Plotting and manipulating data in R

Nicola Romanò

Introduction

Plotting is an extremely important part of data analysis. Indeed, after you finish collecting a dataset one of the first thing you should probably do is plotting the data. This allows you to get a feel for what your data look like and to spot obvious errors in their collection, or with the experimental setup. However, there is so much more that a good graph can do: it can help you support your choice of statistical test used for analysing those data, it can help you better understand the underlying causes of the phenomenon that you are observing, and help you explain those to others. This process is so important that John W. Tukey, one of the fathers of modern statistics coined the phrase *Observational Data Analysis* (ODA) to describe it [1].

[1] Tukey, Observational Data Analysis, 1977

Data quality is of course very important. Remember the phrase "garbage in, garbage out"! If you collect data in a "sloppy" manner you cannot expect to get beautiful and reproducible results. Similarly, if you do not include the appropriate controls you may have some wonderful data... that means nothing! The way you structure your data is also an extremely important issue to consider. Certain formats may be easier to read by a human, but won't be easy for a computer to process (remember what learnt in the lectures). It has been estimated that between 30% and 80% of data analysis consists of data preparation [2]. This is something that you will encounter over and over again in your career as a biomedical researcher, so it is extremely important to get into *good habits* of data collection as early as possible.

[2] Dasu and Johnson 2003

Learning objectives

After completing this workshop you will be able to:

- Read and explore the main features of a dataset
- Identify suspicious data points / outliers and deal with them
- Deal with missing data points
- Plot the data and visually identify relationships between variables
- Convert wide dataset into long datasets

Section 1 - Basic data handling & plotting using R

This first section will be a refresher on how to read, explore and plot data in R. You should be fairly familiar with these commands, but a refresher always helps!

For this first section we are going to use the file called *metab.csv*.

Let's start by loading the dataset using[3]

```
metab <- read.csv("metab.csv")
```

This dataset contains the concentration of a metabolite in the plasma of patients after receiving a specific treatment. We also have information about the patient sex and age. The first thing we should do is examine our data.

head allows us to look at the first few lines of the dataset. Simply use it as follows [4]:

```
head(metab)
```

```
##   Concentration Sex Age Treatment
## 1      550.7270   F  79      CTRL
## 2      260.7356   M  41         A
## 3      450.8036   F  64      CTRL
## 4      324.3586   F  52      CTRL
## 5      228.7021   M  43         B
## 6      325.0527   M  53         A
```

The dataset shows the data for each patient in a row, and has 4 columns showing the measured values or characteristics for that specific patient. We can use *colnames* to see the name of the columns in our dataset.[5]

```
colnames(metab)
```

```
## [1] "Concentration" "Sex"           "Age"           "Treatment"
```

We can now check how many patients we have. Since each line of our dataset represents one patient we can simply count the number of rows using *nrow* [6].

```
nrow(metab)
```

```
## [1] 430
```

Suppose we wanted to know how many of those 430 patients are men and how many are women. There are a few ways of doing this in R. For example you can subset your data and then count the rows.[7]

[3] You will have to change the path to where you put the file, e.g. *read.csv("C:/Workshop1/metab.csv")*. Alternatively you can tell R to look for files in that directory using the *setwd* command. This second approach is very useful if you are reading/writing multiple files in the same script.

[4] Try to use the *n* parameter to see a specific number of lines. If you want to see the end of your dataset you can use *tail* instead of *head*.

[5] Unsurprisingly, there is a *rownames* function as well. Try it!

[6] Can you guess how to count columns instead?

[7] Remember, R uses square brackets to access the content of a dataset (the correct R name for what you are using is a *data frame*). For example writing *data[1,5]* will get the element in row 1, column 5. Using *data[1,]* will get row 1 and all of the columns.

```
# Take all rows for which the _Sex_ = 'F' and all of the columns.
females <- metab[metab$Sex == "F", ]
nrow(females)
```

```
## [1] 215
```

```
nrow(metab) - nrow(females)  # Number of men = total - women
```

```
## [1] 215
```

A more elegant way to do this is to use the *table* function.

```
table(metab$Sex)
```

```
##
##   F   M
## 215 215
```

You can pass multiple parameters to table (separated by ,). Try creating a table of Sex and Treatment[8].

How many men are there in the control group? _____

Another neat way to explore the data is to use the *summary* function:

```
summary(metab)
```

```
##  Concentration   Sex          Age        Treatment
##  Min.   :-10.5   F:215   Min.   :35.00   A   :150
##  1st Qu.:286.9   M:215   1st Qu.:45.00   B   : 80
##  Median :372.6           Median :58.00   CTRL:200
##  Mean   :385.9           Mean   :56.95
##  3rd Qu.:466.7           3rd Qu.:68.00
##  Max.   :862.5           Max.   :80.00
```

This tells us a lot! We have four variables (Concentration, Sex, Age and Treatment). Concentration is a continuous variable, so we get statistics about its distribution. Sex is a discrete variable (in R these are called factors) and has two possible levels: F and M.[9] What type of variables are Age and Treatment? _____

Do you notice any issue with these data? _____

_____

Can you think of a way of correcting the problem? _____

_____

Obviously you can also get summary statistics using R functions such as *mean*, *median*, *range*, *min*, *max*, *quantile*[10]

For example

```
median(metab$Age)
```

```
## [1] 58
```

[8] Note that the order in which you pass parameters to *table* is important! Try passing *metab$Sex* either before or after *metab$Treatment*. What happens?

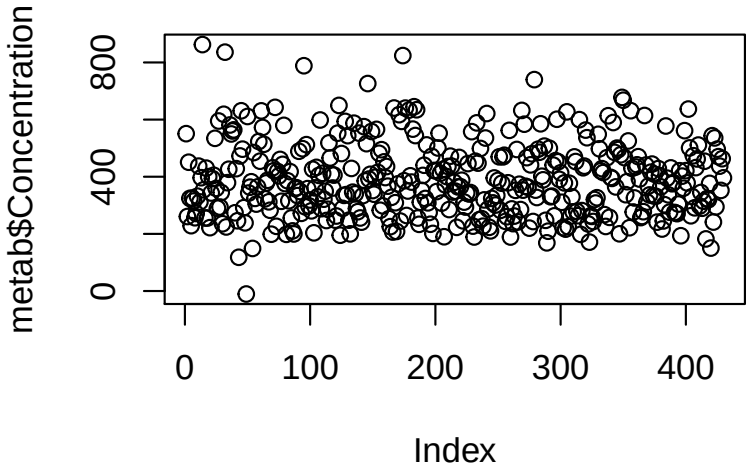
[9] R orders levels alphabetically, we will see how to change this later on.

[10] By default the quantile function returns 5 quantiles (min, 25%, median, 75%, max), but try specifying the *probs* parameter, for example as *quantile(metab$Age, probs = c(0.3, 0.6))*. What happens?

Section 2 - Basic plotting

We now want to start plotting the values we read.

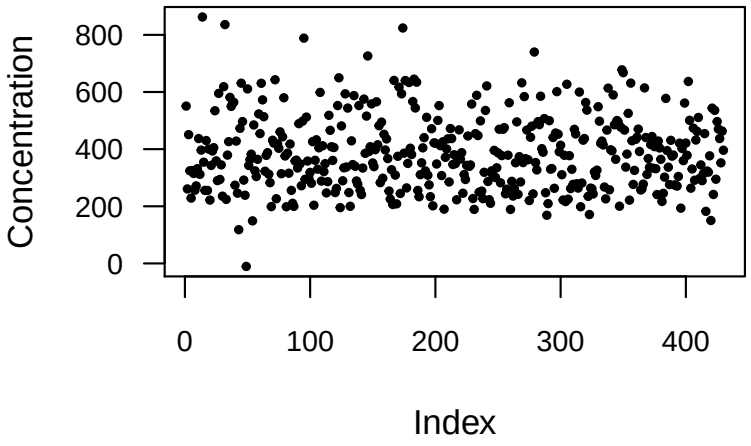
```
plot(metab$Concentration)
```



This plot shows the 430 concentration measurements in our dataset, and easily allows us to estimate their mean[11]. Aside from this, it is not particularly useful, is quite ugly and cluttered. Let's make it prettier!

[11] Could you guess the mean of the values by looking at the plot? Use the *mean* function to calculate their actual mean. Were you close?

```
plot(metab$Concentration, pch = 20, cex = 0.7, cex.axis = 0.8, las = 1, ylab = "Concentration")
```



We have added several parameters:

- *pch* changes the *p*lotting *ch*aracter[12]
- *cex* changes the size of the plotting points. The smaller the value, the smaller the point size, the default value is 1. Depending on the configuration of your computer, different values may give prettier results.
- *cex.axis* changes the size of the axes labels (both x and y).
- *ylab* sets the label for the y axis[13].
- Try changing the *las* parameter and see what it does.

There are many more parameters that you can use, we are going to see a few, but if you want to see the full list type *help(par)*[14].

Now that the plot looks prettier, let's try to make it more useful! We can color the points depending on the value in the Sex column. Let's create a new variable, that contains 430 elements, one for each sample, equal to *"darkgreen"* if the sample is from a man and *"purple"* if it is from a woman.[15]. We assign *"darkgreen"* to everyone, using the *rep* function[16], then change to *"purple"* the values corresponding to samples from women.

[12] There are many different plotting characters, try different numbers for *pch* and see what happens. You can even use a character of your choice, e.g. *pch='x'*.

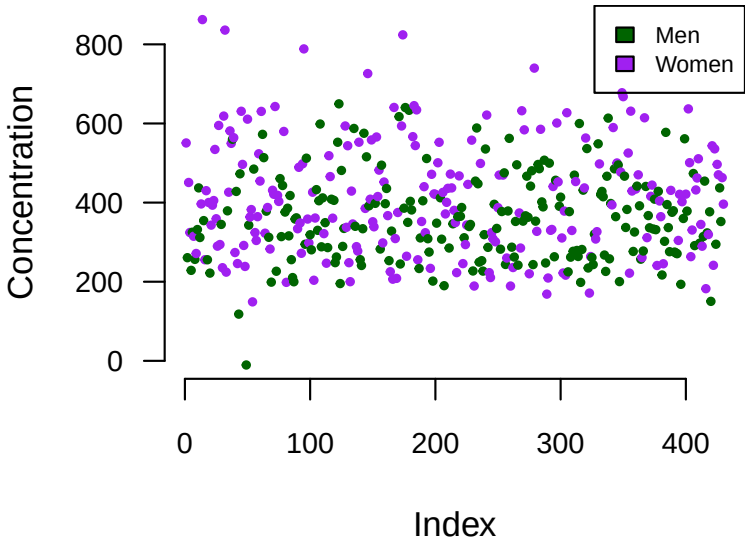
[13] Try changing the label of the x axis

[14] *par* is an extremely powerful function, it allows you to set the default plotting parameters for all of the graphs you will plot after calling it. It also allows you to have multiple plots in the same figure, using the *mfrow* parameter. Try for example writing *par(mfrow = c(1, 3))*, then doing 3 different plots. Can you plot concentration in men on the top and women on the bottom? Pay attention to the y axis scale!

[15] Curious to know which colours are available in R? Type *colours()*!

[16] The *rep* function repeats a value a specific number of times. For example, try, *rep(5, 8)*. Note the quotes around the values, indicating that these are not names of variables but the actual values we want.

```
pointcolor <- rep("darkgreen", nrow(metab))
pointcolor[metab$Sex == "F"] <- "purple"
plot(metab$Concentration, col = pointcolor, bty = "n", pch = 20, cex = 0.7, cex.axis = 0.8,
    las = 1, ylab = "Concentration")
legend("topright", legend = c("Men", "Women"), fill = c("darkgreen", "purple"), cex = 0.7)
```



There are also other ways to generate the color for the points. One is to use the `ifelse` function. We could write:

```
pointcolor <- ifelse(metab$Sex == "F", "purple", "darkgreen")
```

This is equivalent to the first two lines of the code above. It essentially reads "For each element of `metab$Sex`, if it equals F then set `pointcolor` to *purple*, otherwise set it to *darkgreen*". This works because we only have two options, more complex situations with many colors may require a bit more work!

By this time, you should have seen that there is a negative value of concentration! Since this is clearly an error[17], we can safely remove it.

[17] Why is this?

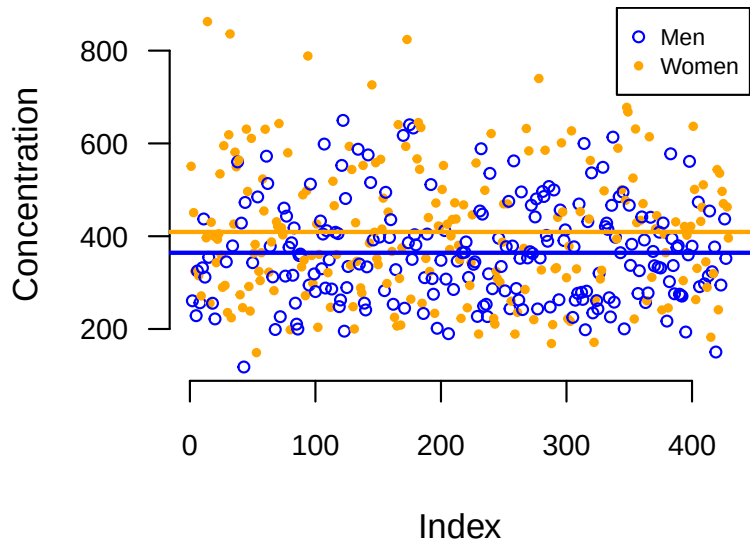
```
# This returns the number of the row(s) containing negative values, in this case
# row 49
wrong.val <- which(metab$Concentration < 0)
# Using - in front of the row (or column) number removes that row (or column). By
# re-assigning the result to metab we are overwriting its previous value
metab <- metab[-wrong.val, ]
```

Finally, we can replot the data, and this time we will also add to horizontal lines corresponding to the mean of the values for men and women. To draw a straight line on an existing plot you can use the `abline` command. For example, these two commands will draw a red line parallel to the x axis at $y = 5$ and a green line parallel to the y axis at $x = 10$

```
abline(h = 5, col = "red")
abline(v = 10, col = "green")
```

Now try to generate the following plot using the commands you have learned so far.[18]

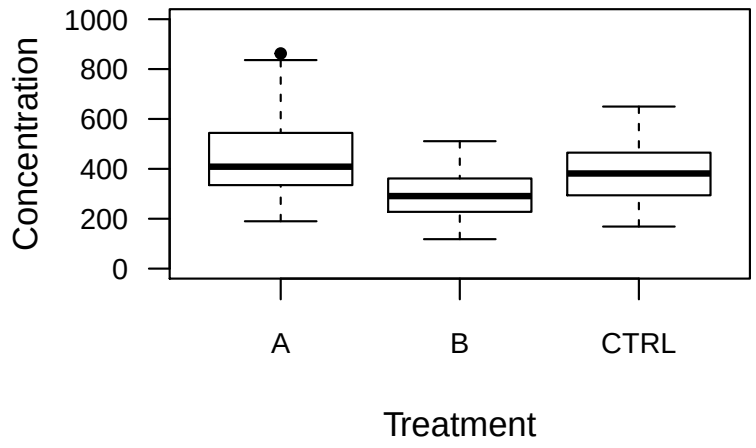
[18] Post your solution in the forum!



Section 3 - Grouping data

Plotting single data points is useful, but most often we want to group the plot by some meaningful factor. Boxplots (a.k.a. box-and-whisker plots) are very good for this.

```
boxplot(Concentration ~ Treatment, metab, las = 1, ylim = c(0, 1000), pch = 20, ylab = "Concentration")
```



The *boxplot* function allows us to plot a value against one or more factor[19]. As we noted above, R orders factors alphabetically, therefore groups appear in the *odd* order A, B, and Control. Should we want to change the order we could do:

[19] Why is there a point on top of the boxplot for group A? We can discuss this in the forum!

```
metab$Treatment <- factor(metab$Treatment, levels = c("CTRL", "A", "B"))
```

After doing this, the boxplot will be correctly ordered [20].

[20] Another point for discussion: what does the line above exactly do?

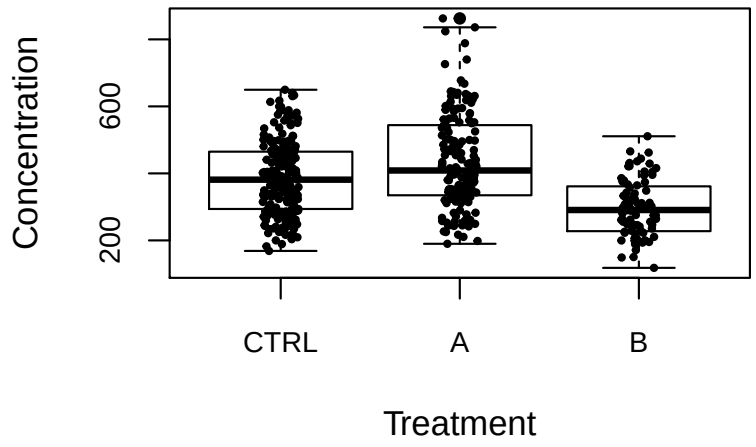
We can also combine more than one factor in the boxplot, using + (e.g. `Concentration ~ Treatment + Sex`)

Try plotting concentration against treatment, separately for men and women.

Now try swapping Sex and Treatment, what happens?

Sometimes it is useful to plot single points over the boxplot. This can be done using the *stripchart* function.

```
boxplot(Concentration ~ Treatment, metab, pch = 20)
stripchart(Concentration ~ Treatment, metab, add = TRUE, vert = TRUE, pch = 20, cex = 0.6,
    method = "jitter", jitter = 0.1)
```



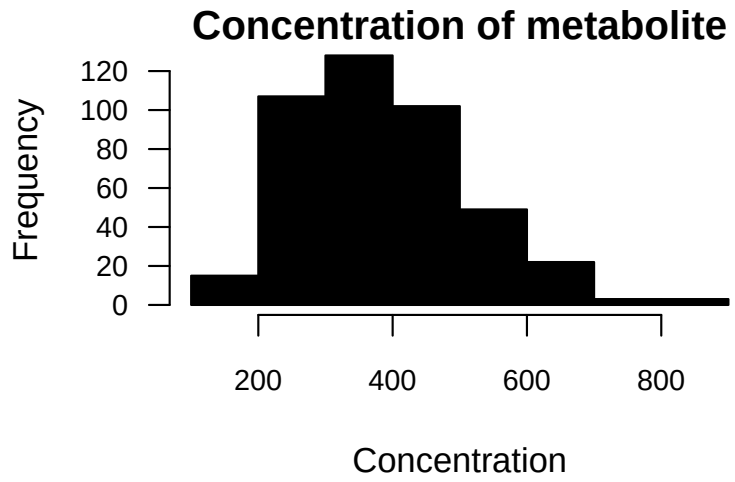
We need to pass two important parameters to *stripchart*: *add* tells R to plot over the existing graph, rather than producing a new one. *vert* specifies to plot the stripchart vertically and not horizontally [21].

[21] The *jitter* parameter moves points around and is very handy when you have a lot of overlapping points. It only works when specifying *method="jitter"*. Try changing the *jitter* value and see what happens!

Section 4 - Histograms

Histograms are a great way to summarise data. Let's generate an histogram of the concentration in our dataset.

```
hist(metab$Concentration, col = "black", las = 1, main = "Concentration of metabolite",
    xlab = "Concentration")
```

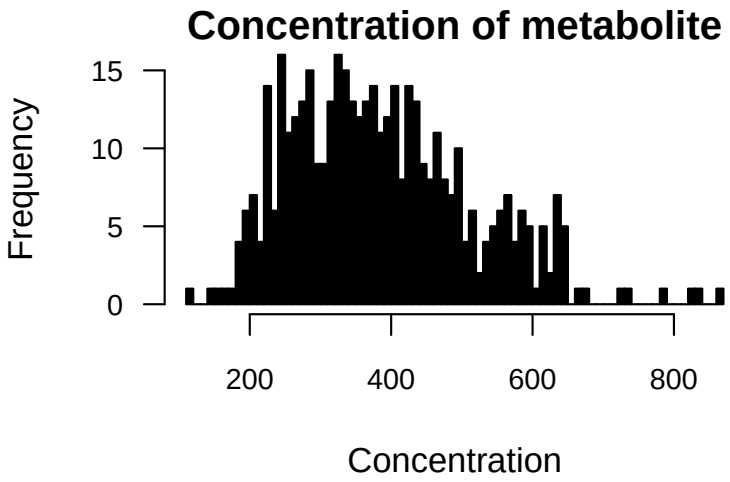


The *br* parameter can be used to specify the number of columns (the 'breaks') in the histogram[22]. It can be used in two ways: by specifying the number of breaks or by specifying the values at which we want to break the histogram.

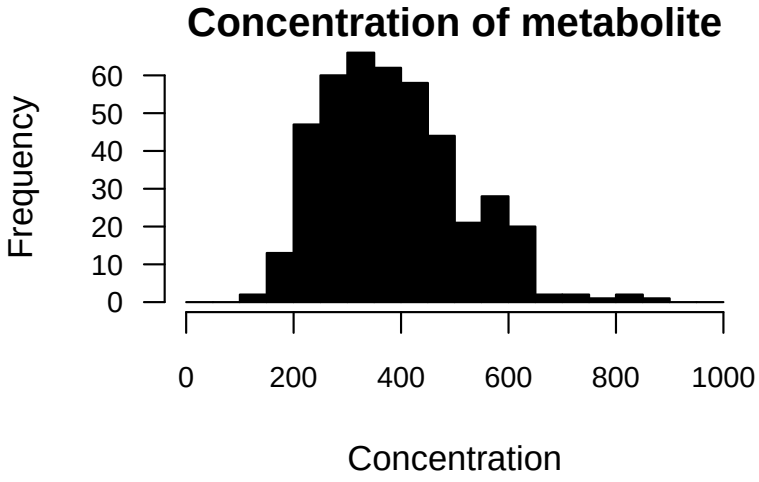
[22] By default R uses Sturge's formula to determine the number of columns. There are many different ways of calculating this number. Wikipedia has a fairly lengthy section about number of bins here: `https://en.wikipedia.org/wiki/Histogram`

For instance:

```
# Histogram with 100 columns
hist(metab$Concentration, col = "black", las = 1, br = 100, main = "Concentration of metabolite",
    xlab = "Concentration")
```




```
# Histogram with columns from 0 to 1000 with width 50
hist(metab$Concentration, col = "black", las = 1, br = seq(0, 1000, 50), main = "Concentration of metaboli
    xlab = "Concentration")
```



Which is the best? Let's discuss this in the forum!

Section 5 - Other plots

There are many other plots that R is able to produce, including some very complex ones. Some plots require extra packages (such as *ggplot2*) to be installed. We will not cover those here, but there are

many online resources that you can use to learn about them should you want to.

What does this code do? Is the resulting plot more or less useful than a boxplot? Why? Try to run it, change things, read the R help, and then we can discuss your solution together!

```
males <- metab$Concentration[metab$Sex == "M"]
females <- metab$Concentration[metab$Sex == "F"]

mean.M <- mean(males)
mean.F <- mean(females)
sd.M <- sd(males)
sd.F <- sd(females)
# barplots returns the x position of each bar
bp <- barplot(c(Men = mean.M, Women = mean.F), col = c("green", "orange"), ylim = c(0,
    600), las = 1, ylab = "Concentration")
arrows(bp, c(mean.M - sd.M, mean.F - sd.F), bp, c(mean.M + sd.M, mean.F + sd.F),
    angle = 90, code = 3)
```

Section 6 - Dealing with missing data

We have talked about missing data in lecture 1.3 and we have seen that the `complete.cases` function can be very useful when dealing with it. Let's practice using it a little bit!

Read and explore the dataset 'neurons.csv'. This contains the number of neurons showing expression of a certain protein in 5 different brain regions. During processing some of the brain sections were lost, before counting could be done.[23]

Start by reading it and exploring it. How many missing values are there? [24]

Now start exploring the dataset; plot the various variables, see if you can spot any trend or relation.

Now, let's try to calculate the mean count per region. You should have all the tools you need to do it by yourself! [25]

[23] What type of missing data is this?

[24] Hint: look at `summary`! Alternatively `which(is.na(neurons$counts))` will tell you which observations (=rows) are NA.

[25] One R function that will make that very easy to do is `aggregate`. See if you can figure out how, and post it in the forum!

Your calculations should result in:

Region	Counts
A	47.8
B	NA
C	9.4
D	26.9
E	NA

Groups B and E have missing data, hence `mean` gives `NA` as a result.

As seen in the lecture, we can use `complete.cases` to remove those data points thus allowing the correct calculations to occur[26].

[26] What are we doing when using `complete.cases`?

Is there a correlation between neuron count and age of the animal? Use the function `cor` as seen in class to calculate it. Try to compare the result of correlation with pairwise omission and with list omission. How does the result change? Why?

```
neurons2 <- neurons[complete.cases(neurons), ]
nrow(neurons)
```

```
## [1] 80
```

```
mean(neurons$counts)
```

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

```
nrow(neurons2)
```

```
## [1] 78
```

```
mean(neurons2$counts)
```

```
## [1] 26.21795
```

Note that we can also use `complete.cases` in conjunction with `which` to know which observations are missing we could also use [27]

[27] This works because `complete.cases` returns `TRUE` if the observation is complete, and `FALSE` if there is missing data.

```
which(complete.cases(neurons) == FALSE)
```

```
## [1] 32 45
```

This tells us that observation 32 and 45 are missing. Indeed we can check that by:

```
neurons[32, ]
```

```
##    mouse sex age region counts
## 32     7   F  94      B     NA
```

```
neurons[45, ]
```

```
##    mouse sex age region counts
## 45     9   M 112      E     NA
```

Section 7 - Converting between wide and long data

The last part of this workshop concerns data formatting. You have learnt about long and wide data formatting during the lectures. Very often you will have to deal with *messy* data, that is not formatted in a way that is easy to work with (e.g. data may be in wide format). This may be because someone else recorded the data, or because often wide data is easier to look at in software such as Excel. Read the file *lizard.csv*. This file contains counts for the number of three different species of lizards found at 5 different locations in a region. If we inspect the data we see that it is in a long format.

```
lizard <- read.csv("lizard.csv")
head(lizard)
```

```
##     Lizard Location1 Location2 Location3 Location4 Location5
## 1 Species1        18        22        15        25        10
## 2 Species2         0         1         6         4         3
## 3 Species3        25        22        30        11        10
```

Each row represent a species and each column a location. This is not a helpful format to use in R, as many functions will be very difficult to run. For instance, if we wanted to plot the number of lizard in each species by location we don't have a straightforward way of doing it[28].

Luckily R comes to the rescue! For reshaping data we need to install an extra R package, called *reshape2* [29]. To install the package we can just run:

```
install.packages("reshape2")
```

You need to do this only once, if you already have the package installed the line above is unnecessary. Once the package is installed we need to tell R to use it; we do so by using the *library* command (this line you will have to call every time).

```
library(reshape2)
```

We can now use all the functions provided by *reshape2*, specifically we want the *melt* function, that converts data from wide to long.

```
lizard.long <- melt(lizard, id.vars = "Lizard", variable.name = "Location", value.name = "Counts")
```

This tells R to use column Lizard as our *ID*, that is the identifier of the subject that is being measured. The *variable.name* parameters specifies the name of the variable(s) that we are changing, in this case the location (this is just the name of the new column in the long

[28] There are ways of plotting wide data, of course, but for instance functions such as *lm* want a long data format.

[29] There are also other packages such as *tidyr* that can perform the same task, feel free to explore those as well if you wish.

dataset). Finally, value.name is the column name for what we are currently measuring.

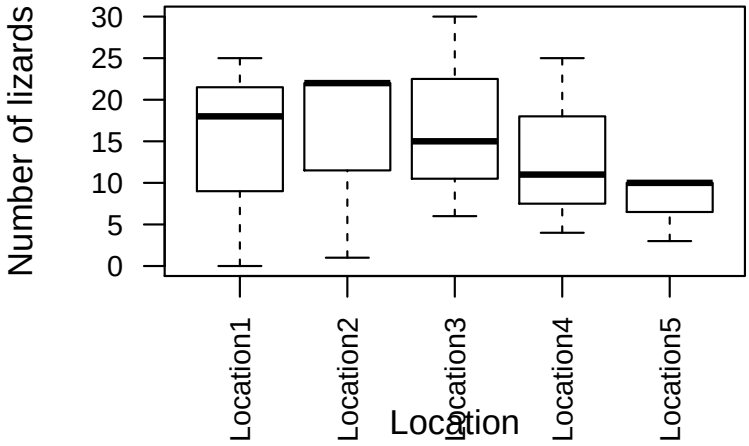
Let's see what we got!

```
head(lizard.long)
```

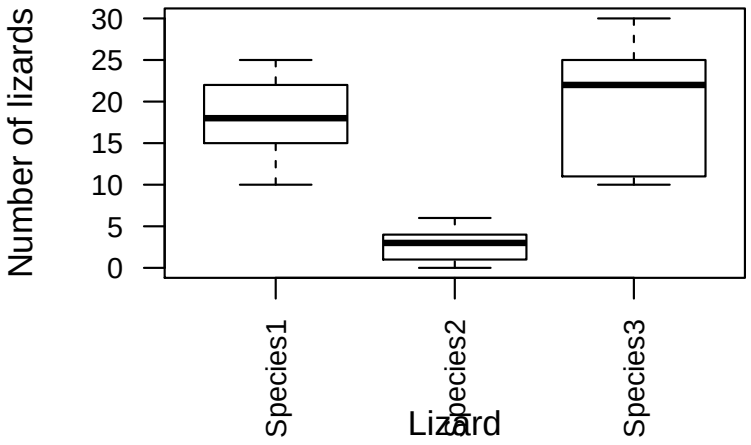
```
##     Lizard  Location Counts
## 1 Species1 Location1     18
## 2 Species2 Location1      0
## 3 Species3 Location1     25
## 4 Species1 Location2     22
## 5 Species2 Location2      1
## 6 Species3 Location2     22
```

We can now easily do:

```
boxplot(Counts ~ Location, lizard.long, las = 2, ylab = "Number of lizards")
```




```
boxplot(Counts ~ Lizard, lizard.long, las = 2, ylab = "Number of lizards")
```



Sometimes conversions from wide to long data are a bit more involved, but *melt* is a very powerful function and most of the time you will be able to reshape your data.

Now try to open the file *BP.csv*. This contains data from an experiment where participants of different sex and age took two different drugs. The data has been anonymised, so that each participant is represented by an ID (2 letters and 3 numbers). Each participant blood pressure was measured before and after taking each of the drugs, and the relative change in blood pressure was calculated. The dataset contains the calculated fold changes for each participants.

- In which format is the data (e.g. wide, long, ...) ?
- If necessary, convert the data into long format
- How many participants were in the study? How many men and how many women?
- How many participants were over 50 years old? How many of those were women?
- Is the response to the drugs different between men and women? What about between younger and older participants?
- Plot the response to the two drugs using an histogram, a boxplot, or a barplot. Which representation is more useful? Why?
- Look at the data for patient OV019. Is there anything striking about this patient? What do you think you should do if you were to continue and analyse these data?