The University Of Sheffield.

Introduction to Data Handling in R (BMS 109) Week4

Today you will learn

- Matrix operations
- Data frame – organisation of data in tables
- Extracting data from data frames
- Preparation to plotting
    - Import data from datasets
- Setting your environment
    - Import/export data from/to files

Recap Matrix and tables

We can organise our data in a table with rows and columns, that is called matrix.

We use the command `matrix()`, to create a matrix or rearrange a set of data. `matrix()` needs the data, nrow, ncol. For example:

```
M1<- matrix( 1:20, nrow = 4, ncol = 5)
 M1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

We can index the elements of M in the same way we used for vectors. The only difference: now we need two indices in the square brackets [,]

The first index corresponds to the rows, the second to the columns.

Recap Matrix and tables

- The 2x2 matrix forming the first two row and the first two column

```
> M3<-M1[1:2,1:2]
> dim(M3)
[1] 2 2
```

- The first two rows with all columns

```
> M4<- M1[1:2,]
> dim(M4)
[1] 2 5
```

- Second column missing

```
> M5<- M1[-2,]
> dim(M5)
[1] 3 5
```

- The first two row and the 4 with third column missing

```
> M6<-M1[c(1,2,4),-3]
> dim(M6)
[1] 3 4
```

Matrix operations

Sum of the matrices

To sum/ subtract two matices they need to have same dimensions and you can use the **+** operator

Exercise : Create another matrix of sequential numbers with dimensionsions 4x5

Sum the matrices M1 and M2

```
M2<- matrix( 21:40, nrow = 4, ncol = 5)
M1+M2
     [,1] [,2] [,3] [,4] [,5]
[1,]   22   30   38   46   54
[2,]   24   32   40   48   56
[3,]   26   34   42   50   58
[4,]   28   36   44   52   60
```

Subtraction of matrices

```
M1-M2
     [,1] [,2] [,3] [,4] [,5]
[1,]  -20  -20  -20  -20  -20
[2,]  -20  -20  -20  -20  -20
[3,]  -20  -20  -20  -20  -20
[4,]  -20  -20  -20  -20  -20
```

Matrix operations (cont…)

Multiplication of the matrices.

In order to multiply S1 and S2 I need to make sure the dimensions are compatible, therefore I need to transpose (change the row with the column) of one of the two. M1=[4x5] and M2=[4x5], the number of col of M1 needs to be the same as the number of row of M2. the operator we use is **%*%**

To do this we use the function **transpose t()**

```
M2t<-t(M2)
> dim(M2t)
[1] 5 4
```

```
M1%*%M2t
     [,1] [,2] [,3] [,4]
[1,] 1465 1510 1555 1600
[2,] 1610 1660 1710 1760
[3,] 1755 1810 1865 1920
[4,] 1900 1960 2020 2080
```

Data Frames

A data frame is a structure that help to hold your data. It a place where you organise and store your data for analysis and visualisation.

Data frames are a collection of vectors. These vectors can be of different types (e.g. numeric, character, logical), but they must all be of the **same length**.

In R you can build data frames, as you do from matrices and vectors, using the data.frame function:

```
> mydf <- data.frame(Var1=1:4, Var2=LETTERS[1:4], Var3=c(1,2,4,8))
> mydf
  Var1 Var2 Var3
1    1    A    1
2    2    B    2
3    3    C    4
4    4    D    8
```

Data frames are central to data analysis in R. They store in each vector (or field) the data you want to analyse. You normally store data corresponding to a statistical variable in each field

Data Frames (cont ...)

We can create data frames from vectors and "force" matrices into data frames.
For example if you want to store the results of a game of cards with four players in a data frame you can use the following:

```
# assigning the vectors
names<- c('Bob','Claire','Luisa','Matt','Marta','Mike')
score<- c(34,82,59,72,50,100)

# forcing vectors into a data frame
 game_cards<- data.frame(names,score,stringsAsFactors=FALSE)
```

When creating data frames R turns names into **factors,** which are statistical variables. R does this by default when creating data frames from string vectors. We use stringsAsFactors=FALSE to avoid it.

Selecting vectors from data frames

```
> game_cards
   names score
1    Bob    34
2 Claire    82
3  Luisa    59
4   Matt    72
5  Marta    50
6   Mike   100
```

We can extract a vector from a data frame in a few different ways:

- Using the name and the $ operator
- Using the name of the vector inside the [[ ]] operator, using quotes
- Using the position inside the [[ ]] operator

Exercise:

Extract the vectors “names” with all three methods.

```
game_cards$names
[1] "Bob"    "Claire" "Luisa"  "Matt"   "Marta"  "Mike"
> game_cards[[1]]
[1] "Bob"    "Claire" "Luisa"  "Matt"   "Marta"  "Mike"
> game_cards[["names"]]
[1] "Bob"    "Claire" "Luisa"  "Matt"   "Marta"  "Mike"
```

We will only use $ to extract data from data frame in this sessions

Data frames are table-like objects.
They have rows and columns. We can extract elements of the data frame using the square brackets [ ]:

```
> game_cards[2,2]
[1] 82
> game_cards[2,1]
[1] "Claire"
```

data sets stored in R

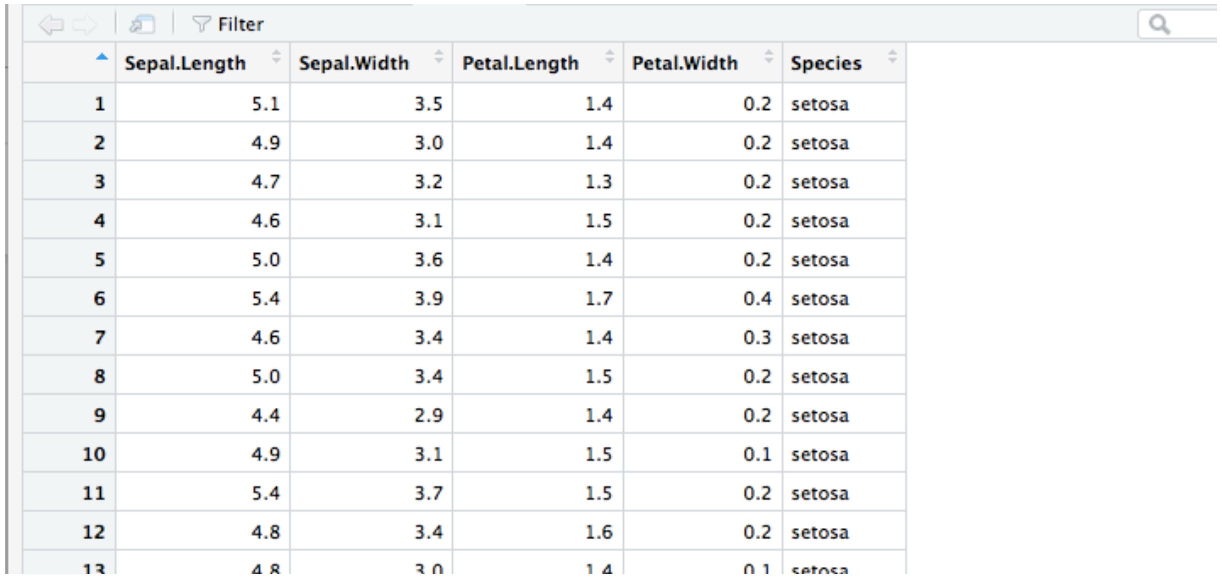
The are many different data sets stored in R. To see which data sets are stored in R, you can use the command data(). This is an "enviromental" function.

Once we have selected a data set we want to use,
Let's choose `iris.` How many rows and how many col it has?

```
> dim(iris)
[1] 150   5
```

we can view its content using the function View(name_dataset)

```
> View(iris)
```



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa

Exploring data frames

We've seen that View(iris) can be used to view a data frame in a spreadsheet like view. There are many other functions that allow us to explore the structure of a data frame. They are particularly useful with large data sets.

Exercise

Exploring a data frame

Using the iris data, experiment with the `head`, `tail`, and `str` functions to see what they do.

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Exploring data frames (cont…)

```
str(iris)
'data.frame':  150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
1 1 1 1 1 1 1 ...
```

The factors are the variables in the iris data set and each variable has 4 measurements.

We can start to visualise the data in the iris datasets and use first basic commands for it. Often useful is to plot the data comparing measures.

For this we use the basic command plot

```
par(mfrow=c(1,2)) # splits the plotting area in two parts

plot(iris$Sepal.Length,iris$Petal.Length, xlab="Sepal Length",
ylab="Petal Length")

plot(iris$Sepal.Width,iris$Petal.Width, xlab="Sepal Length",
ylab="Petal Length")
```

Exploring data frames (cont…)

Its interesting to mark or color in the points by species.

We can use a trick for that using a command called unclass

```
unclass(iris$Species)
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
 [49] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2
 [97] 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3
[145] 3 3 3 3 3 3
attr(,"levels")
[1] "setosa"     "versicolor" "virginica"
```

We can add change the values associated to each class using c()

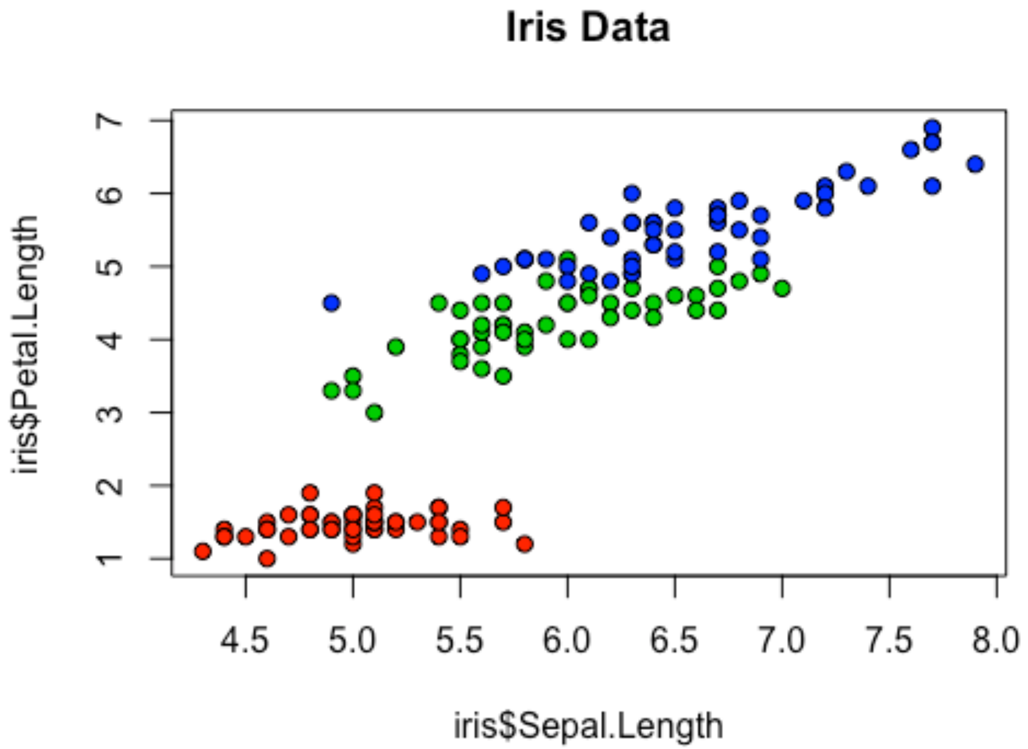
```
c(0,-1,-2)[unclass(iris$Species)]
```

The University Of Sheffield.

Exploring data frames (cont…)

```
par(mfrow=c(1,1)) # make plot area in one block

plot(iris$Sepal.Length, iris$Petal.Length, pch=21,
bg=c("red","green3","blue")[unclass(iris$Species)], main="Iris
Data")
```



```
Legend(4.5, 7, legend=unique(iris$Species),
col=c("red","green3","blue"), pch = 19, cex=0.8,
title=”Species", text.font=4)
```

Working directory

The working directory is a default location where R looks for files you want to use. It is just a folder on your computer.

If you don't set the working directory, R will do it for yo, but needs to be directed to the correct one.

YOU should set the working directory EVERY time you start RStudio. Setting your working directory should be the first thing you do.

Don't do this using R code in your script, because it might be different next time you use it. Instead, use the RStudio menu system (easy and reliable).

Set your working directory now using… Session > Set Working Directory > Choose Directory...

Import data

1. Download the ”Blood_osmosity.xlsx" file from MOLE:

(BMS109 > Data Handling and Visualisation in R)

2. Save this file in you current working directory

Make sure you remember where you put it!

3. Open up your copy of "Blood_osmosity.xlsx" in Excel

Inspect the data:
How many columns ("variables") and rows ("observations") are there?

The University Of Sheffield.

Import data from Excel

STEP 1. Export data from Excel to a CSV file:

Open the ”Blood_osmosity.xlsx" spreadsheet up in Excel

In Excel save the file with *Save As*... and choose the **CSV (Comma Delimited)** option

Keep the file name but change extension (”Blood_osmosity.csv”). Make sure the file is saved in your working directory (you set this earlier)

STEP 2. Now import the CSV file into using RStudio:

Include the following line in your script and run it:

```
blood_data <- read.csv(”Blood_osmosity.csv", stringsAsFactors = FALSE)
blood_data
```

Check Imported data

What happened? Did you see any error messages? If you did, check your spelling and make sure your working directory is set to the right place.

What is the data that you imported? Is is a data frame?
Use str() to check.

```
> str(blood_data)
'data.frame':55 obs. of  3 variables:
 $ Group   : chr  "gamma" "gamma" "gamma" "gamma" ...
 $ sampleA: num  104 118 126 120 104 196 90 95 92 106 ...
 $ sampleB: num  110 126 60 90 80 214 97 108 70 125 ...
```

The University Of Sheffield.

Export data

I r we can export data in a csv format after analysis using a command write.csv()

All data needs to be in data frame before exporting. Also make sure you have clear column names (explore colnames()) and row names (explore rownames())

Let assume that our blood_data is ready to be exported:

```
write.csv(blood_data, file="Blod_data1.csv")
```

Your data will be save in the current working directory

Before you go

Comment well your script.

Save your .R file

Close Rstudio

Log off