, legend = fill, col, bg)
```

Here in the above syntax legend() is a function used for multicurve graph

x and y are the co-ordmates. used to the position of legend. fill is the colours to use for liemg on the boxes beside the legend text, col is the colour of lines and points beside the legend text,

bg is the background colour of legend box.

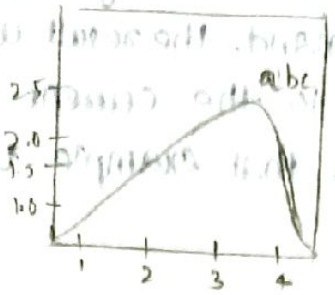
text() function :-

Use the text() function to place some text anywhere in the current graph

Syntax: Eg. text(x, y, labels)

So, x and y are co-ordinates, label is the text to be written.

Eg:- text(2.5, 4, "abc")



locator() function :-

The function used for save the locations at the desire spot in the graph if you click on the spot it returns x and y co-ordinates

locator(1) (get the location of one place)

Eg:- > hist(c(12, 5, 13, 25, 16))
> locator(1)
$x
6.239237
$y
[1] 1.221038

In the above example, we extract the co-ordinates of the point we clicked. If we give locator(2) get the locations of two places and so on

Restoring a plot :-

recordplot R has no "undo" commond. if we need in R we use record_plot() for save the file and then later restore it with replayplot()

Customizing graphs :-

In the above concepts we starting with plot now we want to know enhance graphs using many options R provides.

Changing Character Sizes : cex() (character expand)

this function to expand or shrink the characters with in a graph.

size of character

Eg:- text (2.5, 4, "abc", cex = 1.5)

Changing the Range of Axies :-

xlim and ylim

This function is used for increase the ranges or decrease the ranges on the x and y axises.

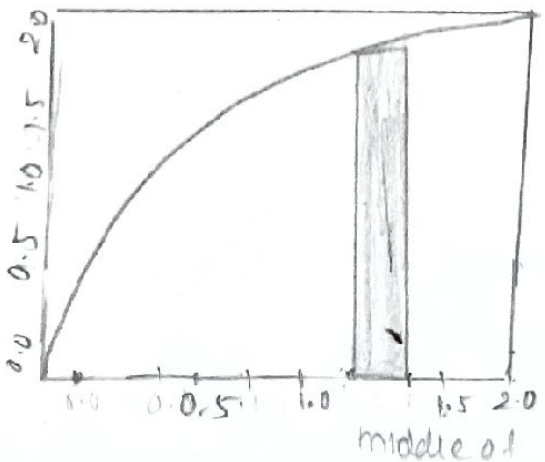
to expand to 60.

Eg:- text (2.5, 4, "abc", cex = 1.5, xlim = 2, ylim = 2)

Polygon() :

we use to draw polygonal objects example

ex :- polygon (c(1.2, 1.4, 1.4, 1.2), c(0, 0, f(1,3), f(1,3)), density = 10)



lowess() and loess()

This two functions ploting a cloud of points collected 'connected' or not may give you nothing but an uninformative mess the two functions are similar but have different defaults and other options

eg:- > plot (testscores)

> lines (lowess (testscores))