Further Pie Chart Options: Adding Shading and Making 3D Pie Charts

This supplementary material builds on the ordinal data example (sections 2.2.2 – 2.2.2.1) from the pie charts material in chapter 2. In the book chapter, for this example we only used code already seen in the previous example using nominal data, but in this extra material will now look at a couple of other ways R enables you to modify your data presentation. Firstly, we consider cases where, instead of using block colours for the segments, we might want to add shading to our pie charts. This might be particularly useful in instances such as when: i) you want to follow a particular colour scheme but are concerned about colours being difficult to differentiate, ii) you want the figure to be equally readable when printed or published in black-and-white, and iii) your figure design is taking readers with colour-blindness into consideration (see section 1.6 for further discussion of making effective colour choices). However, we feel strongly that shading will rarely increase the ability of your graphic to communicate the data effectively (in fact, it can often reduce the clarity of figures). We here demonstrate how to use it in case you are sure you disagree with us.

To start with, you’ll need to run the code used for section 2.2.2.1 so that the data and label names are loaded and ready to go.

***Step 1*:** To add shading to segments to the pie chart we created for Figure 2.8, we will first add the arguments **density** and **angle** of shadingto both the pie chart and legend code from section 2.2.2.1 to see what we get:

**pie(percent, labels=paste(percent,"%"),clockwise=TRUE, col= c("forestgreen", "green", "red", "red4", "cornsilk2"), main="UK adult attitudes towards rewilding", density = c(6,10,15,50,20), angle = c(0,22.5,45,67.5,90))**

**legend("bottomright",legend=responses, cex=0.7, fill=c("forestgreen", "green", "red", "red4", "cornsilk2"),**

**density = c(6,10,15,50,20), angle = c(0,22.5,45,67.5,90))**

The **density** part refers to the density of shading lines, in lines per inch, while **angle** gives the slope of shading lines, given as an angle in degrees (counter-clockwise). In this code, we have given each segment a unique **density** and **angle** of shading lines by providing lists of values for each, in order to help differentiate the segments.

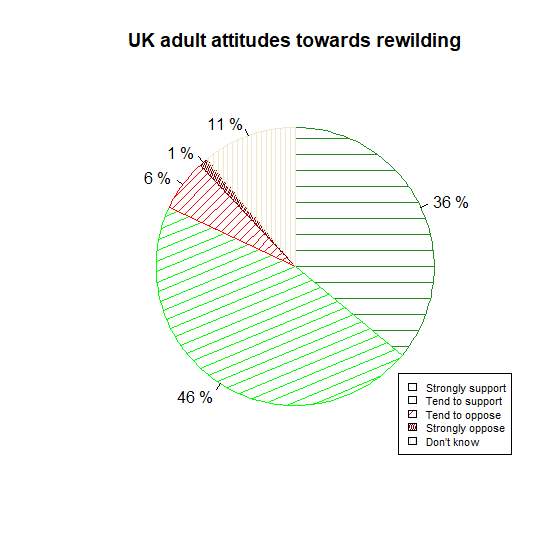


Fig. S.2.1. As Figure 2.8, but with shading added to the segments. The shading lines are, by default, very thin, and the differences in shading are not well conveyed in the legend.

However, we can see in Figure S.2.1 that the shading lines are, by default, very thin, and that the differences in shading patterns are not easy to discern in the legend. If we want to use shading, we may then want to use thicker lines and include the response names as segment labels rather than a legend.

***Step 2*:** Next then, we’ll try re-drawing this chart with thicker lines (see section 7.3.4 for further line customization options). To do this, we first need to tell R that we want its line width (**lwd**) parameter (**par**) for drawing plots to increase to 3 from its default of 1:

**par(lwd=3)**

***Step 3*:** Now, we draw the same plot as above to get a pie chart with thicker shading lines on the segments. The main extra change in the following code is that we are including the responses in the segment labels too, so that R will now **paste** (in order) the corresponding response option, an equals sign, the percentage value, and then a percentage symbol for each segment. See section 6.3.3 for another use of **paste** to present values alongside text, and section 8.6 and Scientific Approach 8.1 for more details on the **paste** function (and its use alongside the **expression** function). We also reduce the font size (**cex**) of these segment labels from the default size of 1 to 0.8 to squeeze them in:

**pie(percent, labels=paste(responses,"=", percent,"%"), cex=0.8, clockwise=TRUE,**

**col= c("forestgreen", "green", "red", "red4", "cornsilk2"),**

**main="UK adult attitudes towards rewilding",**

**density = c(6,10,15,50,20), angle = c(0,22.5,45,67.5,90))**

***Step 4*:** We then need to remember to restore R’s default line width (**lwd**) parameter (**par**) of 1, in case you draw anything else in the current RStudio session:

**par(lwd=1)**

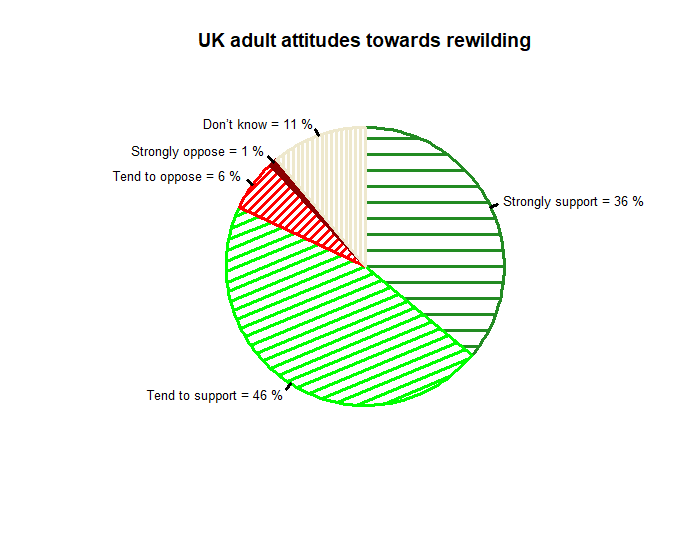


Fig. S.2.2. As Figure S.2.1, but with thicker shading lines and more detailed segment labels in place of a legend.

While Figure S.2.2 is an improvement on Figure S.2.1, we are not sure that shading with lines is ever the best option in terms of interpretation and readability. We have given you the option to do so if you choose, but we would recommend that you instead use carefully-selected highly contrasting solid colours and/or colour-blind friendly palettes when designing figures for the situations outlined earlier (see section 1.6 for more discussion of how to make effective colour choices).

The second optional modification for pie charts we shall briefly outline here is the production of 3D pie charts. Again, we think using 3D will rarely, if ever, make your graphic more effective at communicating the data. Indeed, it usually hampers accurate interpretation. However, you might disagree, and demonstrating its use helps us show-off the flexibility of R in any case.

***Step 1*:** As plotting in three dimensions is quite a ‘fancy’ modification, we will need to download a specialist package to achieve this. The package we need to install is call **plotrix**, and we do so either by going to the 'Packages' tab on the bottom-right window and following the necessary installation steps there, or by running the code below (see the Additional Guide 1.1 R Basics for details on both of these installation routes):

**install.packages("plotrix")**

***Step 2*:** We then need to activate **plotrix**, by running the following code:

**library(plotrix)**

***Step 3*:** Now that **plotrix** is activated in this current RStudio session, we can use its function **pie3D**. This works similarly to the **pie** function we have been using throughout this chapter, but with a few different arguments. We use it here in a slightly modified version of the full code used to produce Figure 2.8 (that is, using solid colours for segments rather than shading):

**percent<-c(36,46,6,1,11)**

**responses<-c("Strongly support", "Tend to support", "Tend to oppose", "Strongly oppose", "Don’t know")**

**pie3D(percent, labels=paste(percent,"%"), col= c("forestgreen", "green", "red", "red4",** **"cornsilk2"), main="UK adult attitudes towards rewilding", sector.order=c(1,5,4,3,2),explode = 0.08, labelcex=0.9)**

**legend("top",legend=responses, cex=0.9,fill=c("forestgreen", "green", "red", "red4", "cornsilk2"), ncol=2, bty="n")**

Looking first at the code associated with the **pie3D** function, the new arguments are **sector.order**, **explode**, and **labelcex**. **sector.order** allows us to specify the order in which the sectors are drawn. To know which order this should be, you can run the code without this argument and see how R arranges the segments without instruction. In the case of the code above, the segments are arranged a bit strangely if you do not include **sector.order**—try this out to see for yourself. Once you see where the segments are laid out, it is easy enough to figure out how to arrange them on top of each other so that the pie fits together nicely. The **sector.order** numbers correspond with the order in which values were added to the values list—in this case we have the segments arranged as: 36%, 11%,1%, 6%, and the 46% segment added last to let it sit on top at the ‘front’ of the 3D pie. **explode** is a measure of how exploded/separate from each other we want the slices of the pie to be, in this case 0.08 looked good to us but you can make it more extreme if you wish. **labelcex** is then simply the font size for labels.

You might have noticed that the argument **clockwise** from the normal **pie** function is missing, as it is not an available argument with **pie3D**. However, although you could spend time playing with a **pie3D** argument (**start**) to specify the angle at which to start drawing segments, we do not need to worry as much about this chart running clockwise or starting from ‘12 o’clock’ because 3D charts make judging differences in area more difficult anyway.

Looking next at the code associated with the **legend** function, we have made a few changes that could be applied to a normal pie chart as well as this 3D pie chart if you wanted. Firstly, the legend is now positioned at the **"top"** of the chart rather than the **"bottomright"** (see section 7.4 for more legend positioning and customization options). We also edit the font size (**cex**) to 0.9. New to this chapter, we have also told R to arrange the legend in 2 columns rather than just one block, using the argument **ncol**. And, additionally, we have drawn the legend without a box by changing the box type (**bty**) to **"n"**.

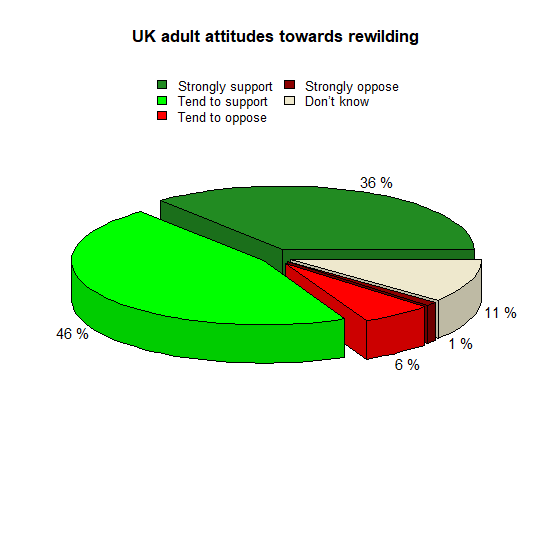


Fig. S.2.3. A 3D pie chart displaying the data from Table 2.2 (YouGov 2019).

While Figure S.2.3 certainly looks stylish, we strongly advise against using 3D pie charts in any scientific context (or indeed 3D figures of any data that can be displayed in 2D form)—see section 1.7.2 for more discussion of this. 3D charts are considered bad practice by many as they distort the apparent values represented by chart elements and make drawing comparisons much more difficult for readers; in particular, the bottom slices appear emphasized. The only instances where we can imagine them proving useful might be in the creation of an eye-catching poster or slide to demonstrate a clearly identifiable trend or pattern in an informal context, such as a presentation aimed more at entertaining than presenting a scientific argument.

**Reference:**

YOUGOV. 2019. *Third of Brits would reintroduce wolves and lynxes to the UK, and a quarter want to bring back bears* [Online]. Available: <https://yougov.co.uk/topics/science/articles-reports/2020/01/28/third-brits-would-reintroduce-wolves-and-lynxes-uk> [Accessed 26/10/2020].