MS4024 Week 9 Class Notes

Relational and Logical Operators

Relational operators allow for the comparison of values in vectors.

greater than	**>**
less than	**<**
equal to	**==**
less than or equal to	**<=**
greater than or equal to	**>=**
not equal to	**!=**

Note the difference of the equality operator "==" with assignment operator "=".

"**&**" and "**&&**" indicate logical AND and "**|**" and "**||**" indicate logical OR.

The shorter form performs element-wise comparisons in much the same way as arithmetic operators. The longer form is appropriate for programming control-flow and typically preferred in "if" clauses.

We can use relational operators to subset vectors (as well as more complex data objects such as ***data frames***, which we will meet later).

We specify the relational condition in square brackets.

We can construct compound relational conditions too, using logical operators

```
> vec=1:19

> vec[vec<5]

[1] 1 2 3 4

> vec[(vec<6)|(vec>16)]

[1]  1  2  3  4  5 17 18 19
```

Lists

Many data objects returned as output are structured as *lists*.

(Recall the output from the eigen() function.)

An *R* list is an object consisting of an ordered collection of objects known as its *components*.

There is no particular need for the components to be of the same mode or type, and, for example, a list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a function, and so on.

Here is a simple example of how to make a list:

```
> Lst <- list(name="Fred", wife="Mary", no.children=3,
child.ages=c(4,7,9))
```

Components are always *numbered* and may always be referred to as such.

Thus if *Lst* is the name of a list with 4 components, these may be individually referred to as `Lst[[1]], Lst[[2]], Lst[[3]]` and `Lst[[4]]`.

If `Lst[[4]]` is a vector, then `Lst[[4]][1]` is its first entry.

```
> Lst
$name
[1] "Fred"

$wife
[1] "Mary"

$no.children
[1] 3

$child.ages
[1] 4 7 9

> Lst[[1]]
[1] "Fred"
> Lst[[4]][1]
[1] 4
```

The function `length(Lst)` gives the number of (top level) components that the list has.

Components of lists may also be *named*, and in this case the component may be referred to either by giving the component name as a character string in place of the number in double square brackets, or, more conveniently, by giving an expression of the form

```
> name$component_name
```

for the same thing.

This is a very useful convention as it makes it easier to get the right component if you forget the number. So in the simple example given above:

- Lst$name is the same as Lst[[1]] and is the string "Fred",
- Lst$wife is the same as Lst[[2]] and is the string "Mary",
- Lst$child.ages[1] is the same as Lst[[4]][1] and is the number 4.

This dollar sign operator is very useful , particularly when looking at the output of a complex statistical function.

To find out the names assigned to a list use the command *names().*

```
> names(Lst)
[1] "name"        "wife"        "no.children" "child.ages"
> Lst$name
[1] "Fred"
```

Data Frames

Technically, a data frame in R is a very important type of data object; a type of table where the typical use employs the rows as observations (or cases) and the columns as variables.

Inter alia, a data frame differs from a matrix in that it can contain character values. Many data sets are stored as data frames.

Let us consider the following two variables; age and height.

```
> age=18:29
> age
[1] 18 19 20 21 22 23 24 25 26 27 28 29
```

In similar fashion, we entered the average heights in a vector called height.

```
>height=c(76.1,77,78.1,78.2,78.8,79.7,79.9,81.1,81.2,81.8,82.8
,83.5)
> height
[1] 76.1 77.0 78.1 78.2 78.8 79.7 79.9 81.1 81.2 81.8 82.8
83.5
```

We will now use *R*'s **`data.frame()`** command to create our first data frame and store the results in the data frame "village".

```
> village=data.frame(age=age,height=height)
```

How do we access the data in each column? One way is to state the variable containing the data frame, followed by a dollar sign, then the name of the column we wish to access (as with Lists earlier).

For example, if we wanted to access the data in the "age" column, we would do the following:

```
 > village$age
 [1] 18 19 20 21 22 23 24 25 26 27 28 29
```

The additional typing required by the "dollar sign" notation can quickly become tiresome, so R provides the ability to "attach" the variables in the data frame to our workspace.

```
> attach(village)
```

Let's re-examine our workspace. (The **`ls()`** command lists all data objects in the workspace)

```
> ls()
[1] "village"
```

No evidence of the variables in the workspace. However, R has made copies of the variables in the columns of the data frame, and most importantly, we can access them without the "dollar notation."

```
> age
 [1] 18 19 20 21 22 23 24 25 26 27 28 29
> height
 [1] 76.1 77.0 78.1 78.2 78.8 79.7 79.9 81.1 81.2 81.8 82.8
[12] 83.5
```

Previously we have seen `rownames()` and `colnames()` to determine the names from an existing data frame. We can use these commands to create names for a new data frame.

Vectors can be bound together either by row or by column.

```
> X=1:3; Y=4:6
> cbind(X,Y)
     X Y
[1,] 1 4
[2,] 2 5
[3,] 3 6
>
> rbind(X,Y)
  [,1] [,2] [,3]
X    1    2    3
Y    4    5    6
```

We can use this approach to create a data frame :

```
> data.frame(rbind(X,Y))
  X1 X2 X3
X  1  2  3
Y  4  5  6
```

We can then use **`rownames()`** and **`colnames()`** to assign meaningful names to this data frame.