MA26620: Practical 2

Tables, Barplots, and the Importance of Plotting

1 The Importance of Plotting

In this week's lecture, we discussed how important it is to *plot* data as well as just producing summary statistics like the correlation coefficient. For instance, a low value of R^2 signifies a weak *linear* relationship between variables, but may miss some other type of relationship.

To illustrate this, let's begin this practical by looking at a dataset specifically designed to emphasize this point: "*The Datasaurus Dozen*"[1]. Download `datasaurus.csv` from the module webpages (`https://stats.vellender.com`) and load it into RStudio via *Import Dataset* > *From Text (base)*. In the resulting window, **be sure to select "Yes" for the Heading option**; this tells RStudio to interpret the first line of the file as variable names.

After importing the data, clicking on "datasaurus" in the top-right pane will display the raw data in the top-left pane. You'll see that it contains thirteen (must be a bakers' dozen) pairs of data called `x1, y1; x2, y2`, and so on. Run the command `attach(datasaurus)`. Recall that this command means that we can thereafter refer to variable names as just `x1` for example, rather than the longer, more tedious `datasaurus$x1`.

Let's generate some summary statistics for some of these variables. Perhaps we'll start with the X variables – run the command:

```
mean(x1)
```

and then:

```
mean(x2)
```

How about the means of `x3`, `x4` and so on (you needn't go all the way to 13 - about four is probably enough to get the gist)? They're very similar indeed, aren't they? Now how about the standard deviations? Run `sd(x1)`, `sd(x2)` and so on. Again, very similar.

Next, do similarly for the `y` variables. You'll hopefully conclude that they're rather similar to each other, too, at least in terms of mean and standard deviation.

How about correlation coefficients? We can get those for the first pair of variables by running

```
cor(x1,y1)
```

and of course can change the 1s for 2s and so on to get the correlation coefficients for other pairs. You'll get a negative value for R, very close to zero. This implies a very weak linear relationship indeed. A poor statistician might conclude that all of the sets of data pairs are pretty much the same, and there's practically no relationship between X and Y in any of the pairs.

BUT... that statistician has forgotten to plot and has forgotten that a small value of R only means a weak **linear** relationship! We're not poor statisticians, we're awesome statisticians. So, year two, we shall plot.

We'll meet more sophisticated methods of plotting scatterplots in the next practical, but in the meantime, run the simple

```
plot(x1,y1)
```

That one admittedly looks like someone dropped a load of smarties on the floor completely randomly; there's no obvious relationship between X and Y here. But how about plotting `x2` vs `y2`? `x3` vs `y3`? `x4` vs `y4` is a particular favourite and explains the name of the dataset. Have a look at a fair few of them.

[1] https://cran.r-project.org/web/packages/datasauRus/vignettes/Datasaurus.html

So the moral here is simple: summary statistics, while very useful, are not the be-all and end-all. Datasets with very similar summary statistics may well have very different characteristics, many of which become apparent via a very quick, simple plot. Moreover, a correlation coefficient close to zero only implies a very weak **linear** relationship, *not* that there's no relationship at all.

2 Hospital Mortality Data

2.1 Introduction

In the rest of today's practical we're going to look at some data concerning hospital mortality (isn't statistics cheery?). The data is available to download under the Practicals section of the modules webpages. Load the data into RStudio as a dataframe called `survival` by using the "Import Dataset > From Text File..." button, as we did in the previous practical.

You should see 'survival' now appears in the top-right Environment pane. Clicking on it there will display the data in the top left pane. Alternatively you can type `head(survival)` in the console to see the first few rows.

The data contains details of patients who were admitted for a certain type of operation at one of two hospitals (A or B). Recorded for each patient is their condition upon being admitted (Good or Poor) and whether they survived the procedure (coded as 1) or died (coded as 0).

We can obtain summary statistics for the data with `summary(survival)`. Note that for numerical data (e.g. `Survived`), R gives summary statistics like those we've met in the lectures. What type of variable is `Survived`? Are these summary statistics useful here?

Attach the data i.e. `attach(survival)`. Again, recall that this allows us to refer to a variable by typing only its name, e.g. `Hospital` instead of `survival$Hospital`.

2.2 Tables of counts

We will now make tables that summarise this discrete data. Let us begin by making a very simple table that counts how many patients were treated at each hospital.

```
table(Hospital) # The count agrees with the summary from earlier.
```

Now try adding more variables:

```
table(Hospital, Survived)
```

Let's give this a name so we can save this for future use:

```
HospSurv <- table(Hospital, Survived)
```

Now try

```
table(Hospital,Condition,Survived) # Note that this makes two tables.
```

Question 1: Which of the following is true?

(i) More patients died in Hospital A.

(ii) More patients survived in Hospital A.

R can also make a *flattened table* for more than two variables; essentially this combines the two tables that your last command made into one neater table:

```
ftable(Hospital,Condition,Survived) # Flattened table.
```

Question 2: Which of the following is true?

(i) Hospitals A and B treat similar numbers of patients in good condition.

(ii) Hospitals A and B treat similar numbers of patients in poor condition.

Try changing the order of the variables in the previous command – what happens? Do all possible orders give potentially useful or interesting outputs? Think which order is the most useful for presenting the data. Copy the most useful table and paste it into a Word document. Edit the table to give clear labels. For example either replace 0 and 1 for Survived by No and Yes or use Died and Survived.

2.3 Proportions and percentages

It is often more illuminating to construct a table containing proportions rather than just frequencies. We can make such a table in R by using the `prop.table` command:

```
prop.table(HospSurv, 1) # Use 1 to mean proportions by row
```

Give this table a name: call it `HospSurvProp`. Note that the `prop.table` command takes a table as its first input, in this case HospSurv. If we hadn't previously created and named this table, the command

```
prop.table(table(Hospital, Survived), 1)
```

would have done the same. Here, the second input ("1" after the comma) tells R to take proportions across rows of the table (so each row sums to 1). To express this in percentages, multiply by 100: `100*prop.table(HospSurv,1)`. If we instead use "2" as the second input, proportions are taken down columns. If we don't put a second input (i.e. nothing after the comma), proportions are taken across the whole table.

It's important to understand the above as we'll use lots of proportional tables. To understand the differences between taking proportions across rows, down columns, or across the whole table, consider the following very simple table:

Table 1: Example table

2	3
4	5

Consider the top-left element of the table. If we take proportions across rows, the top left entry in the resulting proportional table will be 0.4, because $2/(2+3) = 2/5$. If we take proportions down columns, the top left entry in the resulting proportional table will be 0.333, because $2/(2+4) = 1/3$. If we take proportions across the entire table, the top left entry in the resulting proportional table will be 0.143, because $2/(2+3+4+5) = 1/7$.

When interpreting proportional tables, it's important to be clear exactly what you're saying. For instance, looking at the table we created with `100*prop.table(HospSurv,1)`, a good comment regarding what it shows would be:

"Of those patients who were treated at Hospital A, 3.0% died and 97.0% survived. Hospital B displayed a slightly lower mortality rate, with 2.5% of patients dying, while 97.5% survived."

What would be a good comment for the table generated by `100*prop.table(HospSurv,2)`? Copy the table into Word, edit it (too many decimal places should **always** be avoided – you almost never need as many as R gives you), and write your comment there.

Question 3 Which of the following is true?

(i) Nearly 80% of Hospital A's patients died.

(ii) Nearly 80% of those who died were patients in Hospital A.

Finally, construct a flattened table with the patients' original condition as the outer variable and make a proportion table:

```
ByCondition <- ftable(Condition,Hospital,Survived)
100*prop.table(ByCondition,1)
```

Question 4 Which of the following is true?

(i) For patients in good condition, Hospital A is better.

(ii) For patients in poor condition, Hospital B is better.

Copy this table into your Word document, edit it and write a comment.

Question 5 Based on these tables, which hospital would you rather have the operation at? Does this agree with your earlier judgement about which hospital has the higher survival rate?

2.4 Barcharts

We will now construct barcharts to display some of the characteristics of the patients. Your aim is to display some of the features you may have noticed in the tables. Begin by making a barchart representing the data from the frequency table we made that we called "HospSurv".

```
barplot(HospSurv)
```

Note that the barplot command takes a table as its input. The values in the column(s) of the table determine the height of each bar or sub-bar. The resulting plot shows two bars: one for patients who didn't survive and one for patients who did. Each bar has two colours representing the two hospitals. Is this a useful plot? Hopefully you'll agree that the disparity in sample sizes for those who survived and those who didn't makes it difficult to interpret.

Let's attempt to make a barchart with a bar for each hospital, coloured according to the proportion of patients who died/survived. We've already made a suitable table with the data we need for this, which we called HospSurvProp. Remind yourself of this table by typing `HospSurvProp` and then try making a barchart from this data using `barplot(HospSurvProp)`.

Despite our table `HospSurvProp` containing all the correct data, R has not plotted what we intended. Note that the numbers in the first column define the heights of the first stacked bar, while the second column defines the heights of the second stacked bar. We instead wanted the proportions in the rows to define the height of the bars. R treats tables similarly to matrices, so we can take the transpose of `HospSurvProp` by typing

```
TransTable<- t(HospSurvProp) #Transpose, swaps rows and columns
TransTable
```

Making the barplot of this transposed table now gives us what we wanted:

```
barplot(TransTable)
```

To make the vertical axis display percentages rather than proportions, use `barplot(100*TransTable)`.

NB: Using the `t(...)` *command to take the transpose of the table would not have been necessary if we'd constructed the table the other way around. The command* `barplot(100*prop.table(table(Survived,Hospital),2))` *would have given us the same output more directly. Try it!*

We can add titles, labels and colours in the same way as we did last time for boxplots. For example:

```
barplot(100*TransTable, main="Operation survival at two hospitals",
                xlab="Hospital", ylab="Percentage (%)", col=heat.colors(2))
```

Tip: It's very easy to make typing errors in this. Watch out for missing brackets, commas and quotation marks. When editing such long commands, it may be a good idea to use the "Source" pane in the top left region of the screen. To do this go into History in the top right pane, find and click on the line `barplot(100*TransTable)` then click To Source. The Source pane now opens and you can type in the extra subcommands. To submit the line click to select it, then click the Run button. The `col=heat.colors(2)` command (note the American spelling – you'll get an error if you spell it as "colours") assigns two hot-looking colours to the survived/died bars; if we needed more colours we can simply increase the number. There are many such colour palettes in R; you might like to try replacing `heat.colors` by `rainbow` or `terrain.colors` or `cm.colors`.

2.5 Legend

The only thing that our barchart is now missing is a legend. We can add one using another command (note that this is a separate command and shouldn't be inside the barplot brackets). By default, R will only let us place the legend in front of the bars and won't let us place it off to the side – this is not ideal! In fact, quite a few of the built-in plot tools have annoying things like this; we'll counteract this next time by looking at how to create more sophisticated plots with another package within R.

Nevertheless, let's try to put the legend somewhere sensible. The trouble is that the margin where we'd probably like to put it isn't big enough at present. We can override this by using the (not very user-friendly!) command

```
par(xpd=T, mar=par()$mar+c(0,0,0,4))# You can type ?par to see the help file.
```

Don't worry if you don't understand that command, just know that it exists! All it does is make a wider margin to the right of our graph where we can put a legend. **It will seem that nothing has happened – this is expected. Repeat your barplot command and the new plot will have the wider margin**. If our later plots have right margins that are too wide, we can run the above command again but with -4 in place of 4.

Now, to make the legend, type

```
legend(locator(1),c("Died","Survived"),fill=heat.colors(2))
```

After you press Enter, it may seem like nothing has happened, but click where you want to put the top left corner of the legend (perhaps to the right of the letter "B") and it should appear there. Export your barchart, paste it into your Word document and save it. If there is too much white space around your plot, you can crop it in Word by selecting the plot, and then clicking "Crop" on the "Picture Tools - Format" menu that appears at the top of the screen. You can include the table of percentages by copying and pasting `100*TransTable`. Use Word's Table menu (Insert - Table) to construct a more professional table and edit the percentages to 1 decimal place. Write a comment on what the barchart and table show and save your Word document.

2.6 Clustered barchart

Thus far, we have made stacked barcharts. Suppose instead that we want to make a clustered barchart. Having already made the stacked barchart, this is very simple: we only need to add to our earlier command

beside=TRUE. That is:

```
barplot(100*TransTable, main="Operation survival at two hospitals",
                        xlab="Hospital", ylab="Percentage (%)", col=heat.colors(2),
                        beside=TRUE, ylim=c(0,100))
```

Unlike with our stacked barchart earlier, there's plenty of empty space in the top left where we can put our legend. Rather than using `locator(1)` to click-and-place our legend, we can instead ask R to place the legend in the top left:

```
legend("topleft",c("Died","Survived"),fill=heat.colors(2))
```

If this still gets in the way (it may depend upon the size of your plot), remake the barplot and use the `locator(1)` method instead.

3 Titanic passengers data

3.1 Missing values and splitting the data by categories

On Blackboard is a large dataset containing lots of information on the passengers who were aboard the doomed ocean liner "Titanic" when it sank in the Atlantic in 1912. Load this csv file into R in the usual way as a dataframe called passengers. Detach the survival dataframe and attach passengers and have a look at the data to see what information is contained.

Note that the dataset is not complete; for instance if you scroll down the data for "age", many are given as NA (not available, i.e. missing data). If you try to compute something based on age, for instance the mean age, R will give you NA as the output (try it). If you wish to compute the mean age of passengers whose age was known (i.e. ignoring the NAs), you can instruct R to ignore the NAs for its computations by appending `na.rm=TRUE`:

```
mean(age,na.rm=TRUE)
```

Often, we will be interested in separating the dataset into subsets and computing something based on those subsets. For instance, we might be interested in the mean age of Titanic passengers in each of 1st, 2nd and 3rd class travel.

We can use the command `subset` to make a dataframe consisting solely of those passengers who were in 1st class by typing `first<- data.frame(subset(passengers, pclass=="1st"))`

Note the decreased number of observations in this new dataframe in the workspace window and inspect the data in the upper left pane to check that "first" does indeed only contain the 1st class passengers. Alternatively, a neat check would be to run

```
table(first$pclass) # First isn't attached so we need to write $ first
```

Having made this new subsetted dataframe, if we want to know the mean age of just the 1st class passengers, we can type `mean(first$age,na.rm=TRUE)`.

More complicated subsetting is possible. As an example, we could make a dataframe of all under-18 passengers by using the following subset command:

```
U18<- data.frame(subset(passengers, age < 18))
```

R can also combine constraints; it uses the notation "&" to mean "and", and the notation "|" (on most keyboards, type this character using shift with \) to mean "or". For instance, we could obtain data

containing only those passengers who travelled in 3rd class, were female and under 21 years old with the command

```
thirdYoungF<- subset(passengers, age<21 & sex=="female" & pclass=="3rd")
```

Type `summary(thirdYoungF)` to check that the data is as you'd expect.

Now, using your hospital mortality example as a template, make a stacked barchart showing the proportion of survivors for males and females in 1st class aboard the Titanic (ask for help if you get stuck!). Add labels and a title (maybe colours too) and copy your graph to a Word document. Comment on what the graph shows.

3.2 A note on attaching, detaching, and 'masking'

Now that we've subsetted the data, we have multiple dataframes that have the same variable names. This can cause R, and us, some confusion if we're not careful. Why? Well, suppose we type `summary(age)`. There's some data called `age` now in `passengers`, `first` and `thirdYoungF`. How does R know which one you mean?

Well, we could of course just ask for `summary(passengers$age)`, `summary(first$age)`, or `summary(thirdYoungF$age)`, which removes any ambiguity. If we want to do a bit less typing, since it soon gets tedious typing the dataframe's name followed by a dollar sign all the time, we could make sure that the dataframe we want is attached, e.g. if we wanted a summary of first class passengers' ages, we could run `attach(first)` followed by `summary(age)`.

If we take this approach of attaching the dataframe we're working with, we should be pretty fastidious about *detaching* it when we're moving on to a different data frame. By this, I mean that once we've finished doing stuff with `first`, we should run `detach(first)`. This might seem a tricky habit to get in to at first, but it's well worth consciously trying to always remember to do it.

So what happens if you don't? Suppose you attach `passengers`, forget to detach it, and then attach `first`. You'll get a warning message saying something like:

```
The following objects are masked from passengers:
age, boat, body, cabin, embarked, fare, home.dest, name, parch, pclass, sex, sibsp,
survived, ticket, X
```

Whenever you get such a message, it's typically because you forgot to detach something first. What does it mean? Well, before you attached `first`, if you'd typed `age`, R would have taken that to mean `passengers$age`. The warning is telling you that now you've attached `first`, R will now understand `age` to mean `first$age`. This is probably what you would've wanted anyway, but R warns you in case you hadn't realised that both dataframes contained variables with the same name.

3.3 The command `tapply`

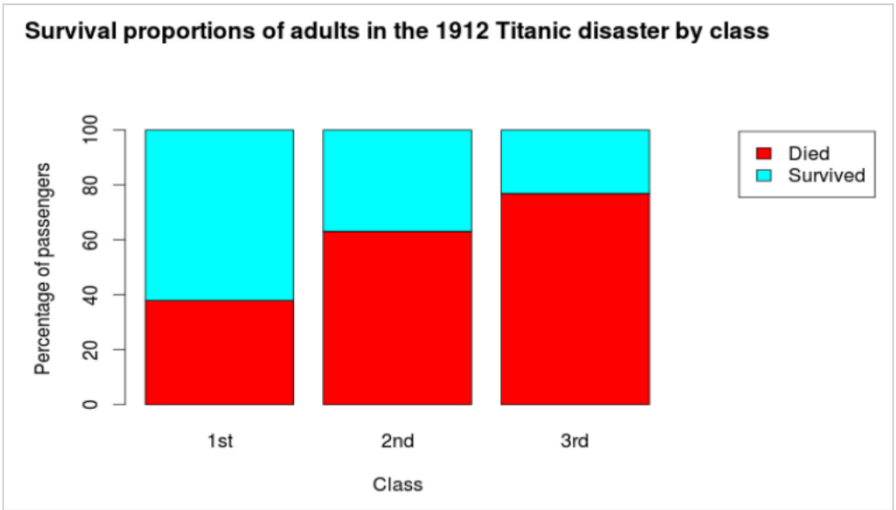
In the earlier example of finding the mean age, we've only considered one class of passengers at a time. Suppose we wanted to repeat the procedure for the other classes... this would be somewhat tedious, especially if there were many classes! The command `tapply` is very useful for such occasions when we want to apply a function (e.g. finding the mean) to some variable (e.g. age), split into groups according to some categorical variable (e.g. class of travel). The syntax for the command in R takes the form: `tapply`([variable we want to apply the function to], [category variable to split by], [function]); thus in our example we would use the command:

```
tapply(age,pclass,mean,na.rm=TRUE)
```

Note that this outputs a table of the average ages of passengers in each class. Use this table to make a suitably labelled barplot of the average ages of Titanic passengers by class.

3.4 Final exercise

Using what you've learned in this practical, make a stacked barchart showing the percentage of survivors aged 21 and above from each class. Your final barchart should look something like this:



Make a further barchart showing survival proportions by class for passengers under 18 years old. Write some comments about what they show and save your Word document and R commands in a suitably named folder on your M drive, as you did in Practical One.