Graphics with the R-package "ggplot2"

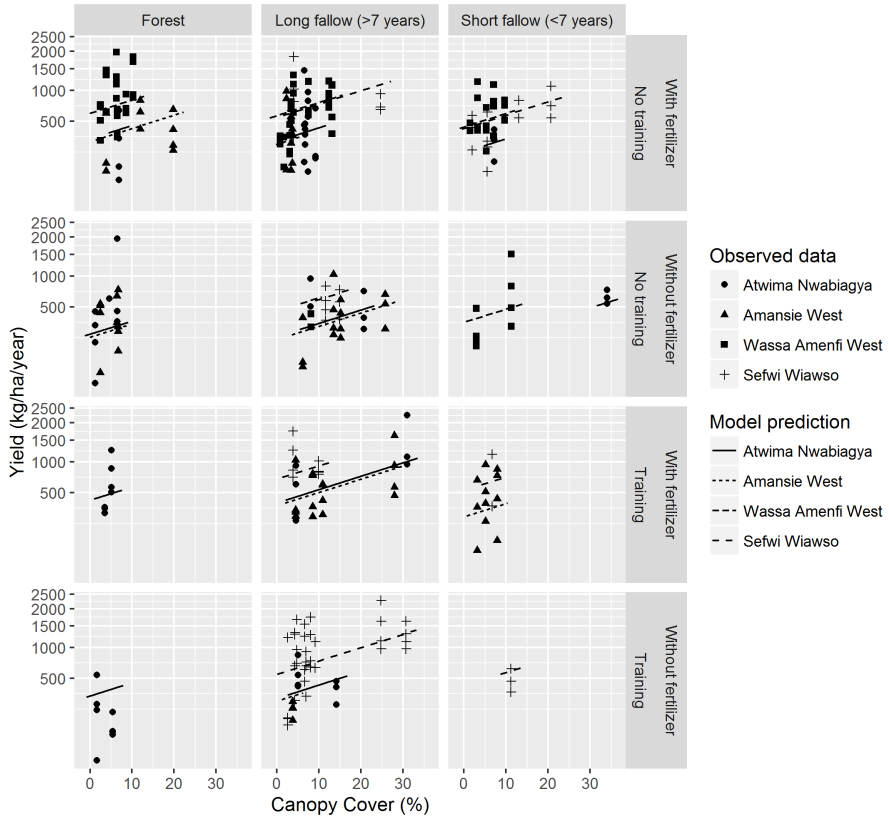
Applied Statistics & Statistical methods for the Biosciences

Anton Rask Lundborg

November 2023

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 it can be faster or simpler to use this. Before ggplot2 became available the lattice package was very popular (but it is not recommended anymore). Bo (the previous lecturer of the course) wanted to make the following plot for one of his publications (Asare et al., 2017):



Probably the only way 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 such a figure is highly informative in 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 is not 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 do not 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.

Exercise

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

1. Discuss the features of the figure on page 1. Do you agree with the description above?
2. In order to try ggplot2 yourselves open RStudio and install the `ggplot2`-package (if you have not installed 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 the 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 lines of R code, and discuss what they do:

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

5. Construct our first plots using ggplot2! Try the following lines of R code 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. The `ggplot()`-function has two primary 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.

   The `ggplot()` function call only sets up the data/plotting situation. To plot anything else, you need to call additional functions. 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. Run the following code:

```
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 is easy to subdivide the plot into several panels. Let us try this according to the categorical variables `cut` and `clarity`. Execute the following lines of R code 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 lines of code 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 should not 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()
```

What happens if you remove either `scale_x_log10()` or `scale_y_log10()`? You are probably 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()
```

[1] it is recommended to use the png format if you use Word, and pdf if you use PDFLaTeX.

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*. To do so, we run

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

Note that the smoothing method has changed 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 times 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, predictions from a linear mixed effects model fitted via the R-package lme4 were used. In order to do tha another data frame with the model predictions was made and used together with the `geom_line()` function. We may see 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/`