R programming
Notes

Contents

1 Lecture 1

1.1 Reading and writing data

Function name: read.table()
Input arguments: file name, header = logical index if the file has a header line, sep = a string indicating how the columns are separated.

Function read.csv() is identical to read.table except that the default separator is comma while for read.table the separator is space.

Function name: readLines(); it is used to read text lines

When the dataset is loaded in the R, it is stored in RAM so it is important to have rough estimate of the data size. Another important argument as colClasses can be used to specify of different data class and R does not have to deal with it automatically that makes the program slow when dealing with large datasets.
Textual formatting: Dumping and Dputing provides the editable textual formats. dput constructs the R code that can be used to get the R object. Similarly multiple objects can be departed using the dump function and read back in using source.
Interfaces: Data are read in using connection interfaces. Connection can made to files or to other sources like webpages. for example file is used to open connections to a file, url to open a connection to we page.

Function name: file() Input arguments: “r” - read only, “w” - write only etc.

2 Lecture 2

2.1 Control structure

IF loop:
General loop structure:-

```
if(condition){
    expression
}
else if(condition){
    expression
}
else {
}
```

FOR loop: For loop can be nested, so a loop can be inside another loop.
General loop structure:-

```
for(i in 1:10){
    expressions
}
```

While loop: It begins by testing a condition. If it is true, the code is executed. While loops can potentially result in infinite loop so one need to be careful when executing while loop.
General loop structure:-
while(condition){
    Expressions
    condition control structure
}

Repeat loop: Repeat initiates an infinite loop. The only way to exit a repeat loop is to call break. It is better to use for loop than using the repeat loop.
General loop structure:-
repeat {
    Expression
    condition of convergence and break
}

Next: It is used to skip an iteration of a loop.
General loop structure:-
for (i in 1:100){
    if(i ¡= 20){
        next
    }
    Expression
}

2.2 Functions

Function are first class object which means that they can passed as arguments to other functions or it can be nested.
General loop structure:-
f ¡- function(arguments){
expressions
}
Function arguments can have the default arguments. If the arguments are named, the order of the arguments can be reversed however having one named argument and another unnamed argument can lead to confusion. To see the arguments of a function type–¿
args(function name)

... represents a variable number of arguments that are usually passed on to other functions.
args(paste) – Paste function allows you to paste different variables. It starts with ... as different arguments can be included in the paste function. However after ... you can use partial matching of the variables. By default sep paste space between the arguments.

3 Lecture 3

Loop functions: Loop functions have apply as the keyword in them. These looping functions make the implementation of loops easier.

lapply: It loops over a list and evaluate a function on each element. lapply always returns a list, regardless of the class of the input.
Function name: lapply
Arguments: (1) a list x (2) a function FUN and (3) other arguments via its ... arguments.
Example code:
x <− 1:4
lapply(x, runic, min= 0, max = 10)
lapply and friends make heavy use of anonymous functions. Anonymous functions are those which do not have a name. For example a function for extracting the first column of each matrix.

lapply(x, function(elt) elt[,1])

sapply: A variant of lapply that simplifies the results of the lapply. If the result of lapply is a list where every element is length 1, then a vector is returned for spply. If the result is a list where every element is a vector of the same length, a matrix is returned.
Example: sapply(x, mean)

Apply: Apply function is applied on array, to apply the function across the entire array.
Example: apply(x, 2, mean) where 2 is used to get the column wise mean.
apply(x,1, quantile, probs = c(0.25, 0.75))

To calculate row mean or row sums we have functions like, rowSums, rowMeans, colSums etc. They are faster than using apply function.

tappply It is used to apply a function over subsets of a vector.
Function name: str(tapply)
usage: function(X, INDEX, fun = NULL, ... , simplify = TURE)
Example: x <− c(rnorm(10), runif(10), rnorm(10,1))
f<− gl(3,10)
tapply(x, f, mean)
Provides the mean for x depending on the levels. If the simplify is not applied we get a list as a solution.

split: It takes a vector or other objects and splits it into groups determined by a factor or list of factors.
Function name: function(x, f, drop = FALSE, ...)
Usage: split(x,f) in above example will give a list, but this does not apply the summary statistics. It splits the x in different levels indicated by levels in f. Once split, tapply and other functions can be used onto it. e.g.

lapply(split(x,f), mean) : here tapply will do the same thing.
Usage: s < − split(airquality, airquality$Month)
lapply(s, function(x) colMeans(x[,c("Ozone", "Solar.R", "wind")])
However it will give a list. Here we use lapply over split along with an anonymous function.

mapply: A multivariate apply function that applies a function in parallel over a set of arguments. It allow one to apply the same functions over different list without using for loop.
Function usage: function(FUN, ... , Moreargs = NULL, simplify = TRUE)
Example: list (rep(1,4), rep(2,3). rep(3,2), rep(4,1)) instead we can also do
mapply(rep, 1:4, 4:1)
Here mapply is used to apply same function over different arguments

3.1 Debugging

How to get information about errors:

- Message: Normal message, diagnostic message
- warning: something is wrong but not necessarily fatal, execution of the function continues
- error: tatar problem has occurred and the execution of the function stops.
- condition: Something unexpected has occur

Functions to debug the program:

- trackback: prints out the function call stack after an error occurs; does nothing if no error.
- debug: flags a function for "debug" mode which allows you to step through execution of a function one line at a time.
- browser: suspends the execution of a function whenever it is called and puts the function in debug mode. Can be started anywhere in the function.
- trace: allows you to insert debugging code into a function a specific places.
- recover: allows you to modify the error behaviour so that you can browse the function call stack.