Data Science: Visualisations in R - Assignment 4

Healy (2018) Data visualization, Chapter 6

dr. Mariken A.C.G. van der Velden

November 22, 2019

Instructions for the tutorial

Read the *entire* document that describes the assignment for this tutorial. Follow the indicated steps. Each assignment in the tutorial contains several exercises, divided into numbers and letters (1a, 1b, 1c, 2a, 2b, etc.).

Write down your answers with **the same number-letter combination** in a separate document (e.g., Google Doc), and submit this document to Canvas in a PDF format.

Note: this is the first time that this course has ever been taught. Because of rapidly changing techniques, the course tries to keep up/ reflect those changes. As a result, the assignments may contain errors or ambiguities. If in doubt, ask for clarification during the hall lectures or tutorials. All feedback on the assignments is greatly appreciated!

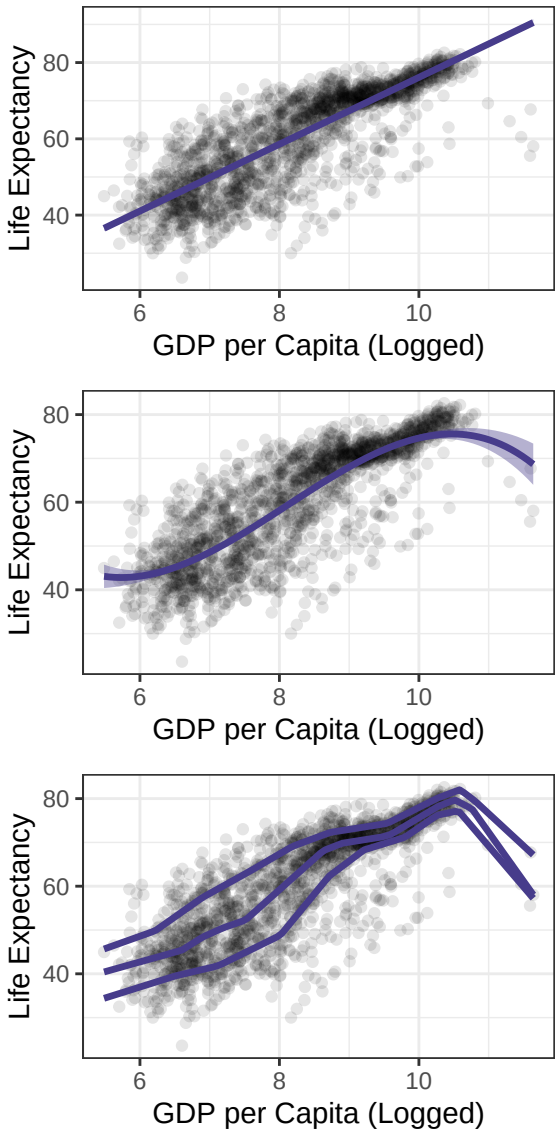
Uncertainty in the Social Science

Data visualization is about more than generating figures that display the raw numbers from a table of data. As we have seen in the last three weeks, it can involve summarizing or transforming parts of the data and plot the results thereof. Statistical models are a central part of the process. The goal of a statistical model is to provide a simple low-dimensional summary of a dataset. Ideally, the model will capture true "*signals*" (i.e. patterns generated by the phenomenon of interest), and ignore "*noise*" (i.e. random variation that you're not interested in).

Today, we will explore how ggplot can be used for various modeling techniques directly with geoms. We will use the `broom` and `margins` libraries to tidily extract and plot estimates from models that we fit ourselves. For this tutorial, it's not necessary to have a solid background or understanding of statistics.[1]

The below displayed graphs demonstrate different ways to show signals and noise in data. For all three graphs, the `geom_point()` displays the relation between the variables `lifeExp` (life expectancy) and `gdpPercap` (GDP per capita). The relation is positive and linear, meaning that the richer a country is (i.e. higher GDP per capita), the higher the citizens are expected to live (i.e. higher levels of life expectancy). This relationship could already be observed by 'eye-balling' the data - i.e. just looking at a simple dotplot. Yet, when we want model the 'exact' relation between the two variables, there are various ways to do that. *First*, you can add a `geom_smooth()` layer. This line tries to adequately capture the underlying pattern in the data - i.e. the signal. This layer gives you many options, the upper and middle plot both have a smoothener added. Still, those two lines differ. The upper plot shows a linear display of the relation, while the middle plot shows a non-linear version. The lighter area around the solid line of the middle plot displays the standard errors around the estimated relation between `gdpPercap` and `lifeExp`. The standard error (SE) of a statistic (i.e. an estimate of a parameter) is an estimate of that standard deviation, it displays the variation around the estimation: The narrower the standard error around an estimate, the more certain we can be that the estimate reflects some pattern in the real world. *Second*, to visualize the relation between `gdpPercap` and `lifeExp`, one could choose to display the different *quantiles*. In the lower plot, three lines are visualized: The lower one displaying the 20th percentile, the middle one the 50th percentile and the upper one the 85th percentile. That means that 65% of the observations are between the lower and upper line.

[1] For a comprehensive, modern introduction to that topic you should work your way through Gelman & Hill (2018). Harrell (2016) is also very good on the many practical connections between modeling and graphing data. Similarly, Gelman (2004) provides a detailed discussion of the use of graphics as a tool in model-checking and validation.



Today, we will discuss some ways to take the models that you fit and extract information that is easy to work with in ggplot. Our goal, as always, is to get from however the object is stored to a tidy table of numbers that we can plot. Most classes of statistical models in R will contain the information we need, or will have a special set of functions, or methods, designed to extract it. We can start by learning a little more about how the output of models is stored in R. Remember, we are always working with objects, and objects have an internal structure consisting of named pieces. Sometimes these are single numbers, sometimes vectors, and sometimes lists of things like vectors, matrices, or formulas. We have been mainly working with tibbles and data frames, like e.g. the `gapminder` data:

```
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
```

```
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Tibbles and data frames store tables of data with named columns, perhaps consisting of different classes of variable, such as integers, characters, dates, or factors. We can use `str()` to learn more about the internal structure of any object. For example, we can get some information on what class (or classes) of object `gapminder` is, how large it is, and what components it has. The output from `str(gapminder)` is somewhat dense:

```
str(gapminder)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22
##  $ gdpPercap: num  779 821 853 836 740 ...
```

There is a lot of information here about the object as a whole and each variable in it. In the same way, statistical models in R have an internal structure. Yet, because models are more complex entities than data tables, their structure is correspondingly more complicated. There are more *pieces* of information, and more *kinds* of information, that we might want to use. All of this information is generally stored in or is computable from parts of a model object.

We can create a linear model, an ordinary OLS regression, using the `gapminder` data. As a side note, if you would work with this data on a paper, you have to be aware that this dataset has a country-year structure, which makes an OLS specification like this the wrong one to use. Never mind that for now. We use the `lm()` function to run the model, and store it in an object called `out`:

```
(out <- lm(formula = lifeExp ~ gdpPercap + pop + continent,
           data = gapminder))
```

The first argument is the formula for the model. `lifeExp` is the dependent variable and the tilde `~` operator is used to designate the left- and right-hand sides of a model (including in cases, as we saw with `facet_wrap()` where the model just has a right-hand side.)

Assignment 1: Models & Their Structure

1a) Try out `summary(out)` and look at the summary. Add a screenshot to your assignment.

1b) When we use `summary()` on `out`, we are not getting a simple feed of what's in the model object. Instead, like any function, `summary()` takes its input, performs some actions, and produces output. What is printed to the console is partly information that is stored inside the model object, and partly information that the `summary()` function has calculated and formatted for display on the screen. Behind the scenes, `summary()` gets help from other functions. Objects of different classes have default methods associated with them. Try `summary(gapminder)`, what type of information is given? What are the similarities and differences with `summary(out)`'?

1c) The output from `summary(out)` gives a precis of the model, but we can't really do any further analysis with it directly. For example, what if we want to plot something from the model? The information necessary to make plots is inside the `out` object, but it is not obvious how to use it. The figure below displays the structure of the `out` object. In this list of items, elements are single values, some are data frames, and some are additional lists of simpler items. List objects could be thought of as being organized like a filing system: cabinets contain drawers, and drawers may contain folders, which may contain pages of information, whole documents, or groups of folders with more documents inside. Try `out$coefficients` and `out$fitted.values` in the console. What do you get?

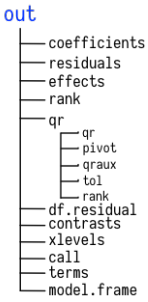


Figure 1: Structure of a object for a linear model

Visually Presenting Models

Figures based on statistical models face all the ordinary challenges of effective data visualization, and then some. This is because model results usually carry a considerable extra burden of interpretation and necessary background knowledge. The more complex the model, the trickier it becomes to convey this information effectively, and the easier it becomes to lead one's audience or oneself into error. Within the social sciences, our ability to clearly and honestly present model-based graphics has greatly improved over the past ten or fifteen years. Over the same period, it has become clearer that some kinds of models are quite tricky to understand, even ones that had previously been seen as straightforward elements of the modeling toolkit (Ai & Norton, 2003; Brambor, Clark, & Golder, 2006).

Plotting model estimates is closely connected to properly estimating models in the first place. This means there is no substitute for learning the statistics! You should not use graphical methods as a substitute for understanding the model used to produce them. While we cannot teach you that material in the scope of the course, we can make a few general points about what good model-based graphics look like, and work through some examples of how `ggplot` and some additional libraries can make it easier to get good results.

1. Useful model-based plots show results in ways that are substantively meaningful and directly interpretable with respect to the questions the analysis is trying to answer. This means showing results in a context where other variables in the analysis are held at sensible values, such as their means or medians. With continuous variables, it can often be useful to generate predicted values that cover some substantively meaningful move across the distribution, such as from the 25th to the 75th percentile, rather than a single-unit increment in the variable of interest. For unordered categorical variables, predicted values might be presented with respect to the modal category in the data, or for a particular category of theoretical interest. Presenting substantively interpretable findings often also means using (and sometimes converting to) a scale that readers can easily understand. There is nothing distinctively graphical about putting the focus on the substantive meaning of your findings.

2. Much the same applies to presenting the degree of uncertainty or confidence you have in your results. Model estimates come with various measures of precision, confidence, credence, or significance. Presenting and interpreting these measures is notoriously prone to misinterpretation, or over-interpretation, as researchers and audiences both demand more from things like confidence intervals and p-values than these statistics can deliver. At a minimum, having decided on an appropriate measure of model fit or the right assessment of confidence, you should show their range when you present your results. A family of related `ggplot` geoms allow you to show a range or interval defined by position on the x-axis and then a `ymin` and `ymax` range on the y-axis. These geoms include `geom_pointrange()` and `geom_errorbar()`, which we will see in action shortly. A related geom, `geom_ribbon()` uses the same arguments to draw filled areas, and is useful for plotting ranges of y-axis values along some continuously varying x-axis.

3. Plotting the results from a model with multiple variable (so-called multivariate model) generally means one of two things. First, we can show what is in effect a table of coefficients with associated measures of confidence, perhaps organizing the coefficients into meaningful groups, or by the size of the predicted

association, or both. Second, we can show the predicted values of some variables (rather than just a model's coefficients) across some range of interest. The latter approach lets us show the original data points if we wish. The way `ggplot` builds graphics layer by layer allows us to easily combine model estimates (e.g. a regression line and an associated range) and the underlying data. In effect these are manually-constructed versions of the automatically-generated plots that we have been producing with `geom_smooth()`.

Having fitted a model, then, we might want to get a picture of the estimates it produces over the range of some particular variable, holding other covariates constant at some sensible values. The `predict()` function is a generic way of using model objects to produce this kind of prediction. In R, "generic" functions take their inputs and pass them along to more specific functions behind the scenes, ones that are suited to working with the particular kind of model object we have. The details of getting predicted values from a OLS model, for instance, will be somewhat different from getting predictions out of a different type of regression, such as a logistic regression. But in each case we can use the same `predict()` function, taking care to check the documentation to see what form the results are returned in for the kind of model we are working with. Many of the most commonly-used functions in R are generic in this way. The `summary()` function, for example, works on objects of many different classes, from vectors to data frames and statistical models, producing appropriate output in each case by way of a class-specific function in the background.

For `predict()` to calculate the new values for us, it needs some new data to fit the model to. We will generate a new data frame whose columns have the same names as the variables in the model's original data, but where the rows have new values. A very useful function called `expand.grid()` will help us do this. We will give it a list of variables, specifying the range of values we want each variable to take. Then `expand.grid()` will generate the will multiply out the full range of values for all combinations of the values we give it, thus creating a new data frame with the new data we need.

In the following bit of code, we use `min()` and `max()` to get the minimum and maximum values for GDP per capita, and then create a vector with one hundred evenly-spaced elements between the minimum and the maximum. We hold population constant at its median, and we let continent take all of its five available values:

```
(min_gdp <- min(gapminder$gdpPercap))
(max_gdp <- max(gapminder$gdpPercap))
(med_pop <- median(gapminder$pop))

(pred_df <- expand.grid(gdpPercap = (seq(from = min_gdp,
                                         to = max_gdp,
                                         length.out = 100)),
                        pop = med_pop,
                        continent = c("Africa", "Americas",
                                      "Asia", "Europe", "Oceania")))
```

Now we can use `predict()`. If we give the function our new data and model, without any further argument, it will calculate the fitted values for every row in the data frame. If we specify `interval = 'predict'` as an argument, it will calculate 95% prediction intervals in addition to the point estimate.

```
(pred_out <- predict(object = out,
                     newdata = pred_df,
                     interval = "predict"))
```

Because we know that, by construction, the cases in `pred_df` and `pred_out` correspond row for row, we can bind the two data frames together by column. Note: this method of joining or merging tables is definitely not recommended when you are dealing with data!

```
pred_df <- cbind(pred_df, pred_out)
```

The end result is a tidy data frame, containing the predicted values from the model for the range of values we specified. Now we can plot the results. Because we produced a full range of predicted values, we can decide

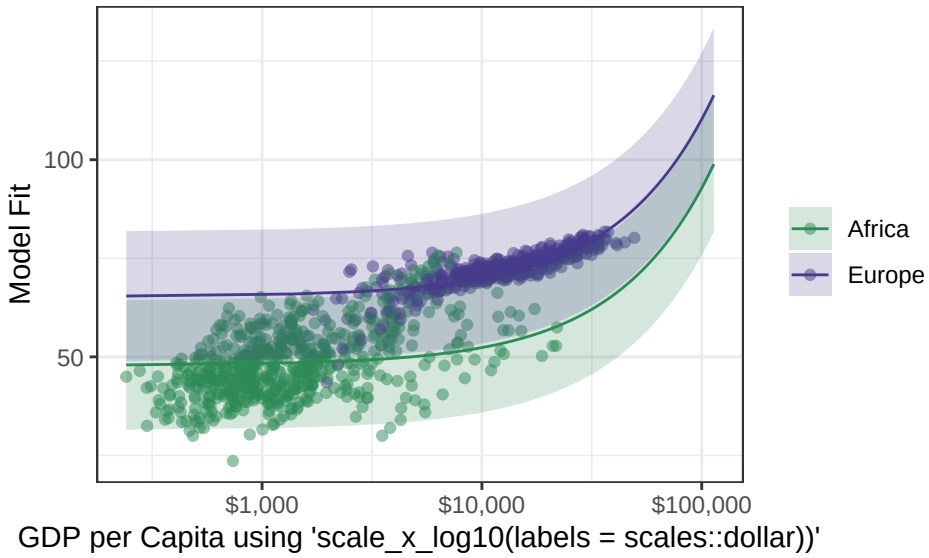
whether or not to use all of them.

Assignment 2: Plot a fitted model

2a) How many columns does the new data set `pred_df` *have? And how many rows? Use* `dim(pred_df)`.

2b) And how many columns does the new data set `pred_out` *have? And how many rows? Use* `dim(pred_out)`.

2c) Look at the plot displayed below. This is based on the predicted data (`pred_df`*) - displayed by the line and the confidence intervals around the line using* `geom_ribbon()` *- as well as on the real data from* `gapminder` *- displayed with* `geom_point()`. *The data is filtered, which data is included and which is not? Tip: see* `?table` *to find out information about a variable (e.g.* `table(gapminder$continent)`.



2d) Describe the layers (i.e. geoms) that you see in the graph.

2e) Replicate the graph. I've done some filtering using `data = subset(NAME DATA, continent %in% c("Europe", "Africa"))`. *Moreover, I manually picked the colors for the two continents. I've used "seagreen" and "slateblue4", but have a look at http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf and pick your own favorite colors. Additionally, I set a theme (*`theme_bw()`*), and removed the title of the legend with* `theme(legend.title=element_blank())`.

Good Luck

The deadline for the assignment is **November 28, 10am**!

References

Healy, K. (2018). *Data visualization: a practical introduction*. Princeton University Press.