A black and white logo

Description automatically generated

**StratoBayes Workshop**

**18th – 20th March 2024, Durham, UK**

**Worksheet for Day 1 Session 3:**

**Visualising and describing data**

Adapted by Andrew Millard from materials prepared by Dr Sally Street,   
Department of Anthropology, Durham University.

This work is licensed under a   
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

[Creative Commons License](http://creativecommons.org/licenses/by-nc-sa/4.0/)

# Session 3: Visualising and describing data

In this session we will continue using the data from Talat n’ Yissi. You will need to have loaded the isotope data and the elemental data, have replaced missing values with NA, and have created a merged dataset as in the last part of the previous session.

## Sample sizes

One of the most basic ways in which we need to describe our data is to provide a sample size. To simply check how many rows are in a dataset, you can use str to examine the data:

str(Talat\_isotopes)

The top row of the output will tell you the number of rows (obs.) and columns (variables):

'data.frame': 479 obs. of 17 variables:

You could also use length to report the number of observations in a single column of data, e.g.

length(Talat\_isotopes$d18O)

However, neither of the above commands excludes missing values, so they will not necessarily give you the sample size for your analyses. To count the number of non-missing observations for a single variable, you can ask for the sum of observations that are not missing as follows:

sum(!is.na(Talat\_isotopes$X))

Alternatively, if you want to check how many variables have complete rows across the whole dataset, you can use complete.cases together with sum, as follows:

sum(complete.cases(Talat\_isotopes))

sum(complete.cases(Talat\_elements))

Here, complete.cases is checking through all the rows in the data and telling us which ones are complete (TRUE) and which are not (FALSE). Try running just complete.cases alone to see for yourself (although be warned that this can bring up a large output!):

complete.cases(Talat\_elements)

## Continuous data

Most of the Talat n’ Yissi data is continuous numerical data and R has numerous tools for summarising and displaying such data.

### Histograms

The most effective way of figuring out how best to describe data can be simply to plot out their distributions using histograms and visually examine them, which we can do using the command hist, e.g.

hist(Talat\_isotopes$d13C)

You should see a histogram pop up in the bottom right window, like the one below. Next, we will explore how to tweak graphical parameters so that R displays the plot in a more useful way.

A graph of a number of gray rectangular objects

Description automatically generated with medium confidence

The first problem with the above plot is that the data are broken up into pretty coarse 1 ‰ ‘bins’. By default, R will choose break-points that it believes are sensible, but you can alter them using the ‘breaks’ argument, which specifies either how many bins you want to divide the data into, or a list of breakpoints. The bigger the number, the more breaks and the smaller the bins.

Try for example:

hist(Talat\_isotopes$d13C, breaks = 50)

Behind the scenes, R actually uses an algorithm to break up the data so the number you give it may not correspond exactly to the number it shows, and at a certain point it will refuse to give you narrower bins, but you can usually tweak this argument to get the data displayed in a more helpful way.

Sometimes, you might want to plot a limited range of the data, for example, to look at the middle peak of the δ13C distribution. We can do this by adjusting the xlim plotting argument (i.e. ‘x-axis limits’) within hist. xlim needs two numbers combined (using c) in a vector, the first referring to the lower limit and the second to the upper limit, e.g.

hist(Talat\_isotopes$d13C, breaks = 50, xlim = c(-1.5, 1.5))

### Measures of central tendency

You can calculate means and medians using the functions mean and median. For example, you can calculate the mean gold concentration in the elements dataset using:

mean(Talat\_elements$Au\_ppb, na.rm = TRUE)

Setting the ‘na.rm’ argument to TRUE as this instructs R to remove missing values. In this case it has no effect as there is no missing data, but if there was, the command would just return ‘NA’.

This tells us that the average gold concentration is around 9.0 ppb. However, trace element data is often skewed. The mean will be strongly pulled upwards by the small number of high concentrations. It might be more appropriate to calculate the median, which we can do in exactly the same way as the mean:

median(Talat\_elements$Au\_ppb, na.rm = TRUE)

As anticipated, the median is lower (8.4 ppb), which probably better captures the concentration in an ‘average’ sample compared with the mean.

Trace element concentrations often appear to have a **log-normal distribution**. This means that the log-transformed values will be approximately normally distributed. We can investigate this by plotting a histogram showing data on a log-10 scale, as below:

hist(log10(Talat\_elements$Au\_ppb), breaks = 20)

This distribution looks closer to a normal distribution than the untransformed data, and calculating a mean of the log-transformed data, may be a useful measure of central tendency:

mean(log10(Talat\_elements$Au\_ppb), na.rm = TRUE)

and converting back to the ppb scale suggests an even lower average value of 7.0 ppb:

10^mean(log10(Talat\_elements$Au\_ppb), na.rm = TRUE)

Finally, for count or categorical data we may wish to calculate the mode instead of the mean or median. Annoyingly, there is no built-in function to calculate the mode in R (there is a function called ‘mode’, but it does something completely different). We can figure out the mode using a frequency table, with the function table, although unfortunately this is quite fiddly.

First let’s look at using table by itself. Use the command below to produce a frequency table for the number of samples from each formation:

table(Talat\_isotopes$formation)

We will need to manipulate this table, so save it as an object:

formation\_freqs <- table(Talat\_isotopes$formation)

Now we can use which.max to find just the element with the maximum value, and then select that from the table:

formation\_freqs[which.max(formation\_freqs)]

Or even just return the name of the most frequent formation as a character object, without the frequency:

names(formation\_freqs[which.max(formation\_freqs)])

### Measures of variation

When reporting descriptive statistics for continuous variables, we typically need to report not only a measure of central tendency but also a measure of how much the data vary around that ‘average’ value. For normally distributed variables we can use the command sd to report the standard deviation, while for non-normally distributed variables it is probably more appropriate to use the IQR function to report the inter-quartile range. We can also report the range using range. These three commands should all be entered in the same format as mean and median, as follows:

*function(data$variable, na.rm = TRUE)*

Try now adapting the template above to calculate the standard deviation and the IQR for δ13C and δ18O values. We don’t have NA values so the ‘na.rm’ argument is not strictly needed, but it is a good habit to use it routinely to avoid the commands returning nothing but ‘NA’ when there are missing values in the data.

### Scatterplots

Often it can be very illuminating to visualise continuous data using scatterplots prior to any analysis. This can help us spot any unanticipated or concerning patterns in the data, potential data errors, outliers and so on. At this point, we will introduce the extremely useful plot command. plot is a highly flexible function, which can produce many different types of plots for you depending on what objects and arguments you feed into it. If we give it two continuous variables, it will create a scatterplot by default. There are three basic formats that may are useful in different situations:

*plot(X\_variable ,Y\_variable)*

*plot(Y\_variable ~ X\_variable, dataframe)*

*plot(two\_column\_data)*

The first form assumes your data comes as two vector objects. In the second R uses the tilde (~) to distinguish the variable on the Y axis (to the left of the tilde) from the variable on the X axis (to the right of the tilde), and both must be columns in one data frame. In the third we simply supply two columns of data in a matrix or data frame. Let’s first try making a basic scatterplot, for δ18O versus gold concentration. Try the command below, using the second form to place δ18O on the Y axis and gold concentration on the X axis:

plot(d18O ~ Au\_ppb, Talat\_merged)

And you should see the following plot appear in the bottom-right corner:

A graph with numbers and points

Description automatically generated

Exactly the same could be achieved using the alternative forms of the plot command:

plot(Talat\_merged$Au\_ppb, Talat\_merged$d18O)

plot(Talat\_merged[, c("Au\_ppb", "d18O")])

From earlier, we know the gold data seems to be log-normally distributed so it might be better to use a log-scale:

plot(d18O ~ log10(Au\_ppb), Talat\_merged)

Now we can see a possible pattern in the data – there seem to be many points falling roughly in a line and then a group below. If this was a real analysis we might want to follow up by investigating other properties of the two possible groups.

Before going further, we’ll make a couple of graphical tweaks to improve the plot further. First, by default, R will display the names of the variables as specified in the dataset on the X and Y axis labels. We probably want to change these slightly, which we can do by specifying ‘xlab’ and ‘ylab’ arguments within plot. For example, try the command below:

plot(d18O ~ log10(Au\_ppb), Talat\_merged, ylab = "delta-18-O", xlab = "log10 Au concentration in ppb")

We might also want to rotate the Y axis numbers so that they are horizontally oriented. We can do this by adjusting the argument ‘las’ (which stands for ‘label axis style’). las requires a number from 0 to 3, in which 0 = both labels parallel to axes, 1 = Y axis perpendicular, X axis parallel, 2 = both perpendicular and 3 = Y axis parallel, X axis perpendicular. In this case, we want to select 1, as follows:

plot(d18O ~ log10(Au\_ppb), Talat\_merged, ylab = "delta-18-O", xlab = "log10 Au concentration in ppb", las = 1)

Finally, we might wish to change the symbol and colour of the plot points. To change the symbol, we need to add the argument ‘pch’ (‘plotting character’), which takes a number from 0 to 25 corresponding to the following symbols:

A number symbols and symbols

Description automatically generated with medium confidence

To change the colour, we need to include the argument ‘col’. We can specify colours in a number of different ways, including using words (e.g. “red”, “blue”, etc.[[1]](#footnote-2)).

For example, the command below will change the plotting symbols to filled, purple circles:

plot(d18O ~ log10(Au\_ppb), Talat\_merged, ylab = "delta-18-O", xlab = "log10 Au concentration in ppb", las = 1, pch = 19, col = "purple")

Sometimes when we have lots of plotting points overlapping, it can be useful to use transparent colours to get a sense of the density of the data. If we want to do this, we can set the colour using the rgb (‘red green blue’) function within the plotting command. rgb requires four arguments – a value for red, green, blue and ‘alpha’, which sets the transparency, all of which vary from 0 to 1.

To illustrate this, we will use the isotope data. Try this plot of δ18O versus δ13C:

plot(d18O ~ d13C, Talat\_isotopes, ylab = "delta-18-O", xlab = "delta-13-C", las = 1, pch = 19, col = "purple")

Now let’s redo that with pink points with 20% transparency:

plot(d18O ~ d13C,T alat\_isotopes, ylab = "delta-18-O", xlab = "delta-13-C", las = 1, pch = 19, col = rgb(1, 0, 1, 0.2))

Now we can see that there are overlapping symbols in the region around δ13C = −0.5, δ18O = −12.

For the plotting symbols 21–25, it is possible to set different colours for the outline and fill. ‘Col’ will set the colour of the outline, while ‘bg’ (‘background’) sets the colour of the filled area. The command below, for example, will plot grey circles with black outlines:

plot(d18O ~ d13C, Talat\_isotopes, ylab = "delta-18-O", xlab = "delta-13-C", las = 1, pch = 21, col = "black", bg = "grey")

R’s plotting facilities are versatile and complex, and we don’t have time to explore fully them here, but one key element is making sure that they are appropriately labelled.

First, let’s make a plot which distinguishes the isotope data by lithology, with red squares (pch = 15) for dolostone and blue circles (pch = 16) for limestone. We need additional variables to encode these for each point:

Talat\_isotopes$colours <- ifelse(Talat\_isotopes$lithology == "dl", "red", "blue")

Talat\_isotopes$shapes <- ifelse(Talat\_isotopes$lithology == "dl", 15, 16)

Then we can make the plot:

plot(d18O ~ d13C, Talat\_isotopes, ylab = "delta-18-O", xlab = "delta-13-C", las = 1, pch = shapes, col = colours)

Note that because we used the data frame version of plot, R automatically uses the shapes and colours columns of the data frame. If we were using the other formats, we would need to explicitly name Talat\_isotopes$shapes and Talat\_isotopes$colours.

A title above the plot would be helpful, and this can be added with the main parameter:

plot(d18O ~ d13C, Talat\_isotopes, ylab = "delta-18-O", xlab = "delta-13-C", las = 1, pch = shapes, col = colours, main = "Carbon and oxygen isotope data from Talat n’ Yissi, Morocco")

Now we need to add a legend to decode the point styles, which we can do using the function legend. legend will ‘paste’ a legend onto the existing plot, so if you make mistakes you will need to create the plot afresh before attempting to re-do the legend (otherwise it will just paste multiple legends on top of one another!). This function first requires a location, which can either be given as x and y coordinates or simply using words (“bottomright”, “topright”, “bottomleft” or “topleft”) as the first argument, followed by the contents of the legend (‘legend’), the plotting symbol (here 15 or 16) and the colours (red or blue). We have to be really careful here that the colours and labels match up, as R won’t help us out if we make a mistake – it will just blindly do what we tell it to! For a more complex plot it may help to place the required values in an object and use the columns to keep consistent alignment of the plotting parameters.

legend("bottomleft", legend = c("dolostone (dl)", "limestone (ls)"), pch = c(15, 16), col = c("red", "blue") )

Note that for the argument indicating the legend’s location, we have not specified the name of the argument (i.e. x = "bottomleft"). This is because R can either interpret arguments based on names or on the order in which they are received. legend ‘expects’ the first argument to contain the location, so unless we tell it otherwise, this is what it will assume. When learning R, however, it is often safer to specify arguments with names so that we have a better idea of what we are doing! You can use the help (e.g. ?legend) command to remind yourself of the order of arguments for a given function.

The default size of the legend is satisfactory, but we might want it a bit larger. To make it larger, we can introduce another very useful plotting argument – ‘cex’ (character expansion factor). ‘cex’ takes one number indicating the relative size of the plotting characters, where 1 is the default, 2 indicates twice the size, 0.5 half the size and so on. Try different values of cex until you get something that looks more reasonable:

legend("bottomleft", legend = c("dolostone (dl)", "limestone (ls)"), pch = c(15, 16), col = c("red", "blue"), cex = 1.5)

### Overlaying plots

Now we have some experience with plotting, let’s return to the gold concentration data and visually compare it to a normal distribution.

First look at the range of the log-transformed data:

range(log10(Talat\_elements$Au\_ppb))

Given this, and what we saw earlier for the mean and sd, a neat histogram might plot the data between -0.5 and +2.0 in steps of 0.1:

hist(log10(Talat\_elements$Au\_ppb), breaks = seq(-0.5, 2.0, 0.1))

Now we want to overlay a normal distribution. The function dnorm allows us to calculate the normal probability density function. We will do this at a finer scale than the histogram bins to draw a smooth curve, and based on the empirical mean and sd:

x\_values <- seq(-0.5, 2.0, by = 0.01)

norm\_dens <- dnorm(x = x\_values, mean = mean(log10(Talat\_elements$Au\_ppb)), sd = sd(log10(Talat\_elements$Au\_ppb)))

The lines command acts very much like the plot command, except that it adds to the current plot, and defaults to drawing lines rather than points:

lines(x\_values, norm\_dens)

That was not very satisfactory, so we can try again, this time expanding the y-values, and adding some colour with a thicker line:

hist(log10(Talat\_elements$Au\_ppb), breaks = seq(-0.5, 2.0, 0.1))

lines(x\_values, norm\_dens \* 5, col = "gold", lwd = 2)

Now we can see that the gold concentrations are more peaked than might be expected for a log-normal distribution, but we only have 26 data points so this is not a very reliable observation.

### Outliers

We can often spot potentially problematic outliers in the data just by visualising the data in scatterplots. However, if we want to use a more formal approach, we could look for values falling outside of a certain range of the data (e.g. more than 2 standard deviations from the mean, or falling outside of 95% of the values in the data). These values aren’t necessarily going to be data entry errors or otherwise problematic, as with large distributions we would expect to get quite a few extreme values anyway. But you might wish to investigate such values further just in case.

The approach below is slightly long-winded – in practice you might wish to find a more ‘ready-made’ function to do this instead (try for example the package ‘outliers’) – but this way is probably more helpful for honing our R skills.

Let’s say first we want to identify samples with δ18O more than 2 standard deviations above the mean. First, we would need to calculate the standard deviation multiplied by two, as follows:

sd(Talat\_isotopes$d18O, na.rm = T) \* 2

Then we’d need to add this to the mean to find our cut-off value:

mean(Talat\_isotopes$d18O, na.rm = T) + sd(Talat\_isotopes$d18O, na.rm = T) \* 2

Now we know that we want to find out which values are above -5.64‰. We could do this using the subset function, which we used earlier:

subset(Talat\_isotopes, d18O > mean(Talat\_isotopes$d18O, na.rm = T) + sd(Talat\_isotopes$d18O, na.rm = T) \* 2)

The above command is asking R to select only those rows for which the δ18O is greater than the mean plus two standard deviations, and it should return five samples in the Lie de Vin (Twd) formation and seven in the Igoudine (K1a) formation. The plot of δ18O versus δ13C shows these are distinct from the main body of data and so might warrant further investigation before using these data for further analysis.

If you wanted to identify values either more than *or* less than 2 SDs from the mean, you could modify the command as follows:

subset(Talat\_isotopes, d18O < mean(Talat\_isotopes$d18O, na.rm = T) -sd(Talat\_isotopes$d18O, na.rm = T) \* 2 | d18O > mean(Talat\_isotopes$d18O, na.rm = T) + sd(Talat\_isotopes$d18O, na.rm = T) \* 2)

This command is now getting a bit unwieldy and hard to follow! Here we’ve added an additional condition, specifying that we want to subset values either less than 2 SDs below the mean *or* (using ‘I’) more than 2 SDs above the mean. It is clearer to calculate the limits separately:

d18O\_lims <- mean(Talat\_isotopes$d18O, na.rm = T) + c(-2, 2) \* sd(Talat\_isotopes$d18O, na.rm = T)

subset(Talat\_isotopes, d18O < d18O\_lims[1] | d18O > d18O\_lims[2])

Because of the distribution of the δ18O values, this lower limit does not add any samples to our list. In this case perhaps it would make more sense to identify outliers using quantiles, without implicitly assuming that the data are normally distributed. For example, we might want to find out which samples have δ18O values that fall outside of the range of 95% of the data. We can do this using the quantile function. If we want to identify values falling below or above the central 95% of the distribution, we want to identify the 2.5% and 97.5% quantiles, which we could do as follows (not forgetting to set na.rm to TRUE):

quantile(Talat\_isotopes$d18O, c(0.025, 0.975), na.rm = TRUE)

So now we know that 95% of the δ18O values in the sample are between -15.1 ‰ and -5.7 ‰. Putting this together with the subset function, we can identify samples falling outside this range:

d18O\_lims <- quantile(Talat\_isotopes$d18O, c(0.025, 0.975), na.rm = TRUE)

subset(Talat\_isotopes, d18O < d18O\_lims[1] | d18O > d18O\_lims[2])

Note how by using the intermediate object d18O\_lims to store the limits we make the code simpler to read and we can reuse the subset command that we had before.

## Categorical data

### Frequencies & percentages

We can use R to calculate frequencies and percentages when describing categorical variables. The most important function to get to know here is table, which we already encountered earlier when calculating the mode. We saw before using:

unique(Talat\_isotopes$formation)

that there are eight different formation names in this dataset. To create a frequency table, we just need to feed the variable to the function table, as follows:

table(Talat\_isotopes$formation)

This variable happens not to have any missing values, but it is good practice to always include an additional argument telling table to also list how many cases are missing, if there are any, as follows:

table(Talat\_isotopes$formation, useNA = "ifany")

If we wanted to express the frequencies as proportions, we need to feed the whole command above to the function prop.table, as follows:

prop.table(table(Talat\_isotopes$formation, useNA = "ifany"))

This emphasises how little of the data comes from the Amouslek (K2a) formation. If we wanted to express these as percentages, we just need to multiply the whole thing by 100:

prop.table(table(Talat\_isotopes$formation, useNA = "ifany"))\*100

Quite often we are interested in cross-tabulated frequencies for more than one variable. For example, perhaps we want to know the frequencies/proportions for lithologies by formation. We could do so by feeding both variables of interest to the table function, as follows:

table(Talat\_isotopes$formation, Talat\_isotopes$lithology, useNA = "ifany")

Now that we have two variables, if we want to calculate proportions we should specify whether we want the proportions across the rows or down the columns. We therefore need to include an extra argument ‘margin’ within prop.table, in which 1 indicates rows and 2 columns. For example, the command below will figure out the proportions across the rows, i.e. the proportions of dolostone and limestone samples, for each formation:

prop.table(table(Talat\_isotopes$formation, Talat\_isotopes$lithology, useNA = "ifany"), margin = 1)

R has functions to produce plots from the outputs of table and prop.table, or other forms of contingency tables We give these as examples but will not explore them further:

Piechart:

pie(table(Talat\_isotopes$formation, useNA = "ifany"))

Barplot:

barplot(t(prop.table(table(Talat\_isotopes$formation, Talat\_isotopes$lithology, useNA = "ifany"), margin = 1)), las = 2)

Note: t() transposes its argument and is needed here because barplot expects proportions or frequencies in the columns. The default output here is not very pretty and needs adjusting to be useful.

## Descriptives for data subsets

In the last section for this session, we’ll cover how to calculate descriptive statistics for specific subsets of data. Some of this section will overlap with what we have already covered on subsetting and indexing. Here we’ll cover three ways to do this: subsetting, aggregating and indexing.

Perhaps the simplest way to calculate descriptives for data subsets is to create a subset, then use whatever functions we need to use on that subset to calculate the descriptives. For example, here is how you would calculate the mean δ18O for the Lie de Vin (Twd) formation using this method:

LdV\_Twd <-subset(Talat\_isotopes, formation == "lie de vin (Twd)")

mean(LdV\_Twd$d18O, na.rm = TRUE)

Although relatively straightforward, this method has the disadvantage of becoming cumbersome if you want to calculate descriptives for lots of subsets. If that’s the case, you might be better off using the function aggregate to perform the same function across multiple data subsets all at once. For example, to calculate mean δ18O for each formation:

aggregate(d18O ~ formation, Talat\_isotopes, FUN = mean, na.action = na.omit)

The first part inside the brackets specifies the formula you want to use to aggregate, where the variable on the right of the tilde is the grouping variable (think of it as standing for ‘by’ in this case). Then we include the name of the dataset, the function we wish to use (‘FUN =’) and finally what we want to do with missing data (in this case ‘na.action = na.omit’ i.e. exclude NAs). This will produce a handy table (which we could also use to create a barplot if we wanted to). We can replace mean with any function we wish to, such as median, sd, range and so on.

Finally, we could calculate descriptives for subsets using indexing. This can be neater because it does not require you to create lots of objects (as does subsetting) and it doesn’t provide with you with a whole lot of potentially irrelevant information (as might aggregating). It is however the least beginner-friendly option. First let’s remind ourselves how to select subsets using indexing. We need the name of the dataset, followed by some square brackets containing the conditions used to select cases. For example, the command below will select just the Lie de Vin (Twd) formation:

Talat\_isotopes[Talat\_isotopes$formation == "lie de vin (Twd)", ]

Note the comma after "lie de vin (Twd)" – it indicates we are selecting rows rather than columns, so is really important yet very easy to misplace! This will produce all the rows that meet the criterion, but we are only interested in one column in particular – the δ18O values. So, we can add $d18O on the end of the command to extract only those values:

Talat\_isotopes[Talat\_isotopes$formation == "lie de vin (Twd)", ]$d18O

Now, we just need to paste all the above into the brackets following our chosen descriptive function, such as ‘mean’ (not forgetting the na.rm argument!). Putting this all together, we could use the following lines to calculate the mean δ18O for the Igoudine (K1a) and Igoudine (K1b) formations separately:

mean(Talat\_isotopes[Talat\_isotopes$formation == "igoudine (K1a)", ]$d18O)

mean(Talat\_isotopes[Talat\_isotopes$formation == "igoudine (K1b)", ]$d18O)

## Session 3: Questions and exercises

1. What is the mean δ13C for the Issafene (K3r) formation?

2. How many samples have both sodium and calcium concentrations above the means?

3. What is the median number of observations per formation?

4. What are the highest and lowest vanadium concentrations?

5. What is the inter-quartile range for cerium concentrations?

6. What is wrong with the command below?

plot(Talat\_isotopes$lithology)

7. What is wrong with the command below?

plot(m, d13C, Talat\_isotopes)

8. Which two formations contain the samples with the top 1% of δ18O values?

9. What percentage of Igoudine (K1b) formation samples are dolostone?

1. See e.g. <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf> for a full list. [↑](#footnote-ref-2)