GEOG 6000 Lab Plotting with ggplot2

Simon Brewer

September 22, 2017

Introduction

The base installation of R contains several functions for plotting and visualizing data. In addition to these, there are several addon packages that greatly extend the basic functionality. Here, we will look at one the most widely used of these, **ggplot2**.

The **ggplot2** package was written by Hadley Wickham, and implements the Grammar of Graphics framework developed by Leland Wilkinson. The idea behind the grammar of graphics is that all plots can be described by a common language, rather than considering them as separate barplots, line charts, etc. What differs is the coordinate system and *geometry* used to place the data on the page. Using this package requires a little more work than standard plots, but the results are usually worth while.

The **ggplot2** package comes with two main functions: `qplot()`, a simple plotting routine which is designed to replace the base `plot()` function in R; and `ggplot()`, which provides a much greater range of functions. We'll look briefly at the first of these before going on examples with `ggplot()` in more detail.

Datasets

Start by installing and loading the **ggplot2** package:

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

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

Datasets

We'll use several datasets in this lab:

- the Iris morphological dataset (*iris.csv*)
- a dataset on the growth of five Orange trees (*orange.csv*)
- Gapminder data on countries population, GDP and life expectancy for the past 60 years (*gapminderData.csv*)
- The VADeaths dataset (*VADeaths.csv*), showing death rates in West Virginia

Load these data sets, and use `str()` and `names()` to look at what they contain:

```
iris = read.csv("iris.csv")
str(iris)
orange = read.csv("orange.csv")
str(orange)
gapdata = read.csv("gapminderData5.csv")
str(gapdata)
VADeaths = read.csv("VADeaths.csv")
VADeaths
```

Reshaping tables into dataframes for plotting

Note that the Virginia death rate dataset is simply a table. To use this with ggplot, we need to change it from a table into a data frame, with columns of age groups, social groups and death rates. When you installed the **ggplot2** package, it installed a helper package called **reshape2** with it. This has a number of routines for reshaping data into suitable organizations for plotting. One of these functions `melt()` takes tabular data and converts it into a data frame:

```
library(reshape2)
VADeaths2 = melt(VADeaths, variable.name="DemClass", value.name="DeathRate")
```

Have a look at the new data frame to see how the `melt()` function has reorganized it.

```
VADeaths2
```

```
##      Age     DemClass DeathRate
## 1  50-54   Rural.Male      11.7
## 2  55-59   Rural.Male      18.1
## 3  60-64   Rural.Male      26.9
## 4  65-69   Rural.Male      41.0
## 5  70-74   Rural.Male      66.0
## 6  50-54 Rural.Female       8.7
## 7  55-59 Rural.Female      11.7
## 8  60-64 Rural.Female      20.3
## 9  65-69 Rural.Female      30.9
## 10 70-74 Rural.Female      54.3
## 11 50-54   Urban.Male      15.4
## 12 55-59   Urban.Male      24.3
## 13 60-64   Urban.Male      37.0
## 14 65-69   Urban.Male      54.6
## 15 70-74   Urban.Male      71.1
## 16 50-54 Urban.Female       8.4
## 17 55-59 Urban.Female      13.6
## 18 60-64 Urban.Female      19.3
## 19 65-69 Urban.Female      35.1
## 20 70-74 Urban.Female      50.0
```

Simple plots with `qplot()`

Scatterplots

A basic scatterplot. Note that the syntax matches the original `plot()` function, but the output is quite different:

```
qplot(Sepal.Length, Petal.Length, data = iris)
```

Coloring the symbols by species (note that a legend is automatically added):

```
qplot(Sepal.Length, Petal.Length, data = iris,
      color = Species)
```

Adding further information by resizing the symbols with a third variable (petal width):

```
qplot(Sepal.Length, Petal.Length, data = iris,
      color = Species, size = Petal.Width)
```

Finally, we add labels and a title, and make the symbols transparent using the parameter `alpha`, in order to make it clear where these overlap:

```
qplot(Sepal.Length, Petal.Length, data = iris,
      color = Species, size = Petal.Width, alpha=I(0.7),
      xlab = "Sepal Length", ylab = "Petal Length",
      main = "Sepal vs. Petal Length in Fisher's Iris data")
```

By default, `qplot()` makes a scatterplot, but this can be changed to a different plot type or geometry, by using the `geom` argument. For example, to make a histogram:

```
qplot(Sepal.Width, data=iris, geom='histogram', binwidth=0.1)
```

Or a set of boxplots:

```
qplot(Species, Sepal.Width, data=iris, geom='boxplot')
```

Line plots can be made with `geom="line"`. Try this with the set of Orange trees:

```
qplot(age, circumference, data=orange, geom='line')
```

As this is plotting all five trees together, the line doesn't make much sense. We can separate out the different trees by telling `qplot` to use the tree number as a color. As this is a continuous variable, we first convert this into a factor to indicate the different groups (trees):

```
orange$Tree = factor(orange$Tree)
qplot(age, circumference, data=orange, color=Tree, geom='line')
```

More advanced figures with `ggplot()`

The `ggplot()` function offers more control over your plots. It works in quite a different way to the other plotting functions, that start with a base plot, then add other points, lines, etc to the figure. Instead, `ggplot` creates a plot *object*, which can be adjusted and added to as you proceed.

In order to understand how `ggplot` makes a figure, we need to establish what the fundamental parts are of every data graph. They are:

1. Aesthetics – these are the roles that the variables play in each graph. A variable may control where points appear, the color or shape of a point, the height of a bar and so on.
2. Geometries – these are the geometric objects which represent the data: points, lines, bars, ...
3. Statistics – these are the functions which add some interpretation to the data, e.g. best fit line, location of median, etc
4. Scales – these are legends that show the relationship between variables and different symbols or colors (e.g. circular symbols represent females while squares represent males)
5. Facets – these are groups in your data which may be used to make multiple graphs, each for one of the groups. For example, faceting by gender would cause the graph to repeat for the two genders.

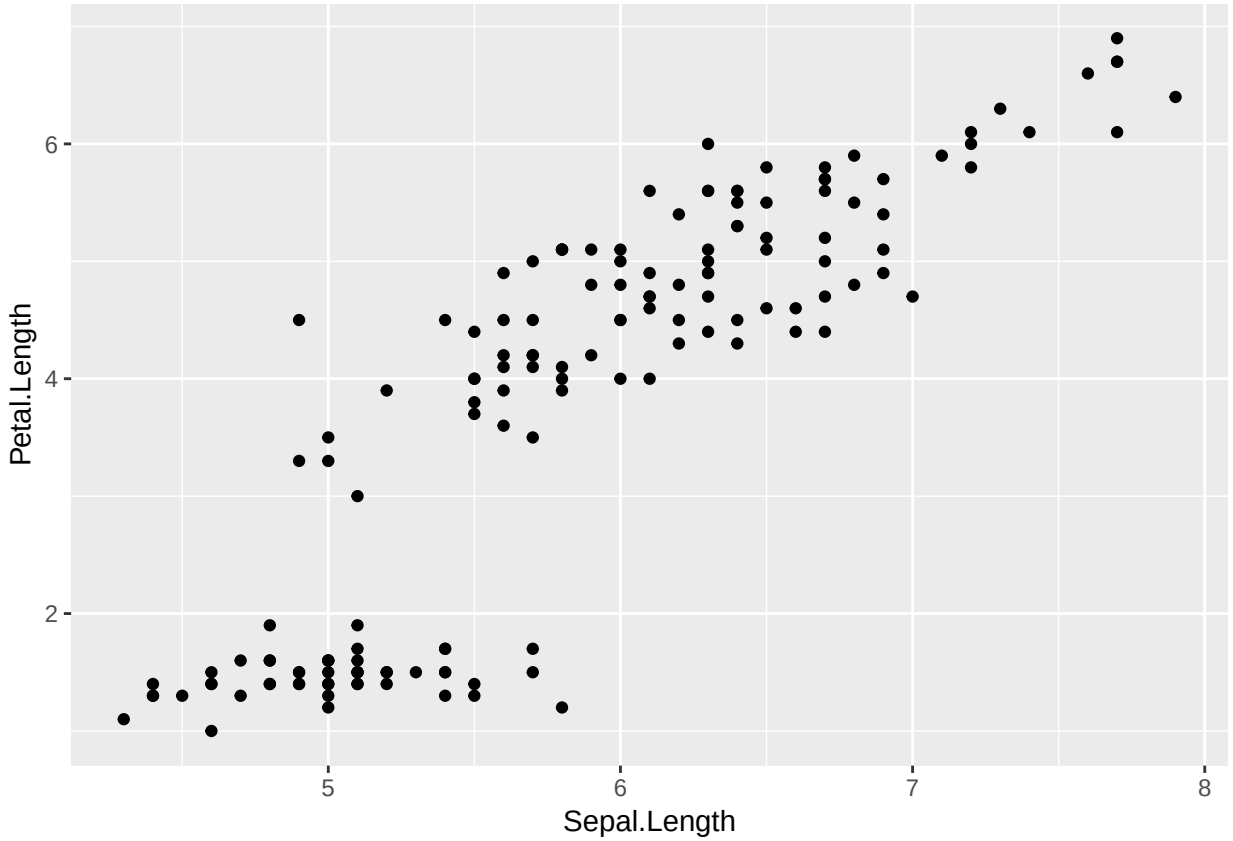
When making a `ggplot` figure, we generally start by creating the base figure. To do this we need to tell the function where the data is coming from, and the base aesthetic (i.e. which variable is `x`, which is `y`?). To remake the first iris plot with `ggplot()`, we would do the following:

```
myplot = ggplot(iris, aes(x=Sepal.Length, y=Petal.Length))
```

If you know type `myplot` at the command line, R will complain that there are no layers in the plot. This is because we need to tell `ggplot` what geometry to use. To do this, we simply 'add' one of `ggplot`'s geometry functions. In this case, we use `geom_point()`, to make a scatterplot:

```
myplot + geom_point()
```

And you should see the following figure:



A very important thing to note is that this has not changed our base object `myplot`, but simply displayed it with the point geometry added to it. If you retype `myplot` at the command line, you will again be told that there are no layers. In order to make this change stick (i.e. keep the use of the point geometry), we need to resave the plot object:

```
myplot = myplot + geom_point()
```

Now typing `myplot` makes the plot appear. This allows you to experiment with different displays, and only update (or resave) the plot object when you have changes that you want to keep.

We could now add other geometries, statistics or other effects to the plot figure to change the way that it looks. Instead, we'll work through several different examples now, using the Gapminder dataset to look at the different functions that are available.

Histograms

A simple histogram of life expectancy for all countries:

```
myplot = ggplot(gapdata, aes(x = lifeExp))
myplot + geom_histogram()
```

Let's add a title and axis labels to the base plot and remake the histogram. Note that by adding these to the base plot object, all subsequent plots will have these labels and title:

```
myplot = myplot + scale_x_continuous("Life Expectancy") + ggtitle("GapMinder Life Expectancy")
myplot + geom_histogram()
```

The output should tell you that the histogram binwidth is set to 1/30 of the data range. This can easily be adjusted:

```
myplot + geom_histogram(binwidth=1)
```

We might be interested in looking at the histogram of values by continent. There are a couple of ways to do this. The first is to include an aesthetic which links the continent to the color used to fill the histograms. Note that we do this by adding the aesthetic to the `geom_*` function (you could equally remake the original `ggplot()` command with this included):

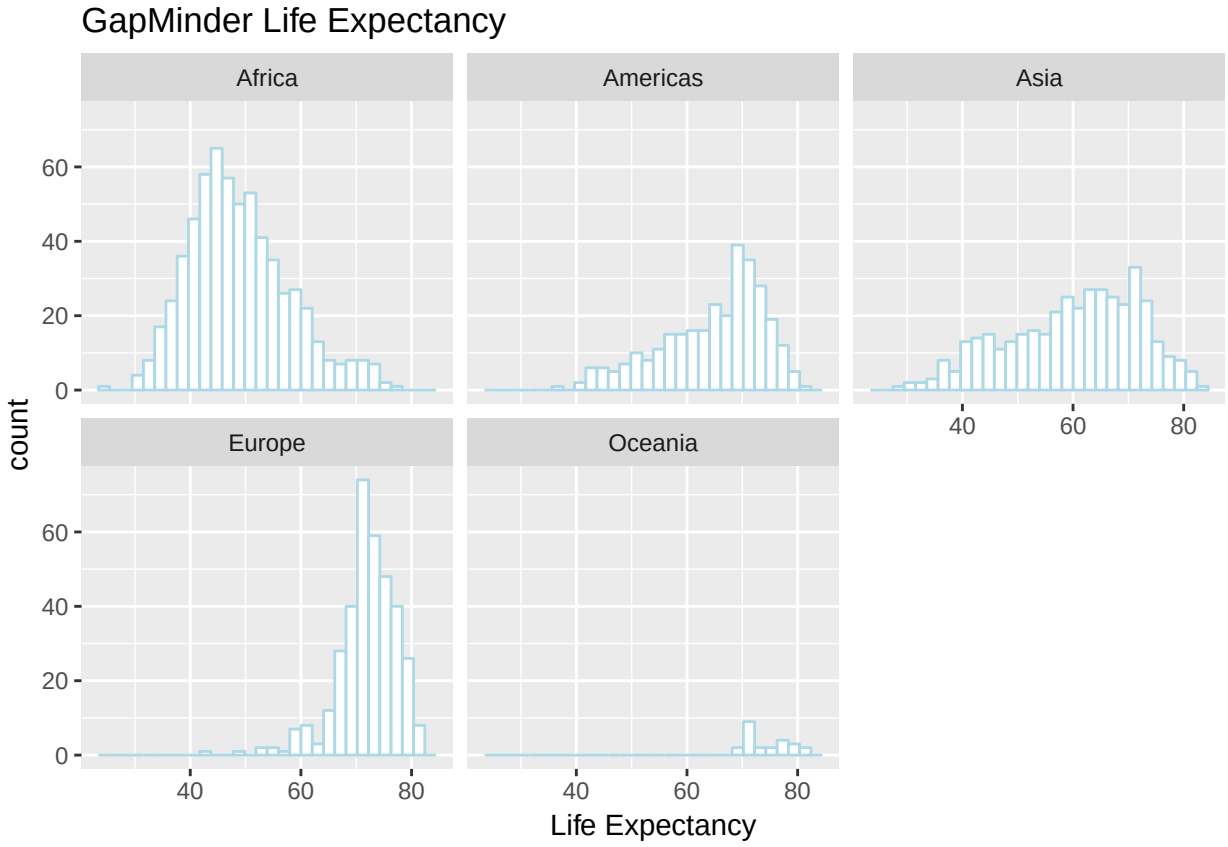
```
myplot + geom_histogram(aes(fill=continent))
```

By default, `ggplot` stacks the histograms - a better way to do this is have all bars starting at zero and superimposed. This can be achieved with the `position` parameter:

```
myplot + geom_histogram(aes(fill=continent), position = "identity")
```

An alternative display with several groups is to use *facets*. There are two types of facets in **ggplot2**: `facet_wrap` which splits plots based on one group, and `facet_grid` which splits plots based on two groups. We'll start by using `facet_wrap` to display the histograms by continent. We also add two parameters to `geom_histogram` to change the border and fill color of the histograms:

```
myplot + geom_histogram(color='lightblue', fill='white') + facet_wrap(~ continent)
```



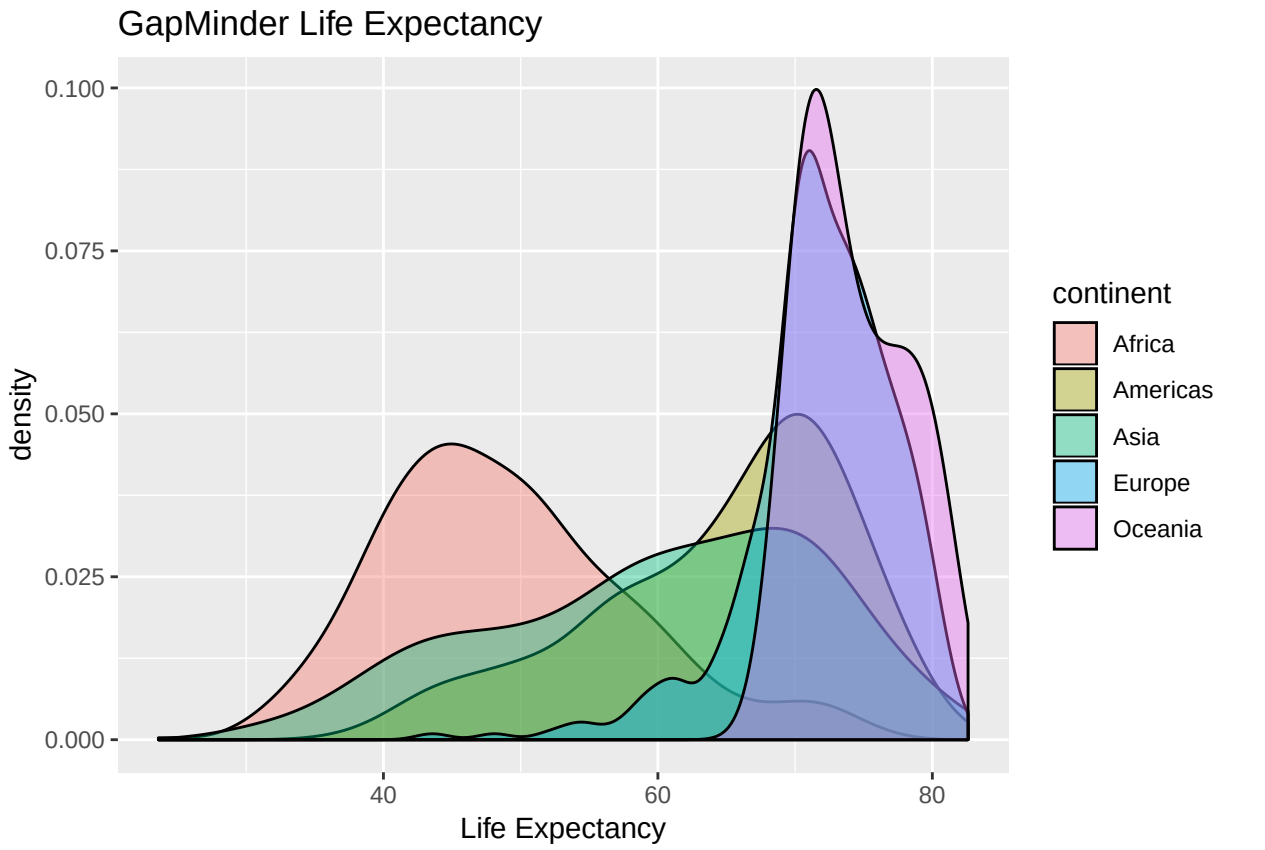
The dataset also includes the year that the data was taken in. We can use this as a second grouping, which allows us to make histograms per continent for every year that data was recorded:

```
myplot + geom_histogram(fill='darkorange') + facet_grid(year ~ continent)
```

Density plots

An alternative display of univariate, continuous data is through density plots (`geom_density()`). These fit a small Gaussian window around each observation, then add the value of all the windows across the range of data. The result is a smoothed, continuous looking histogram. As before, we can use the `fill` parameter to separate out continents. We add an alpha value again to make the results transparent:

```
myplot + geom_density(aes(fill=continent), alpha=0.4)
```



Boxplots

We have already looked at boxplots using the base R functions. Here, we need to specify the grouping within the plot aesthetic as `x`, and the continuous variable as `y`. As we need both `x` and `y`, we need to make a new `ggplot` object:

```
myplot = ggplot(gapdata, aes(x=continent, y=lifeExp))
myplot + geom_boxplot()
```

By default, the order of the groups is alphabetical, but it might be useful to order the boxplots to reflect something about the data, for example from low to high average life expectancy. The R function `reorder` can be used here. This require three parameters: a factor to be reorganized, a variable to be used to reorganize it, and a function to be calculated on that variable. Here we reorder by median life expectancy:

```
myplot + geom_boxplot(aes(x=reorder(continent, lifeExp, median)))
```

Barcharts

Two types of barchart can be produced by **ggplot2**. By default, the barcharts are presented as summary of the variable of interest. With factor or categorical variables, this is simply a count of each category. For example:

```
myplot = ggplot(gapdata, aes(x=continent))
myplot + geom_bar()
```

Often, however, we want to use some precalculated value for the height of the bar. In the Virginia death rate dataset, for example, the death rates have already been calculated by age and demographic class. To plot death rates per age groups, we can do the following:

```
myplot = ggplot(VADeaths2, aes(x = Age, y = DeathRate))
myplot + geom_bar(stat="identity")
```

We can split the plot into the different demographic groups. As before, we can do this in two ways. First, by including the demographic group as a fill in the aesthetic setup:

```
myplot + geom_bar(aes(fill=DemClass), stat="identity")
myplot + geom_bar(aes(fill=DemClass), stat="identity", position="dodge")
```

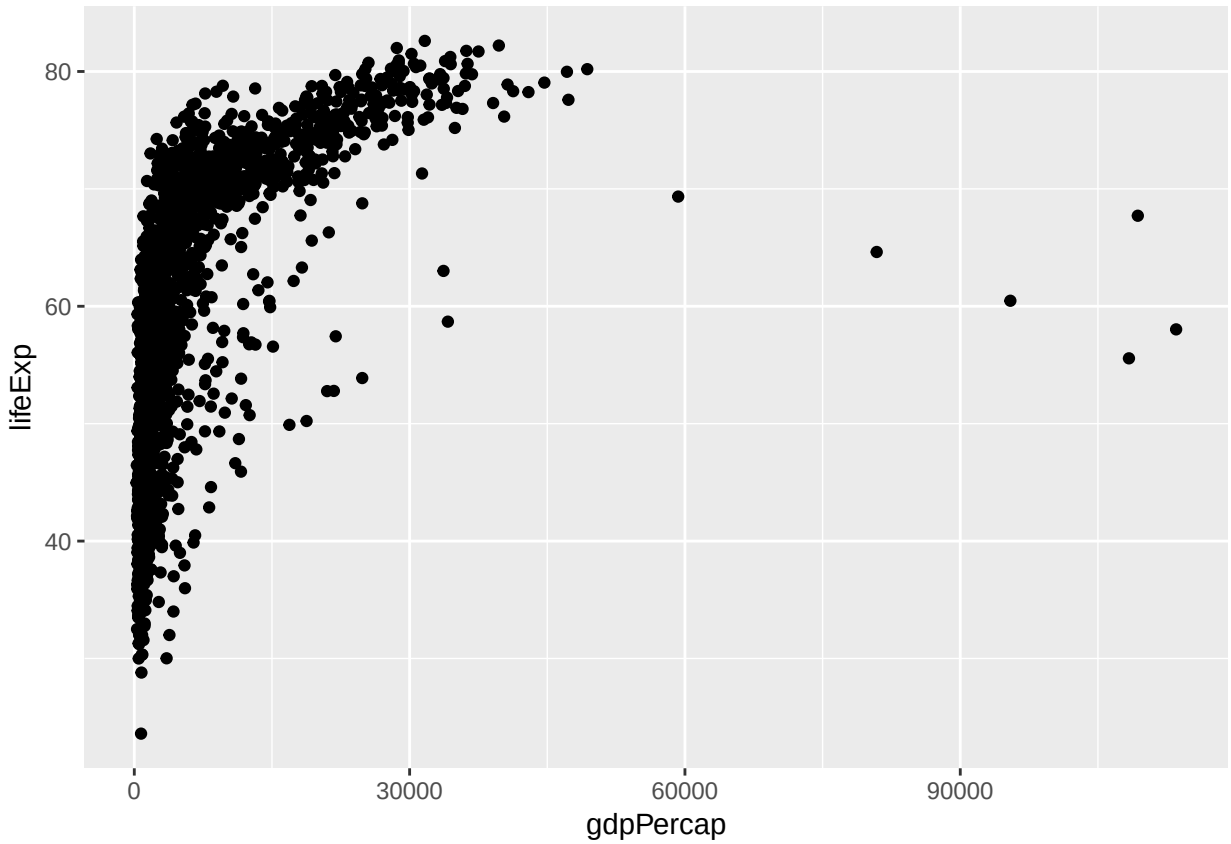
Second, by using faceting the data using `facet_wrap()` (we also use `coord_flip()` to make the bars horizontal):

```
myplot + geom_bar(stat="identity") + facet_wrap(~ DemClass) + coord_flip()
```

Scatterplots

Scatterplots are the simplest way to display bivariate data. **ggplot2** has several different geometries, including points and lines. To make a simple figure of life expectancy against per capita GDP from the gapminder dataset:

```
myplot = ggplot(gapdata, aes(x = gdpPercap, y = lifeExp))
myplot + geom_point()
```



Given the distribution of the GDP variable, a clearer display could be obtained by transforming it into log10 values. We can do this simply by adding the transformation in call to `ggplot()`:

```
ggplot(gapdata, aes(x = log10(gdpPercap), y = lifeExp)) + geom_point()
```

But we can also manipulate the axis scale directly, which keeps the original values on the axis labels:

```
myplot + geom_point() + scale_x_log10()
```

As this will be useful for future plots, will modify the underlying `ggplot` object:

```
myplot = myplot + scale_x_log10()
```

A further type of geometry (`geom_smooth()`) allows us to add trend lines to the plot.

```
myplot + geom_point() + geom_smooth()
```

The basic method uses either a type of spline or a local regression (loess) to fit the line. A simple linear fit can be added using the parameter `method='lm'`

As before, we can use an extra variable to identify different parts of the data set. The Gapminder data contains a column indicating which continent the observations came from. We'll use this first to color the points by changing the aesthetic:

```
myplot + geom_point(aes(color = continent))
```

If we now add a best fit line, however, we only get a single line for the whole dataset:

```
myplot + geom_point(aes(color = continent)) + geom_smooth(method='lm')
```

This is a little tricky to explain, but basically, we have only changed the aesthetic in the geometry of the points, rather than the underlying `ggplot` object. The simplest way around this is to remake the object with

the color included, then add the smoother:

```
myplot = ggplot(gapdata, aes(x = gdpPercap, y = lifeExp, color=continent)) + scale_x_log10()
myplot + geom_point() + geom_smooth(method='lm')
```

Note that if you drop the `geom_point()`, you will just obtain the trend lines.

We can also split the data into facets, which will automatically split the data when adding trend lines:

```
myplot = ggplot(gapdata, aes(x = gdpPercap, y = lifeExp)) + scale_x_log10()
myplot + geom_point() + facet_wrap(~ continent)
```

And we can add a linear fit by continent:

```
myplot + geom_point() + facet_wrap(~ continent) + geom_smooth(method='lm')
```

Finally, we can facet by both continent and year by using `facet_grid()`:

```
myplot + geom_point() + facet_grid(year ~ continent) + geom_smooth(method='lm')
```

Line plots are a good method for displaying data over time. We'll make a spaghetti diagram of the change in life expectancy for each country, broken out by continent. To do this, we tell the `geom_line()` function to use groupings based on the country name for each line:

```
myplot = ggplot(gapdata, aes(x = year, y = lifeExp))
myplot + facet_wrap(~ continent) + geom_line(aes(group=country, color=continent))
```

We can also look at different subsets of the data using a couple of different methods. Both of these examples used the R function `subset()` to extract just part of a dataframe; the first just extracts a single country, the second all countries from a list:

```
ggplot(subset(gapdata, country == "Zimbabwe"),
       aes(x = year, y = lifeExp)) + geom_line() + geom_point()
```

```
jCountries <- c("Canada", "Rwanda", "Cambodia", "Mexico")
ggplot(subset(gapdata, country %in% jCountries),
       aes(x = year, y = lifeExp, color = country)) + geom_line() + geom_point()
```

Saving your ggplot2 figures

In this section we'll make a figure and save it to a file. We'll start by building the figure, then add various elements, including axis labels and a title, and finally save it to a file using `ggsave()`. Note that as we make the figure, we update the 'ggplot" object each time:

Plot life expectancy against GDP with colors by continent

```
myplot = ggplot(gapdata, aes(x = gdpPercap, y = lifeExp, color=continent)) + scale_x_log10()
myplot = myplot + geom_point() + geom_smooth(method='lm')
```

Now add a title and labels:

```
myplot = myplot + ggtitle("Life Expectancy vs. GDP") +
  xlab("GDP") + ylab("Life Expectancy")
myplot
```

To save the image, you can dump it directly from the R plot window, but a better way is to use the `ggsave()` function. This provides more control over the final figure. To save as a png:

```
ggsave("lifeexp_vs_gdp.png", myplot)
```

As a pdf (jpeg, svg, eps also available):

```
ggsave("lifeexp_vs_gdp.pdf", myplot)
```

To adjust the output size:

```
ggsave("lifeexp_vs_gdp.png", myplot,
       width=6, height=4)
```

Themes

ggplot2 has a default theme with a gray background to the plot. There are a number of included themes that can simply applied to change this. Two of the built in themes are 'classic' and 'bw':

```
myplot + theme_classic()
myplot + theme_bw()
```

The addon package **ggthemes** includes a number of different themes, including themes inspired by the Economist, the Wall Street Journal and Nate Silver's blog 538. You will need to install and load this package, for this to work.

```
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 3.4.4
```

```
myplot + theme_calc()
myplot + theme_economist()
myplot + theme_wsj()
myplot + theme_fivethirtyeight()
```
