Lab: Introduction to ggplot

Introduction

In this lab we will delve more deeply into the ggplot2[1] graphics package and how we can use it to create elaborate graphs.

ggplot2 builds on the idea of a grammar of graphics[2]. The basic concept in the grammar of graphics is that we build a graph iteratively by specifying various components of it:

- We start by specifying the **data** for it. That would be a data frame.
- We use **aesthetics** to determine which variables will be shown on the graph and how. For example one variable could be assigned to correspond to the x axis, another to correspond to a color, one to correspond to size and so on.
- We use **geometries**, usually abbreviated as **geom**s to determine the kinds of shapes we want to have included in the graph (points, lines, bars etc). We can have multiple geoms on the same graph, creating various layers.
- We use **stats** to determine what statistic of the variable will be visualized. This often is done at the same time as specifying a geom, but it can also be done separately.
- We get to specify a **facet** to create separate panels based on values of one or more discrete variables.
- We can use **scales** to control how data values are mapped to visual values. Often default scales work just fine.
- We can use **annotations** to add text and similar elements to a graph.
- We use **themes** to control the overall graph appearance (axes ticks, labels, legend etc).

This will make more sense as we move along.

You can load ggplot2 itself directly, or you can load it as part of the hanoverbase package:

```
library(hanoverbase)
```

A first example: Color vs Price for diamonds

We will use the built-in diamonds data for this lab.

```
data(diamonds)
View(diamonds)     # Do the View in the console
?diamonds          # If you want to open up the dataset documentation
```

We would like to investigate how the color relates to the price of the diamonds. We start by defining the dataset to use:

```
ggplot(diamonds)
```

This will not show much (and empty plot), but it tells ggplot to use the diamonds dataset. Next we will specify the "aesthetics": In this case we will tell it to use color for the x axis and price for the y axis:

```
ggplot(diamonds) + aes(x=color, y=price)
```

[1] https://ggplot2.tidyverse.org/
[2] http://vita.had.co.nz/papers/layered-grammar.html

This should have produced axes, but it has plotted no data yet. This is because we have not specified any geom layer yet. We add this with another “plus”. We will spread this onto a new line to keep the lines short. To do that, you need each previous line to end with the plus.

```
ggplot(diamonds) +
    aes(x=color, y=price) +
    geom_boxplot()
```

Note that the price seems to increase as the color gets *worse* (J is worse than D). This sounds opposite to what it should be, and we’ll examine that later.

For now let us add one more layer, namely a point for each color, representing the mean price:

```
ggplot(diamonds) +
    aes(x=color, y=price) +
    geom_boxplot() +
    geom_point(stat="summary", fun.y=mean, color="red", fill="red", size=3)
```

We are here telling the graph to add “points” (geom_point) where it uses a specific stat function, namely “summary” which actually calls the function stat_summary. This specifies a statistical summary to compute at each grouping, and fun.y=mean specifies what computation should occur, namely the mean function for each individual color value. The remaining parameters customize the look and feel of the points.

Let’s also add lines connecting the means:

```
ggplot(diamonds) +
    aes(x=color, y=price) +
    geom_boxplot() +
    geom_line(stat="summary", group=1, fun.y=mean, size=1, linetype="dashed") +
    geom_point(stat="summary", fun.y=mean, color="red", fill="red", size=3)
```

As the price range is quite skewed, let us adjust the y scale to use a logarithmic scale:

```
ggplot(diamonds) +
    aes(x=color, y=price) +
    geom_boxplot() +
    geom_line(stat="summary", group=1, fun.y=mean, size=1, linetype="dashed") +
    geom_point(stat="summary", fun.y=mean, color="red", fill="red", size=3) +
    scale_y_log10()
```

This performed a log transform on the prices before it computed the boxplot, then drew a the graph with a logarithmic y-scale but showing the actual price values there.

Next, let us specify some tick marks. We do this by setting the breaks parameter of the scale transform:

```
ggplot(diamonds) +
    aes(x=color, y=price) +
    geom_boxplot() +
    geom_line(stat="summary", group=1, fun.y=mean, size=1, linetype="dashed") +
    geom_point(stat="summary", fun.y=mean, color="red", fill="red", size=3) +
    scale_y_log10(breaks=c(1000, 3000, 5000, 70000, 9000, 11000, 13000))
```

Now, let us use a custom theme to change the look and feel of the graph We’ll remove the vertical grid lines, set the background to while and the horizontal grid lines to light green:

```
ggplot(diamonds) +
    aes(x=color, y=price) +
    geom_boxplot() +
    geom_line(stat="summary", group=1, fun.y=mean, size=1, linetype="dashed") +
    geom_point(stat="summary", fun.y=mean, color="red", fill="red", size=3) +
    scale_y_log10(breaks=c(1000, 3000, 5000, 70000, 9000, 11000, 13000)) +
    theme_light() +
    theme(
        panel.grid.major.y = element_line(color="lightgreen"),
        panel.grid.major.x = element_blank()
    )
```

It's worth mentioning that you can save any part of this graph-building, and reuse it in other graphs. For example, we can create our own theme and store in a variable:

```
ourTheme <- theme_light() +
    theme(
        panel.grid.major.y = element_line(color="lightgreen"),
        panel.grid.major.x = element_blank()
    )

ggplot(diamonds) +
    aes(x=color, y=price) +
    geom_boxplot() +
    geom_line(stat="summary", group=1, fun.y=mean, size=1, linetype="dashed") +
    geom_point(stat="summary", fun.y=mean, color="red", fill="red", size=3) +
    scale_y_log10(breaks=c(1000, 3000, 5000, 70000, 9000, 11000, 13000)) +
    ourTheme
```

We can also save our own boxplot with the extra mean lines added. For that we will need to to store the three components that we want to include into a list. ggplot2 will unpack the list and put the components with pluses:

```
boxplotWithMeans <- list(
    geom_boxplot(),
    geom_line(stat="summary", group=1, fun.y=mean, size=1, linetype="dashed"),
    geom_point(stat="summary", fun.y=mean, color="red", fill="red", size=3)
)

priceLogScale <- scale_y_log10(breaks=c(1000, 3000, 5000, 70000, 9000, 11000, 13000))

ggplot(diamonds) +
    aes(x=color, y=price) +
    boxplotWithMeans +
    priceLogScale +
    ourTheme
```

Practice: Do a similar graph to compare price and clarity as well as price and cut. What would be your initial observations about the relations?

Scatterplots and faceting

Let us now examine the relation between price and carats:

```
ggplot(diamonds, aes(carat, price)) +
    geom_point()
```

This relation will look better in logarithmic scale:

```
ggplot(diamonds, aes(carat, price)) +
    geom_point() +
    scale_x_log10() +
    scale_y_log10()
```

Let us try to add a transparency to each point:

```
ggplot(diamonds, aes(carat, price)) +
    geom_point(alpha=0.3) +
    scale_x_log10() +
    scale_y_log10()
```

Or we can try to use single dots for each point rather than circles:

```
ggplot(diamonds, aes(carat, price)) +
    geom_point(shape=".") +
    scale_x_log10() +
    scale_y_log10()
```

A better approach is to use bin2d:

```
ggplot(diamonds, aes(carat, price)) +
    geom_bin2d(bins=50) +
    scale_x_log10() +
    scale_y_log10()
```

Now let us add a linear regression line with error estimates:

```
ggplot(diamonds, aes(carat, price)) +
    geom_bin2d(bins=50) +
    geom_lm(interval="prediction") +
    scale_x_log10() +
    scale_y_log10()
```

Or we might prefer a “smooth” line fit instead:

```
ggplot(diamonds, aes(carat, price)) +
    geom_bin2d(bins=50) +
    geom_smooth(color="red") +
    scale_x_log10() +
    scale_y_log10()
```

We will now use facets to create a panelled graph and investigate this relation based on a third variable (e.g. cut, clarity, or color):

```
ggplot(diamonds, aes(carat, price)) +
    geom_bin2d(bins=50) +
    geom_smooth(color="red") +
    scale_x_log10() +
    scale_y_log10() +
    facet_wrap(~cut)
```

We can do two factors:

```
ggplot(diamonds, aes(carat, price)) +
    geom_bin2d(bins=50) +
    geom_smooth(color="red") +
    scale_x_log10() +
    scale_y_log10() +
    facet_wrap(~cut + clarity)
```

With two factors, it is best to use `facet_grid`

```
ggplot(diamonds, aes(carat, price)) +
    geom_bin2d(bins=50) +
    geom_smooth(color="red") +
    scale_x_log10() +
    scale_y_log10() +
    facet_grid(cut ~ color) +
    ggtitle("Diamond price vs carat separated by cut and color")
```

Practice: Use these techniques to compare the width (y) and length (x) of diamonds. You may have to filter the data to get a better view.

Barcharts for categorical variables

We can use barcharts to investigate relations between the categorical variables. Let's start with individual barcharts:

```
ggplot(diamonds) +
    aes(x=color)
```

Question: How would we add color to the bars?

```
ggplot(diamonds) +
    aes(x=color) +
    geom_bar(aes(fill=color))
```

Let us briefly talk about choosing the color palette to use. This is basically setting a **scale** for the fill variable. It would be a discrete scale. There are four different ways of specifying a discrete color scale:

- Directly listing the colors: `scale_fill_manual(values=c("blue", "green", "orange", "red", "magenta", "purple", "pink"))`
- Using one of the preset "color brewer" palettes `scale_fill_brewer(palette="Set1")` (see options with `display.brewer.all()`)
- Using a grey-scale: `scale_fill_grey(start=0.3, end=0.7)`
- Using a scale based on the HCL color wheel: `scale_fill_hue`

Practice: Try adding one of these now.

Let us compare two categorical variables, say cut and clarity:

```
ggplot(diamonds) +
    aes(x=color, fill=clarity) +
    geom_bar()
```

Let's make it 100% stacked:

```
ggplot(diamonds) +
    aes(x=color, fill=clarity) +
    geom_bar(position="fill")
```

You can also try to combine this with a facet:

```
ggplot(diamonds) +
    aes(x=cut, fill=clarity) +
    geom_bar(position="fill") +
    facet_wrap(~color)
```

We can also add lines at the bar locations, to follow the trends:

```
ggplot(diamonds) +
    aes(x=cut, fill=clarity) +
    geom_bar(position="fill") +
    geom_line(aes(group=clarity), stat="count", position="fill") +
    facet_wrap(~color)
```

We can change the theme to “angle” the x axis labels:

```
ggplot(diamonds) +
    aes(x=cut, fill=clarity) +
    geom_bar(position="fill") +
    geom_line(aes(group=clarity), stat="count", position="fill") +
    facet_wrap(~color) +
    theme(
        axis.text.x = element_text(angle=-45, vjust=0.5)
    )
```

Pie Charts

Creating a pie chart is somewhat more elaborate. The main idea is as follows: We create a bar chart of one stacked bar, and then we make a change to polar coordinates, which wraps the y axis into a circle.

On a first step, let us create a bar chart of one stacked bar based on cut. Each bar chart however needs an x variable. We “fake” such a variable by using `factor(1)`, which creates a single common level (labeled `1`) for all the data rows.

```
ggplot(diamonds) +
    aes(x=factor(1), fill=cut) +
    geom_bar()
```

The current graph has extra spaces on the bar sides, which is the default behavior for bar charts as the bars are not supposed to connect to each other. In our case this extra space will present problems when we turn the graph to polar coordinates, we therefore eliminate the space by specifying the bar width. We will also remove the extraneous top and bottom spacing by setting the `expand` parameter of the `y` scale:

```
ggplot(diamonds, aes(x=factor(1), fill=cut)) +
    geom_bar(width=1) +
    scale_y_continuous(expand=c(0, 0))
```

Next we will remove the axis labeling and ticks, by specifying a theme:

```
ggplot(diamonds, aes(x=factor(1), fill=cut)) +
      geom_bar(width=1) +
      scale_y_continuous(expand=c(0, 0)) +
      theme_void()
```

Lastly, we turn into polar coordinates. We also add borders to the bars/slices:

```
ourPieChart <-
      ggplot(diamonds, aes(x=factor(1), fill=cut)) +
      geom_bar(width=1, color="white", size=0.2) +
      scale_y_continuous(expand=c(0, 0)) +
      theme_void() +
      coord_polar(theta="y")
```

To add percentages, a lot more work needs to be done. We can obtain the percentages as follows:

```
diamonds$cut %>% table() %>% prop.table()
```

To incorporate the percentages as slice labels, we need a bit more work. We start by creating a new data frame that computes the needed counts and proportions:

```
cutCounts <- diamonds %>% group_by(cut) %>%
                summarize(count=n(), percent= 100 * n()/nrow(diamonds))
cutCounts
```

We will want to format that percent so that it has fewer digits and a percent symbol after it:

```
cutCounts <- diamonds %>% group_by(cut) %>%
                summarize(count=n(), percent= 100 * n()/nrow(diamonds)) %>%
                mutate(percent = format(percent, digits = 2)) %>%
                mutate(percent = paste(percent, "%", sep=""))
cutCounts
```

Then we add a new geom to our graph, which draws from a specific dataset rather than the dataset used in the pie chart:

```
ourPieChart +
     geom_text(data=cutCounts,
                 aes(x=1.2, y=count, label=percent),
                 position=position_stack(vjust=0.5))
```

Solving the mystery

So let us try to now answer the question on why diamonds with worse color/clarity are actually more expensive. The answer lies in the effect of the diamond size (carat) on the price: Bigger diamonds are more expensive but also harder to get in the best color/clarity.

So we would like to condition on the size as we examine the relationship between price and color. In order to do that we would need to "bin" the carat variable to turn into a categorical variable. We can use one of the cut_ methods for this:

- cut_interval lets you specify how many bins to use.

- cut_width lets you specify how wide the bins will be.
- cut_number lets you specify a uniform bin frequency.

First let us restrict the carat range to exclude outliers. Here is the carat range:

```
ggplot(diamonds, aes(x=carat)) + geom_histogram(bins=40)
```

We will restrict to diamonds up to 2.5 carats. Let us look at how these are distributed amongst the different colors:

```
diamonds %>% filter(carat <= 2.5) %>%
    ggplot(aes(x=carat, fill=color)) +
        geom_density() +
        scale_fill_brewer(palette="Blues")
```

This graph shows us the overlapping density curves for each color. A better view will stack the curves:

```
diamonds %>% filter(carat <= 2.5) %>%
    ggplot(aes(x=carat, fill=color)) +
        geom_density(position="fill") +
        scale_fill_brewer(palette="Blues")
```

We can see from this graph that most of the best color diamonds are small.

Practice: What if we used clarity instead of color?

Now we will facet the graph based on a carat grouping, using the cut_interval method:

```
diamonds %>% filter(carat <= 2.5) %>%
    ggplot(aes(x=color, y=price)) +
        geom_boxplot() +
        facet_wrap(~cut_interval(carat, 6))
```

The vast difference in value ranges on the various panels makes it hard to clearly see the patterns. To that end, we would like to set each panel to have independent scales:

```
diamonds %>% filter(carat <= 2.5) %>%
    ggplot(aes(x=color, y=price)) +
        geom_boxplot() +
        facet_wrap(~cut_interval(carat, 6), scales="free")
```