Day 1: Data Visualization in R

FSU Summer Methods School

Therese Anders

5/6/2019

Getting an overview of the data

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 over 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. Lets 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 ...
```

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" (Data Visualization with ggplot2 Cheat Sheet.

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.
- `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)
```

Showing data distributions

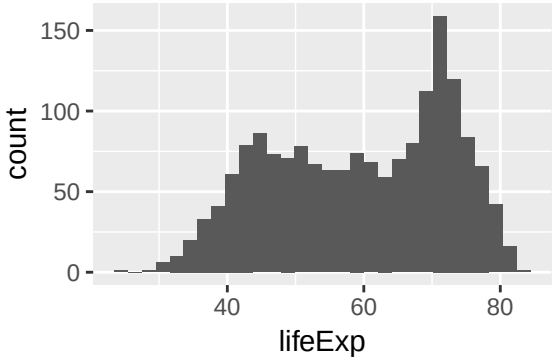
Histograms

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 Can you make sense of this graph? What is plotted on the x-axis? What is plotted on the y-axis? What specifies the width of each bar? What specifies the height of each bar?

A histogram plots the distribution of a variable. The x-axis specifies the values of the variable. The y-axis specifies the number of observations for each value (or group of values) of the variable. The width of the bar specifies which values of the variable are grouped into one bin. The height of the bar specifies the number of observations in each bin.

Question 2 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).

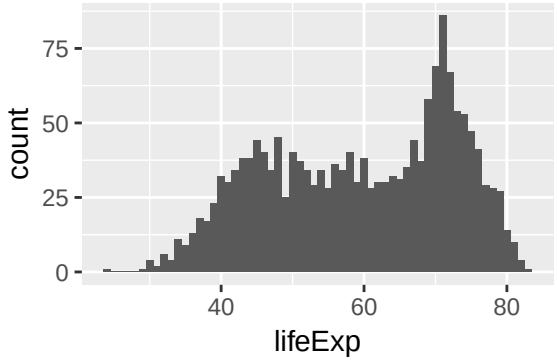
Adjusting the number of bins

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) # 60 years
```

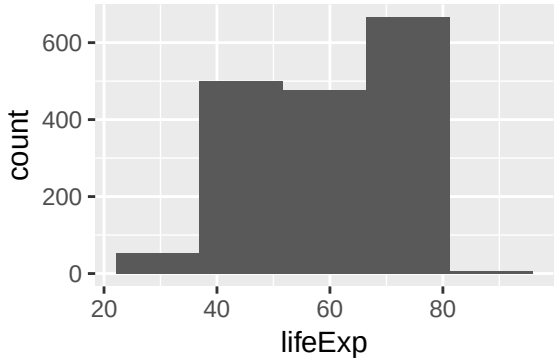
```
## [1] -59.004
```

```
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)
```

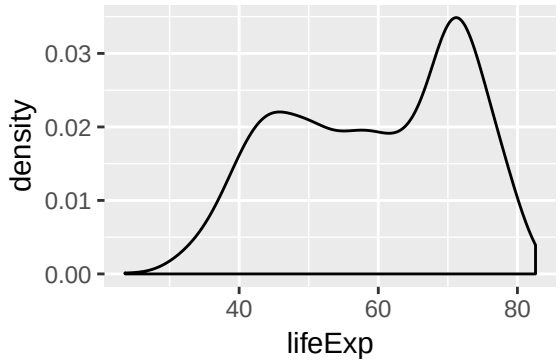


Density plots

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.

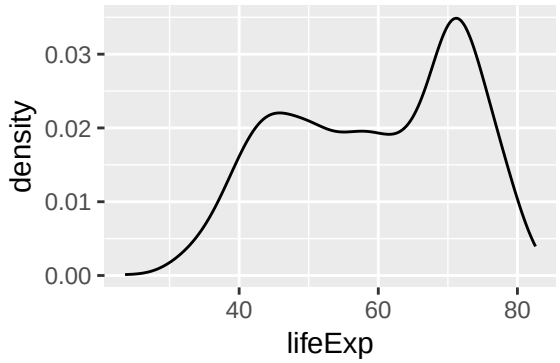
Density plots are continuous alternatives to histograms that do not rely on bins. We will cover details about the mechanics behind density plots and their estimation here. Just know that we can interpret the height of the density curve in a similar way that we interpreted the height of the bars in a 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")
```



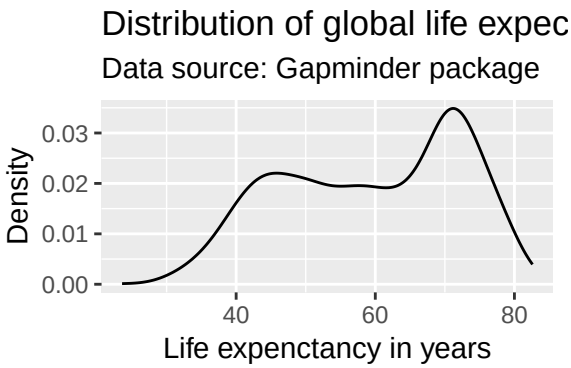
Controlling the appearance of graphs

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") +
```

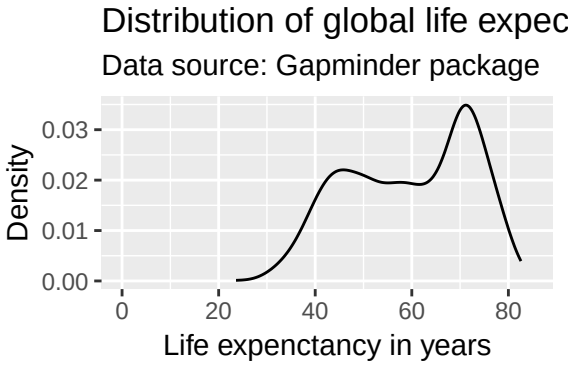
```
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
       y = "Density")
```



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 expenctancy in years",
       y = "Density") +
  coord_cartesian(xlim = c(0, 85))
```

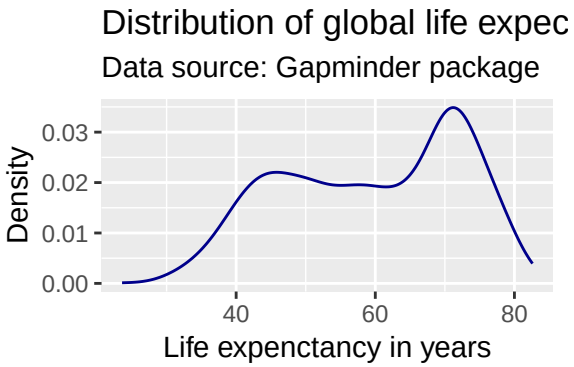


Caution!! You will sometimes see the command `scale_y_continuous(limits = c(0, 85))` instead of `coord_cartesian(ylim = c(0, 85))`. Note that these are not the same. `coord_cartesian()` only adjusts the range of the axes (it "zooms" in and out), while `scale_y_continuous(limits = c())` subsets the data. 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.

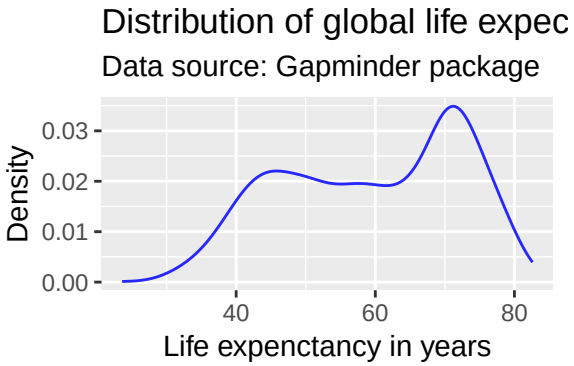
```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "darkblue") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
       y = "Density")
```



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 expenctancy in years",
       y = "Density")
```

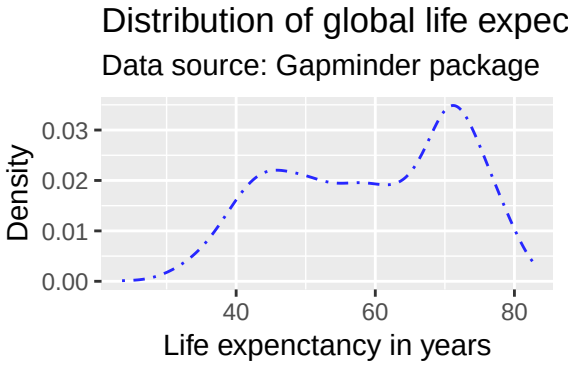


We will talk more about color schemes later in the workshop.

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 expenctancy in years",
       y = "Density")
```

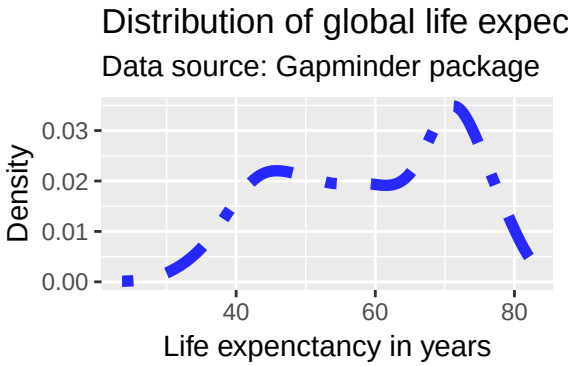


Changing the width of the line

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",
```

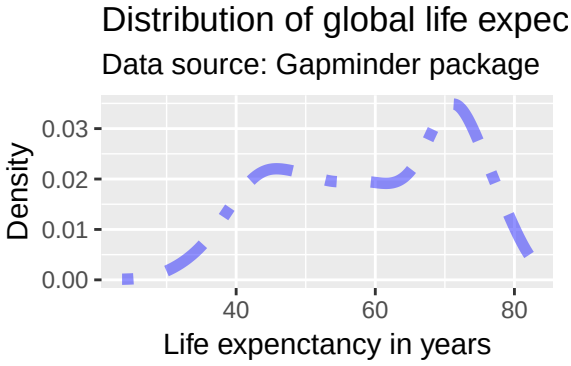
```
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
       y = "Density")
```



Changing the opacity of the line

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.

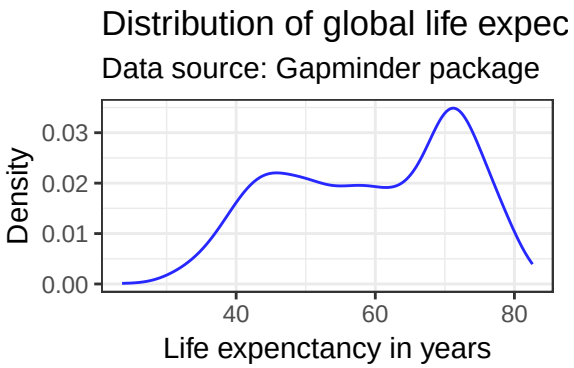
```
ggplot(df,
       aes(x = lifeExp)) +
  geom_line(stat = "density",
            color = "#2727ff",
            linetype = "dotdash",
            size = 2,
            alpha = 0.5) +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy 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_minimal()`. For a full list of themes, see https://ggplot2.tidyverse.org/reference/ggtheme.html. We can change all parameters manual using the `theme()` function.

```
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 expenctancy in years",
       y = "Density") +
  theme_bw()
```



Graphing distributions across groups

Using different colors

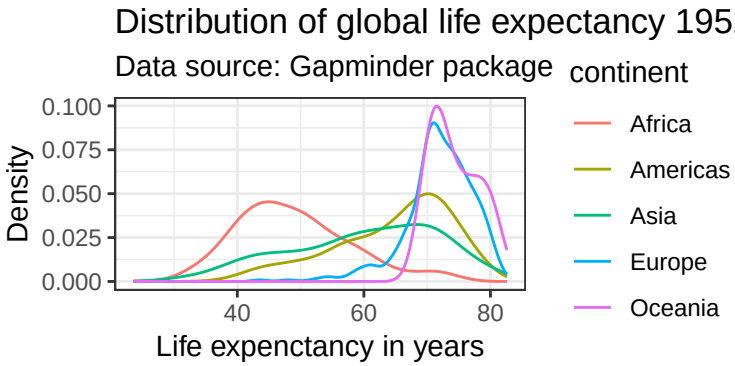
Sometimes, we want to compare distributions across different groups in our data set. Suppose, we wanted to assess the distribution of the life expectancy on different continents. We can use the `table()` function to get an overview over the groups in our data.

```
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 expenctancy in years",
       y = "Density") +
  theme_bw()
```

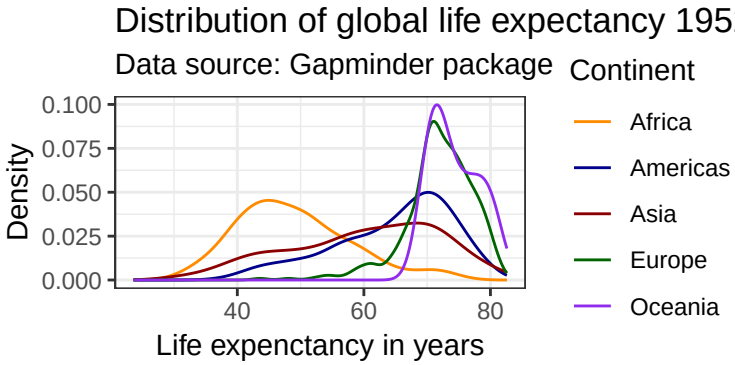


Question 3 What is the difference between specifying the `color` parameter outside the `aes()` argument versus within the `aes()` argument?

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.

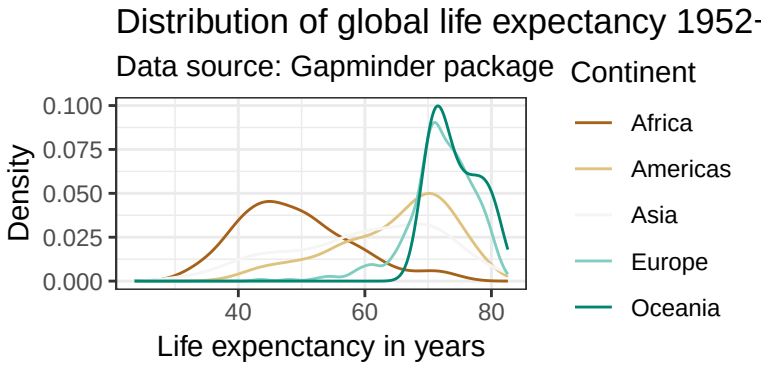
We can adjust the colors used in the plot in a variety of ways. Below, we first use the `scale_color_manual()` function. This will change the colors in both the plot and the legend, based on our manual specification. Within the `scale_color_manual()` argument, we can also specify a name and labels for 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 expenctancy 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 expenctancy in years",
       y = "Density") +
  theme_bw() +
  scale_color_brewer(palette = "BrBG",
                     name = "Continent")
```

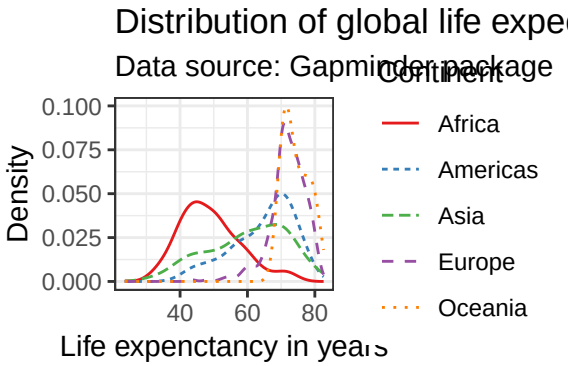


Check out the list of color palettes compiled by Emil Hvitfeldt. There is even a LaCroix inspired color scheme available using the package `LaCroixColoR`! Another popular option are the color schemes from the `viridis` package due to their desirable properties with respect to colorblindness and printability.

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 `lintype` and `color` aesthetics, we need to pass the same name within the `scale` function.

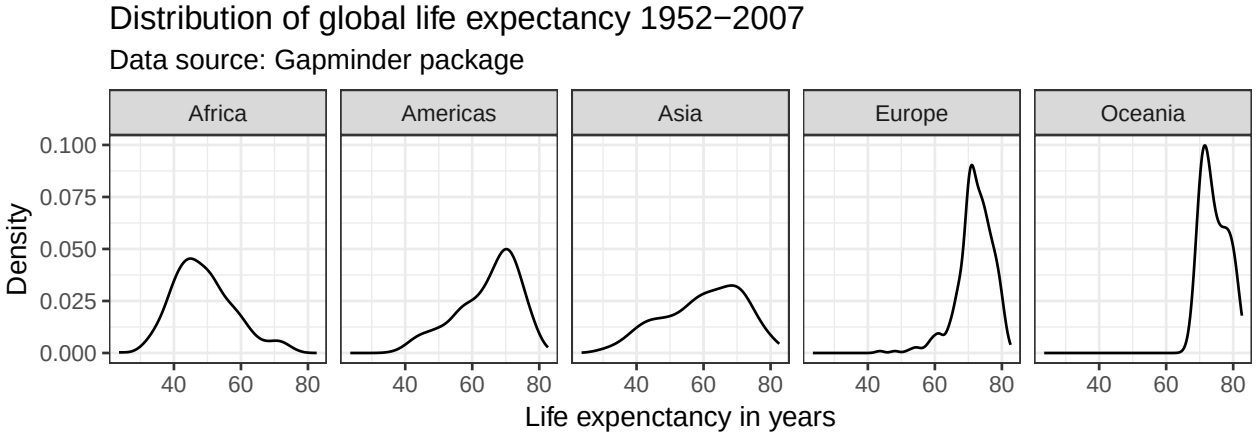
```
ggplot(df,
       aes(x = lifeExp,
           color = continent,
           linetype = continent)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
    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



Faceting

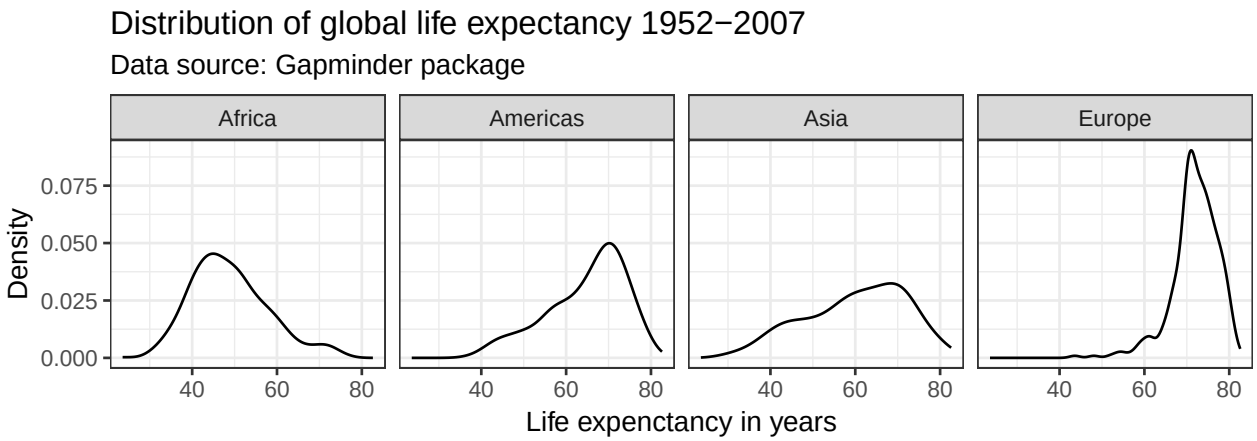
Another option to graph different groups is to use faceting. This means to plot each value of the variable upon which we facet in a different panel within the same plot. Here, we will use the `facet_wrap()` function. We could also use the `facet_grid()` which allows faceting across more than one variable.

```
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 expenctancy in years",
       y = "Density") +
  theme_bw() +
  facet_wrap(~ continent, nrow = 1)
```



Suppose, we wanted to exclude the plot for 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()`.

```
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 expenctancy in years",
       y = "Density") +
  theme_bw() +
  facet_wrap(~ continent, nrow = 1)
```



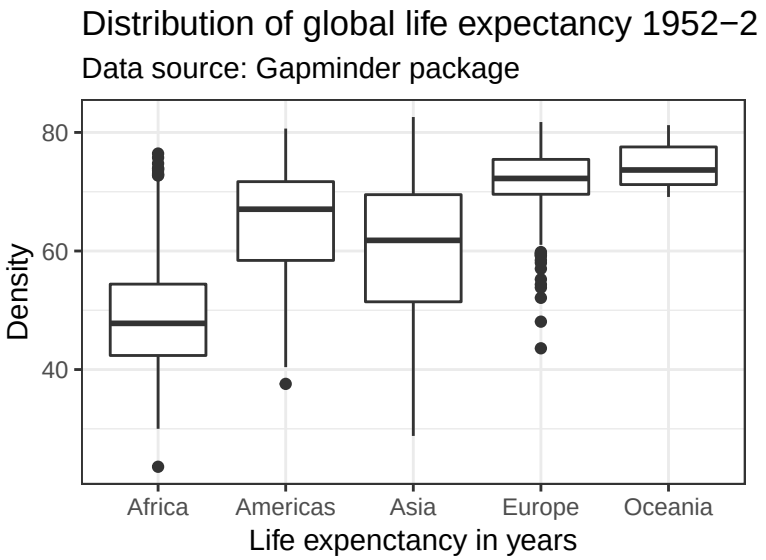
Boxplots

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.

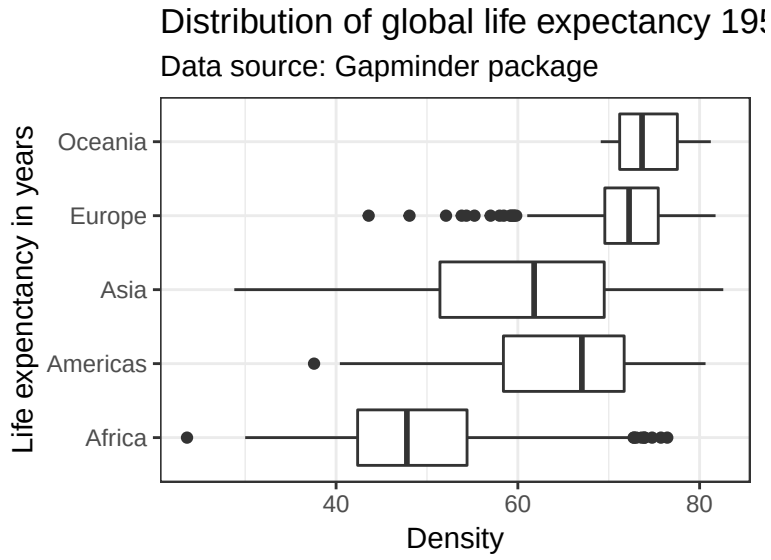
In `ggplot2` we can graph boxplots across multiple variables using the `geom_boxplot()` geometric object. Here, the continuous variable (i.e. `lifeExp`) should be specified as the `y` variable, and the categorical variable (i.e. `continent`) as the `x` variable.

```
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 expenctancy in years",
       y = "Density") +
  theme_bw()
```



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

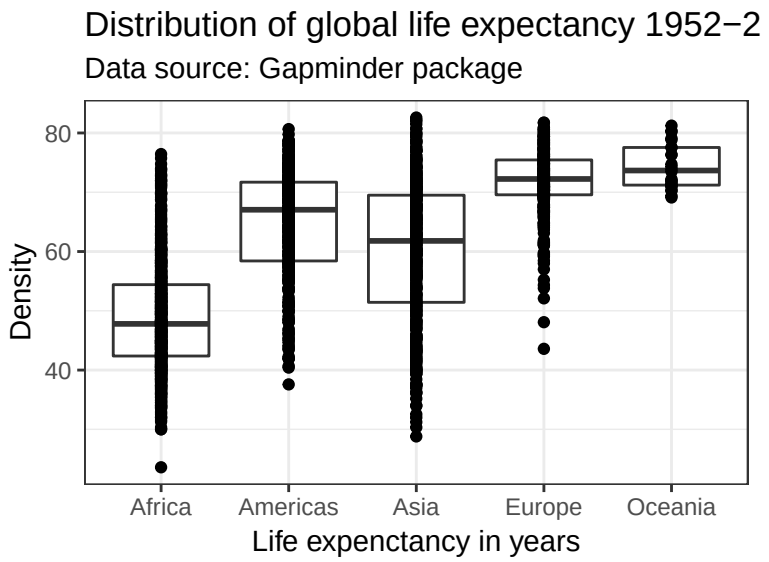
```
ggplot(subset(df),
       aes(x = continent,
           y = lifeExp)) +
  geom_boxplot() +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
       y = "Density") +
  theme_bw() +
  coord_flip()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



The boxplot denotes outlier with points. We could also overlap the boxplot with the original observations using the `geom_point()` aesthetic. This illustrates how many observations are included in each group. Make sure to specify `outlier.shape = NA` within `geom_boxplot()` distinguish between "regular" and outlier observations.

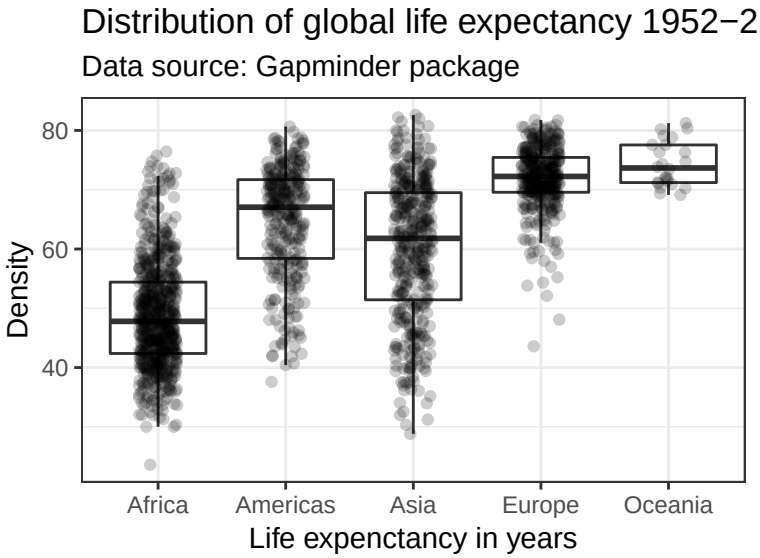
```
ggplot(subset(df),
       aes(x = continent,
```

```
          y = lifeExp)) +
  geom_boxplot(outlier.shape = NA) +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
       y = "Density") +
  theme_bw() +
  geom_point()
```



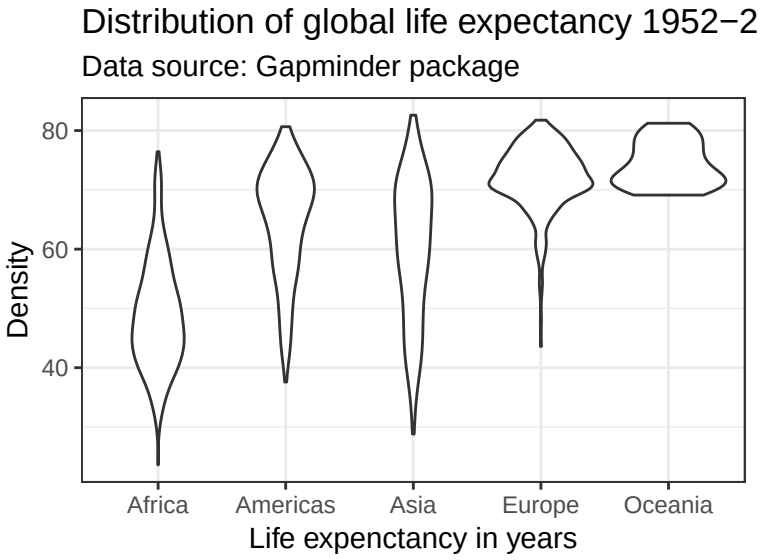
This is confusing because there is a lot of overplotting when we add all original observations to the point. We can add jitter to the points. This will add a small random value to each point in either direction and enhance the appearance of the graph. We can control the spread by using the `width` argument. We can also decrease the opacity of the points. The plot below shows that while Oceania has the highest median life expectancy, this value is based on a lot fewer observations as compared to other continents.

```
ggplot(subset(df),
       aes(x = continent,
           y = lifeExp)) +
  geom_boxplot(outlier.shape = NA) +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
       y = "Density") +
  theme_bw() +
  geom_point(position = position_jitter(width = 0.15),
             alpha = 0.2)
```



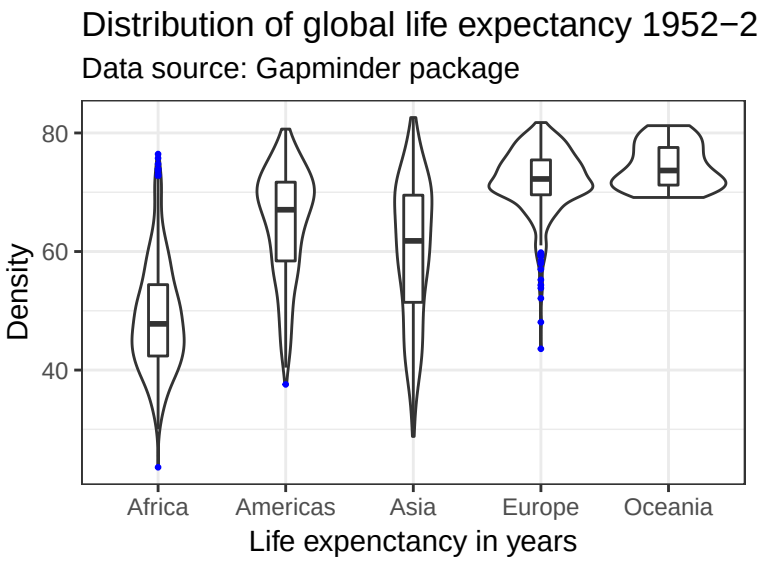
Finally, we could use violin plots to better show the distribution of life expectancy within each continent. Violin plots are similar to boxplot in that they show the distribution of a variable. However, in addition to range and median, they add kernel density plot on each side.

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

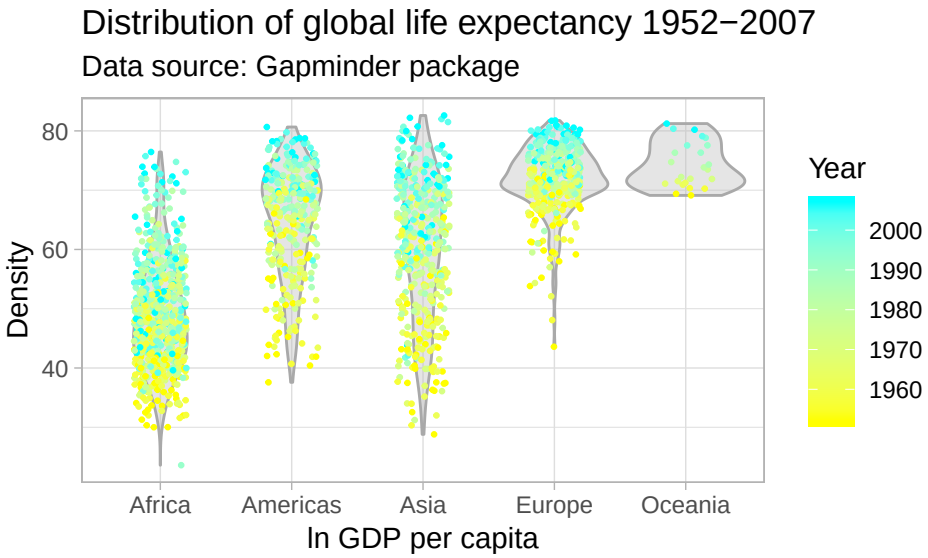


We can add additional statistical properties to the violin plot to make them more informative. Below, we add a boxplot on top to show the median and quartile of the data. We have to manually adjust the width of the boxplot to avoid overplotting. Below, I am also changing the size and color of the outliers displayed in the plot.

```
ggplot(subset(df),
       aes(x = continent,
           y = lifeExp)) +
  geom_violin() +
  geom_boxplot(width = 0.15,
               outlier.size = 0.5,
               outlier.color = "blue") +
  labs(title = "Distribution of global life expectancy 1952-2007",
       subtitle = "Data source: Gapminder package",
       x = "Life expenctancy in years",
       y = "Density") +
  theme_bw()
```



Practice Please try to re-create the plot below as closely as possible.



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.

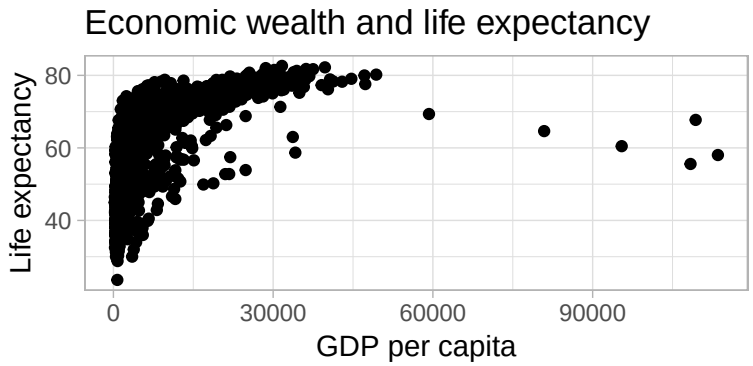
```
# ggsave("violin_lifeexp_continent.png", width = 6, height = 3, dpi = 400)
```

Showing relationships in data

Scatter plots

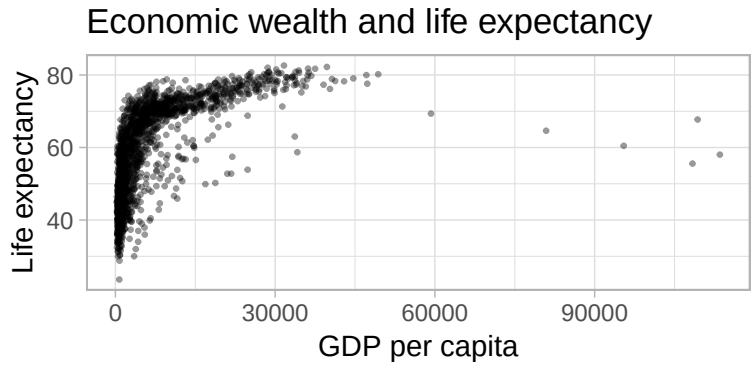
In their basic form, scatter plots are used to display values of two variables on a Cartesian coordinate system. Below, we inspect the relationship between GDP per capita and life expectancy.

```
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()
```

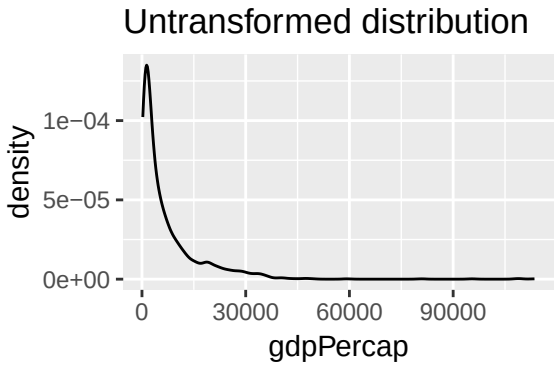


Scaling the data

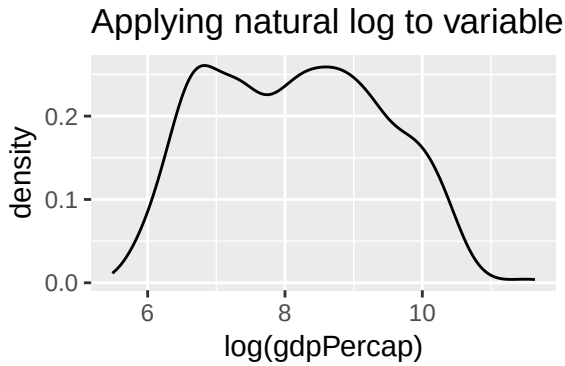
One reason why the plot above is hard to read is rooted in the shape of the distribution of the GDP per capita variable. GDP per capita has a strong right skew. We can correct for this skew and transform the variable to have a more "normal" distribution by taking the natural logarithm. There are multiple 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

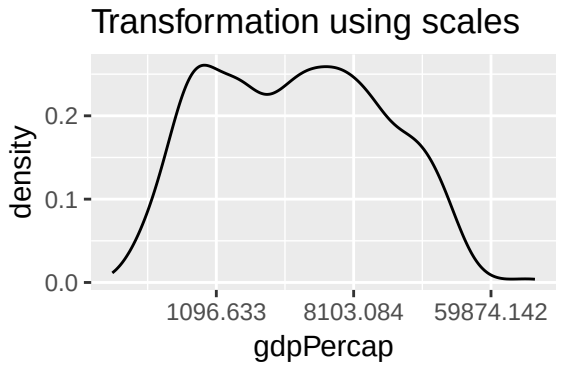
```
ggplot(df,
       aes(x = gdpPercap)) +
  geom_line(stat = "density") +
  labs(title = "Untransformed distribution")
```




```
ggplot(df,
       aes(x = log(gdpPercap))) +
  geom_line(stat = "density") +
  labs(title = "Applying natural log to variable directly")
```



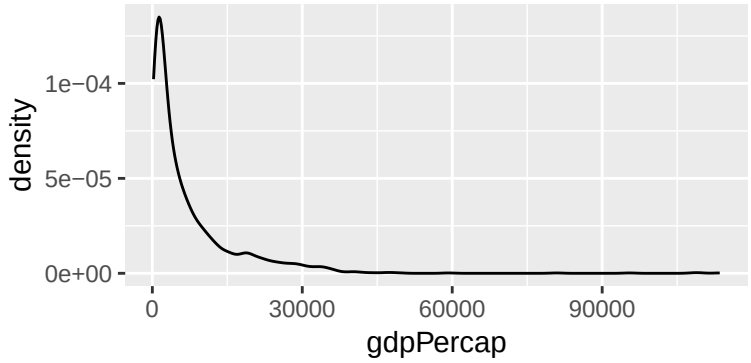

```
ggplot(df,
       aes(x = gdpPercap)) +
  geom_line(stat = "density") +
  labs(title = "Transformation using scales") +
  scale_x_continuous(trans = "log")
```



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()`.

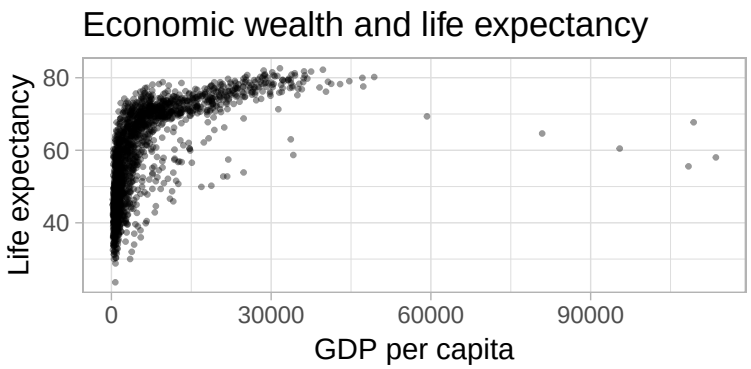
Transforming the variable using the natural logarithm within `aes()` causes the x-axis to be displayed in log values. Using `scale_x_continuous()`, the data is transformed in the same way, however, the x-axis is displayed in the original, non-logged version.

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




```
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()
```

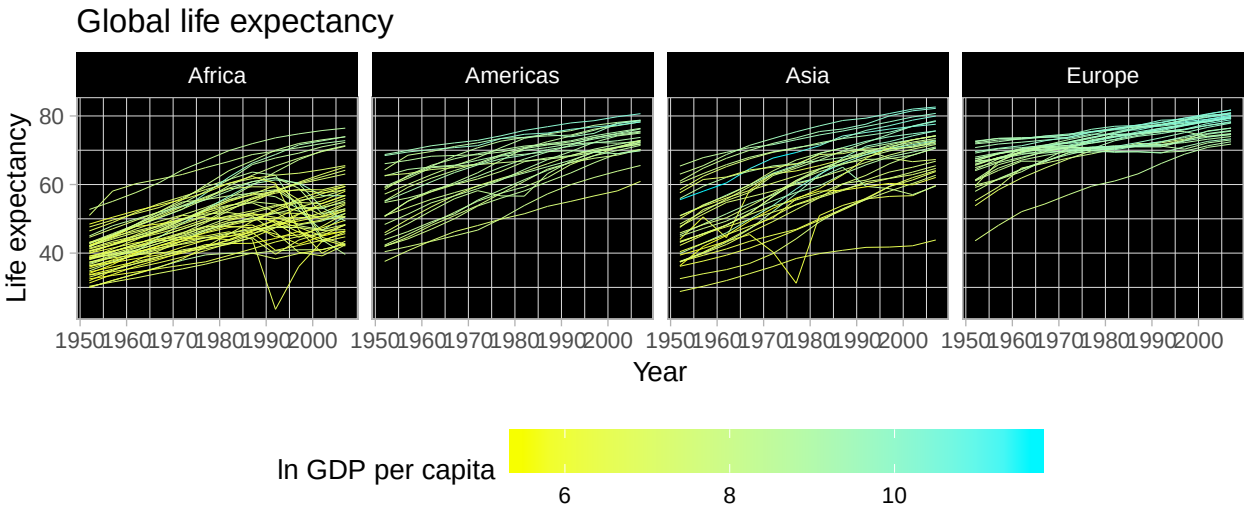


Adding a trend line

Spaghetti plots

Talk about scaling data versus axes

```
ggplot(subset(df, continent != "Oceania"),
       aes(x = year,
           y = lifeExp,
           group = country,
           color = log(gdpPercap))) +
  geom_line(alpha = 1,
            size = 0.1) +
  theme_light() +
  theme(panel.background = element_rect(fill = "black"),
        panel.grid = element_line(size = 0.1)) +
  scale_color_gradient(low = "#f7ff00",
                       high = "#00f7ff",
                       name = "ln GDP per capita") +
  labs(x = "Year",
       y = "Life expectancy",
       title = "Global life expectancy") +
  theme(legend.position = "bottom",
        strip.background = element_rect(fill = "black"),
        strip.text = element_text(color = "white"),
        legend.key.width = unit(1.5, "cm")) +
  facet_wrap(~continent, nrow = 1)
```


Global life expectancy
Africa
Americas
Asia
Europe
Life expectancy
80
60
40
1950
1960
1970
1980
1990
2000
Year
ln GDP per capita
6
8
10