Introduction to R

Outline

- Processing Data in R
- Programming in R
- Graphical Analysis in R
- Statistical Analysis in R

Workspace: Save() and load()

```
yourname <- readline("What is your name?")
What is your name? Mohan

> save(yourname,file= "yourname.rda")
> rm(yourname)
> yourname
Error: object 'yourname' not found

> load("yourname.rda")
> yourname
[1] “Mohan"
```

scan() and clipboard

- Default type numeric
- X<- scan()

- Other Data type
- X<- scan(what=(‘character’))

- X<-scan(what=(0,0,0,””))
- X<-
  scan((what=list(cust_id=0,sales_total=0,num_of_
  orders=0,gender='logical'), sep=",")

Import / Export

To set working directory

setwd(D:/Odd Sem 2015/R")

getwd()

R can read data stored in text (ASCII) files, files in other formats (Excel, SAS, SPSS, . . .), and access SQLtype databases.

Reading / Writing data in a file

Reading data from a file

```
mydata <- read.table("data.dat")
mydata <- read.table("data.txt“, header=TRUE)
mydata <- read.table("data.txt“, header=TRUE, sep=“\t”)
```

```
sales<- read.csv(“yearly_sales.csv”, header = TRUE);
sales<- read.csv(file.choose(), header = TRUE);
```

Saving data in a file

```
write.table(x, file=”data.txt”, sep=“\t”)
```

Executing scripts

- File → New script
- Type couple of commands in R Editor:
- x<-1:10
- x=x+1
- x
- ls()
- Save as script1.R
- Open script1.R. Edit → run all

- A note pad can also be used to create a script

Grouping

- Grouped expressions
  - R is an expression language in the sense that its only command type is a function or expression which returns a result.
  - Commands may be grouped together in braces, {expr 1, . . ., expr m}, in which case the value of the group is the result of the last expression in the group evaluated.

Example:

```
x<-
+ {y=10
+ z=10
+ }
> x
[1] 10
```

Control Statements

- Control statements
  - –if statements
  - –The language has available a conditional construction of the form

    if (expr 1) expr 2 else expr 3

    where expr 1 must evaluate to a logical value and the result of the entire expression is then evident.
  - –a vectorized version of the if/else construct, the ifelse function. This has the form
    - ifelse(condition, a, b)

Branching

```
if (logical expression) {
   statements
}
else {
   alternative statements
}
```

else branch is optional

{ } are optional with one statement

```
ifelse (logical expression, yes
statement, no statement)
```

Repetitive execution

- for loops, repeat and while
  - for (name in expr 1) expr 2

    where name is the loop variable. expr 1 is a vector expression, (often a sequence like 1:20), and expr 2 is often a grouped expression with its sub-expressions written in terms of the dummy name. expr 2 is repeatedly evaluated as name ranges through the values in the vector result of expr 1.
- Other looping facilities include the
  - repeat expr statement and the
  - while (condition) expr statement.
  - The break statement can be used to terminate any loop, possibly abnormally. This is the only way to terminate repeat loops.
  - The next statement can be used to discontinue one particular cycle and skip to the “next”.

Loops

When the same or similar tasks need to be performed multiple times; for all elements of a list; for all columns of an array; etc.

```
for(i in 1:10) {
   print(i*i)
}

i<-1
while(i<=10) {
   print(i*i)
   i<-i+sqrt(i)
}
```

Also: `repeat, break, next`

lapply

- When the same or similar tasks need to be performed multiple times for all elements of a list or for all columns of an array.
  - May be easier and faster than “for” loops
- lapply(li, *function*)
  - To each element of the list li, the function *function* is applied.
  - The result is a list whose elements are the individual *function* results.

```
> li = list("klaus","martin","georg")
> lapply(li, toupper)
> [[1]]
> [1] "KLAUS"
> [[2]]
> [1] "MARTIN"
> [[3]]
> [1] "GEORG"
```

sapply

sapply(li, fct)
Like apply, but tries to simplify the result, by converting it into a vector or array of appropriate size

```
> li = list("klaus","martin","georg")
> sapply(li, toupper)
[1] "KLAUS"  "MARTIN" "GEORG"

> fct = function(x) { return(c(x, x*x, x*x*x)) }
> sapply(1:5, fct)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    4    9   16   25
[3,]    1    8   27   64  125
```

Functions

Functions do things with data

"Input": function arguments (0,1,2,…)

"Output": function result (exactly one)

Example:

```
add <- function(a,b) {
      result <- a+b
      return(result)
}
```

Hello.R: MyFirstFunction()

Hello.R

```
--------------------------------
MyFirstFunction<-function()
{
print("Hello")
}
--------------------------------
>source(Hello.R)
>MyFirstFunction()
```

displayTable.R

```
-------------------------------------
showTable<- function()
{
z <- 1:10
table <- round(z*2)
result<-paste(table, sep="")
print("Table of 2")
return(result)
}
-----------------------------------------
source(displayTable.R)
showTable()
```

Modified displayTable.R

Modify displayTable for user defined x to display “Table of x”

```
showTable<- function(x)
{
If(!is.numeric(x))
Return(NULL)
z <- 1:10
table <- round(z*x)
result<-paste(table, sep="")
print("Table of 2")
return(result)
}
```

Assigning the Function Objects

- st<- showTable
- st
- st(4)

- To edit the function object:
- edit(st)

Default Arguments

- calCube<- function(x, y=3) return(x^y)
- calCuble(3)

Frequently used Built-in functions

`c`	Concatenate
`cbind, rbind`	Concatenate vectors
`min`	Minimum
`max`	Maximum
`length`	# values
`dim`	# rows, cols
`floor`	Max integer in
`which`	TRUE indices
`table`	Counts

`summary`	Generic stats
`Sort, order, rank`	Sort, order, rank a vector
`print`	Show value
`cat`	Print as char
`paste`	`c()` as char
`round`	Round
`apply`	Repeat over rows, cols

Problems

- Write an R script for the following:
  - Generate vector x of 1 to 10.
  - Generate vector enames of 10 names.
  - Generatae a vector salary of 10 numeric values
  - Create a dataframe employee from the above vectors
  - Display total , max , min and average salary
  - Display employee name with max. salary
  - Display name of employee with salary greater than average salary

- Write an R script to update the salary of employees in a file employee.csv by 15%(10000 and above) / 10% (below 10000).