MAS115 R programming, Lab Class 5

Bryony Moody (Original notes by Prof. P. G. Blackwell) [*]

2020-21

This lab session covers a number of data structures that are more sophisticated than those seen previously.

1 Factors

A *factor* is a variable which can take only one of a discrete set of values called *levels*. When you enter them you will either use numbers or character strings but they are encoded differently, using the integers that index the levels. For example:

- In a clinical trial you might have different doses of a drug administered to patients e.g. 1mg, 2 mg, 5mg.
- In a sample of flowers you might have the particular species of flower e.g. Iris setosa, versicolor or virginica.

While factors are stored as numbers in R they are not *numeric* - for example it wouldn't make sense to ask for the mean type of flower.

Factors are extremely useful though, as they can be used to split up the data into different groups i.e. you might want to find just the survival times of all patients who had 5mg of a drug or look at the heights of only those virginica flowers. Often they are used in specifying linear and other models that you will come across later in your studies but they also allow us to create easy boxplots.

1.1 Using and creating factors

The set of levels associated with a factor can be checked using the `levels()` function, and we can count the number of data points taking each level of a factor using the `table()` command e.g.

```
Treatment <- as.factor(rep(c("A","B"), c(3,7))) # Create a factor variable
levels(Treatment)
```

```
[1] "A" "B"
```

```
table(Treatment)
```

```
Treatment
A B
3 7
```

[*]Thanks to Dr. T Heaton & Dr. R Ripley for suggestions

We can also create factor variables by using the `cut()` command. This is useful for splitting people into groups e.g.

```r
Age <- runif(100) * 100
AgeGroup <- cut(Age, c(0,20,40,60,80,100)) # Split into age groups
# What does this variable look like?
table(AgeGroup) # Create a table of age groups
```

```
AgeGroup
  (0,20]  (20,40]  (40,60]  (60,80] (80,100]
      24       25       13       18       20
```

Finally we can create separate boxplots for each level of our factor as below. You should have a look at the `iris` data set of flower sizes, using `?iris`.

```r
str(iris) # Note the species is a factor variable with 3 levels
boxplot(Sepal.Length ~ Species, data = iris, ylab = "Sepal Length (cm)")
```

In the last `boxplot()` command, the first argument has a special syntax; it is known as a *formula*, with two sides separated by a `~`. It is mostly used in R in a modelling context; here, we can very loosely think of "modelling'' length by allowing it to depend on species.

2 Data Frames

2.1 Creating a data frame

Data frames are the most common way to store data from a statistical experiment in R. They are objects rather like matrices but (unlike matrices) each column can have a different mode i.e. some can be integers, some doubles, some factors, ...

To make this easier to relate to, suppose that we have done an experiment where we observe n_j characteristics about n_i individuals. Then we would create a data frame to store that information where each column relates to one of the characteristics measured and in the i^{th} row we would put in the values for person i. Every entry in the data frame has to have a value but we can use the special value `NA` if it is missing (as might be the case in a lot of statistical experiments). We'll start by creating a data frame containing information about salaries of four graduates which will hopefully make it easier to understand:

```r
grads <- data.frame(subject = c("Maths", "English", "Physics", "Maths"),
  age = c(25, 47, 64, 28), uni = c("Sheffield", "Leeds", "Manchester", "Nottingham"),
  salary = c(27000, 45000, 60000, 42000))
grads
```

```
  subject age        uni salary
1   Maths  25  Sheffield  27000
2 English  47      Leeds  45000
3 Physics  64 Manchester  60000
4   Maths  28 Nottingham  42000
```

As we can see we have a series of columns with each column representing one variable. A single row corresponds to all the values for a single individual. Note also that the names of the columns are those that we specified when creating the data frame.

If we want to find out a bit more information about each column (variable) then we can use the `str()` command:

```
str(grads)
```

```
'data.frame':   4 obs. of  4 variables:
 $ subject: Factor w/ 3 levels "English","Maths",..: 2 1 3 2
 $ age    : num  25 47 64 28
 $ uni    : Factor w/ 4 levels "Leeds","Manchester",..: 4 1 2 3
 $ salary : num  27000 45000 60000 42000
```

we see that R thinks that `subject` and `uni` are factor variables while `age` and `salary` are numeric.

Indexing data frames

In order to access the particular columns/rows in our data frame we have several options.

To access one complete row:

```
grads[1,]
```

```
  subject age       uni salary
1   Maths  25 Sheffield  27000
```

```
grads[grads$salary == 27000,]
```

```
  subject age       uni salary
1   Maths  25 Sheffield  27000
```

To access one complete column (can use specified name if we want):

```
grads$age # Just a vector
```

```
[1] 25 47 64 28
```

```
grads[,2] # Just a vector
```

```
[1] 25 47 64 28
```

```
grads[, "uni", drop = FALSE] # Still data frame
```

```
         uni
1  Sheffield
2      Leeds
3 Manchester
4 Nottingham
```

To access several rows/columns:

```
grads[2:3,] # Still a data frame
```

```
  subject age        uni salary
2 English  47      Leeds  45000
3 Physics  64 Manchester  60000
```

```
grads[,c(1,4)] # Still a data frame
```

```
  subject salary
1   Maths  27000
2 English  45000
3 Physics  60000
4   Maths  42000
```

To access rows which satisfy a certain condition:

```
grads[grads$age > 30, ]
```

```
  subject age        uni salary
2 English  47      Leeds  45000
3 Physics  64 Manchester  60000
```

To omit a row/column:

```
grads[-2,]
```

```
  subject age        uni salary
1   Maths  25  Sheffield  27000
3 Physics  64 Manchester  60000
4   Maths  28 Nottingham  42000
```

We can also edit a data frame. Suppose that the last two people got pay rises and we had to increase their salaries. *Note for any such replacement the replacement elements must be of the right dimension.*

```
grads[c(3,4), "salary"] <- c(65000, 44000)
```

We can also add/remove variables to our data frame quite easily. To add a new variable in a column

```
grads$married <- c(TRUE, TRUE, TRUE, FALSE)
```

while to remove a column we use

```
grads$married <- NULL
```

2.2 Reading a data frame from a file

The function `read.table` is designed to read data in a regular format from a file, creating a data frame.

To reproduce the same data frame as above, create a text file (not a script), called `Salary.dat` say, containing the following information.

```
subject age uni salary
Maths 25 Sheffield 27000
English 47 Leeds 45000
Physics 64 Manchester 60000
Maths 28 Nottingham 42000
```

Then use the command

```
grads <- read.table("Salary.dat", header=TRUE)
```

What happens if you omit the `header` argument?

2.3 Tasks

1. Create a data frame `PlantTrial` containing the following information

Species	Treatment	Yield	Height
Green	A	10.1	8.2
Green	B	15.5	12.3
Blue	A	2.3	9.1
Blue	B	7.6	20.3

2. R has a lot of built in datasets which can be seen by typing `data()`. Several of these store the information in the form of a data frame. One of these is `mtcars` which contains data on 32 different cars produced in 1974.

   (a) Look at the help file for `mtcars`. What does `qsec` measure?

   (b) Have a look at the first few lines of the dataframe by typing `head(mtcars)`. What is `qsec` for the `Datsun 710`?

   (c) Using the `str()` command, in what mode is the variable `cyl` stored?

   (d) Extract the rows corresponding to those cars with 4 cylinders.

   (e) Calculate the mean mpg for those cars with automatic transmission.

   (f) Extract the row corresponding to the Ferrari Dino. Note that as well as using the column names to index variables you can also use the row names i.e. `mtcars["Datsun 710", ]` will give you the row corresponding to the Datsun.

3 Lists

3.1 Creating a list

Lists are similar to vectors but where each element can not only be of a different mode but actually a complex object in itself i.e. a vector, matrix, data frame, another list. They are very useful when we are writing functions and want to return several different values but for now they are of limited use. Note, however, that a data frame is a special case of a list.

Lists are mainly created by using the `list()` command. For an example we can put three completely different objects in a list

```
a <- seq(1:10)
b <- matrix(runif(4), nrow = 2)
c <- data.frame(Sex = c("M", "F"), Mark = c(60, 72))
retlist <- list(a, b, c)
```

This creates a list where the first element is the vector `a` the second the matrix `b` and the third the data frame `c`. We can access the elements in our list in three different ways: If we want to extract elements while still keeping them as lists use:

```
retlist[1:2] # Note the output is a list
```

```
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
         [,1]      [,2]
[1,] 0.633782 0.2313184
[2,] 0.324330 0.3791337
```

If we want to extract the elements in the object form that they went in:

```
retlist[[1]] # Note the slightly different output
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

If we gave the elements in our list names:

```
retlist <- list(myvec = a, mymat = b, mydat = c)
retlist$mymat  # We can choose any names, they don't have to be e.g. myvec
```

```
         [,1]      [,2]
[1,] 0.633782 0.2313184
[2,] 0.324330 0.3791337
```

The function `unlist` can be very useful when we have an object that is a list but where it would still make sense if it had a simpler structure. It needs to be used with care; if it's used where it doesn't really make sense, R will attempt to coerce the contents to fit a simpler structure, which may not be what you intended. Always look at the object you have created, perhaps using `str` or `head`.

3.2 Tasks

1. Create a vector `mysamp` containing a vector of 20 random numbers sampled from a continuous uniform distribution, $U[0, 1]$.
2. Create a list with elements named `a` and `b` containing `mysamp` and the data frame `PlantTrial` you created earlier respectively.
3. Extract the first element in the list as a list.
4. Extract the second element in the list as a data frame.

4 Homework - due 12pm (Midday) Thursday week 6

Your solutions must be in the form of a print-out of a PDF document produced from an R markdown file, including a suitable title and your name in the header and code, results and explanation for each task. Please submit your PDF via blackboard by **12pm (Midday) Thursday 18th March 2021**.

Homework Tasks

1. (a) Look at the ToothGrowth data frame help file using `?ToothGrowth`.
   (b) Using `str()` have a look at the various column modes. In what mode is the dose variable stored?
   (c) Create a table to show the number of guinea pigs who had each supplement.
   (d) Create boxplots to show the lengths of teeth split by supplement type.
   (e) What is the mean tooth length of those individuals who got their Vitamin C from Orange Juice?
   (f) What is the maximum tooth length of those individuals who had a dose of 2mg?
2. (a) The length of time X before a lightbulb fails can be modelled by an exponential distribution with mean $E[X] = 50$. Using `rexp()`, create a vector `FailTime` of 100 random such times.
   Note that `rexp()` *asks you to specify the rate* $\lambda = 1/E[X]$.
   (b) Using `cut()` split `FailTime` into four factors groups according to whether the failure time falls into $(0, 10], (10, 30], (30, 70], (70, \infty)$. Create a table counting the number of occurrences in each of these classes
   Note that when specifying the last class you will need to use the `Inf` *value in R.*