Module 1

Overview and History of R

R is a dialect of the S language.

What is S?

S is a language that was developed by John Chambers and others at Bell Labs. S was initiated in 1976 as an internal statistical analysis environment - originally implemented as Fortran libraries. Early versions of the language did not contain functions for statistical modeling. In 1988 the system was rewritten in C and began to resemble the system we see today. In 1993, Bell Labs gave StatSci an exclusive license to develop and sell the S language. In 2004 Insightful purchased the S language from Lucent and is the current owner. In 2006 Alcatel purchased Lucent technologies. Insightful sells its implementation of S under the name S-PLUS. In 2008, Insightful is acquired by TIBCO for $25 million. The makers of S wanted to create an interactive environment for users to work with data and perform basic analyses. Later the user could use the languahe to develop their own tools.

A short history of R

R was created in 1991 by Ross Ihaka and Robert Gentleman. In 1993, R was announced to the public for the first time. In 1995, Martin Machler convinced Ross and Robert to use the GNU General Public License; thus making it a free software. In 1996, a mailing list was created for developers using R. This led to the formation of the R core group in 1997; this group controls the source code of R

Features of R

- Syntax is very similar to S, making migration easier.
- Semantics are superficially similar to S, but in reality are quite different.
- Can run on any standard computing platform or operating system.
- There is active development with frequent bugfix releases.
- The software itself is quite lean; the functionalities are organized in modular packages
- It has much better graphic and visualization capabilities when compared to most other stat packages.
- While it is useful for working interactively (Markdown and Console), it is also a powerful programming language to develop new tools
- It has a vibrant and active user community.
- It is free!!

What is meant by “free software”?

Free software, libre software, libreware sometimes known as freedom-respecting software is computer software distributed under terms that allow users to run the software for any purpose as well as to study, change, and distribute it and any adapted versions.Free software is a matter of liberty, not price; all users are legally free to do what they want with their copies of free software (including profiting from them) regardless of how much is paid to obtain the program.

- The freedom to run the program, for any purpose.
- The freedom to study how the program works; adapt it to your needs. Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbour.
- The freedom to improve the program, and release your improvements to the public so that the whole community benefits.

Drawbacks of R

- Essentially based on a 40 year old technology.
- Minimal built-in support for dynamic or 3D graphics.
- New functionality is based on consumer demand and user contributions.
- Objects must be stored in physical memory; however there are methods to deal with this.
- Not ideal for all situations.

Design of the R system

The R system divided into two conceptual parts: base R and everything else. R functionality is divided into a number of packages.

- The "base" R system contains the **base** package required to run R, and contains the most fundamental actions
- The other packages contained in the base system include utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk, stats4.
- There are also recommended packages like boot, class, cluster, codetools, foreign, KernSmooth, lattice, mgcv, nime, rpart, survival, MASS, spatial, nnet, Matrix.

There are about 4000 packages on CRAN that have been developed by users and programmers around the world. There are also many packages associated with the Bioconductor project. Additionally, a lot of people also make packages available on their personal websites; there is no reliable way to keep track of how many packages exist in this way.

R console input and evaluation

Expressions can be typed into the R prompt for evaluation. The <- operator is used to assign a value to a variable

```
x <- 1
x
```

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

```
msg <-"Hello"
```

Entering the name of a variable in the console will display its value. One can also use the print( ) function.

```
print(x)
```

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

When a complete expression is entered at the prompt, it is evaluated and the result of the expression is returned. It may sometimes be auto-printed (e.g. when entering a variable, we get its value).

```
msg
```

```
## [1] "Hello"
```

[1] indicates that msg has one element and that it is the string Hello.

The colon is used to create a sequence of integers.

```
x <- 6:25
x
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

The result has [1] before printing the elements.

Data types in R

Objects and Attributes

R has five basic categories of objects: Character, numeric, integer, complex, logical. The most basic object in R is a vector. A vector can only contain objects of the same class. The only exception is a list; which can be a collection of objects belonging to different classes.

Numbers in R are generally treated as numeric objects i.e. double precision real numbers. If you need an integer, specify the L suffix. 1 will return a numeric type while 1L will return an integer type. R also lets you use a special number called Inf. It can be used in regular calculations. 1/0 will return Inf while 1/Inf will return 0. NaN stands for undefined values. 0/0 will return NaN.

Objects in R can have attributes associated with them. Attributes are: names or dimnames, dimensions (for arrays and matrices), class, length and any other attribute that is user-defined. Attributes of an object can be accessed using the attributes( ) function.

Vectors and Lists

The c( ) function is used to create vectors of objects. It concatenates the elements together.

```
x <- c(0.5,0.6)          ##numeric
x <- c(TRUE, FALSE)      ##logical
x <- c(T,F)              ##logical
x <- c("a", "b", "c")    ##character
x <- 9:29                ##integer
x <- c(1+0i, 2+4i)       ##complex
```

We can also use the vector( ) function to create a vector of a certain type and length. In the example below we see a numeric vector of size 10. It will be initialized with the default value of its type. For numeric the default is 0.

```
x <- vector("numeric", length = 10)
x
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

When you mix different types of objects in a single vector, the general rule is that R will try to coerce the vector to get each element into the same class. This is called coercion.

```
y <- c(1.7, "a")
y
```

```
## [1] "1.7" "a"
```

```
z <- c(TRUE, 3)
z
```

```
## [1] 1 3
```

```
x <- c("a", FALSE)
x
```

```
## [1] "a"     "FALSE"
```

We can see in the first example that the resultant vector is a vector of strings. This is because R can convert 1.7 into character but not convert "a" into a number. In the second example, the logical value TRUE is converted to the number 1; FALSE would have been converted to zero. In the third example, the logical value FALSE is converted to a string,

```
x <- 0:6
x
```

```
## [1] 0 1 2 3 4 5 6
```

```
class(x)
```

```
## [1] "integer"
```

```
as.logical(x)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

Explicit coercion can be done using the as.* functions in R.

```r
x <- c("A", "B", "C")
as.numeric(x)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA NA NA
```

```r
as.logical(x)
```

```
## [1] NA NA NA
```

If a conversion is attempted that is not possible, we get NaN and corresponding warnings.

Lists are a special type of vector that can contain elements of different classes

```r
x <- list(1, "a", TRUE, 1+2i, FALSE)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+2i
##
## [[5]]
## [1] FALSE
```

Here you can see that the first element is a vector of integers, the second is a vector of characters, the third is a vector of logical TRUE and so on. Elements of a list will have double brackets around them.

```r
x <- list("a", "b", 0.5, 1, FALSE, "d")
x
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] "b"
##
## [[3]]
## [1] 0.5
##
## [[4]]
## [1] 1
##
```

```
## [[5]]
## [1] FALSE
##
## [[6]]
## [1] "d"
```

Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is an integer vector of length 2 (nrow, ncol)

```
m <- matrix(nrow = 2, ncol = 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```
dim(m)
```

```
## [1] 2 3
```

```
attributes(m)
```

```
## $dim
## [1] 2 3
```

Matrices are constructed column-wise, so entries can be thought of starting in the upper-left corner and running down the columns.

```
m <- matrix(1:6, nrow=2, ncol=3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Matrices can also be created by column-binding or row-binding with cbind( ) and rbind( )

```
x <- 1:3
y <- 10:12
cbind(x,y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```
rbind(x,y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

Factors

Factors are used to represent categorical data. Factors can be unordered or ordered. Gender (male, female) would be an unordered factor wheras something like a grade (A, B, C, D) is ordered because it has a rank. One can think of of a factor as an integer vector where each integer has a label. Factors are treated specially by modeling functions like lm( ) and glm( ). Using factors with labels is better than using integers because factors are self-describing: having a variable with values "male" and "female" is better than a variable with values 1 and 2.

A factor can be created using the factor( ) function in R. Its input should usually be a character vector

```
x <- factor(c("yes", "yes", "no", "yes", "no"))
x
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

The function table will give a frequency count of how many elements of each level are present in the factor.

```
table(x)
```

```
## x
##  no yes
##   2   3
```

Unclass reduces the factor to an integer vector. In the example below, yes is coded as 2 and no is coded as 1. A factor at its core is an integer vector with the levels attribute.

```
unclass(x)
```

```
## [1] 2 2 1 2 1
## attr(,"levels")
## [1] "no"  "yes"
```

The order of the levels of a factor can be set by using the levels( ) argument to a factor( ). This can be important in linear modeling because the first level is used as the baseline level. If levels are not mentioned then R processes them alphabetically. So because N comes before Y, no would be the baseline level and yes will be the second level

```
x <- factor(c("yes", "yes", "no", "yes", "no", "yes", "no"),
            levels = c("yes", "no"))
x
```

```
## [1] yes yes no  yes no  yes no
## Levels: yes no
```

In the factor initialized above, yes is the baseline level and no is the second level. It can be verified using the unclass function.

```
unclass(x)
```

```
## [1] 1 1 2 1 2 1 2
## attr(,"levels")
## [1] "yes" "no"
```

Missing values

Data Frames