KOSTAT-UNFPA Summer Seminar on Population

Workshop 1. Demography in R

Day 4: Visualizing data using ggplot2

Instructor: Tim Riffe
tim.riffe@gmail.com

Assistant: Rustam Tursun-Zade
rustam.tursunzade@gmail.com

29 July 2021

Contents

1 Summary

So far we've focused on getting data tidy, processing within the tidy framework, and using custom functions in the tidy framework. Today, we focus on visualizing data that is already tidy and ready-to-go. It could be raw data, or a calculated or estimated quantity on your part. For most of this, you could imagine a tidy pipeline preceding the visualization code.

2 Example data

Here's a small example, using a nice tidy dataset delivered by the *gapminder* project https://www.gapminder.org/, which is used often in the excellent book that I recommend you use to reinforce this class, *Data Visualization, a practical introduction* (Healy (2018)). You can get a free online version of the book here: https://socviz.co/

```
# install.packages("gapminder")
library(tidyverse)
library(gapminder)
glimpse(gapminder)
```

```
## Rows: 1,704
## Columns: 6
## $ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ~
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ~
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ~
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8~
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12~
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ~
```

3 ggplot2 principles

The `ggplot2` package is an implementation of a theoretical framework to data visualization, a so-called grammar of graphics (Wilkinson (2012), Wickham (2011)). It loads automatically when you load the `tidyverse`, or else you can directly load `ggplot2`.

Recall the definition of *tidy* data: observations in rows, and variables in columns. Sometimes the idea of what is an observation and what is a variable only becomes clear when we get to the step of plotting data at the end of the pipeline. Variables *map* to different aesthetic aspects of the plot: they get translated to coordinates and geometric elements, and potentially to colors, widths, or other visual properties. In order to map to such qualities in a plot, we need to be able to achieve a 1:1 correspondence from variables to aspects of the plot. Mappings are declared inside the function `aes()` (aesthetics), which you've seen me do inside `ggplot()`, but which I've not yet explained.

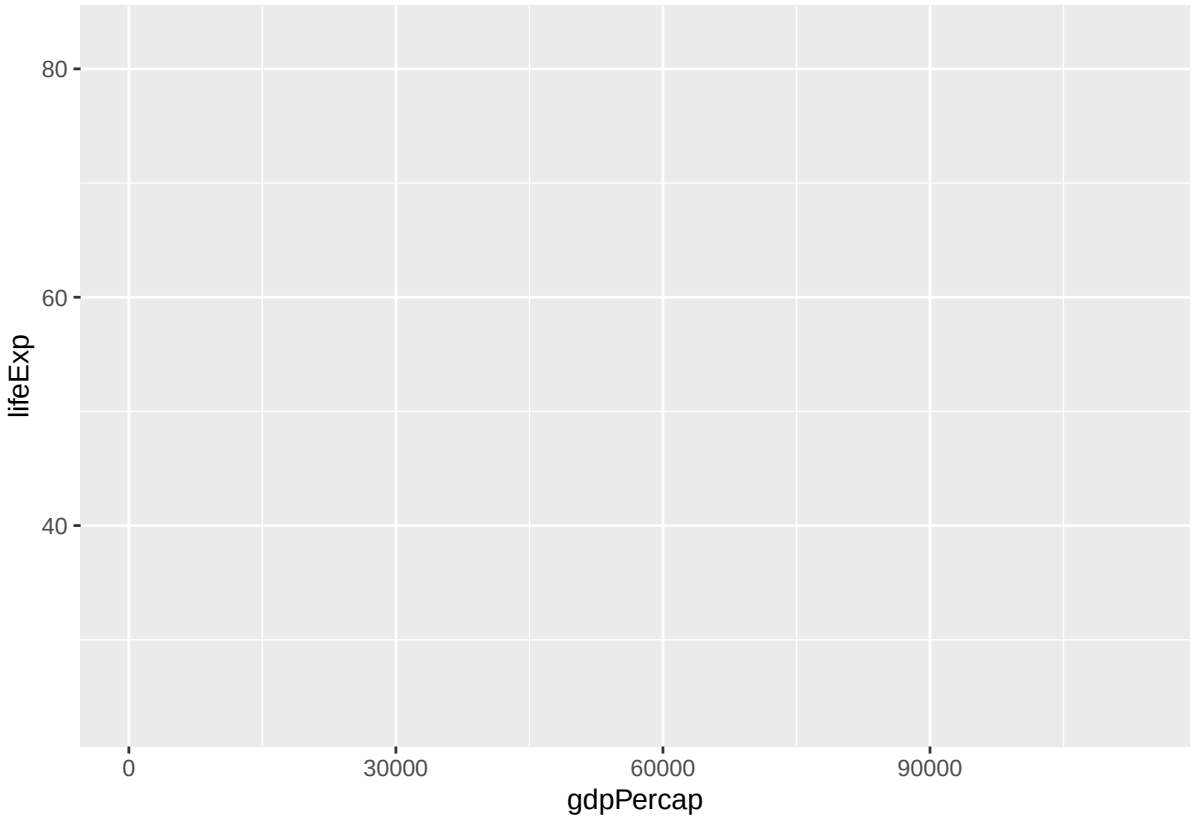
3.1 Setting up a plot

Setting up a simple plot will serve to show the basic anatomy of a `ggplot()`.

Here we map the variable for average GDP per person `gdpPercap` to the abscissa (x coordinate) and the variable for life expectancy at birth e_0 `lifeExp` to the y coordinate. Executing this

opens a plot device that is empty, but which fits the values ranges in the data, and has some nice default settings.

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp))
```

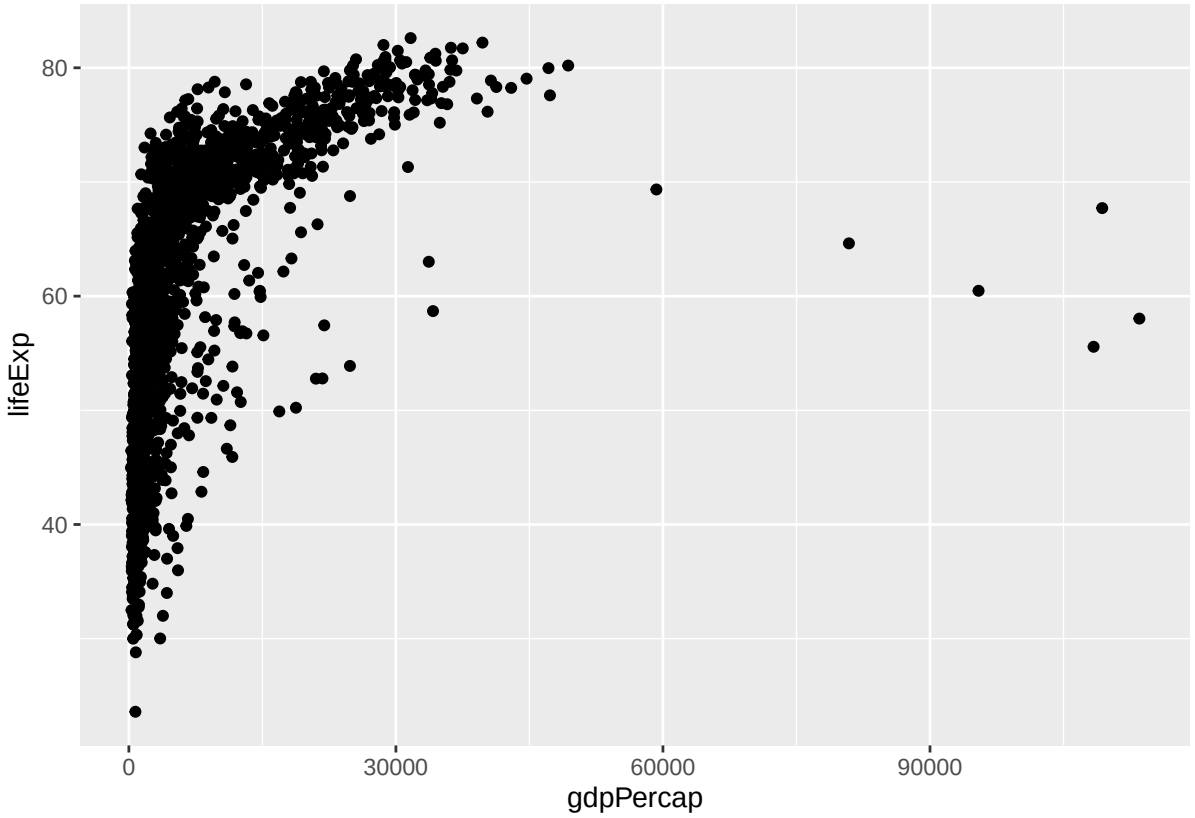


Since there is no default geometric mapping, the plot remains empty; in this case a declared Cartesian space.

3.2 Adding a geometric mapping

Elements can be **added** `+` to the plot using the concept of layering. In this case, it will do to add points, using the `geom_point()` function. Observe:

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```

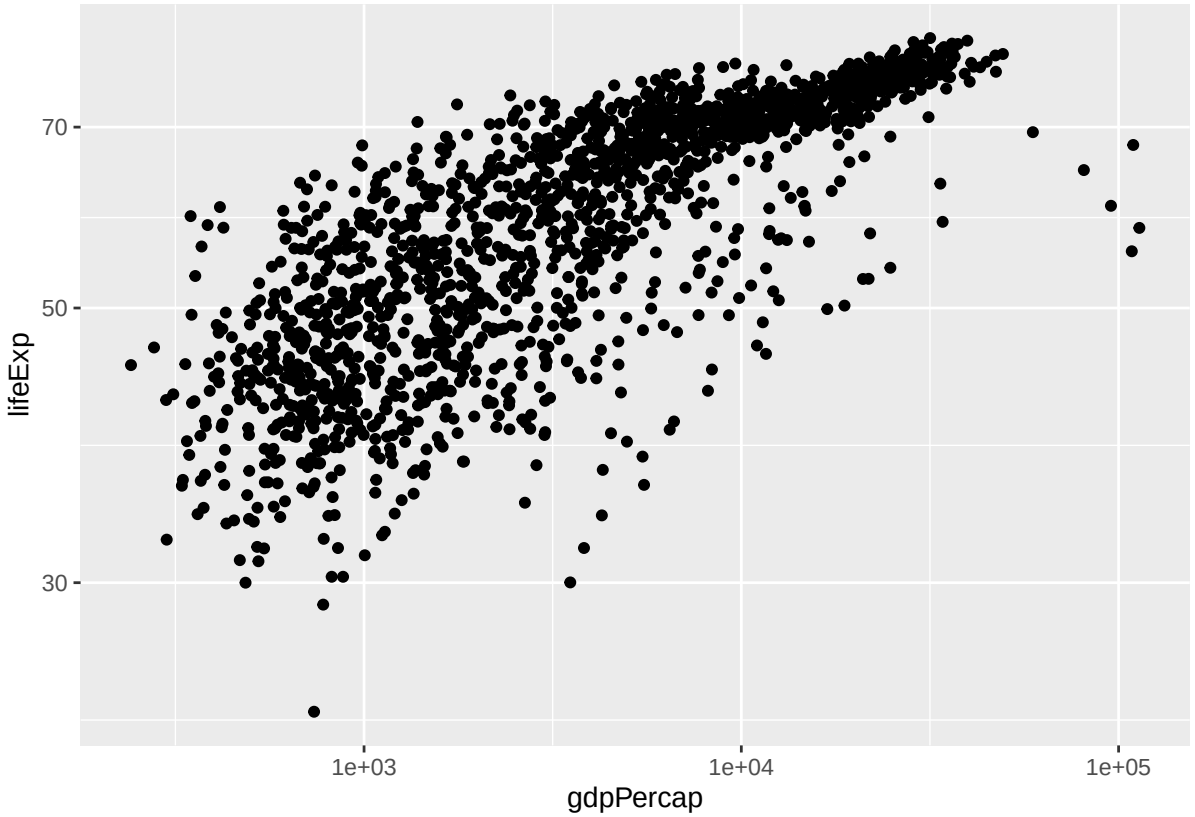


Excellent, our first plot!

3.3 Setting scales

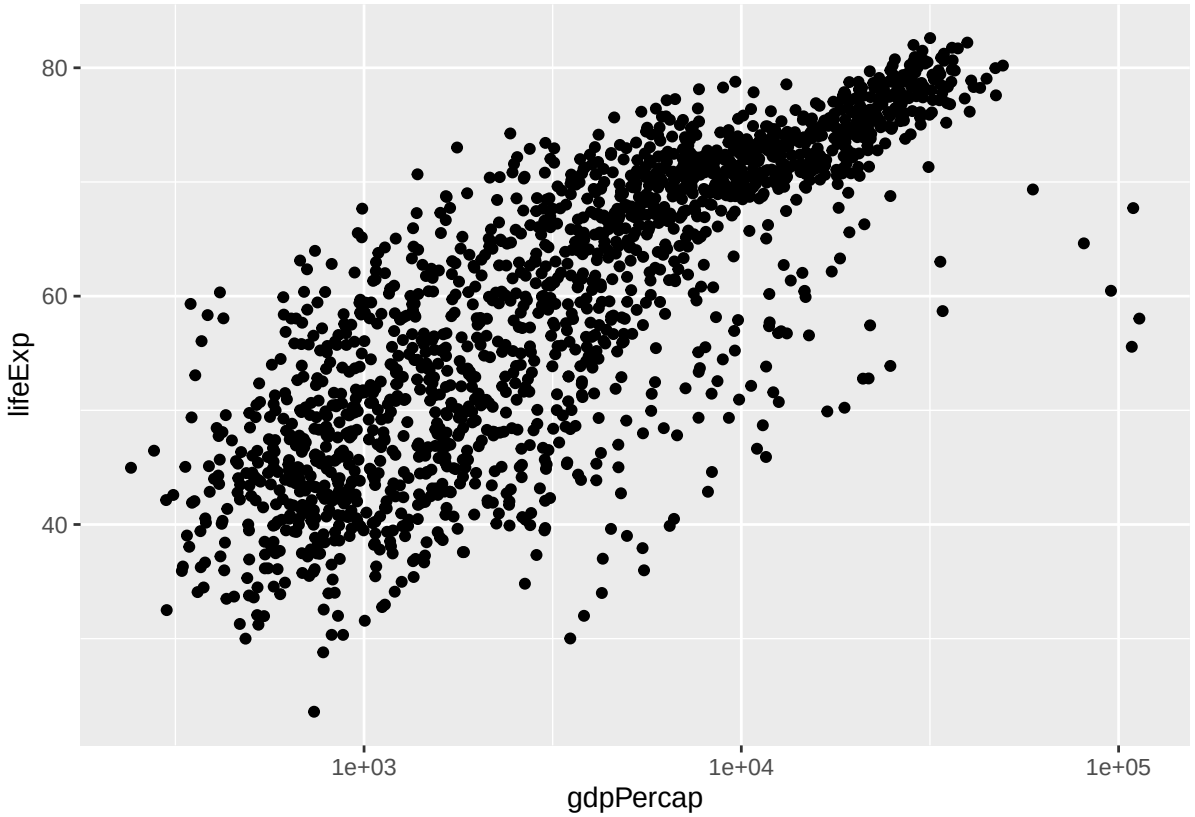
Sometimes mapping the data to the desired `geom` is the hard part. See how the data is bunched up around low values of `gdpPercap` and tapers off in the high range of `lifeExp`? Often in this situation, logging one or another scale reveals the pattern in the data better. We can alter the x and y scales by adding this information to the plot with `scale_x_log()` (my preference here), for example. Here we log both axes:

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_log10() +
  scale_y_log10()
```



Having both axes logged lends itself to interpreting potential trends in terms of proportional changes. This dataset might not benefit much from logging life expectancy:

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_log10()
```



That's a matter of taste, though. Notice how `ggplot` logs the values, but the tick labels are still in the original units? I love this feature.

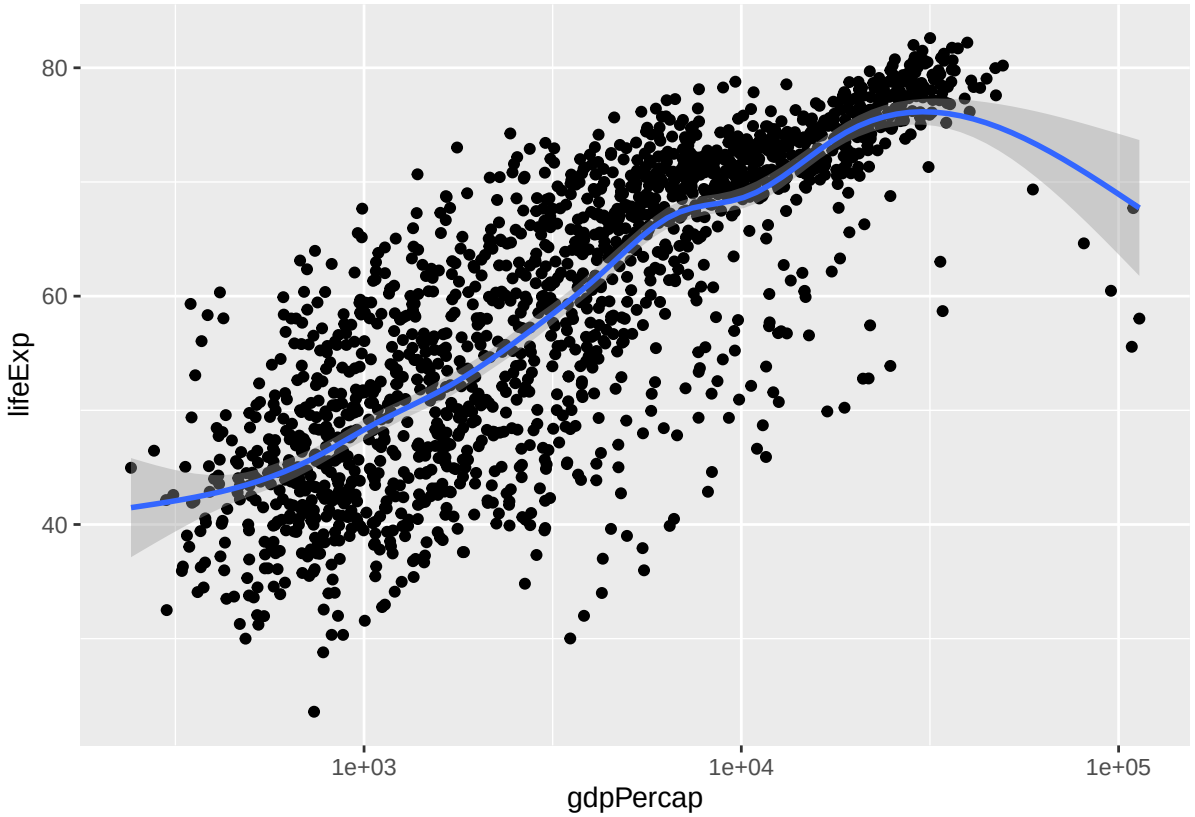
To get a sense of what other scales are available, check out the autocomplete options after typing `scale_` in the console. You can scale other things like colors or fill colors, and possibly other things too.

3.4 `geom_smooth()`

You can add a smoother to the series using another `geom`, `geom_smooth()`. As you type it, go ahead and observe the pop-up list from RStudio. Scroll down the suggestions menu to get a sense of some of the options, or basically to understand that there are *many* of them.

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  geom_smooth() +
  scale_x_log10()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



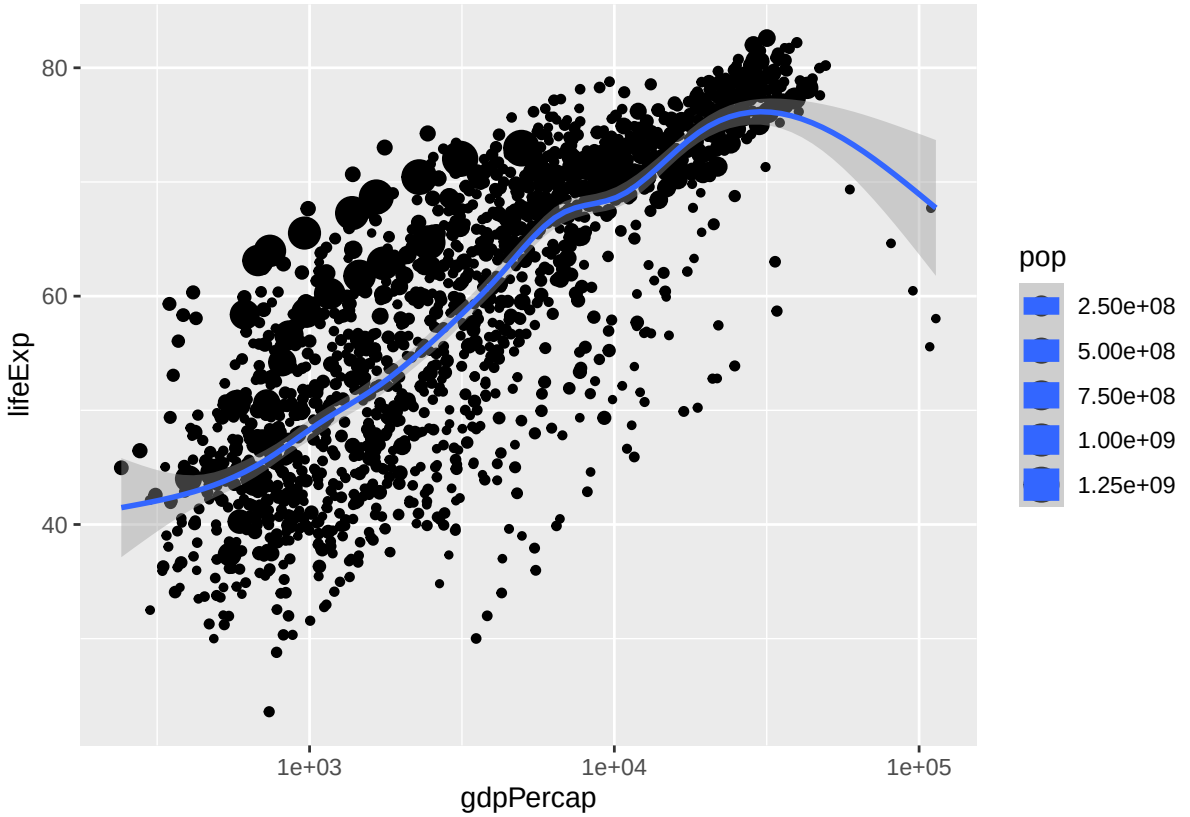
By default, a GAM smoother is used, but others are available to you, see `?geom_smooth` for more options. These smoothers are in fact doing some statistics on top of the data, but only for the purposes of plotting the result as an enhancement of the legibility of the plot, just to be clear on that point.

3.5 Mapping to `size`

Let's see how variables can map to more than just coordinates:

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, size = pop)) +
  geom_point() +
  geom_smooth() +
  scale_x_log10()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Here for each point we have an associated population size. Mapping population `pop` to `size` will scale the *area* of each point.

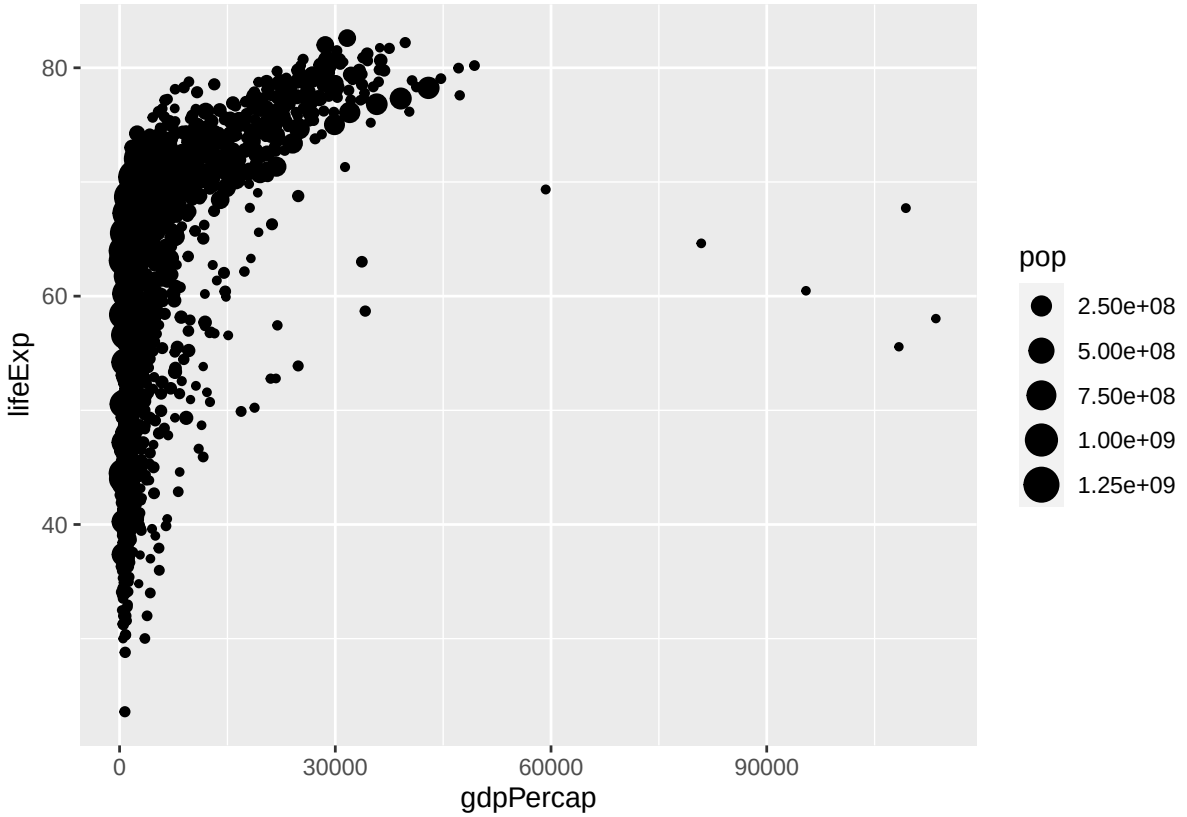
3.6 Plots can be assigned and modified

We'll be modifying this plot several times to point out different features of `ggplot2`. It will be handy to point out that a `ggplot` can be assigned to an object, and it can be modified. See:

```
p <-
  gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, size = pop)) +
  geom_point()
```

No plot was printed because we just assigned it to an object. To actually view the plot, just type `p`:

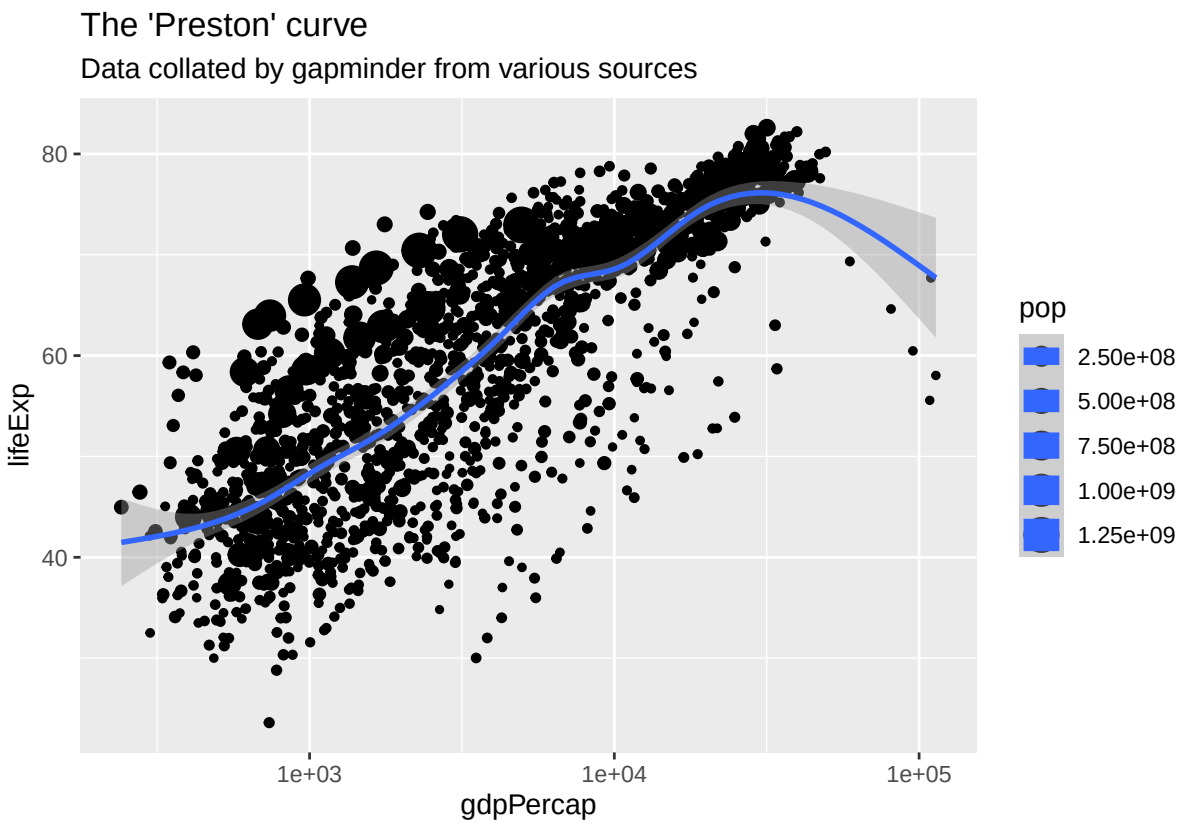
```
p
```



We can simply add more aesthetics to this plot

```
p +
  geom_smooth() +
  scale_x_log10() +
  labs(title = "The 'Preston' curve",
       subtitle = "Data collated by gapminder from various sources")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



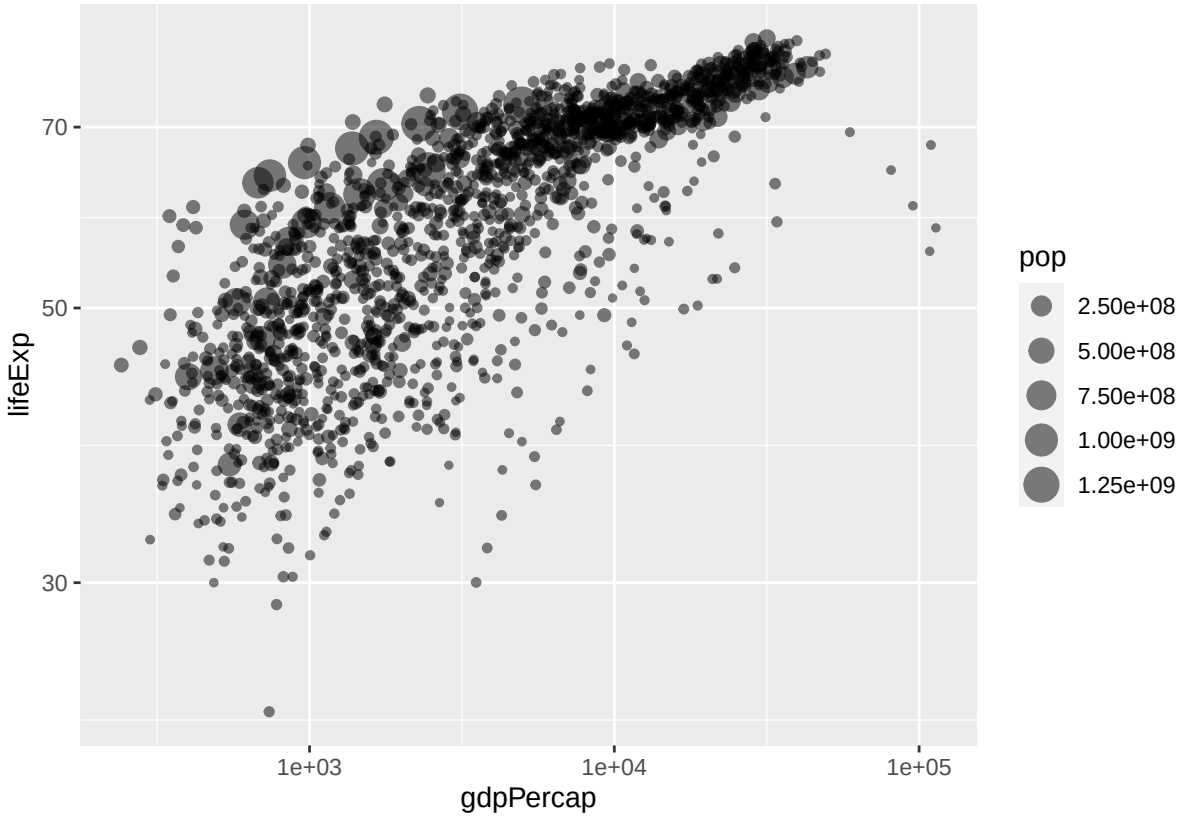
But you can't modify elements that have already been drawn without using other packages or digging into the data structure of `p`. It's far easier to just re-create it as desired.

3.7 *Setting* versus *mapping* aesthetics

3.7.1 Setting transparency, `alpha`

Currently our plot has lots of overlapping points. We probably have very dense areas, but can't really know which is the densest, because the points are all pushed together. This is called overplotting, and it reduces the information content of the plot. There are some easy things we can do to offset this effect. We could deal with this by setting points to be semi-transparent, using an argument called `alpha` in this case, we might not want to *map* a variable to alpha (although we could), but rather just set a level for all points. To do this, we'll want to specify `alpha` (a number between 0 and 1) outside the `aes()`, directly inside the relevant `geom`:

```
gapminder %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, size = pop)) +
  geom_point(alpha = .5) +
  scale_x_log10() +
  scale_y_log10()
```

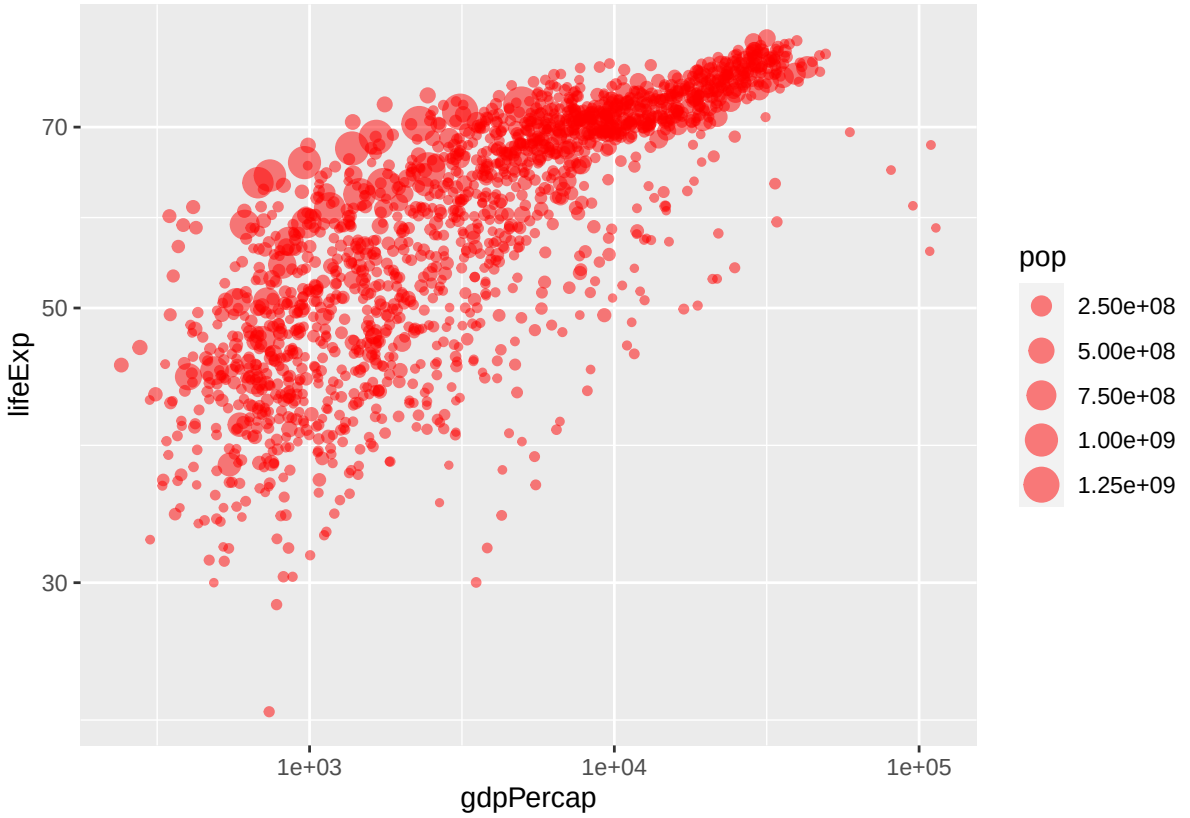


Setting aesthetic properties outside of `aes()`, whether in the `geom_*` or `ggplot()` part of the plot construction, will apply the setting wherever applicable with no mapping. It's not mapping because it doesn't derive from a *variable*.

3.7.2 Setting `color`

We could do the same with `color`:

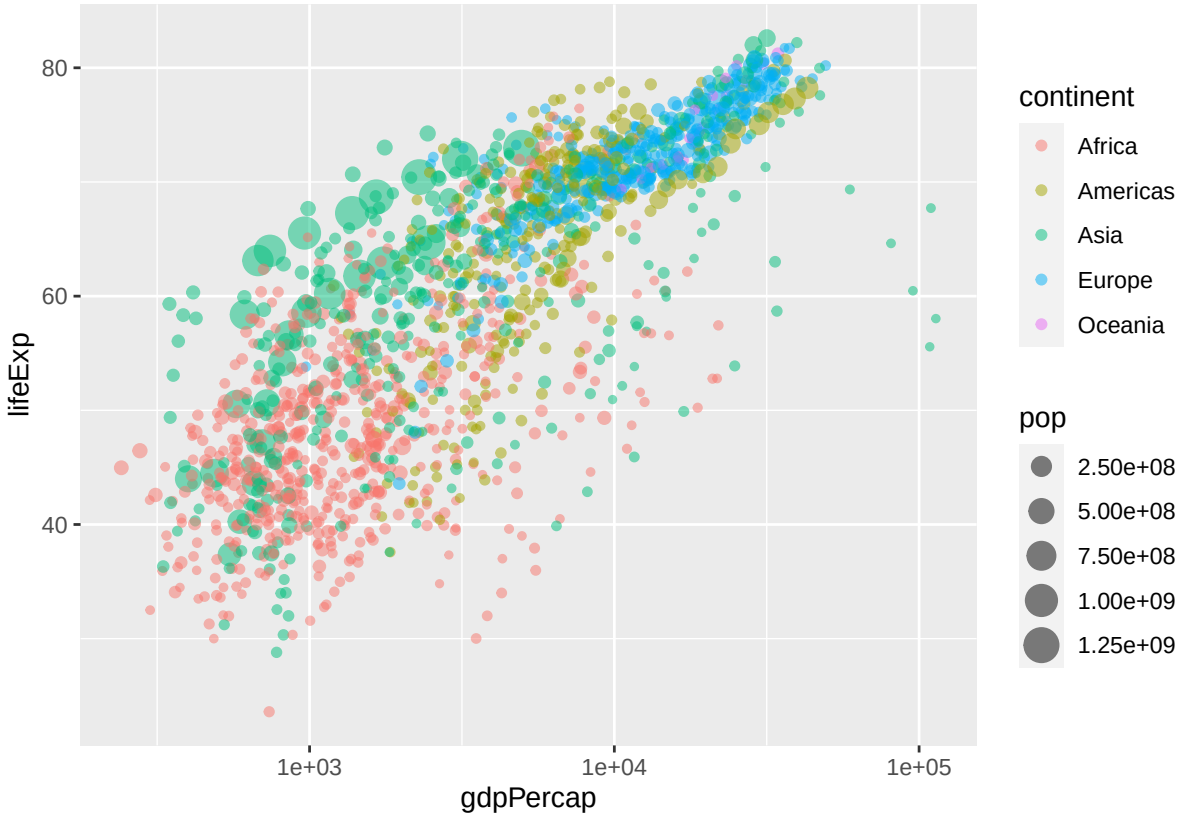
```
gapminder %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp,
             size = pop)) +
  geom_point(alpha = .5, color = "red") +
  scale_x_log10() +
  scale_y_log10()
```



Mapping `color`

There is clearly a lot of heterogeneity in this data cloud. We can map other variables to an aesthetic such as color to see more patterns in the data. In this example, since there's only one `geom`, all of the aesthetic mappings can be handled in the initial `aes()` statement.

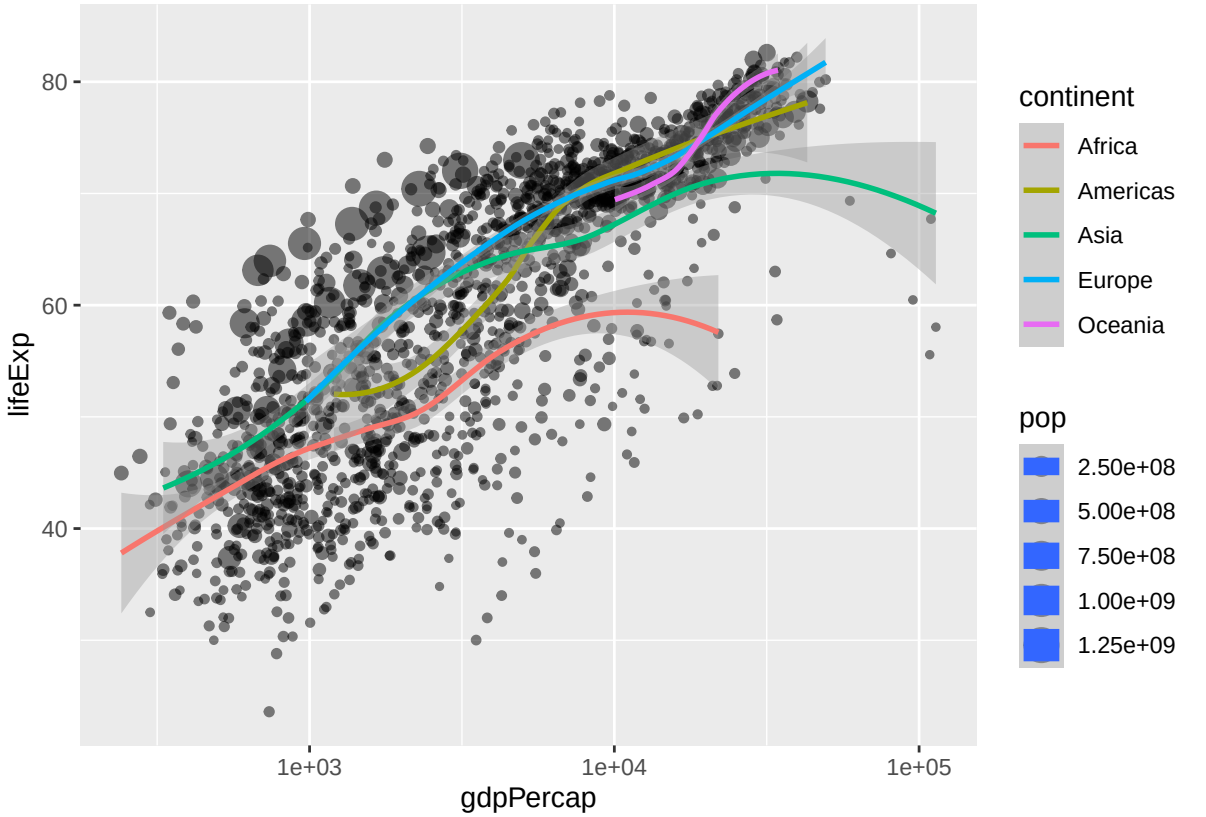
```
gapminder %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp,
             size = pop,
             color = continent)) +
  geom_point(alpha = .5) +
  scale_x_log10()
```



3.7.3 Mapping inside the `geom`

You can also place `aes()` inside of `geom_*` functions, and this makes sense especially if the mappings only apply to the given `geom`. Here's an example building from our `geom_smooth()` use earlier:

```
gapminder %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp,
             size = pop)) +
  geom_point(alpha = .5) +
  geom_smooth(aes(color = continent)) +
  scale_x_log10()
```



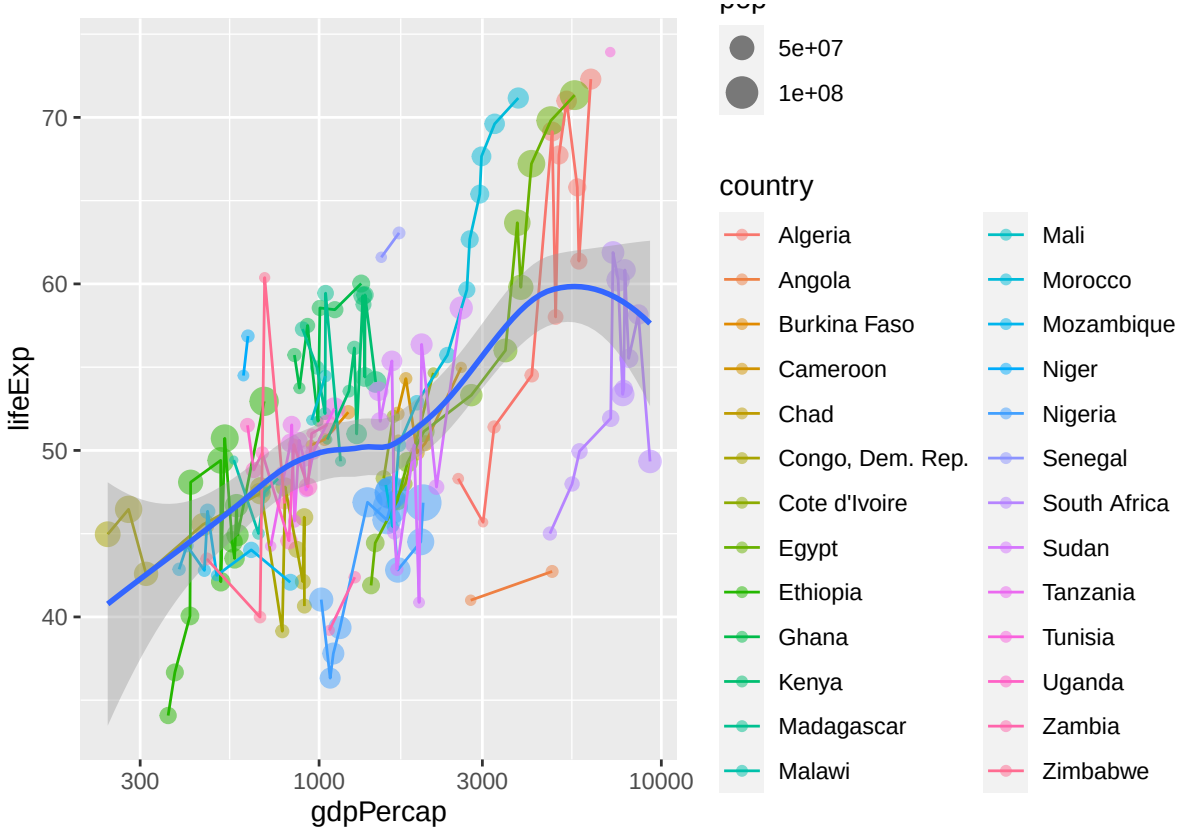
In this example, points are undifferentiated by continent, but the smoother color is mapped to continent, which means that it needs to be generated separately for each continent subset. So, you have control over the mappings in each `geom`, that's nice to know!

3.8 Adding within-country trends

Does the overall continent trend match within-country trends? This will work better if we can reduce the number of series being plotted. To add `country` paths, we include `geom_line()`, which needs its own `aes()` mapping. To make sure each country is its own series, we specify `group = country` in the mapping. This serves to point out that country trajectories appear far steeper than the overall continent would suggest. It's also easier to see that some paths are linear while others are complex.

```
gapminder %>%
  filter(continent == "Africa",
         pop > 1e7) %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp)) +
  geom_point(aes(size = pop, color = country),alpha = .5) +
  geom_line(aes(group = country, color = country)) +
  geom_smooth() +
  scale_x_log10()
```

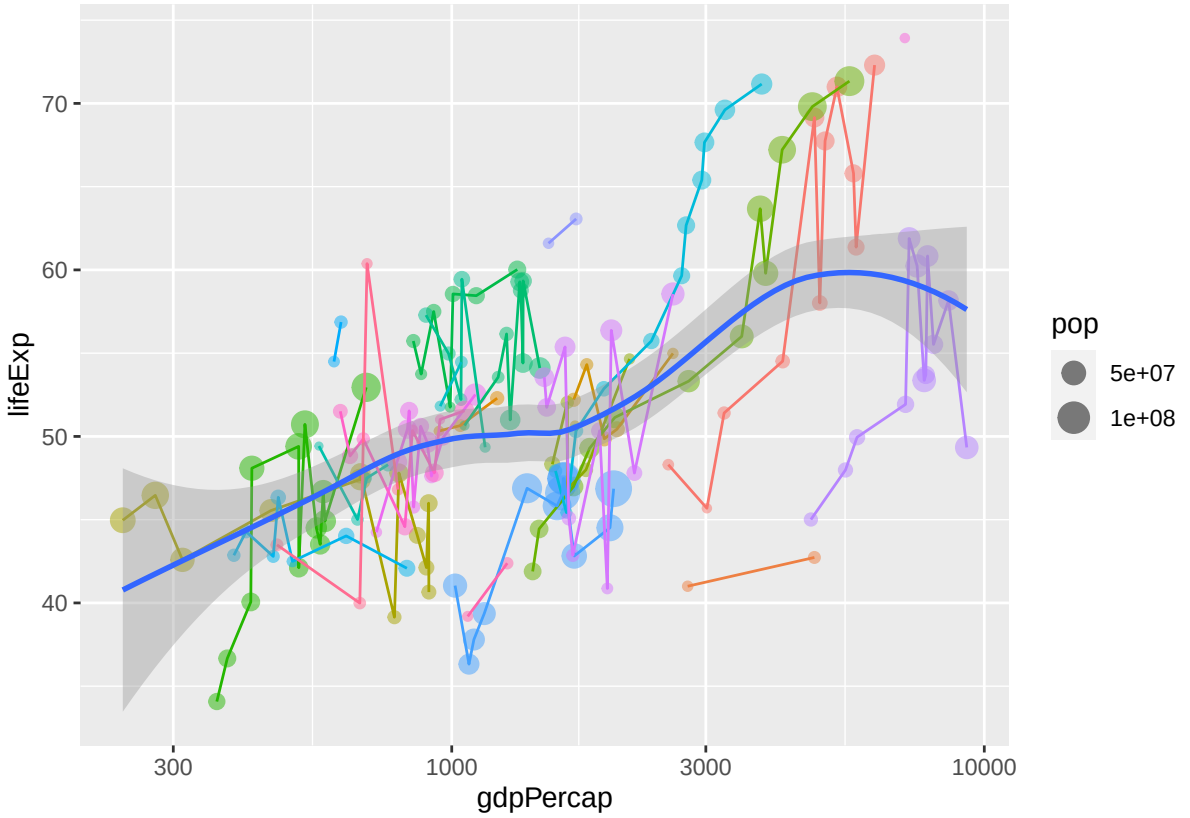
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



In such plots with many series, the color legend is basically pointless. There are different ways to customize legends or turn them off. The `guides()` layer can toggle legends for specific mapped variables if we want. Just add `guides(color = "none")` to eliminate the pointless color legend.

```
gapminder %>%
  filter(continent == "Africa",
         pop > 1e7) %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp)) +
  geom_point(aes(size = pop, color = country),alpha = .5) +
  geom_line(aes(group = country, color = country)) +
  geom_smooth() +
  scale_x_log10() +
  guides(color = "none")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Adjusting legend units

If we want to remove the scientific notation from the size legend, we can use the super-helpful `scales` package. This gives access to the `number_format()` function, where we can scale down to per 1000 population sizes.

```
library(scales)
```

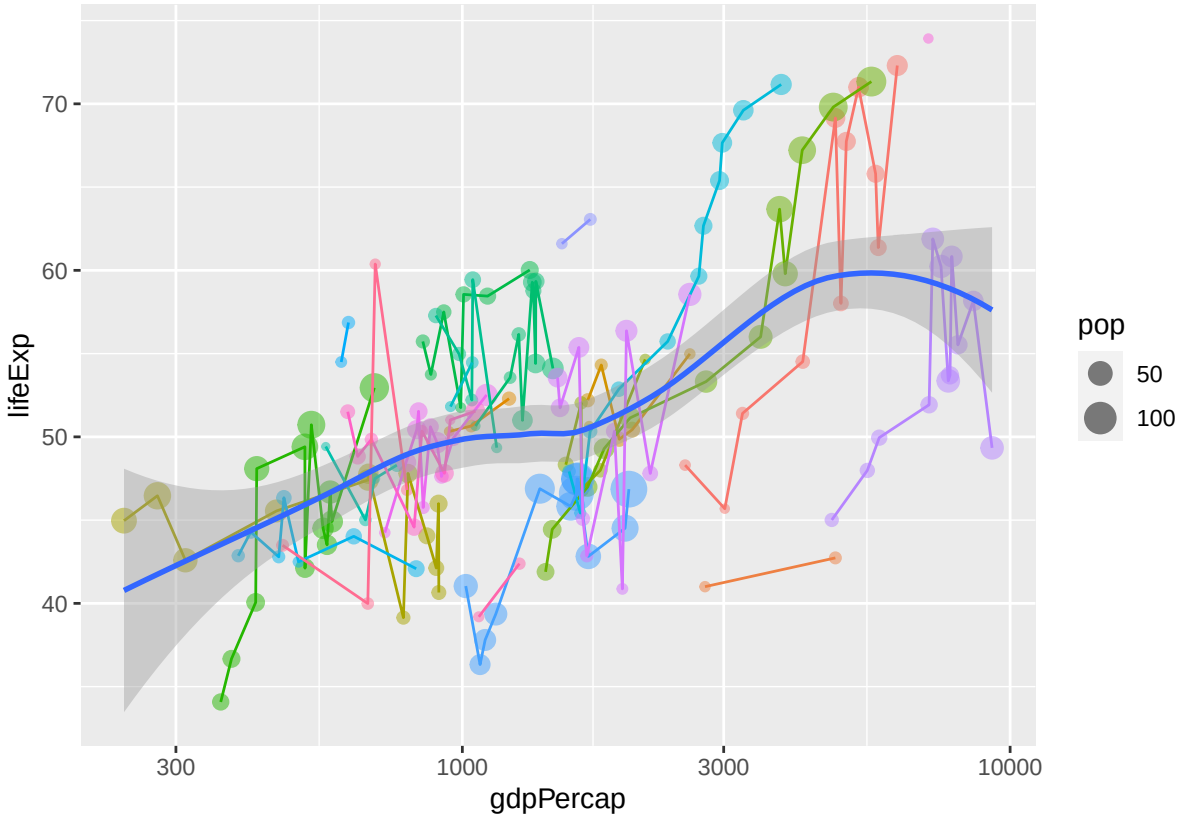
```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
gapminder %>%
  filter(continent == "Africa",
         pop > 1e7) %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp)) +
  geom_point(aes(size = pop, color = country), alpha = .5) +
  geom_line(aes(group = country, color = country)) +
  geom_smooth() +
  scale_x_log10() +
  guides(color = "none") +
  scale_size_continuous(labels = number_format(scale = 1/1e6))
```

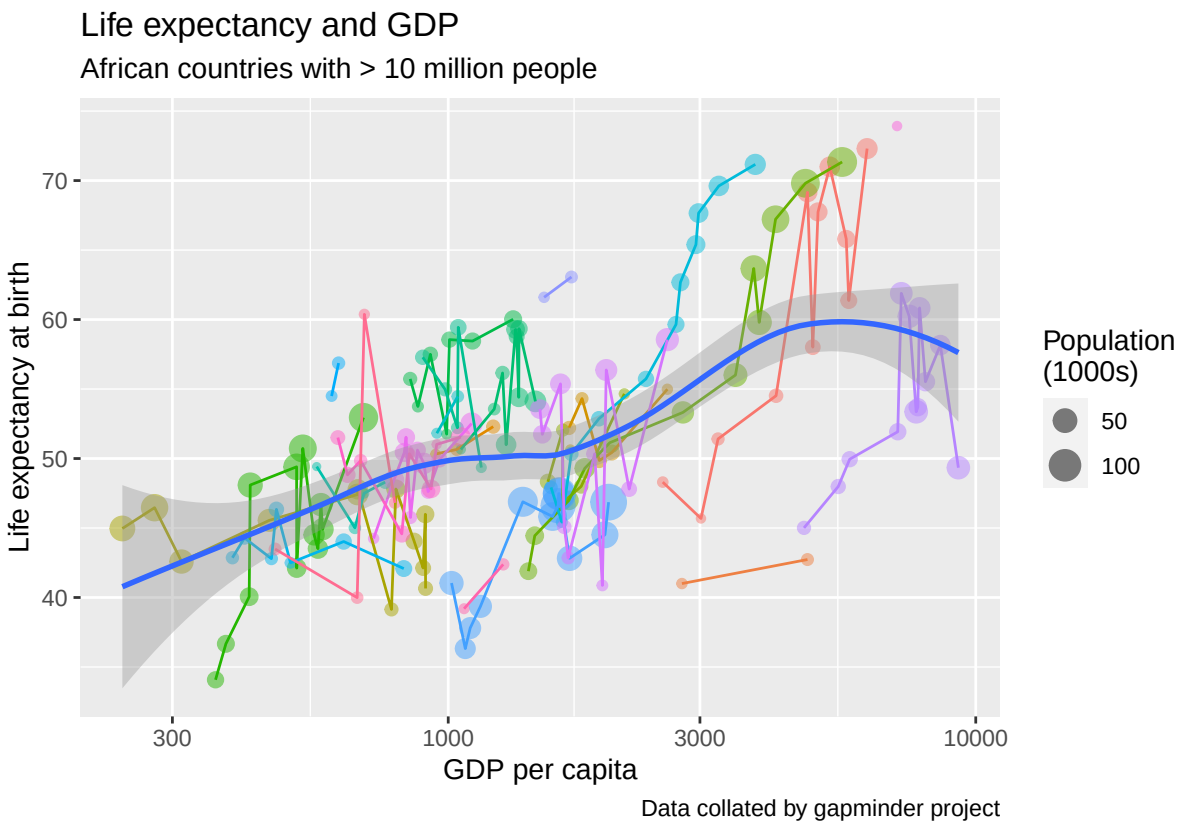
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Adjust legend title and axis labels It's nice to also adjust the legend title. We can control this also from the `guides()` function, just specify which aesthetic this applies to `size = ...`, and here the `guide_lenged()` gives access to this aspect of the plot. Note `\n` makes a line break. `labs()` lets us specify axis labels and titles, subtitles, and captions.

```
gapminder %>%
  filter(continent == "Africa",
         pop > 1e7) %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp)) +
  geom_point(aes(size = pop, color = country), alpha = .5) +
  geom_line(aes(group = country, color = country)) +
  geom_smooth() +
  scale_x_log10() +
  guides(color = "none",
         size = guide_legend(title = "Population\n(1000s)")) +
  scale_size_continuous(labels = number_format(scale = 1/1e6)) +
  labs(x = "GDP per capita",
       y = "Life expectancy at birth",
       title = "Life expectancy and GDP",
       subtitle = "African countries with > 10 million people",
       caption = "Data collated by gapminder project")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

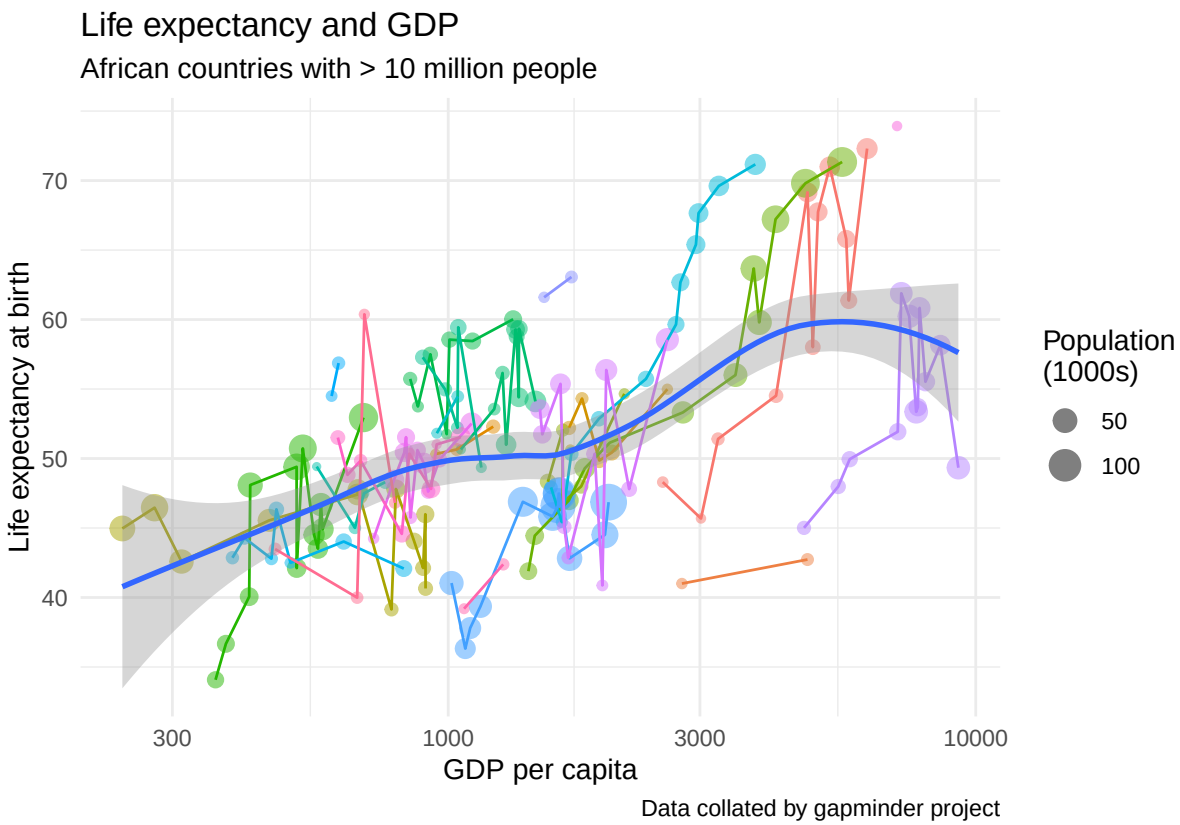


More customization with `theme()`

To have more control over how the plot turns out, you can use the `theme()` function, which has many potential arguments, see `?theme`. Or you could use some of the default themes. Type `theme_` to see some autocomplete suggestions. I'll apply `theme_minimal()`. There are many more themes available in the `ggthemes` package.

```
gapminder %>%
  filter(continent == "Africa",
         pop > 1e7) %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp)) +
  geom_point(aes(size = pop, color = country), alpha = .5) +
  geom_line(aes(group = country, color = country)) +
  geom_smooth() +
  scale_x_log10() +
  guides(color = "none",
         size = guide_legend(title = "Population\n(1000s)")) +
  scale_size_continuous(labels = number_format(scale = 1/1e6)) +
  labs(x = "GDP per capita",
       y = "Life expectancy at birth",
       title = "Life expectancy and GDP",
       subtitle = "African countries with > 10 million people",
       caption = "Data collated by gapminder project") +
  theme_minimal()
```

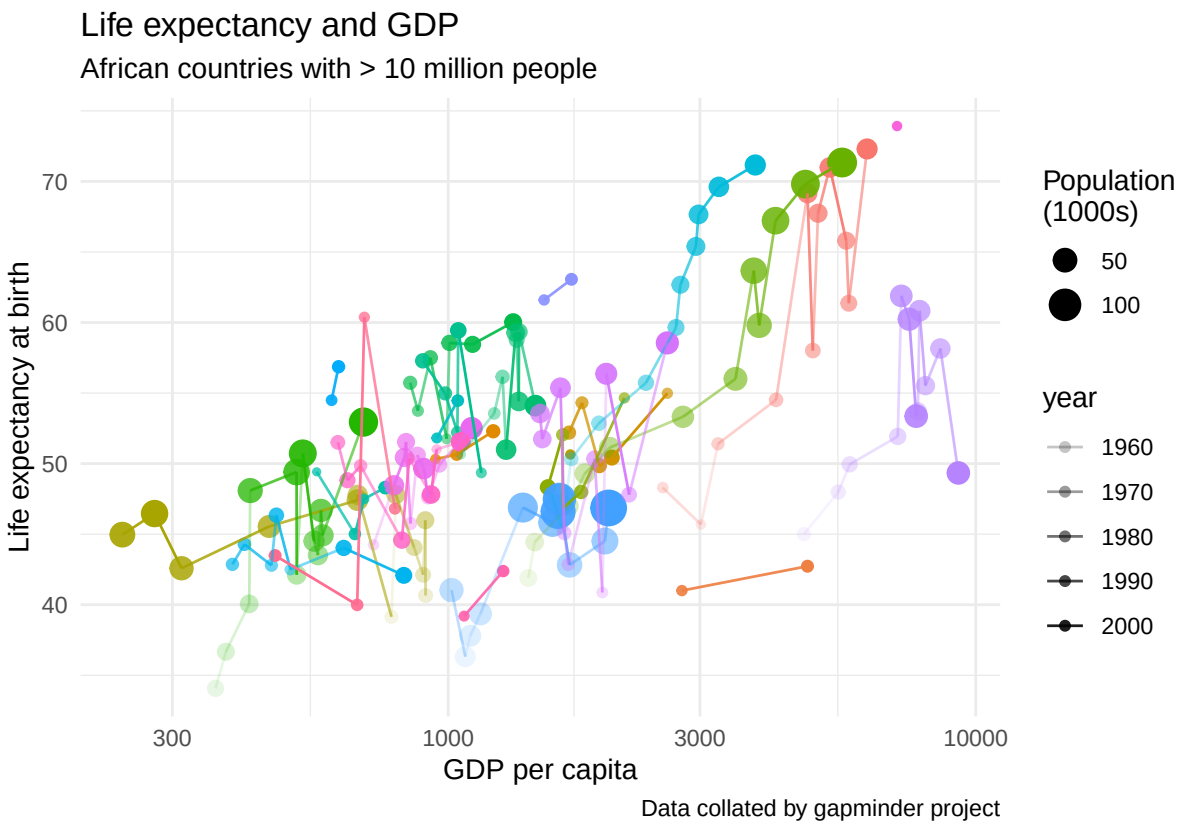
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



mapping transparency, `alpha`

Now in this plot we can detect the time trend quite well due to our priors, but the plot isn't telling us itself what the direction of movement is. Instead of setting `alpha`, we could *map* it to `year`. Here I'll do so for both the lines and the points. You see how transparency can even change along a line? I'm getting sick of that smooth trend, so now it's gone.

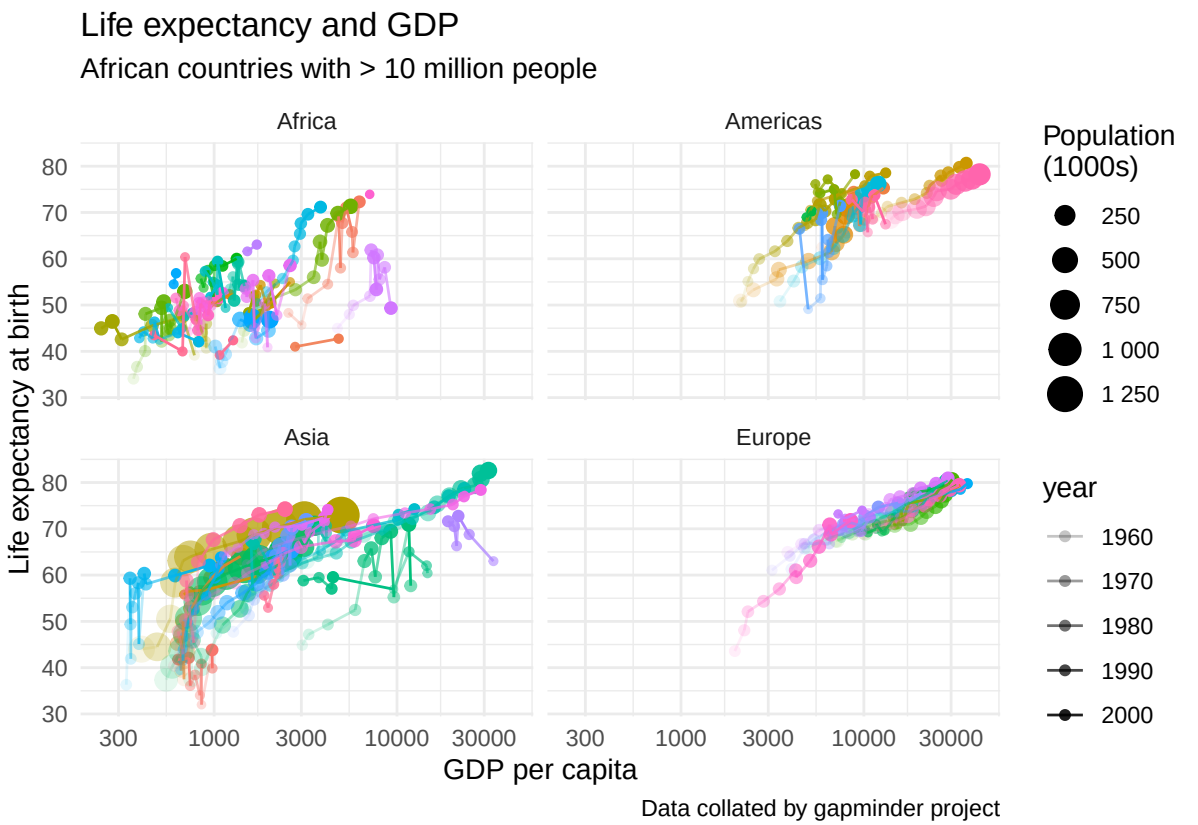
```
gapminder %>%
  filter(continent == "Africa",
         pop > 1e7) %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp)) +
  geom_point(aes(size = pop, color = country, alpha = year)) +
  geom_line(aes(group = country, color = country, alpha = year)) +
  scale_x_log10() +
  guides(color = "none",
         size = guide_legend(title = "Population\n(1000s)")) +
  scale_size_continuous(labels = number_format(scale = 1/1e6)) +
  labs(x = "GDP per capita",
       y = "Life expectancy at birth",
       title = "Life expectancy and GDP",
       subtitle = "African countries with > 10 million people",
       caption = "Data collated by gapminder project") +
  theme_minimal()
```



3.9 making panels, `facet_`

Say we're happy with this plot form and now we want to repeat it for all the large countries of the world? Then `facet_wrap()` will help, we add `facet_wrap(~continent)`. From this we see that Simpson's paradox was biggest in Africa.

```
gapminder %>%
  filter(pop > 1e7,
         continent != "Oceania") %>%
  ggplot(aes(x = gdpPercap,
             y = lifeExp)) +
  geom_point(aes(size = pop, color = country, alpha = year)) +
  geom_line(aes(group = country, color = country, alpha = year)) +
  scale_x_log10() +
  guides(color = "none",
         size = guide_legend(title = "Population\n(1000s)")) +
  scale_size_continuous(labels = number_format(scale = 1/1e6)) +
  labs(x = "GDP per capita",
       y = "Life expectancy at birth",
       title = "Life expectancy and GDP",
       subtitle = "African countries with > 10 million people",
       caption = "Data collated by gapminder project") +
  theme_minimal() +
  facet_wrap(~continent)
```



4 Further learning

`ggplot2` is a powerful and extensive graphical system, which deserves an entire workshop. I think that this introduction should serve to get you started. Further learning will happen when you look into further `geom` types, learn to control color palettes, and so forth. I suggest the Healy book Healy (2018) for a very complimentary resource for this whole course, and I also suggest the book by Claus Wilke *Fundamentals of Data Visualization* Wilke (2019) . Most of your problems can be solved using Stack Overflow https://stackoverflow.com/, where most common questions have been answered. You can also refer to the data visualization course materials from Ilya Kashnitsky https://github.com/ikashnitsky/dataviz-mpidr. Finally, always remember you can see example code for making particular kinds of plots in the help files, or with internet searches.

In my opinion the most involved aspect of making the plots is in the data prep, and that is the focus of this course. More examples will be done spontaneously or on request in the session.

References

Healy, Kieran. 2018. *Data Visualization: A Practical Introduction*. Princeton University Press.

Wickham, Hadley. 2011. "Ggplot2." *Wiley Interdisciplinary Reviews: Computational Statistics* 3 (2): 180–85.

Wilke, Claus O. 2019. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media.

Wilkinson, Leland. 2012. "The Grammar of Graphics." In *Handbook of Computational Statistics*, 375–414. Springer.