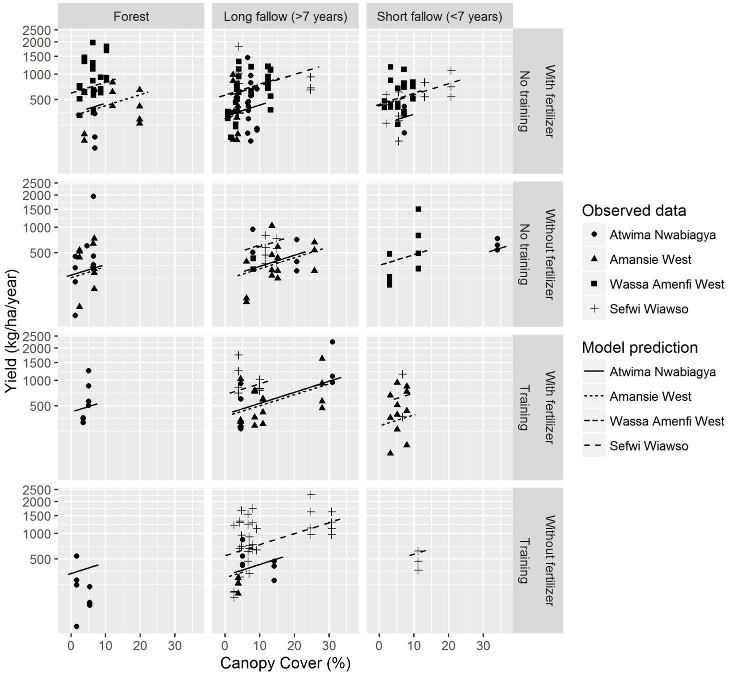
Applied Statistics   Bo Markussen
Statistical Methods for the Biosciences   November, 2022

GRAPHICS WITH THE R-PACKAGE "GGPLOT2"

GGPLOT2 is an R package for making statistical plots. It is created by Hadley Wickham, and it is based on the *Grammar of Graphics* (Wilkinson, 2005). A grammar is part of a language, just like *Danish* and *English*, with syntax and semantics. The purpose of a Grammar-of-Graphics is to describe a statistical plot to the computer. After the computer knows the details of the wanted plot it can make the plot. But it is also possible to store the grammatical description of the plot, which then e.g. may be extended later.

Using GGPLOT2 is not the only way to make graphics in R. The old *Base Graphics* system inside R also produce very nice (but not as nice) plots, and sometimes I prefer to use this. And before GGPLOT2 became available the LATTICE package was very popular (but it is not recommended any more). For one of my publications (Asare et al., 2017) I wanted to make the following plot:



The only way I know of making such a plot in a visually appealing layout is via GGPLOT2. From a statistical point of view the plot contains two things:

1. Observed data (the points).
2. Predictions from a statistical model (the lines).

In general I like to include such a figure in my scientific papers. It not only presents the underlying data to the reader of paper, but it also displays the applicability of the used statistical model as well as the biological variation. The figure above has quite a few features:

- The y-axis is transformed. More precisely using the cubic root. Although the cubic root isn't a standard transformation (like the logarithm), it can still be implemented rather easily (but we will not try this in this exercise). Note also, that the reader of the figure don't need to know the precise transformation in order to perceive the information contained in the graphics.
- The plot has been split into $3 \times 4 = 12$ panels arranged in 3 columns and $2 \times 2 = 4$ rows. This has been done according to the levels of 1 and $1 + 1 = 2$ categorical variables (with 3, 2 and 2 levels, respectively).
- The same x- and y-axes are used in all 12 panels. This makes it possible to compare data across the panels.
- Within each panel 4 different plotting symbols and 4 different line types are used to distinguish observed data and model predictions from 4 different regions. These symbols and line types are summarized in the legend placed to the right of the 12 panels.
- The model predictions (i.e. the lines) are only made in the range of the observed data. And omitted if there are no observed data in the corresponding panel.

Now it is time for you to work

Please work on the following questions in small groups of 2 to 4 persons:

1. Discuss the features of the figure on page 1. Do you agree with description I made above?
2. In order to try GGPLOT2 yourselves open RStudio and install the `ggplot2`-package (if you haven't got the package already).
3. The basic reference on GGPLOT2 is the book (which should be freely available to students at KU via the link embedded in this pdf):

   Hadley Wickham, "ggplot 2", Springer, Use R! series, 2009.

The main data example in Chapter 2 of this book is the `diamonds` dataset, which contains information on the price of about 54,000 diamonds. We will also be using this dataset for this exercise. Execute the following R codes in the *Console* window, and read about the `diamonds` dataset:

```
> library(ggplot2)
> ?diamonds
```

4. The data frame `diamonds` is very big with almost 54,000 observations. Sometimes it can be useful to make a random selection of the observations in order to avoid having a lot of points plotted on top of each other. Execute the following R codes, and discuss what they do:

```
> mypoints <- diamonds[sample(1:nrow(diamonds),1000),]
> mypoints
> head(mypoints)
```

5. Let's make our first plots using GGPLOT2! Try the following R codes one-by-one (!) and discuss the relations between graphical output and the R code. Can you see what is plotted?

```
> ggplot(mypoints,aes(x=carat,y=price)) + geom_point()
> ggplot(mypoints,aes(x=carat,y=price,shape=color)) + geom_point()
> ggplot(mypoints,aes(x=carat,y=price,color=color)) + geom_point()
> ggplot(mypoints,aes(x=carat,y=price,color=color,size=cut)) + geom_point()
```

6. Basically the `ggplot()`-function has two arguments:
   - The first argument is a data frame. This is somewhat similar to an Excel sheet, that is, a spreadsheet like organization of data. In our example this is the data frame that we called `mypoints` in item 4 above.
   - The second argument is a so-called *aesthetic mapping*, which tells R how the variables inside the data frame should be used. This is defined using another R function, namely `aes(x=carat,y=price)`. Here we tell R that the *carat* value should be on the x-axis, and the *price* should be on the y-axis. Ok?

   The `ggplot()` only sets up the data/plotting situation. To plot anything you should tell what to plot. In the example above this is done by *adding* points. If we instead add lines, then we get something less useful (do you agree?) in this example:

```
> ggplot(mypoints,aes(x=carat,y=price,color=color)) + geom_line()
```

7. Above we have written (almost) the same `ggplot()` code 5 times. To avoid doing this even more times, let's define it as a variable. Please try

```
> myplot <- ggplot(mypoints,aes(x=carat,y=price,color=color))
> myplot + geom_point() + aes(shape=cut)
```

   Note, that after executing the first line the variable "`myplot`" appears in the *Environment* window. This variable now contains the results of the `ggplot`-call, and can be used instead of writing this.

8. It's easy to subdivide the plot into several panels. Let's try this according to the categorical variables `cut` and `clarity`. Execute the following R codes one-by-one and discuss the output:

```
> myplot + geom_point() + facet_grid(cut~clarity)
> myplot + geom_point() + facet_grid(clarity~cut)
```

9. To export the most recent figure, such that it can be inserted in a paper or a report, use the `ggsave()` function. The following codes saves the plot to your working directory in PNG and PDF format[1], respectively:

```
> ggsave("diamonds.png")
> ggsave("diamonds.pdf")
```

   Try to insert the generated figure in a WORD document.

10. As already hinted at several places above the output of a call to `ggplot()` is not a graphical output, but a grammatical description of that output. What you see on the screen is a `print()` of that description. One implication of this is that you can add more *"layers"* to the description before it is printed. The symbol for adding components is "+", which in this context shouldn't be confused with the mathematical operation of adding numbers. To change the axes to be logarithmic you add this information. As above; try and think about:

```
> myplot + geom_point() + scale_x_log10() + scale_y_log10()
```

---

[1] I recommend PNG format if you use WORD, and PDF if you use PDFLaTeX.

What happens if you remove either `scale_x_log10()` or `scale_y_log10()`? I guess that you are unhappy with the appearance of the x-axis. There are too few tick points, right? This can be fixed by hand via

```
> myplot + geom_point() +
  scale_x_log10(breaks=seq(0.5,3,0.5)) + scale_y_log10()
```

11. You can also add smoothing lines and other statistical output to the graph:

```
> myplot + geom_point() + geom_smooth()
```

Note, that a smoothing line is generated for each of the diamond colors. The reason for this is that the separation into distinct colors is *inherited* from the `aes()` code inside our variable "`myplot`". To make a single smoothing line for all diamonds we must turn down the *inheritance*, and restate the necessary *aesthetics.* Thus,

```
> myplot + geom_point() +
  geom_smooth(aes(x=carat,y=price),inherit.aes = FALSE)
```

Note that the smoothing method has change from *loess* (= local polynomial regression fitting) to *gam* (= generalized additive model).

- Can you find out why this is the case? Hint: see the help page by executing `?geom_smooth` in the *R console.*
- Can you change back to *loess*-smooting? Hint: use the option `method="loess"`

12. A linear relation between *price* and *carat* appears to be plausible on the log-log scale:

```
> myplot + geom_point() + scale_x_log10() + scale_y_log10() +
  geom_smooth(aes(x=carat,y=price), inherit.aes = FALSE, method="lm")
```

Now try to make a figure that contains both a non-linear smoothing line (either *loess* or *gam*) and the linear regression line in the same plot!

13. We have written the two options "`aes(x=carat,y=price)`" and "`inherit.aes = FALSE`" many time above. Discuss whether it would have been more clever to define "`myplot`" as

```
> myplot <- ggplot(mypoints,aes(x=carat,y=price))
```

How would this change the solution code for the above questions?

Please note that the lines inserted on the figure on page 1 were not generated automatically by GGPLOT2. Instead I used predictions from a linear mixed effects model fitted via the R-package LME4. In order to do that I made another data frame with the model predictions, and used this new data frame together with the `geom_line()` function. I hope to be able to give an example of this technique later in the course.

If you want to read more about GGPLOT2, then you might start at the homepage:

`http://www.r-bloggers.com/basic-introduction-to-ggplot2/`

Or perhaps even better read Chapter 3 in the book:

`http://r4ds.had.co.nz/`

End of exercise