Hertie School/SCRIPTS Data Science Workshop Series

Session 3: Data Visualization with `ggplot2`

*Therese Anders**
Allison Koh†

March 6, 2020

In this first part of the workshop, we will go over basic principles of `ggplot2`. We will work with data from the `gapminder` package. First, install `gapminder` and get an overview of the data. The dataset contains information on life expectancy, GDP per capita, and population by country from 1952 to 2007 in increments of 5 years. Let's use the help function to get an overview of the data.

```
# install.packages("gapminder")
library(gapminder)
?gapminder # getting an overview
```

Start by making a copy of the original data in a data frame called `df`. Then use the `str()` function to get an overview over the variable types in the data frame. The dataframe has 1704 observations and 6 variables.

```
df <- gapminder
str(df)
```

```
## 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 ...
```

Intro to the ggplot2 package

`ggplot2` was developed by Hadley Wickham based on Leland Wilkinson's "grammar of graphics" principles. According to the "grammar of graphics," you can create each graph from the following components: "a data set, a set of geoms–visual marks that represent data points, and a coordinate system" (Wilkinson 2012). You can access the data visualization with `ggplot2` cheat sheet here.

For most applications, the code to produce a graph in `ggplot2` is roughly structured as follows:

```
ggplot(data = , aes(x = , y = , color = , linetype = )) +

geom() +

[other graphical parameters, e.g. title, color schemes, background]
```

- `ggplot()`: Function to initiate a graph in `ggplot2`.
- `data`: Specifies the data frame from which the plot is produced.
- `aes()`: Specifies aesthetic mappings that describe how variables are mapped to the visual properties of the graph. The minimum value that needs to be specified (for univariate data visualization) is the `x` parameter, where `x` specifies the variable to be plotted on the x-axis. Analogously, the `y` parameter specifies the variable to be plotted on the y-axis. Other examples include the `color` parameter, which specifies the variable to be onto different colors, or the `linetype` parameter, which specifies the variable to be mapped onto different line types in case of line graphs.

*Instructor, Hertie School/SCRIPTS, anders@hertie-school.org.
†Teaching Assistant, Hertie School, kohallison3@gmail.com.

- `geom()`: Specifies the type of plot to use. There are many different geoms ("geometric objects") to be specified with the `geom()` layer. Some of the most common ones include `geom_point()` for scatterplots, `geom_line()` for line graphs, `geom_boxplot()` for Boxplots, `geom_bar()` for bar plots for discrete data, and `geom_histogram()` for continuous data.

For an overview of the most important functions and geoms available through `ggplot2`, see the `ggplot2` cheat sheet.

`ggplot2` is part of the `tidyverse` collection of `R` packages. You can load the entire collection by downloading the `tidyverse` package and loading it using the `library(tidyverse)` command, but for this workshop we will be downloading and calling each package separately.

```
# install.packages("ggplot2")
library(ggplot2)
```

Illustrating distributions

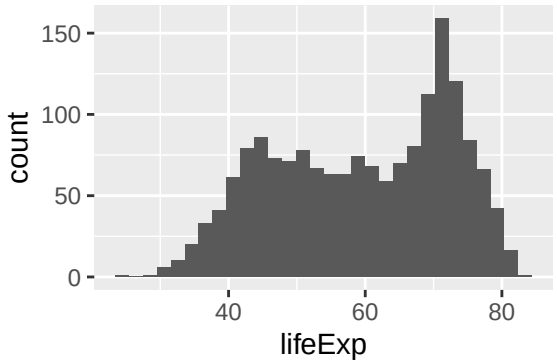
Histograms

Histograms graph the distribution of continuous variables. In this first example, we graph the distribution of the life expectancy variable (i.e. `lifeExp`).

```
summary(df$lifeExp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   23.60   48.20   60.71   59.47   70.85   82.60
```

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_histogram()
```



Question 1 Which conclusions do you draw from the histogram above about the distribution of life expectancy in the world?

The distribution is not normal (i.e. not a bell curve). It is bimodal with a skew to the left. There is a cluster of country-year observations that has a lower life expectancy (approximately 45-60 years), and a cluster of countries with much higher life expectancies (approx 70 years).

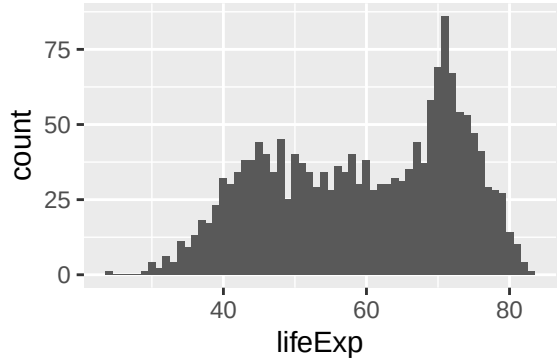
Adjusting the number of bins

The default number of bins is 30, which means that the entire range of the variable (here 23.60 to 82.60) is split into 30 equally spaced bins. We can change the number of bins manually. Below, we specify 60 bins to approximate a binwidth of 1 year, taking into account the range of the variable `lifeExp`.

```
min(df$lifeExp) - max(df$lifeExp) # approx 60 years
```

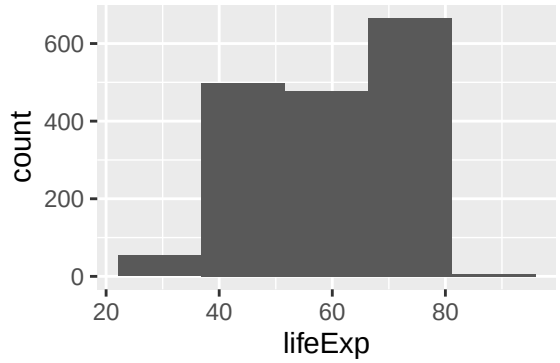
```
## [1] -59.004
```

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_histogram(bins = 60)
```



What if we specified just 5 bins?

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_histogram(bins = 5)
```

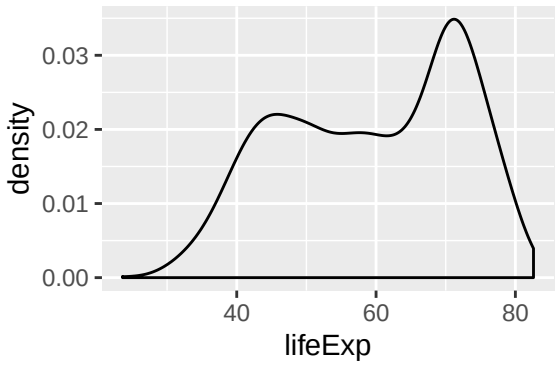


Density plots

We saw that the shape of the distribution is highly influenced by *how many bins* we specify. If we specify too few bins, we run the risk of masking a lot of variation within the bins. If we specify too many bins, we trade parsimony for detail – which might make it harder to draw conclusions about the overall distribution of the variable of interest from the graph.

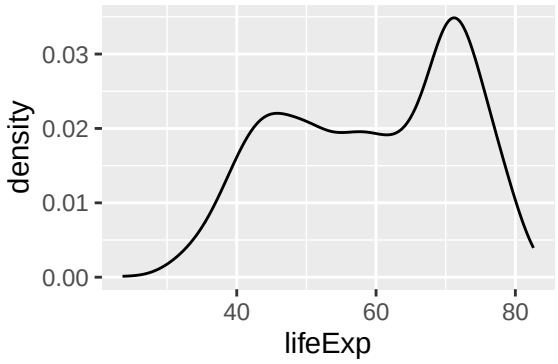
Density plots are continuous alternatives to histograms that do not rely on bins. We will not cover details about the mechanics behind density plots and their estimation here. Just know that we can interpret the height of the density curve in a similar way that we interpreted the height of the bars in a histogram: The higher the curve, the more observations we have at that specific value of the variable of interest. In this first example, we use the `geom_density()` function to create the density plot.

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_density()
```



If you do not want the density graph to be plotted as a closed polygon, you can instead use the `geom_line()` geometric object function with the `stat = "density"` parameter.

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density")
```

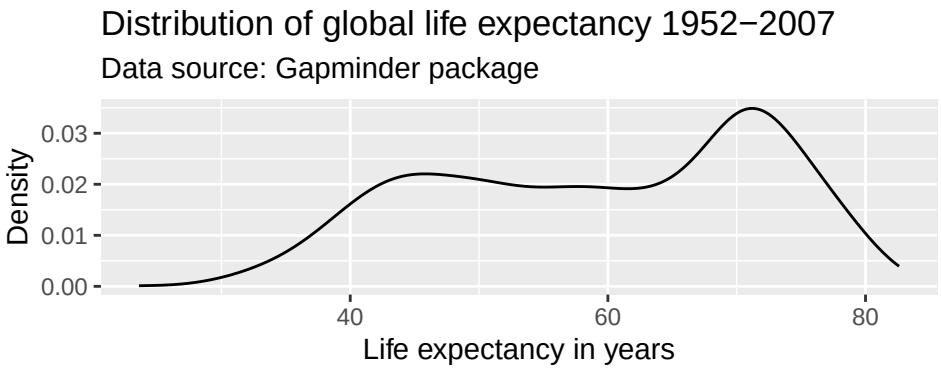


Controlling the appearance of graphs

The default graphs we have produced so far are not (yet) ready for publication. In particular, they lack informative labels. In addition, we might want to change the appearance of the graph in terms of size, color, linetype, etc.

Adding title, subtitle, and axes titles

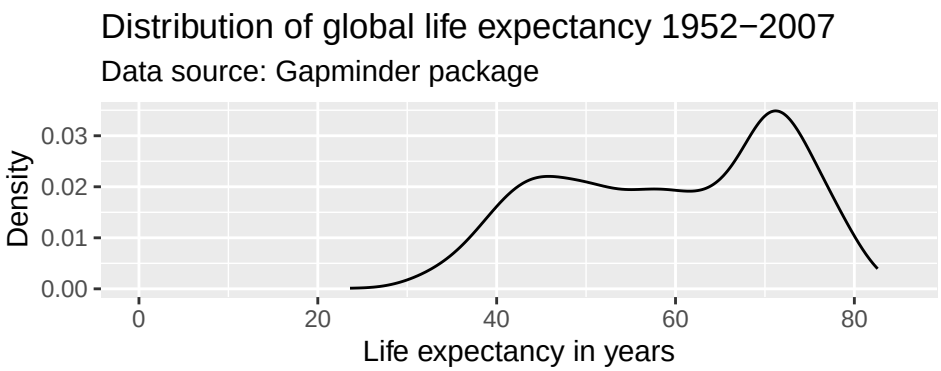
```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density")
```



Adjusting the range of the axes

By default, `ggplot()` adjusted the x-axis to start not at zero but at approximately 23 to reduce the amount of empty space in the plot. We can manually adjust the range of the axes using the `coord_cartesian()` parameter.

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  coord_cartesian(xlim = c(0, 85))
```



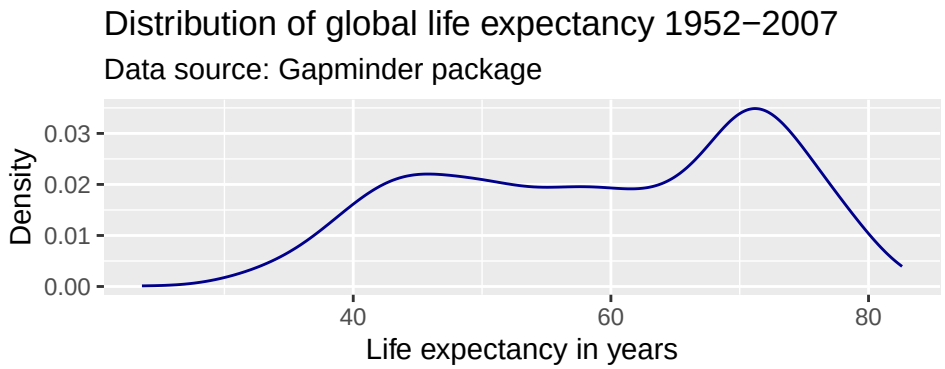
Caution!! You will sometimes see the command `scale_y_continuous(limits = c(0, 85))` instead of `coord_cartesian(ylim = c(0, 85))`. Note that these are not the same. `coord_cartesian()` only adjusts the range of the axes (it "zooms" in and out), while `scale_y_continuous(limits = c())` subsets the data. For density plots, this does not make a difference. But there are other examples where it alters the actual shape of the graph, rather than just the part of the graph that is visible.

Changing the color

Any changes to the appearance of the curve itself are made within the argument that specifies the geometric object to be plotted, here `geom_line()`. `R` knows many colors by name; for a great overview see http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf.

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "darkblue") +
```

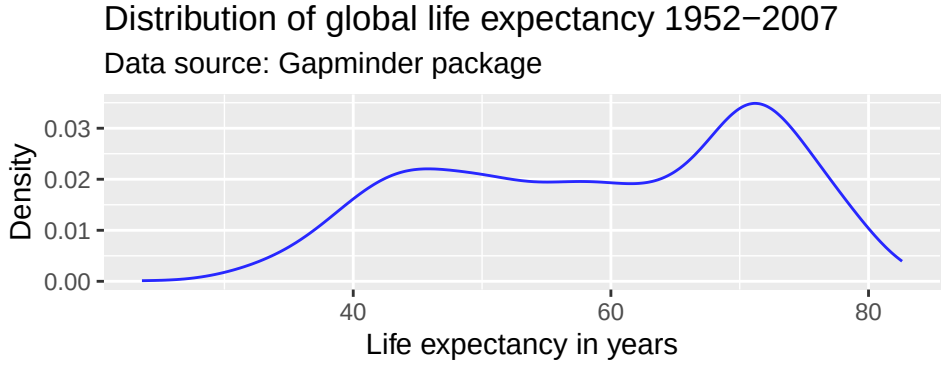
```
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density")
```



We can also use hexadecimal or RGB (red, green, blue) strings to specify colors. There are plenty of online tools to pick colors and extract hexadecimal or RBG strings. One of my favorites is http://www.colorhexa.com. This online tool allows you to specify a color name, hexadecimal, or RGB string, and returns information on color schemes, complementary colors, as well as alternative shades, tints, and tones. It also offers a color blindness simulator.

Suppose, I like the general tone of the darkblue color above, but am worried that it is a bit too dark for my plot. I enter the color "darkblue" into the search field at http://www.colorhexa.com and look for a brighter alternative. Suppose I really like the color displayed in the second tile from the left on the tints scale. I can extract this color's hexadecimal value of `#2727ff` by hovering over the tile of that color.

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "#2727ff") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density")
```

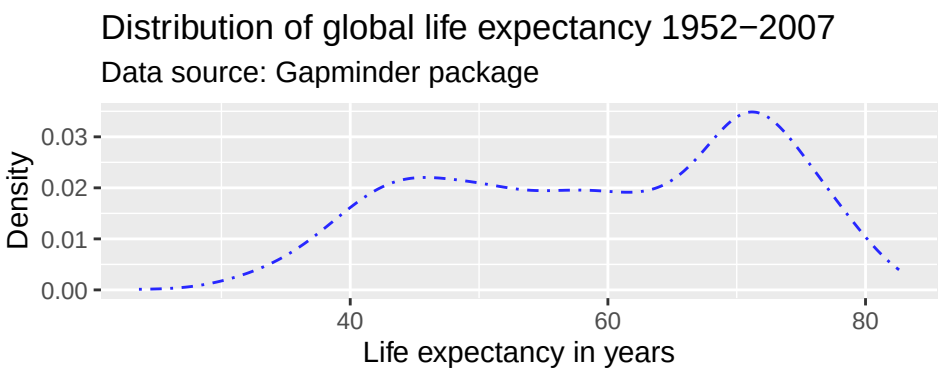


We will talk more about color schemes later in the workshop.

Changing the line type

We can adjust the type of the line via the `linetype` parameter within `geom_line()`. For an overview of line types see http://sape.inf.usi.ch/quick-reference/ggplot2/linetype.

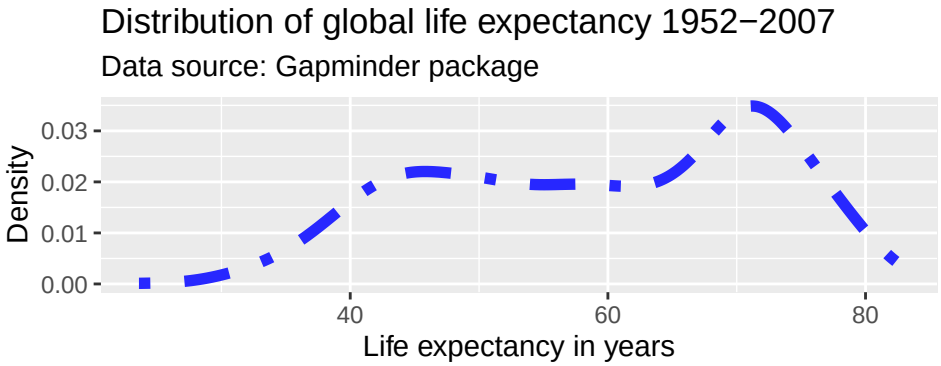
```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "#2727ff",
            linetype = "dotdash") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density")
```



Changing width and opacity of the line (not shown in class)

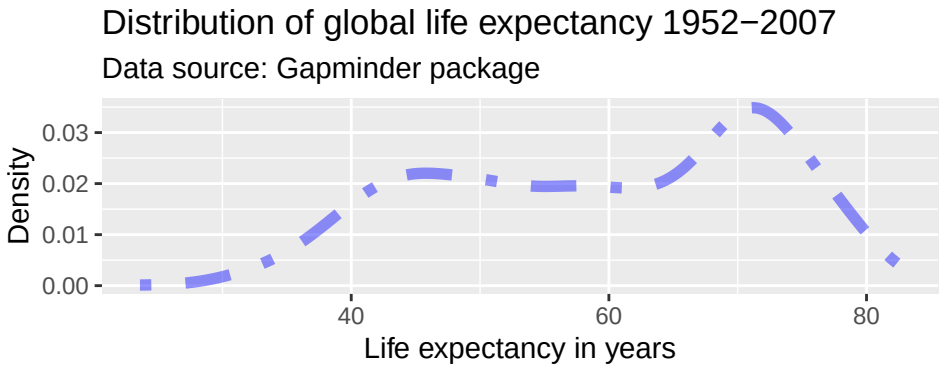
We can adjust the *width* of the line via the `size` parameter within `geom_line()`. Note that the `size` parameter is universal in the way that it controls line width in line plots and point size in scatter plots.

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "#2727ff",
            linetype = "dotdash",
            size = 2) +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density")
```



We can adjust the *opacity* of the line via the `alpha` parameter within any geometric object. The `alpha` parameter ranges between zero and one. Adjusting the opacity of the geometric objects is especially important when plotting multiple lines (or objects) in the same graph to reduce overplotting.

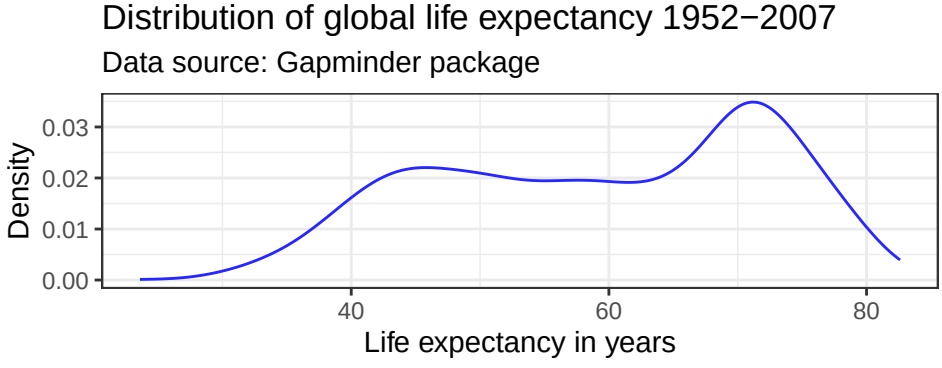
```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "#2727ff",
            linetype = "dotdash",
            size = 2,
            alpha = 0.5) +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density")
```



Themes

We can alter the appearance of any element in the plot. Below, we change the pre-specified `theme` that `ggplot2` uses to determine the appearance of the plot. Popular options are `theme_bw()` or `theme_light()`. For a full list of themes, see https://ggplot2.tidyverse.org/reference/ggtheme.html.

```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "#2727ff") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw()
```



Graphing distributions across groups

Using different colors

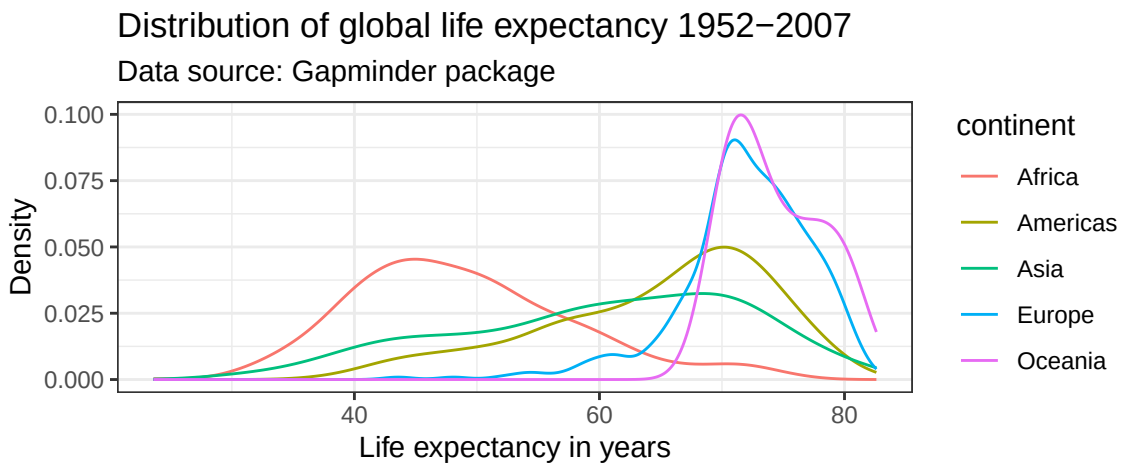
Sometimes, we want to compare distributions across different groups in our data set. Suppose, we wanted to assess the distribution of the life expectancy on different continents. We can use the `table()` function to get an overview over the groups in our data.

```
table(df$continent)
```

```
##
##   Africa Americas     Asia   Europe  Oceania
##      624      300      396      360       24
```

We pass a separate color to the distribution of the `lifeExp` for each continent by specifying the `color` parameter within the aesthetics. Remember, to remove the `color` parameter from the `geom_line()` function. The ability to pass a second variable to the graph with just one aesthetic (here: color) is where the true power of `ggplot2` for data visualization lies.

```
ggplot(df,
       aes(x = lifeExp,
           color = continent)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw()
```



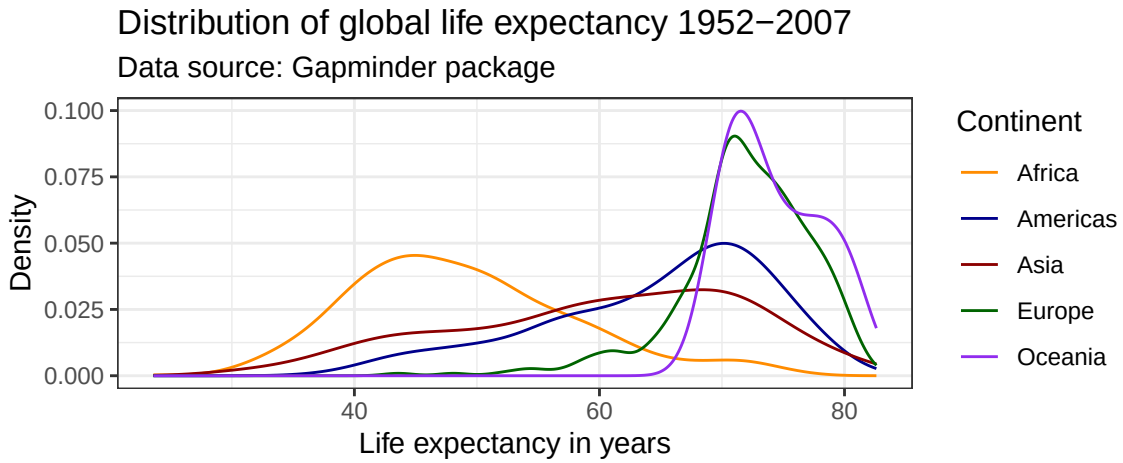
Question 3 What is the difference between specifying the `color` parameter outside the `aes()` argument versus within the `aes()` argument?

If the color parameter is specified outside the `aes()` argument, one color is passed all geometric objects of the same type. If the color parameter is specified within the `aes()` argument, different colors are passed to each value of the variable that is passed to the `color` parameter. A separate geometric object will be plotted for value–each in a different color.

We can adjust the colors used in the plot in a variety of ways. Below, we first use the `scale_color_manual()` function. This will change the colors in both the plot and the legend, based on our manual specification. Within the `scale_color_manual()` argument, we can also specify a name and labels for the legend.

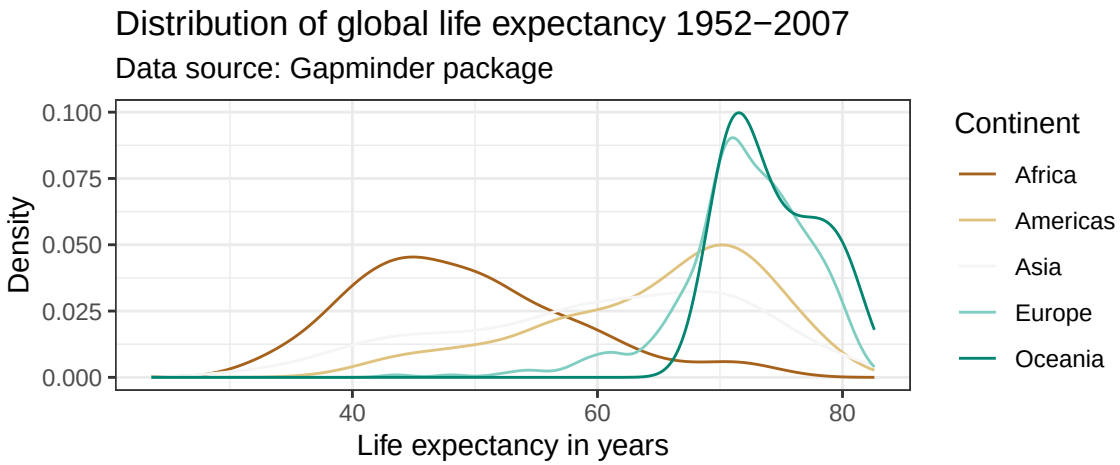
```
ggplot(df,
       aes(x = lifeExp,
```

```
          color = continent)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw() +
  scale_color_manual(values = c("Africa" = "darkorange",
                                "Americas" = "darkblue",
                                "Europe" = "darkgreen",
                                "Asia" = "darkred",
                                "Oceania" = "purple2"),
                     name = "Continent")
```



There are a ton of resources and packages with pre-defined color schemes. The most popular is www.colorbrewer2.org. You can either pick the desired colors manually, or use the `scale_color_brewer()` function in `ggplot2()`.

```
ggplot(df,
       aes(x = lifeExp,
           color = continent)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw() +
  scale_color_brewer(palette = "BrBG",
                     name = "Continent")
```

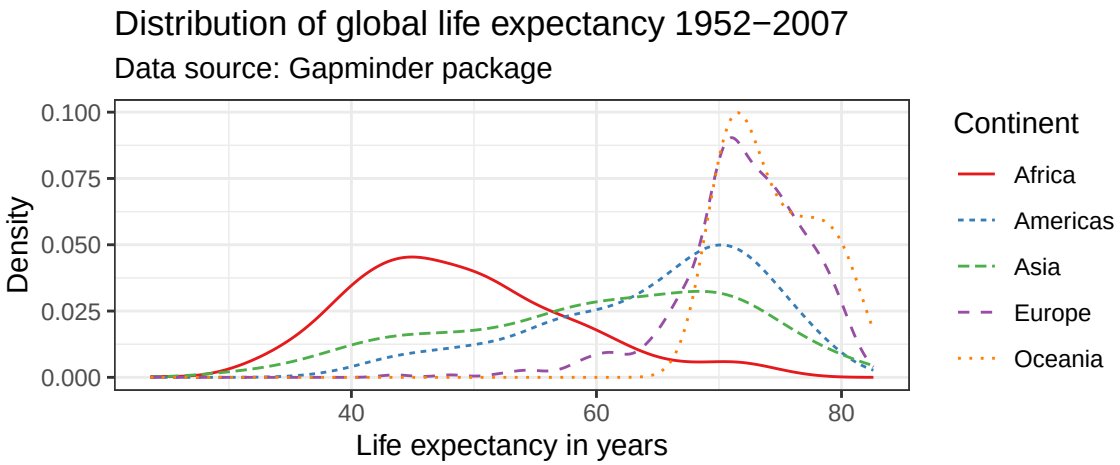


Check out the list of color palettes compiled by Emil Hvitfeldt. There is even a Wes Anderson movies inspired color scheme available using the package wesanderson! Another popular option are the color schemes from the viridis package due to their desirable properties with respect to colorblindness and printability.

Using different linetypes

Many academic journals will only accept graphs on a gray scale. This means that color will not be enough to differentiate five lines. We can use different line types instead by specifying the `linetype` parameter within the `aes()` argument. This also makes the graph more color blind friendly. Notice below that in order to combine the legends for the `linetype` and `color` aesthetics, we need to pass the same name within the `scale` function.

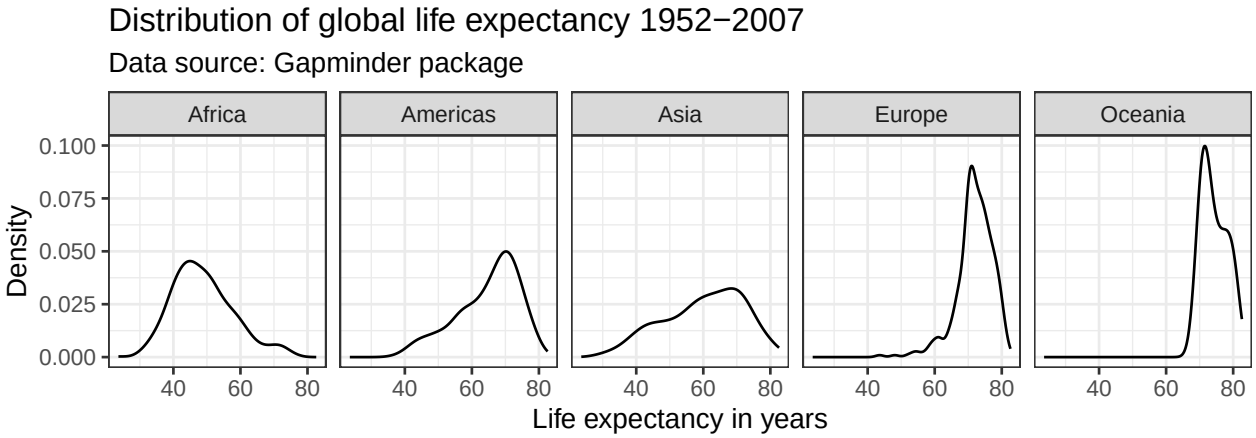
```
ggplot(df,
       aes(x = lifeExp,
           color = continent,
           linetype = continent)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw() +
  scale_color_brewer(palette = "Set1",
                     name = "Continent") +
  scale_linetype_discrete(name = "Continent")
```



Faceting

Another option to graph different groups is to use faceting. This means to plot each value of the variable upon which we facet in a different panel within the same plot. Here, we will use the `facet_wrap()` function.

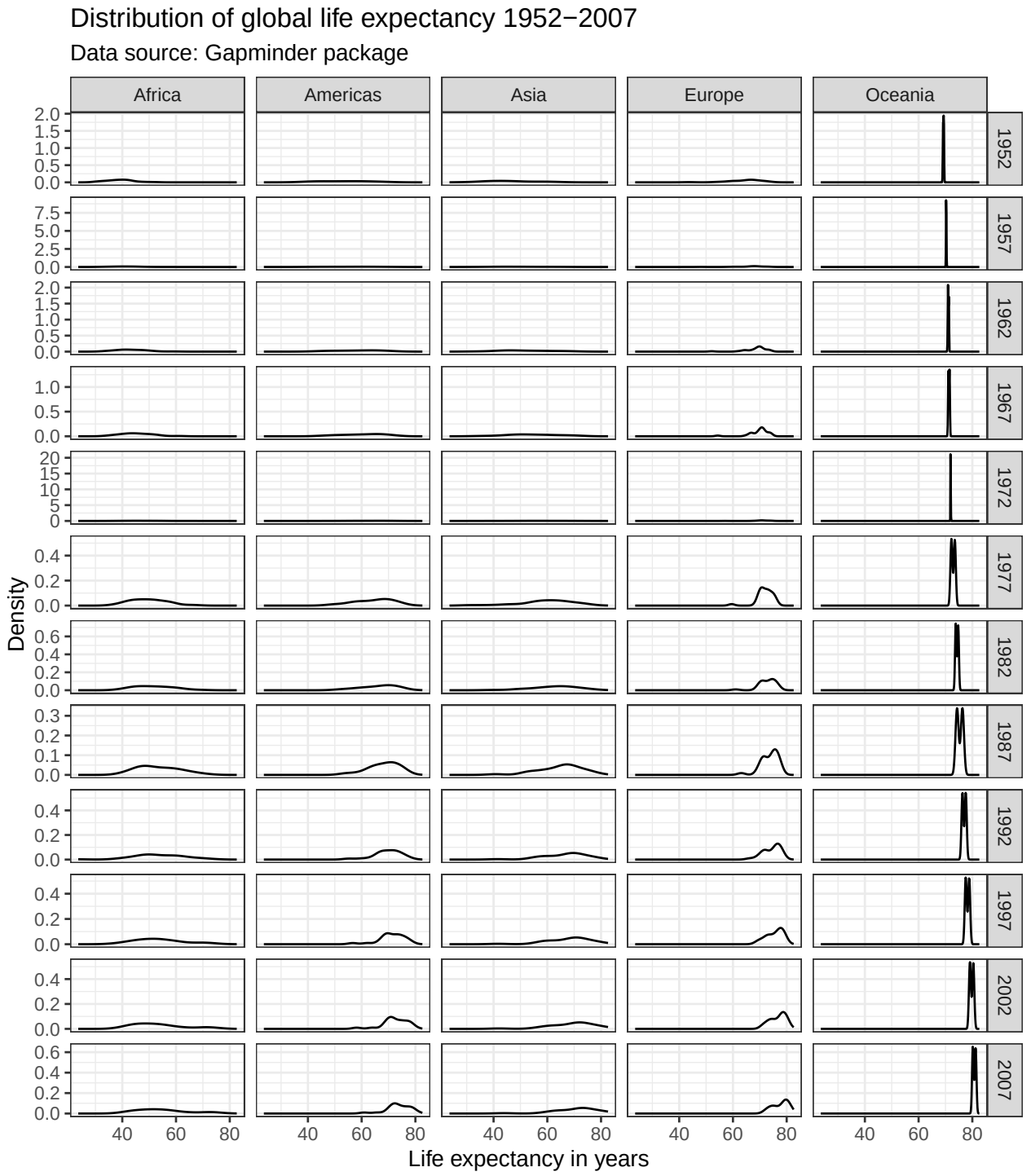
```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw() +
  facet_wrap(~ continent, nrow = 1)
```



We can use the `facet_grid()` to create facets across more than one variable. Suppose, we were interested in the evolution of the distribution of the life expectancy over time for each continent.

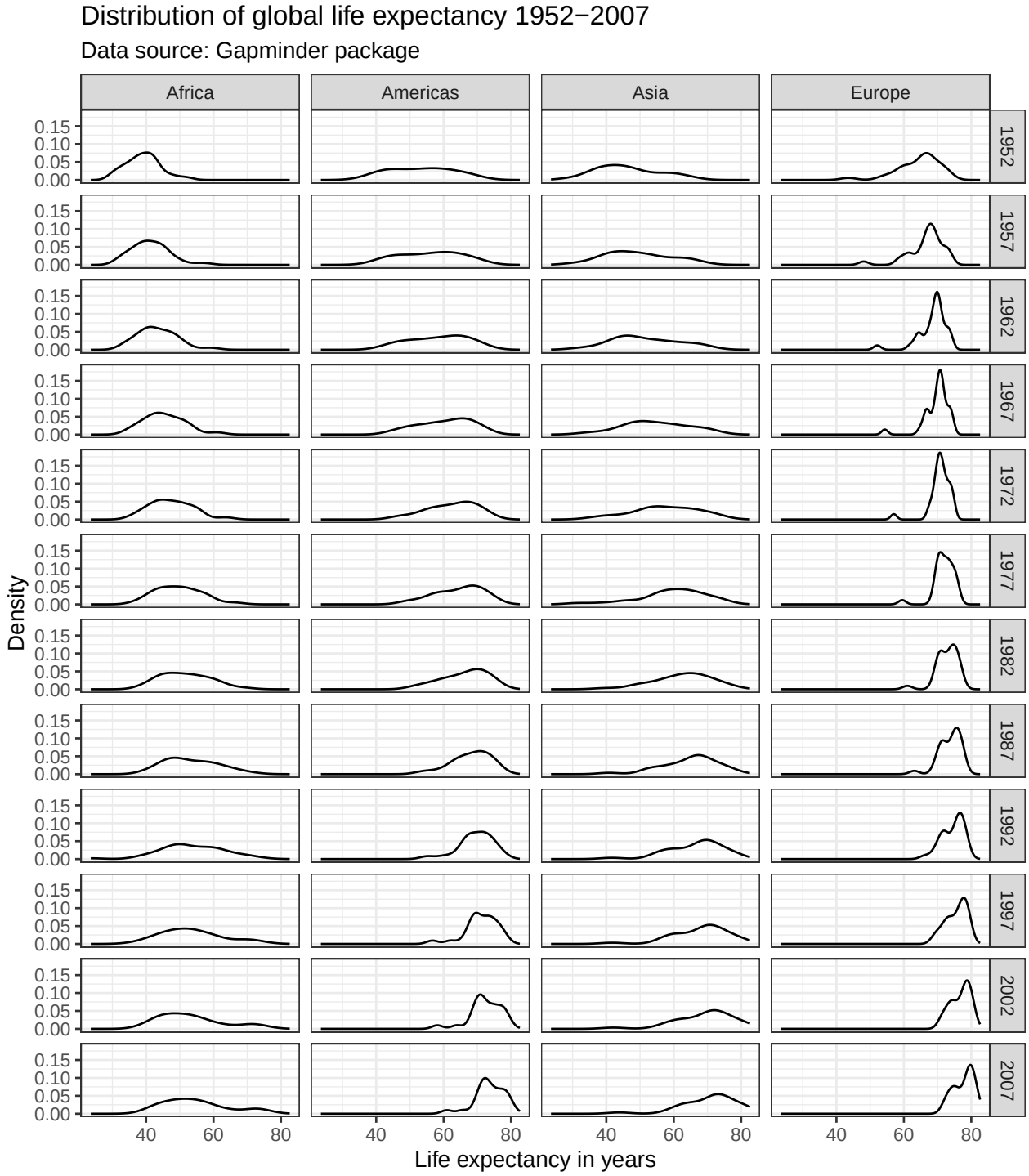
```
ggplot(subset(df),
       aes(x = lifeExp)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw() +
```

```
facet_grid(year ~ continent, scales = "free_y")
```



Oceania causing the y-axis to have a large range, which makes the values for the other continents hard to see. There are different ways to deal with this (hint: check out the `scales = "free"` command). Below, we simply exclude Oceania, since it is only comprised of Australia and New Zealand. We can either create a new subsample data frame, or use the `subset()` command directly within `ggplot()`.

```r
ggplot(subset(df, continent != "Oceania"),
       aes(x = lifeExp)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw() +
  facet_grid(year ~ continent)
```



Saving plots

We can output your plots to many different format using the `ggsave()` function, including but not limited to `.pdf`, `.jpeg`, `.bmp`, `.tiff`, or `.eps`. Here, we output the graph as a Portable Network Graphics (.png) file. We can specify the size of the output graph as well as the resolution in dots per inch (dpi). If no graph is specified, `ggsave()` will save the last graph that was executed. For us, this is the boxplot in horizontal orientation. If we no not specify the complete file path, the plot will be saved to your working directory.

```
# ggsave("panel_lifeexp_continent.png", width = 6, height = 3, dpi = 400)
```

Alternatively, we could save the plot as an `R` object and pass the object name to `ggsave()`.

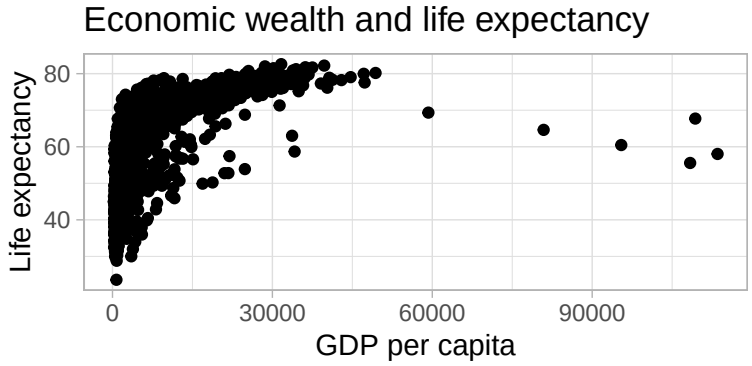
```
p1 <- ggplot(df,
       aes(x = lifeExp)) +
  geom_density()
#p1
#ggsave("lifeexp_dens.png", width = 3, height = 2, dpi = 300, p1)
```

Showing relationships in data

Scatter plots

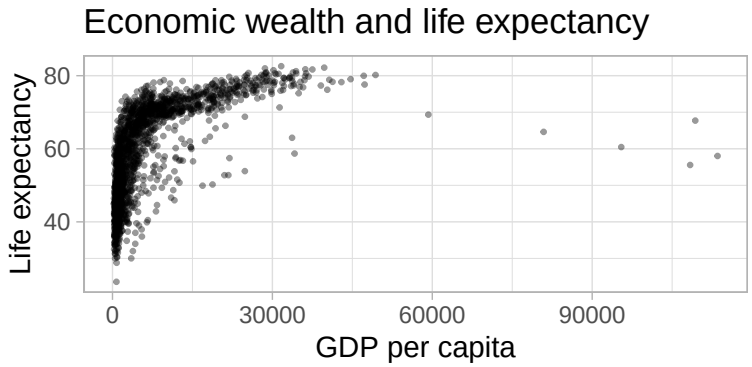
In their basic form, scatter plots are used to display values of two variables on a Cartesian coordinate system. Below, we inspect the relationship between GDP per capita and life expectancy.

```
ggplot(df,
       aes(x = gdpPercap,
           y = lifeExp)) +
  geom_point() +
  labs(title = "Economic wealth and life expectancy",
       x = "GDP per capita",
       y = "Life expectancy") +
  theme_light()
```



The plot above shows a large amount of clustering (and overplotting) on the left side of the plot, while the right side of the plot is sparsely populated with data. This makes it hard to gauge the relationship between the two variables. Below, we make a number of adjustments to the graph to better display the relationship.

```
ggplot(df,
       aes(x = gdpPercap,
           y = lifeExp)) +
  geom_point(alpha = 0.4,
             size = 0.5) +
  labs(title = "Economic wealth and life expectancy",
       x = "GDP per capita",
       y = "Life expectancy") +
  theme_light()
```



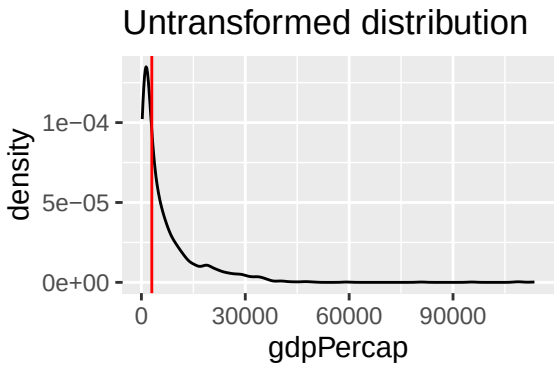
Scaling the data

One reason why the plot above is hard to read is rooted in the shape of the distribution of the GDP per capita variable. GDP per capita has a strong right skew. Below I am plotting the average on top of the graph using the `geom_vline()`.

```
av = mean(df$gdpPercap)
av
```

```
## [1] 7215.327
```

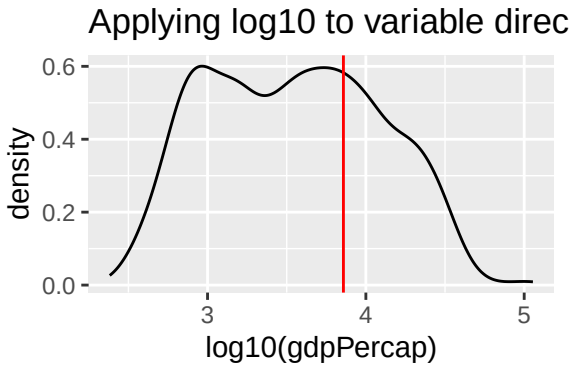
```
ggplot(df,
       aes(x = gdpPercap)) +
  geom_line(stat = "density") +
  labs(title = "Untransformed distribution") +
  geom_vline(xintercept = 3000, color = "red")
```



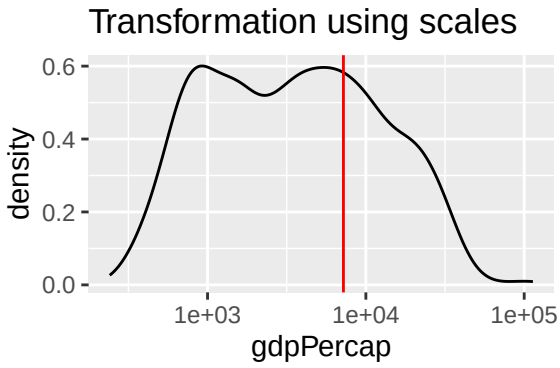
We can correct for this skew and transform the variable to have a more "normal" distribution by taking the logarithm with base 10. There are multiple ways to do this.

1. Create a new variable [not shown below]
2. Take the natural logarithm within the `aes()` statement when specifying the variable to be displayed.
3. Using (scales)[https://ggplot2.tidyverse.org/reference/scale_continuous.html] to transform the display. Note that the data is transformed before properties such as the range of the axis are determined.

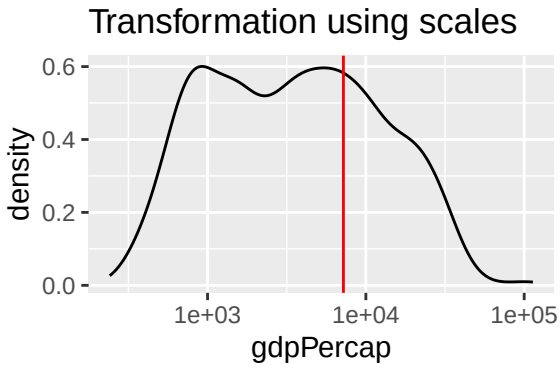
```
ggplot(df,
       aes(x = log10(gdpPercap))) +
  geom_line(stat = "density") +
  labs(title = "Applying log10 to variable directly") +
  geom_vline(xintercept = log10(av), color = "red")
```




```
# Note below that we do NOT need to specify the av in terms of log10
# The entire x-axis is transformed
ggplot(df,
       aes(x = gdpPercap)) +
  geom_line(stat = "density") +
  labs(title = "Transformation using scales") +
  scale_x_log10() +
  geom_vline(xintercept = av, color = "red")
```




```
# Bonus: alternatively could also use scale_x_continuous(trans = "log10")
ggplot(df,
       aes(x = gdpPercap)) +
  geom_line(stat = "density") +
  labs(title = "Transformation using scales") +
  scale_x_log10() +
  geom_vline(xintercept = av, color = "red")
```

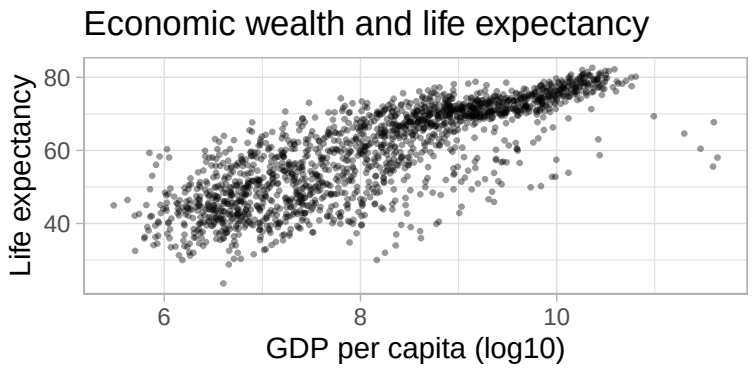


Question 4 Can you explain the differences between the plot applying the natural log to the variable within the `aes()` function versus using `scale_x_continuous()`.

Transforming the variable using the natural logarithm within `aes()` causes the x-axis to be displayed in log values. Using `scale_x_continuous()`, the data is transformed in the same way, however, the x-axis is displayed in the original, non-logged version.

We can use the same principle in bivariate (or multivariate) displays of data. Below, I use the `scale` transformation on the variable and reflect it in the axis label clarify that it is the relationship between life expectancy and the logarithm of GDP per capita that has a strong positive relationship.

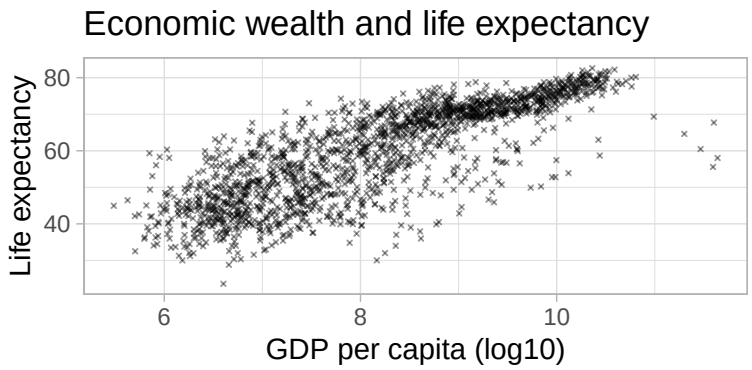
```
ggplot(df,
       aes(x = log(gdpPercap),
           y = lifeExp)) +
  geom_point(alpha = 0.4,
             size = 0.5) +
  labs(title = "Economic wealth and life expectancy",
       x = "GDP per capita (log10)",
       y = "Life expectancy") +
  theme_light()
```



Shape

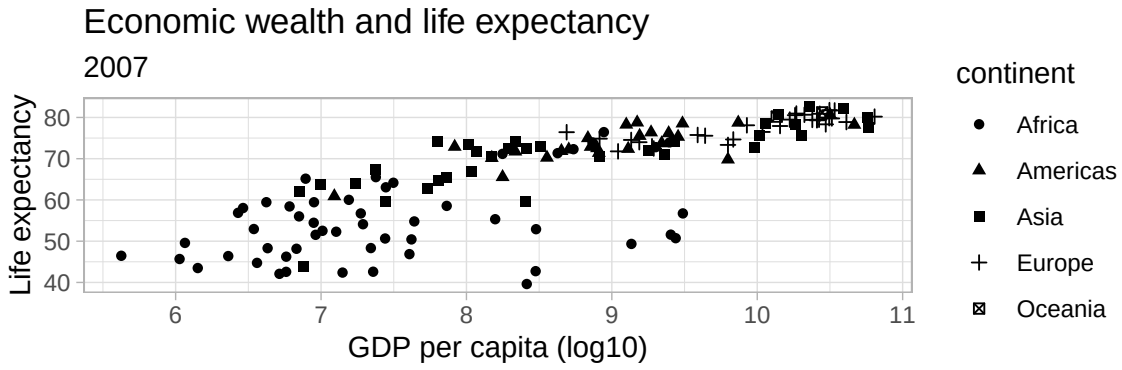
We can adjust the default symbol used by `ggplot2` to display the points. The parameter is called `shape`.

```
ggplot(df,
       aes(x = log(gdpPercap),
           y = lifeExp)) +
  geom_point(alpha = 0.4,
             size = 0.5,
             shape = 4) +
  labs(title = "Economic wealth and life expectancy",
       x = "GDP per capita (log10)",
       y = "Life expectancy") +
  theme_light()
```



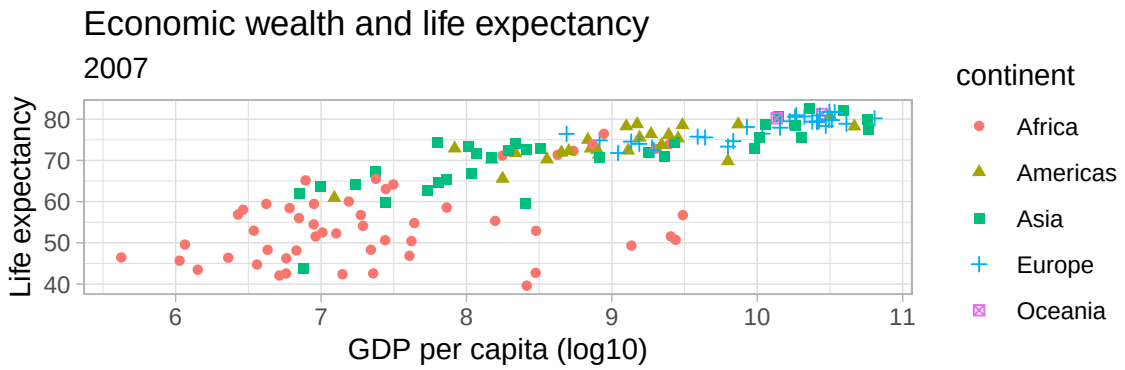
We can also have groups of data displayed using different point shapes. Below, we group by continent. We subset the data to just the year 2007 to de-clutter the plot.

```
ggplot(subset(df, year == 2007),
       aes(x = log(gdpPercap),
           y = lifeExp,
           shape = continent)) +
  geom_point() +
  labs(title = "Economic wealth and life expectancy",
       subtitle = "2007",
       x = "GDP per capita (log10)",
       y = "Life expectancy") +
  theme_light()
```



If I decided I wanted to distinguish continents by `shape` and `color`, I can add an additional aesthetics statement.

```
ggplot(subset(df, year == 2007),
       aes(x = log(gdpPercap),
           y = lifeExp,
           shape = continent,
           color = continent)) +
  geom_point() +
  labs(title = "Economic wealth and life expectancy",
       subtitle = "2007",
       x = "GDP per capita (log10)",
       y = "Life expectancy") +
  theme_light()
```



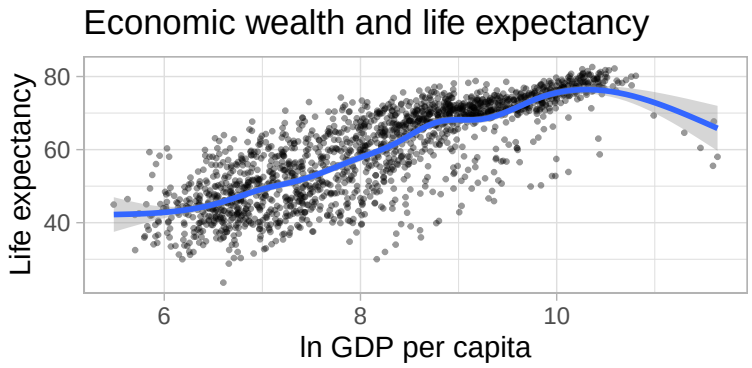
Adding trend lines

The plot above illustrates a strong positive relationship between GDP per capita and life expectancy. We can highlight the direction and strength of the relationship by adding a trend line using the `geom_smooth()` aesthetic.

The default smoothing method is `loess` for less than 1,000 observations and `gam` (Generalized Additive Models) for observations greater or equal to 1,000. `ggplot2` informs us which smoothing method was used via a message. By default, a 95% confidence interval is added to the trend line. It shows that the negative relationship at higher values of GDP per capita has a much lower precision than the positive relationship we observe for the majority of the observations.

```
ggplot(df,
       aes(x = log(gdpPercap),
           y = lifeExp)) +
  geom_point(alpha = 0.4,
             size = 0.5) +
  labs(title = "Economic wealth and life expectancy",
       x = "ln GDP per capita",
       y = "Life expectancy") +
  theme_light() +
  geom_smooth()
```

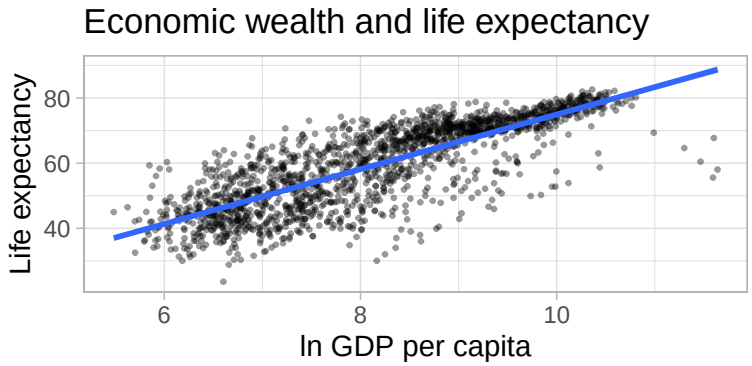
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Alternatively, we can add a linear trend line to the data.

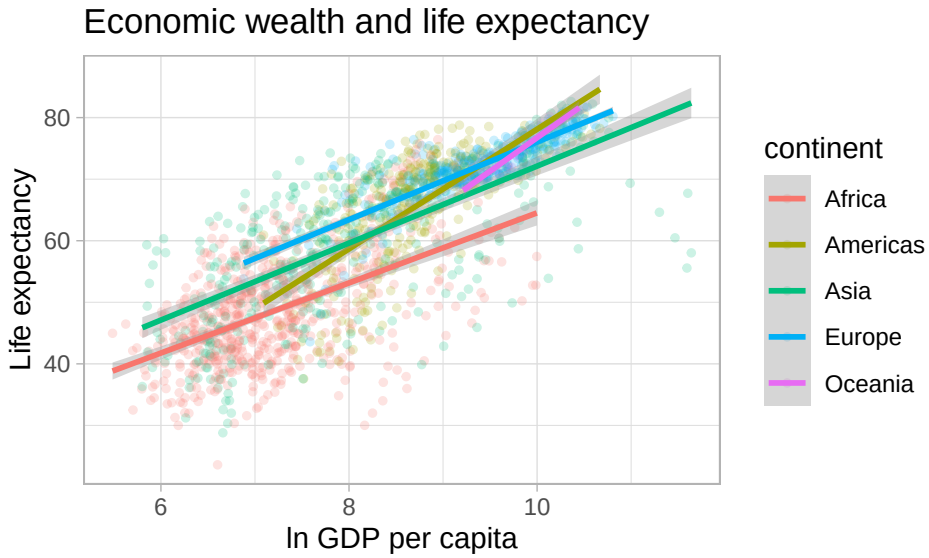
```
ggplot(df,
       aes(x = log(gdpPercap),
           y = lifeExp)) +
  geom_point(alpha = 0.4,
             size = 0.5) +
```

```
  labs(title = "Economic wealth and life expectancy",
       x = "ln GDP per capita",
       y = "Life expectancy") +
  theme_light() +
  geom_smooth(method = "lm")
```



Finally, we can display separate trendlines for groups of data. For example, suppose we wanted to know how the relationship between GDP per capita and life expectancy varies by continent. We can pass the grouping variable to the `color` (and/or `linetype`) parameter within the `aes()` function. Below, I further reduce the opacity of the points to avoid overplotting. Note that the color grouping is passed to both the `geom_point()` and the `geom_smooth()` aesthetic.

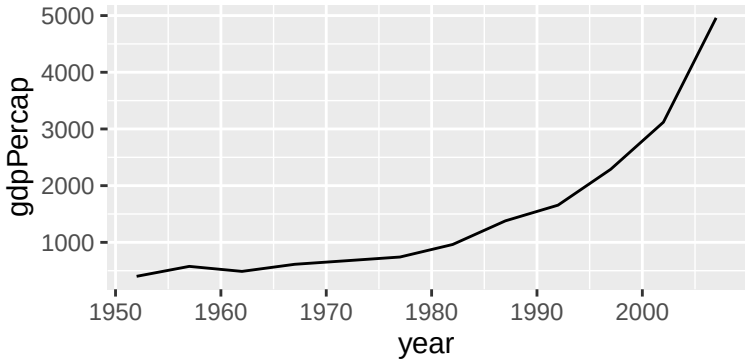
```
ggplot(df,
       aes(x = log(gdpPercap),
           y = lifeExp,
           color = continent)) +
  geom_point(alpha = 0.2,
             size = 1) +
  labs(title = "Economic wealth and life expectancy",
       x = "ln GDP per capita",
       y = "Life expectancy") +
  theme_light() +
  geom_smooth(method = "lm")
```



Line plots

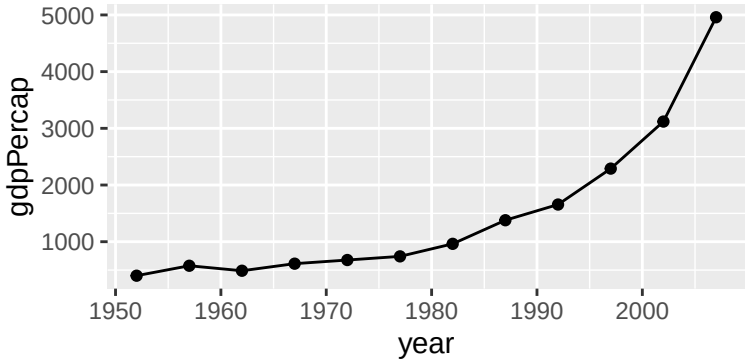
Line plots are particularly useful for time series data. Below, we will graph the GDP per capita development of China from 1952 to 2007. We select the data for China by using the `subset()` function on the original data frame.

```
ggplot(subset(df, country == "China"),
       aes(x = year,
           y = gdpPercap)) +
  geom_line()
```

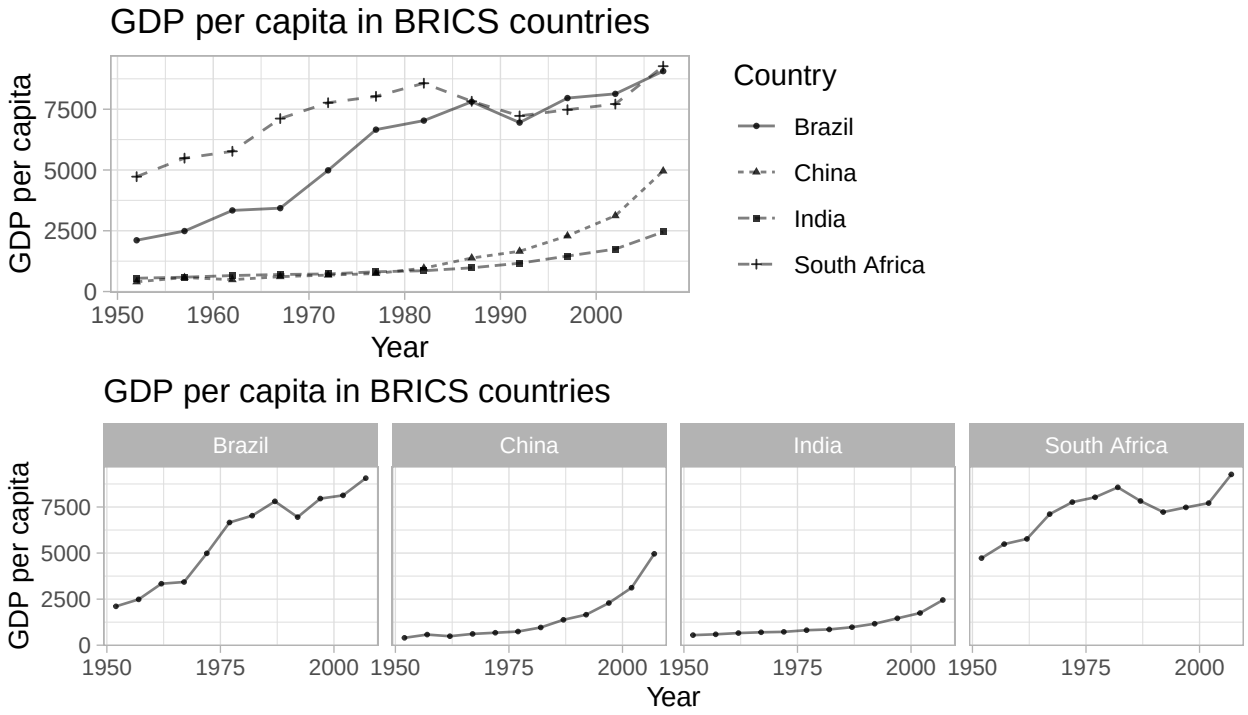


We can add points to the line to highlight which observations are available in the underlying data.

```
ggplot(subset(df, country == "China"),
       aes(x = year,
           y = gdpPercap)) +
  geom_line() +
  geom_point()
```



Practice 2 Create a plot to compare the GDP per capita development of the BRICS countries (Brazil, Russia, India, China, South Africa). Unfortunately, Russia (or previously the Soviet Union) is not part of the `gapminder` data, so we cannot display it in the plot. Please create a publication-ready graph that can be printed using grayscale.



NOTE: For advanced examples of line graphs using *spaghetti plots* please see this [GitHub page]: https://github.com/thereseanders/workshop-dataviz-fsu/tree/master/Day1#spaghetti-plots.

Heatmaps

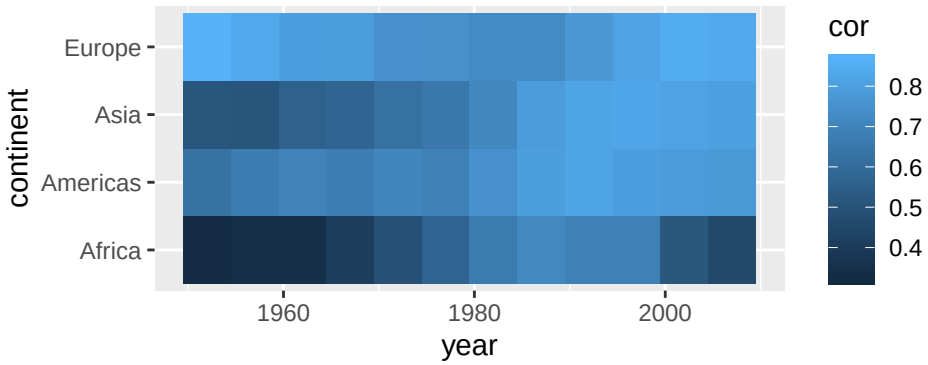
Heatmaps are another great way to illustrate trends for many different groups in data. Suppose, we were interested in the strength of the correlation between life expectancy and GDP per capita over time and space.

Below, we use our data wrangling skills from the last workshop to compute the correlation between the variables `lifeExp` and `gdpPercap` for each continent. Note that we exclude "Oceania" for this exercise.

```
# Compute Pearson correlation coefficient by year and continent
cors <- df %>%
  filter(continent != "Oceania") %>%
  group_by(continent, year) %>%
  summarise(cor = cor(lifeExp, log10(gdpPercap)))
```

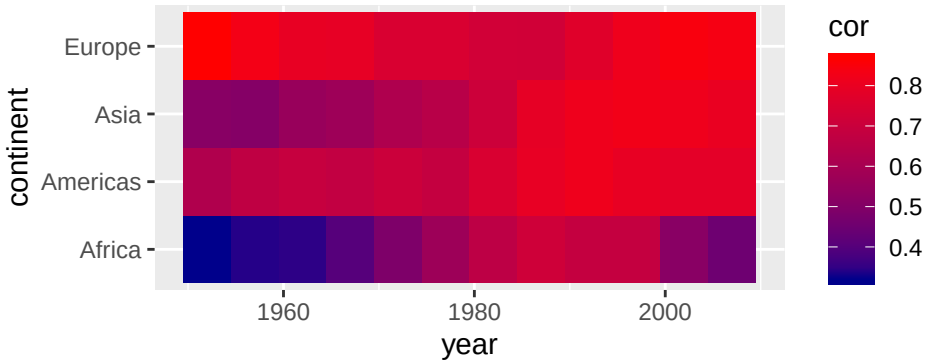
We can use the `geom_tile()` geom to create the heatmap; specifying the variable we want to display in color via the `fill` command.

```
ggplot(cors,
       aes(x = year, y = continent, fill = cor)) +
  geom_tile()
```



We can improve this plot in a number of ways. First, the color scheme is not necessarily intuitive. The colors aren't separated enough to best display smaller differences in the correlation coefficient, because they are based on the same hue. We can customize out colors to display a gradient with multiple hues.

```
ggplot(cors,
       aes(x = year, y = continent, fill = cor)) +
  geom_tile() +
  scale_fill_gradient(low = "darkblue", high = "red")
```



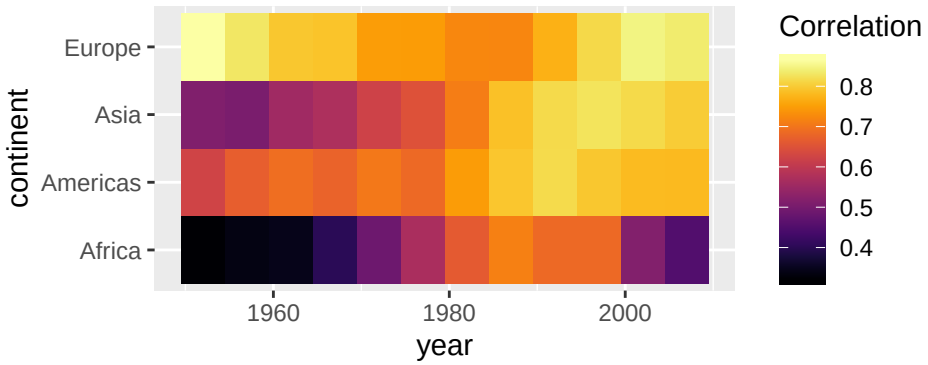
We can also use existing color gradient schemes to better distinguish values in our plot. Below, we use color scales from the viridis package. We also give the legend a more informative title.

Using color scales from the `viridis` package is a favorite among many who use `R` for data visualization. First developed for `matplotlib` in Python, this palette offers the following advantages:

- It is visually appealing
- *Perceptual uniformity*: visual perception of change is proportionate to incremental changes in the data.
- High contrast: optimal for printing in black-and-white and easier to read for people with color blindness[1]

```
ggplot(cors,
       aes(x = year, y = continent, fill = cor)) +
  geom_tile() +
  scale_fill_viridis(option = "inferno", name = "Correlation")
```

[1]For more information about the logic behind developing the viridis palette, see this blog post.

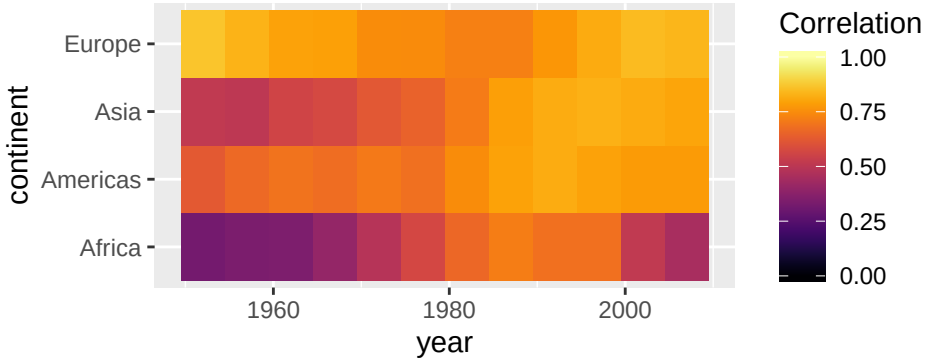


Second, we know that the correlation coefficient ranges from -1 to 1. We only have positive values here and they range from approximately 0.3 to 0.9. It is good practice to show at least one end point of the possible values in legends or axes. Therefore, below we extend the legend to display values from 0 to 1.

```
range(cors$cor)
```

```
## [1] 0.3221508 0.8649859
```

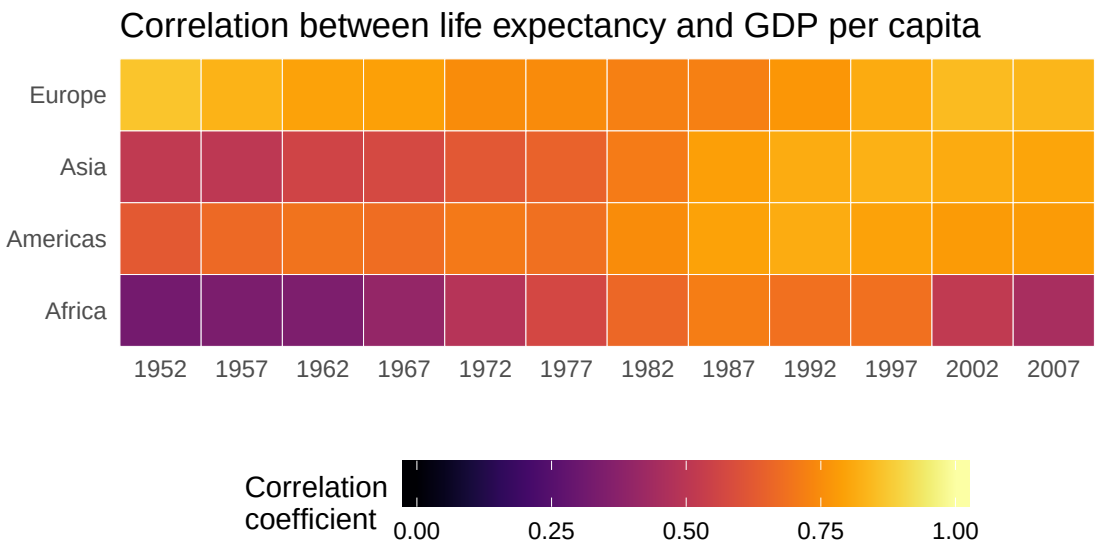
```
ggplot(cors,
       aes(x = year, y = continent, fill = cor)) +
  geom_tile() +
  scale_fill_viridis(option = "inferno", name = "Correlation",
                     limits = c(0, 1))
```



Bonus: To improve on this graph, we can add some of the other elements offered by the `ggplot2` package.

```
ggplot(cors,
       aes(x = year, y = continent, fill = cor)) +
  geom_tile(color = "white") +
  scale_fill_viridis(option = "inferno", name = "Correlation\ncoefficient",
                     limits = c(0, 1)) +
  labs(x = "",
       y = "",
       title = "Correlation between life expectancy and GDP per capita") +

  # Changing appearance of the plot
  theme_light() +
  theme(panel.grid = element_blank(),
        legend.position = "bottom",
        legend.key.width = unit(1.5, "cm"),
        panel.border=element_blank(),
        axis.ticks = element_blank()) +
```

```
  # Adjust x axis labels
  scale_x_continuous(breaks = unique(cors$year)) +

  # Reduce space between plot and labels
  coord_cartesian(expand = 0)
```



Barplots

Suppose we wanted to visualize global population growth over time. We might first want to compute the total population per continent and year.

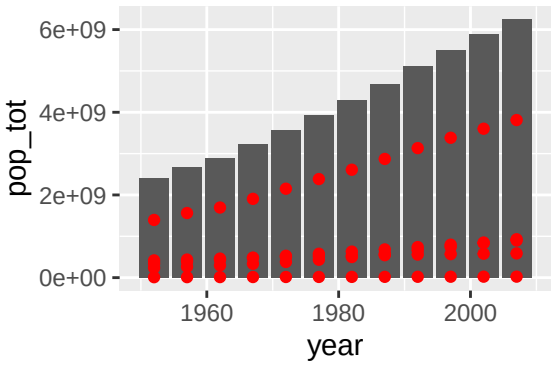
```
str(df)
```

```
## 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 ...
```

```
globalpop <- df %>%
  group_by(continent, year) %>%

  # Need to transform int to num to prevent integer overflow
  summarise(pop_tot = sum(as.numeric(pop)))
```

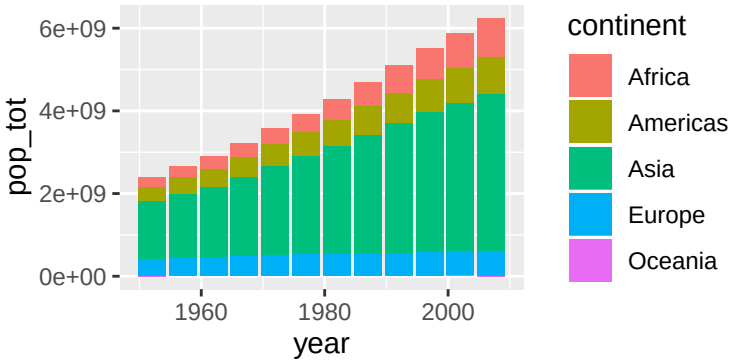
`ggplot2()` is pretty nice and it just stacked each continent's population on top of each other. This is nice because it it automatically allows us to visualize the sum across continents. Try overlaying the plot with a `geom_point()` layer to verify that the height of each bar is truly the sum of all continents' population.

```
ggplot(globalpop,
       aes(x = year, y = pop_tot)) +
  geom_col() +
  geom_point(col = "red")
```



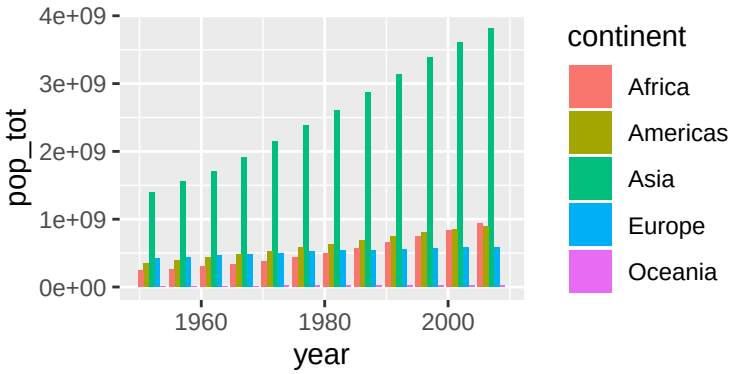
We could illustrate that these are indeed separate continents by passing a `fill` argument within `aes()`.

```
ggplot(globalpop,
       aes(x = year, y = pop_tot, fill = continent)) +
  geom_col()
```



If we instead wanted a separate bar for each continent, we can use the `position` parameter within `geom_col()`.

```
ggplot(globalpop,
       aes(x = year, y = pop_tot, fill = continent)) +
  geom_col(position = position_dodge())
```

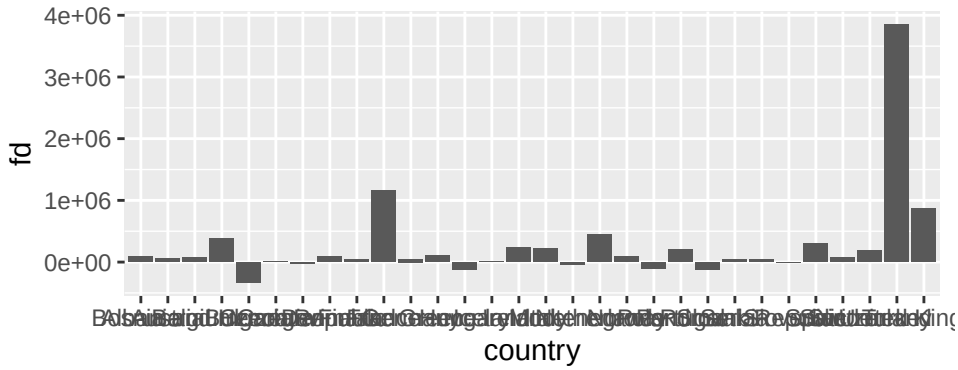


Suppose we wanted to know which countries in Europe are shrinking and which countries are growing their population. We can use our data wrangling skills to compute the first difference of population, i.e. the current value minus the previous year's value.

```
diff07 <- df %>%
  group_by(country) %>%
  arrange(year) %>%
  mutate(fd = pop - dplyr::lag(pop))
```

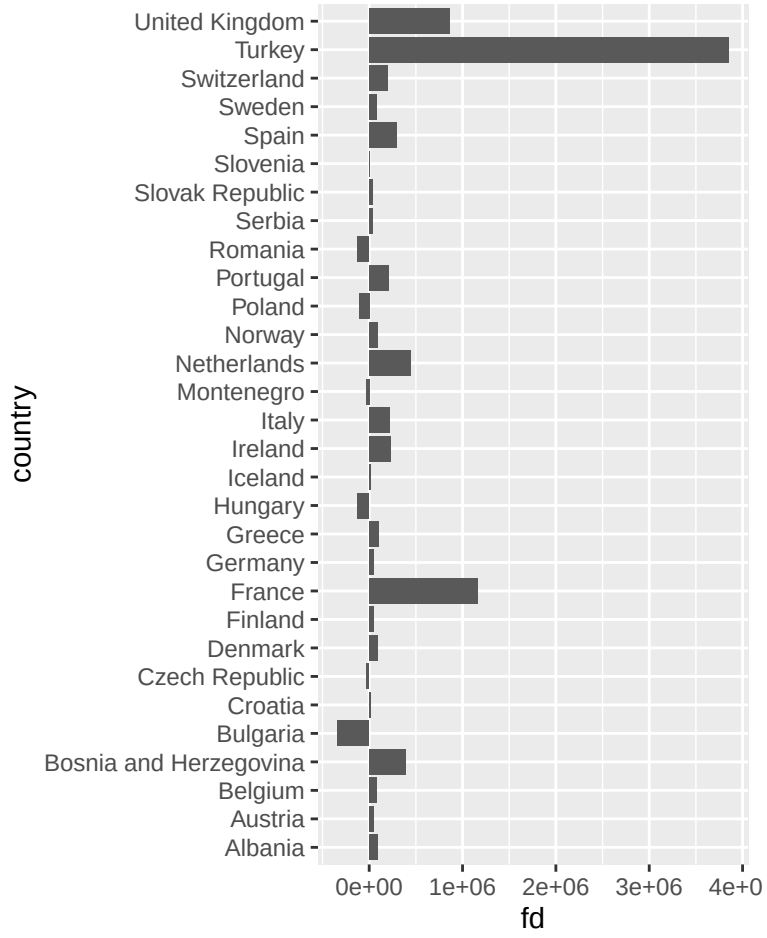
Below, we plot the first difference for European countries in 2007.

```
ggplot(subset(diff07, continent == "Europe" & year == 2007),
       aes(x = country, y = fd)) +
  geom_col()
```



This is really hard to see. Lets flip the axes using `coord_flip()`.

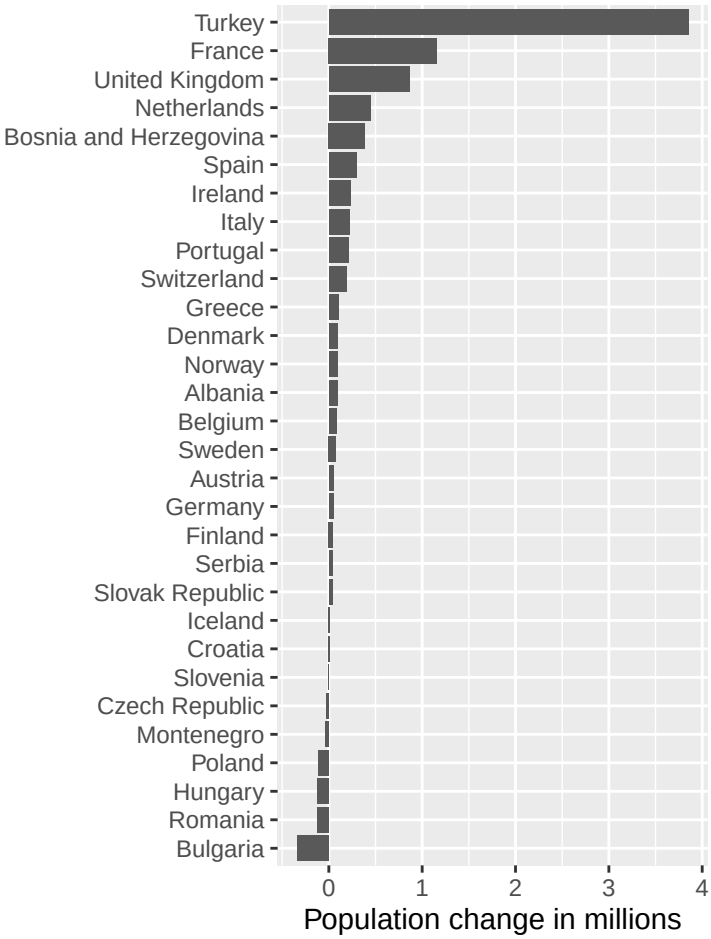
```
ggplot(subset(diff07, continent == "Europe" & year == 2007),
       aes(x = country, y = fd)) +
  geom_col() +
  coord_flip()
```



This could useful because countries are ordered alphabetically, but visually, it is is confusing. Let's reorder

the country axis based on the value of the population change. The default is ro order the points in ascending order from the origin.

```
ggplot(subset(diff07, continent == "Europe" & year == 2007),
       aes(x = reorder(country, fd), y = (fd/1e+6))) +
  geom_col() +
  coord_flip() +
  labs(x = "", y = "Population change in millions")
```



Visualizing regression results

The histograms and density plots above show us that there is substantial variation in life expectancy across countries and continents. International organizations and governments are interested in investigating these differences to get an idea of how to allocate resources for addressing related issues. Against this background, suppose we are interested in using the `gapminder` data to learn more about the determinants of a country's life expectancy.

The model specifications for this example are as follows:

- Outcome variable y: Life expectancy for $n = 142$ countries. (`log(lifeExp)`)
- Explanatory variables x:
  - Population (`log(pop)`)
  - GDP per capita (`log(gdpPercap)`)
  - Continent (`factor(continent)`, reference category: Africa)

Disclaimer: **This example is meant to illustrate how to visualize regression results—in practice, this may not be the most appropriate model for analyzing the determinants of our dependent variable.**

We use the `broom` package to convert the results of our model to a tidy data frame.

```
library(broom)

mod1 <- lm(log(lifeExp) ~ log(pop) + log(gdpPercap) + factor(continent) + factor(year), data = df)
summary(mod1)
```

```
##
## Call:
## lm(formula = log(lifeExp) ~ log(pop) + log(gdpPercap) + factor(continent) +
##     factor(year), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72697 -0.06355  0.00578  0.07150  0.27029
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                3.055604   0.039283  77.785  < 2e-16 ***
## log(pop)                   0.002876   0.001881   1.529  0.12646
## log(gdpPercap)             0.086535   0.003179  27.217  < 2e-16 ***
## factor(continent)Americas  0.165301   0.009281  17.811  < 2e-16 ***
## factor(continent)Asia      0.126959   0.008238  15.412  < 2e-16 ***
## factor(continent)Europe    0.216690   0.010316  21.005  < 2e-16 ***
## factor(continent)Oceania   0.214864   0.024932   8.618  < 2e-16 ***
## factor(year)1957           0.038561   0.013423   2.873  0.00412 **
## factor(year)1962           0.069956   0.013451   5.201 2.22e-07 ***
## factor(year)1967           0.097671   0.013503   7.233 7.12e-13 ***
## factor(year)1972           0.120608   0.013583   8.880  < 2e-16 ***
## factor(year)1977           0.145626   0.013653  10.666  < 2e-16 ***
## factor(year)1982           0.176569   0.013701  12.888  < 2e-16 ***
## factor(year)1987           0.203253   0.013734  14.800  < 2e-16 ***
## factor(year)1992           0.215759   0.013755  15.686  < 2e-16 ***
## factor(year)1997           0.222027   0.013832  16.052  < 2e-16 ***
## factor(year)2002           0.224147   0.013908  16.117  < 2e-16 ***
## factor(year)2007           0.231250   0.014058  16.450  < 2e-16 ***
## ---
```

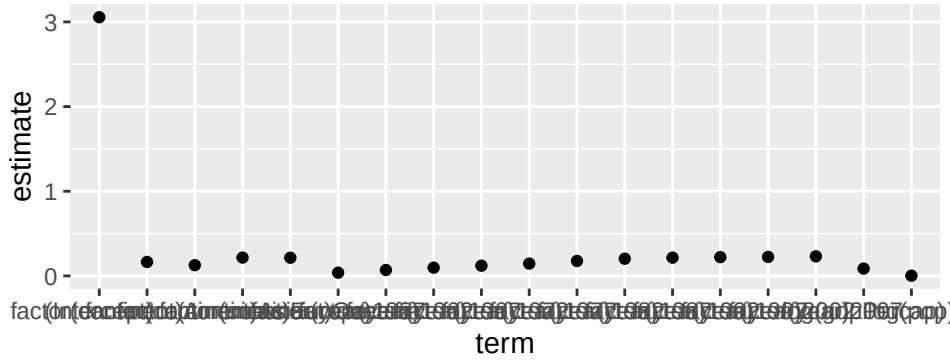
```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.113 on 1686 degrees of freedom
## Multiple R-squared:  0.7656, Adjusted R-squared:  0.7632
## F-statistic: 323.9 on 17 and 1686 DF,  p-value: < 2.2e-16
```

```
df_mod1 <- tidy(mod1)
```

Coefficient plots

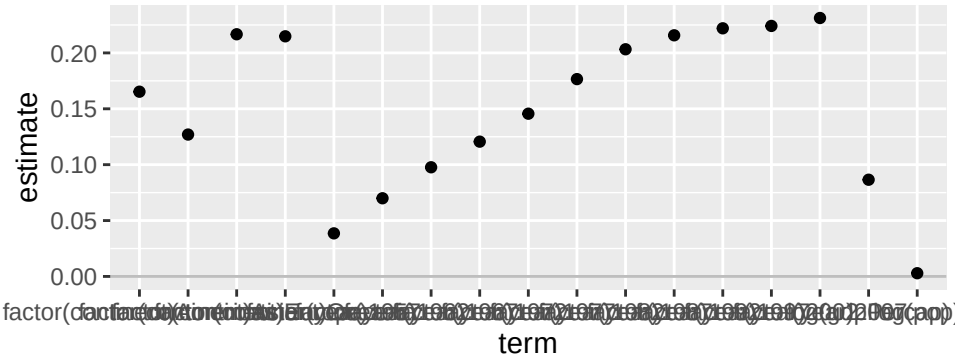
First, lets plot the size of the coefficients (`broom` stored them in the `estimate` column for us).

```
ggplot(df_mod1,
       aes(x = term, y = estimate)) +
  geom_point()
```



Let's drop the intercept along the way and add a line for the null effect.

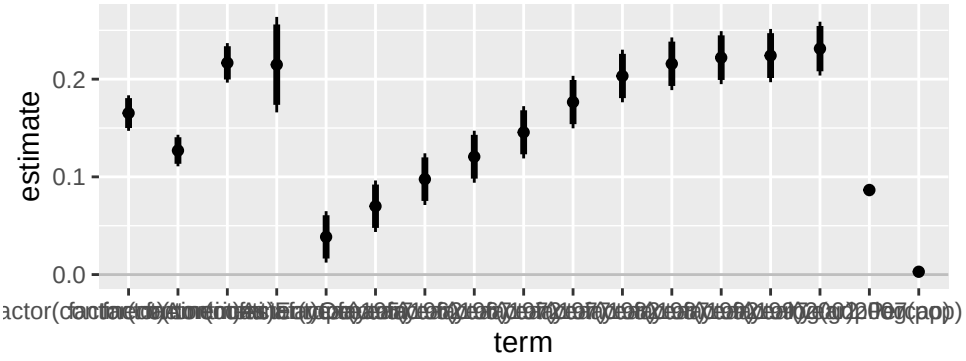
```
df_mod1 %>%
  dplyr::filter(term != "(Intercept)") %>%
  ggplot(aes(x = term, y = estimate)) +
  geom_hline(yintercept = 0, color = "grey") +
  geom_point()
```



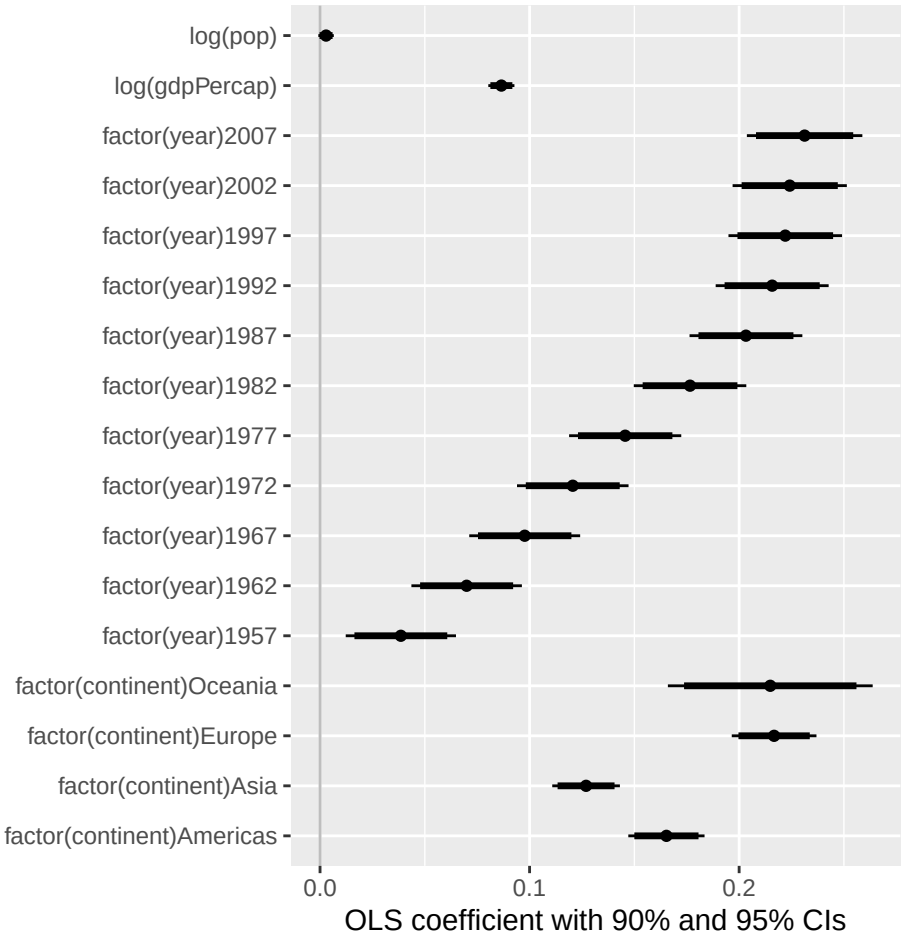
Let's add error bars to each estimate. Below we first plot the 95% CI (z-value of 1.96). We can then also add the 90% CI on top (z-value of 1.65).

```
df_mod1 %>%
  dplyr::filter(term != "(Intercept)") %>%
  ggplot(aes(x = term, y = estimate)) +
  geom_hline(yintercept = 0, color = "grey") +
  geom_point() +
  geom_linerange(aes(ymin = estimate - 1.96*std.error, ymax = estimate + 1.96*std.error)) +
```

```
  geom_linerange(aes(ymin = estimate - 1.65*std.error, ymax = estimate + 1.65*std.error),
                 size = 1.2)
```



We can make the plot prettier by flipping the axes.



Fitted values

Now that we have our baseline model, suppose we want to better understand the interaction between population (`pop`) and GDP per capita (`gdpPercap`). Holding `gdpPercap` constant at two different values (1st and 3rd quantile), we would like to estimate our outcome variable `lifeExp` across a range of values for `pop`.

First, we use `expand.grid()` to generate the relevant data frame to predict values. We then use the `predict()` function to estimate life expectancy across the specified range of values.

```r
mod2 <- lm(lifeExp ~ pop*gdpPercap, data = df)

range(df$pop)
```

```
## [1]      60011 1318683096
```

```r
summary(df$gdpPercap)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    241.2   1202.1   3531.8   7215.3   9325.5 113523.1
```

```r
quantile(df$gdpPercap, 0.25)
```

```
##     25%
## 1202.06
```

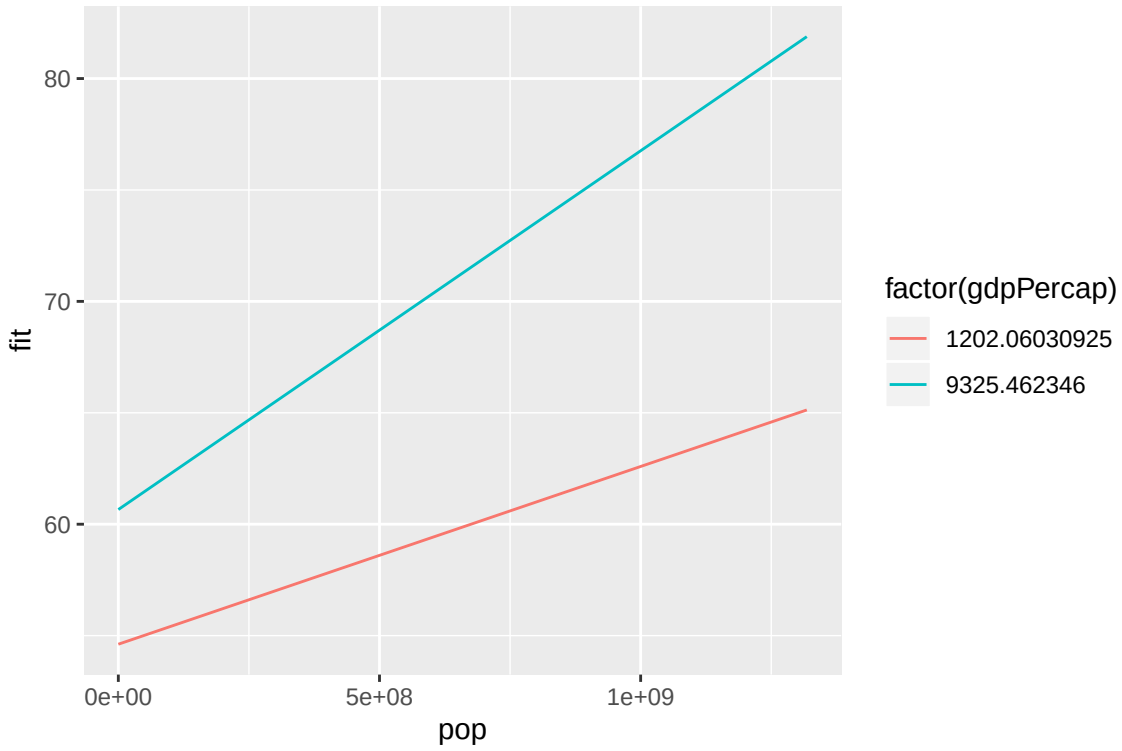
```r
df_new <- data.frame(expand.grid(pop = seq(min(df$pop), max(df$pop), 1000000),
                                 gdpPercap = c(quantile(df$gdpPercap, 0.25),
                                               quantile(df$gdpPercap, 0.75))))

pred <- predict(mod2, df_new, se = T)
df_pred <- cbind(df_new, pred)
```

Plotting it

```r
head(df_pred)
```

```
##       pop gdpPercap      fit    se.fit   df residual.scale
## 1   60011   1202.06 54.61432 0.3056841 1700       10.42918
## 2 1060011   1202.06 54.62229 0.3050714 1700       10.42918
## 3 2060011   1202.06 54.63027 0.3044779 1700       10.42918
## 4 3060011   1202.06 54.63825 0.3039037 1700       10.42918
## 5 4060011   1202.06 54.64622 0.3033489 1700       10.42918
## 6 5060011   1202.06 54.65420 0.3028136 1700       10.42918
```

```r
ggplot(df_pred,
       aes(x = pop, y = fit, color = factor(gdpPercap))) +
  geom_line()
```

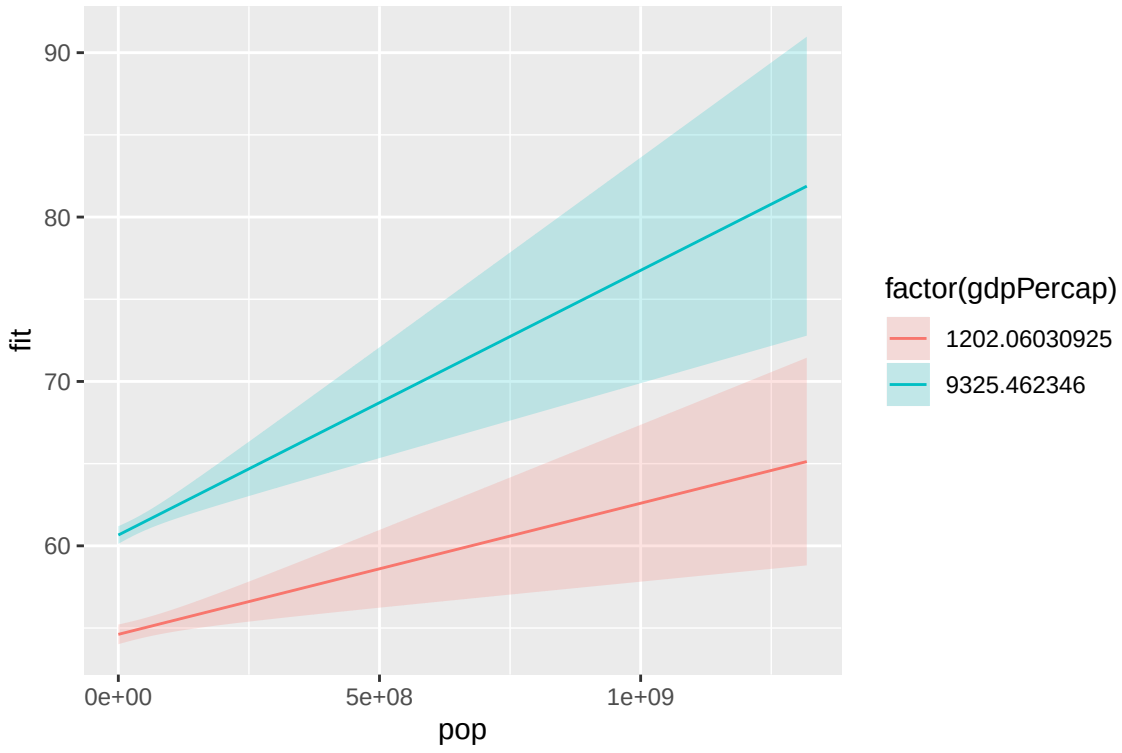


To show our degree of confidence in these estimates, let's add confidence intervals to this visualization. The `ggplot2` package offers several options for illustrating confidence intervals that differ based on the type of data we are working with. Since `pop` is a continuous variable, we use `geom_ribbon()` to plot ranges of the y-axis values along the continuously varying x-axis.

```
head(df_pred)
```

```
##       pop gdpPercap      fit    se.fit   df residual.scale
## 1   60011   1202.06 54.61432 0.3056841 1700       10.42918
## 2 1060011   1202.06 54.62229 0.3050714 1700       10.42918
## 3 2060011   1202.06 54.63027 0.3044779 1700       10.42918
## 4 3060011   1202.06 54.63825 0.3039037 1700       10.42918
## 5 4060011   1202.06 54.64622 0.3033489 1700       10.42918
## 6 5060011   1202.06 54.65420 0.3028136 1700       10.42918
```

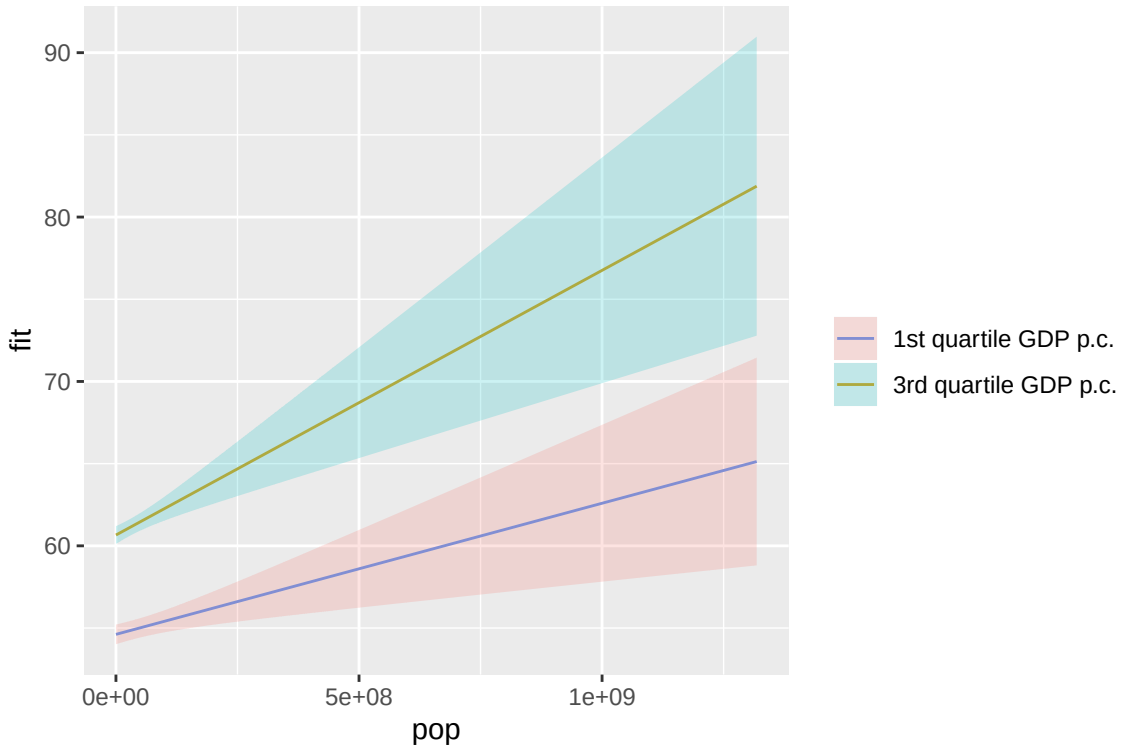
```
ggplot(df_pred,
       aes(x = pop,
           y = fit,
           color = factor(gdpPercap))) +
  geom_line(aes()) +
  geom_ribbon(aes(ymin = fit - 1.96*se.fit, ymax = fit + 1.96*se.fit, fill = factor(gdpPercap)),
              alpha = 0.2,
              color = NA)
```



To turn the plot into a publication-ready graph, we should give it appropriate labels and prettify it. You should consult the help files for each function, for example `?scale_color_manual()` or use Google to find the relevant parameters to control the appearance of the graph.

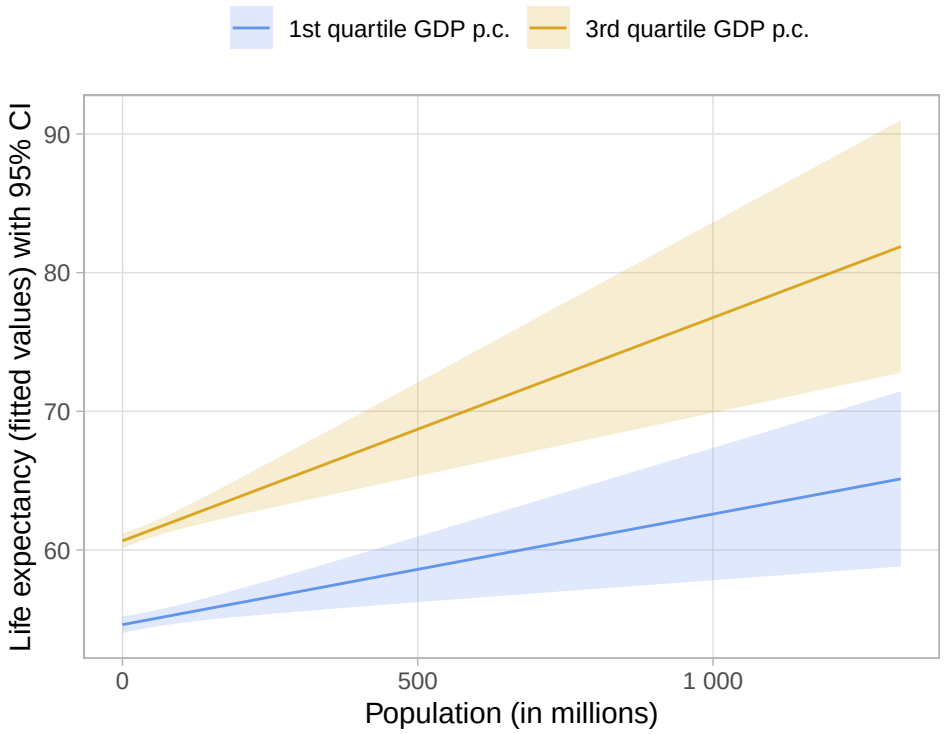
Below, let's give the legend better labels and manually change the color of the fitted value lines. Note the difference between `scale_fill_discrete()` and `scale_color_manual()`.

```
ggplot(df_pred,
       aes(x = pop,
           y = fit,
           color = factor(gdpPercap))) +
  geom_line(aes()) +
  geom_ribbon(aes(ymin = fit - 1.96*se.fit, ymax = fit + 1.96*se.fit, fill = factor(gdpPercap)),
              alpha = 0.2,
              color = NA) +
  scale_fill_discrete(labels = c("1st quartile GDP p.c.",
                                 "3rd quartile GDP p.c."),
                      name = "") +
  scale_color_manual(labels = c("1st quartile GDP p.c.",
                                "3rd quartile GDP p.c."),
                     name = "",
                     values = c("cornflowerblue",
                                "goldenrod"))
```



We need to specify `scale_*_manual()` for both if we want to display the confidence interval in the same shades as the fitted value curves. Below, we are also using the `label_number()` function from the `scales` package to make the x-axis easier to read.

```
library(scales)
mycols <- c("cornflowerblue", "goldenrod")
ggplot(df_pred,
       aes(x = pop,
           y = fit,
           color = factor(gdpPercap))) +
  geom_line(aes()) +
  geom_ribbon(aes(ymin = fit - 1.96*se.fit, ymax = fit + 1.96*se.fit, fill = factor(gdpPercap)),
              alpha = 0.2,
              color = NA) +
  scale_fill_manual(labels = c("1st quartile GDP p.c.",
                               "3rd quartile GDP p.c."),
                    name = "",
                    values = mycols) +
  scale_color_manual(labels = c("1st quartile GDP p.c.",
                                "3rd quartile GDP p.c."),
                     name = "",
                     values = mycols) +
  labs(x = "Population (in millions)",
       y = "Life expectancy (fitted values) with 95% CI") +
  theme_light() +
  theme(legend.position = "top",
        panel.grid.minor = element_blank()) +

  # Using the scales package
  scale_x_continuous(labels = label_number(scale = 1/1e6))
```


1st quartile GDP p.c.
3rd quartile GDP p.c.
Life expectancy (fitted values) with 95% CI
90
80
70
60
0
500
1 000
Population (in millions)

Appendix

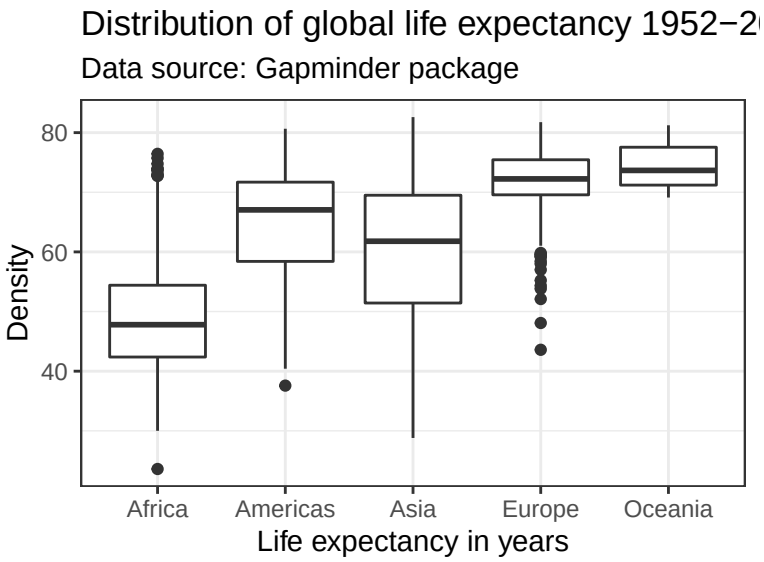
Boxplots

Another way to show the distribution of variables across groups are boxplots. Boxplots graph different properties of a distribution:

- The borders of the box denote the 25th and 75th percentile.
- The line within the box denotes the median.
- The position of the whiskers (vertical lines) denote the first quartile value minus 1.5 times the interquartile range and the third quartile value plus 1.5 times the interquartile range. We will not go into details here.
- Dots denote outliers (values that lie outside the whiskers), if applicable.

In `ggplot2` we can graph boxplots across multiple variables using the `geom_boxplot()` geometric object. Here, the continuous variable (i.e. `lifeExp`) should be specified as the `y` variable, and the categorical variable (i.e. `continent`) as the `x` variable.

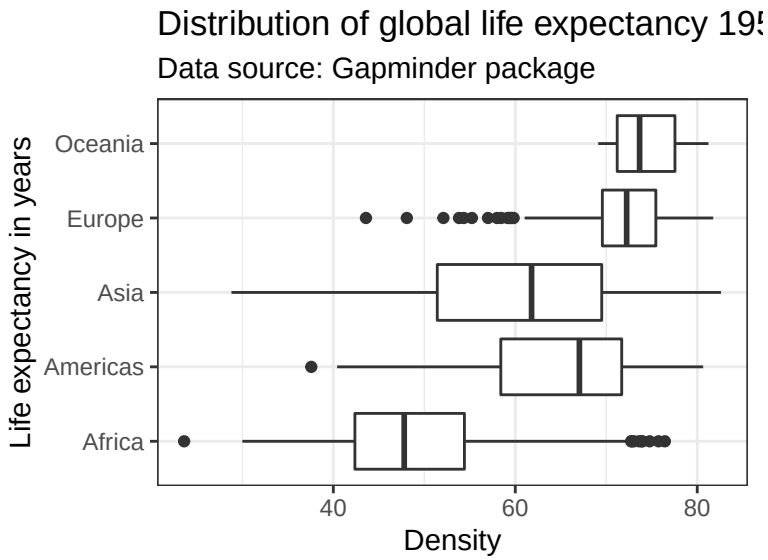
```
ggplot(subset(df),
       aes(x = continent,
           y = lifeExp)) +
  geom_boxplot() +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw()
```



We can flip the axes by using the `coord_flip()` command.

```
ggplot(subset(df),
       aes(x = continent,
           y = lifeExp)) +
  geom_boxplot() +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expectancy in years",
       y = "Density") +
  theme_bw() +
```

```
coord_flip()
```



Sources

Wilkinson, L., 2012. The grammar of graphics. In *Handbook of Computational Statistics* (pp. 375-414). Springer, Berlin, Heidelberg.