Lab Session 5

Lists, Arrays, Data Frames

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.
list1 <- list(c(2,5,3),21.3,”hello”,20L)
# Print the list.
print(list1)
list1[[1]][1]
```

When we execute the above code, it produces the following result:

```
[[1]]
[1] 2 5 3
[[2]]
[1] 21.3
```

```
> x <- list(1:3, TRUE, "Hello", list(1:2, 5))
```

Here x has 4 elements: a numeric vector, a logical, a string and another list.

We can select an entry of x with double square brackets:

```
> x[[3]]
[1] "Hello"
```

To get a sub-list, use single brackets:

```
> x[c(1,3)]
[[1]]
[1] 1 2 3
[[2]]
[1] "Hello"
```

Notice the difference between x[[3]] and x[3].

We can also name some or all of the entries in our list, by supplying argument names to list():

```
> x <- list(y=1:3, TRUE, z="Hello")
> x$y
```

[1] 1 2 3

[[2]]

[1] TRUE

x$z

[1]

Notice that the [[1]] has been replaced by $y, which gives us a clue as to how we can recover the entries by their name. We can still use the numeric position if we prefer:

```
> x$y
[1] 1 2 3
> x[[1]]
[1] 1 2 3
```

The function names() can be used to obtain a character vector of all the names of objects in a list.

```
> names(x)
[1] "y" "" "z"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.
a <- array(c('green','yellow'),dim=c(3,3,2))
print(a)
```

When we execute the above code, it produces the following result:

```
, , 1
[,1] [,2] [,3]
[1,] "green" "yellow" "green"
[2,] "yellow" "green" "yellow"
[3,] "green" "yellow" "green"
, , 2
[,1] [,2] [,3]
[1,] "yellow" "green" "yellow"
[2,] "green" "yellow" "green"
[3,]  "yellow" "green" "yellow"
```

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the **factor()** function.The **nlevels** functions gives the count of levels.

```
# Create a vector.
apple_colors <- c('green','green','yellow','red','red','red','green')
# Create a factor object.
factor_apple <- factor(apple_colors)
# Print the factor.
print(factor_apple)
print(nlevels(factor_apple))
```

When we execute the above code, it produces the following result:

```
[1] green green yellow red red red yellow green
Levels: green red yellow
# applying the nlevels function we can know the number of distinct values
[1] 3
```

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
# Create the data frame.
BMI <- data.frame(
                    gender = c("Male", "Male","Female"),
                    height = c(152, 171.5, 165),
                    weight = c(81,93, 78),
                    Age =c(42,38,26)
                  )

print(BMI)
```

```
gender = c("Male", "Male","Female")
height = c(152, 171.5, 165)
weight = c(81,93, 78)
Age =c(42,38,26)
df=data.frame(gender,height, weight,age)
Print(df)
```

When we execute the above code, it produces the following result:

```
  gender height weight Age
1 Male 152.0 81 42
2 Male 171.5 93 38
3 Female 165.0 78 26
```

Accessing data from Dataframe:

Syntax: Df[row,column]

Extract first two rows:

BMI[1:2,]

Extract 3rd and 4th row with 2nd and 4th column

BMI[c(3,5),c(2,4)]

BMI$height

Access subframe from an existing frame

Syntax: Newdf=subset(df,conditions)

Create a new dataframe with age>30 from the existing dataframe BMI

newbmi=subset(bmi,age>30)

Print(newbmi)

Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

BMI$place <- c("Blore","Mlore","Mysore",”Delhi”)

v <- BMI

print(v)

Add column using cbind

v=cbind(BMI, place= c("Blore","Mlore","Mysore"))

Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the rbind() function.

Newdf=rbind(df, new entries)

newbmi=rbind(bmi, data.frame(gender='M', height=156, weight=67,age=23,place='Delhi'))

newbmi=rbind(bmi,c('M',156,67,23,'Delhi'), c('F,156,67,23,'Delhi'))

R comes with many datasets built-in, particularly in the MASS package. A package is a collection (or library) of functions, datasets, and other objects; most packages are not loaded automatically, so you have to do it yourself:

> library(MASS)

You can now access various datasets from this package. Try looking at the dataset called hills.

> head(hills)

To find out what the data represent, use the help function ?hills.

Data Frames

The object hills is something called a data frame. A data frame is a series of records represented by rows (in this case one for each race), each containing values in several fields (in this case dist, climb, time).

You can check that hills is a data frame by inspecting its class.

> class(hills)

[1] "data.frame"

or more reliably by using an is() command

> is(hills, "data.frame")

[1] TRUE

Data frames share many of the characteristics of matrices. We can select rows or columns in the same way:

> **hills[3,]**

dist climb time

Craig Dunain 6 900 33.65

Hills[hills$dist>=12,1:2] #to display first two columns

```
> hills[hills$dist >= 12,]

              dist climb   time
Bens of Jura    16  7500 204.62
Lairig Ghru     28  2100 192.67
Seven Hills     14  2200  98.42
Two Breweries   18  5200 170.25
Moffat Chase    20  5000 159.83
```

However, they also behave like lists indexed by the columns:

```
> hills$time

 [1]  16.08  48.35  33.65  45.60  62.27  73.22 204.62  36.37  29.75  39.75
[11] 192.67  43.05  65.00  44.13  26.93  72.25  98.42  78.65  17.42  32.57
[21]  15.95  27.90  47.63  17.93  18.68  26.22  34.43  28.57  50.50  20.95
[31]  85.58  32.38 170.25  28.10 159.83
```

Manipulating Data using with()

We often want to use functions on the columns of a data frame, and it quickly becomes inconvenient to repeatedly type (for example) hills$ before every such event. For example, the command below will give a scatter plot of the race times against climbs, amongst only those races less than 10 miles long.

```
> plot(hills$climb[hills$dist < 10], hills$time[hills$dist < 10])
```

```
> with(hills, plot(climb[dist < 10], time[dist < 10]))
```

The command data.frame() is used to create a data frame, each argument representing a column.

```
> books <- data.frame(author=c("Ripley", "Cox", "Snijders", "Cox"),
+ year=c(1980, 1979, 1999, 2006),
+ publisher=c("Wiley", "Chapman", "Sage", "CUP"))
> books
   author year publisher
1 Ripley 1980      Wiley
2     Cox 1979   Chapman
3 Snijders 1999      Sage
4     Cox 2006       CUP
```

Exercise (a) Create a data frame representing a database of films. It should contain the fields title, director, year, country, and at least three films.

Create a second data frame of the same format as above, but containing just one new film.

(b) Merge the two data frames using rbind().

(c) Add two more rows to the data frame

(d) Add one more column genre

(d) Try sorting the titles

order() syntax:

#order(dataframe, decreasing=TRUE)

film1=film[order(film$title),] # sort the dataframe based on title in ascending order

film1=film[order(film$title, decresing=TRUE),]   # descending order

film1=film[order(film$title,film$year),] #sort on multiple columns