Post 1: Communication through the R Package ggplot2

Kate Li

October 29, 2017

Introduction

> Having a good graph is crucial to any statistical work. Not only is it the most colorful and best looking part, it is also worth a thousand words. A good graph will convey the main idea of the analysis in a simple way that even audiences without background information can understand. ggplot2 is based on Grammar of Graphics, you just need to specify variables, aesthetics, and graphical primitives. However, you need to put in a little more effort for an excellent expository graphic. ggplot2, package written by Hadley Wickham, is designed to work iteratively, a good ggplot2 graph is composed of multiple layers of aesthetics for geometric elements and statistical transformation, which makes it different than simple exploratory graphs. Below are different settings can be used to modify your graphs, assuming you have completed the data exploration steps.

I will use the data set that came with the ggplot2 package: mpg. It includes information about the fuel economy of popular car models in 1999 and 2008, collected by the US Environmental Protection Agency, http://fueleconomy.gov.

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggrepel)
library(viridis)
```

```
## Loading required package: viridisLite
```

```
mpg
```

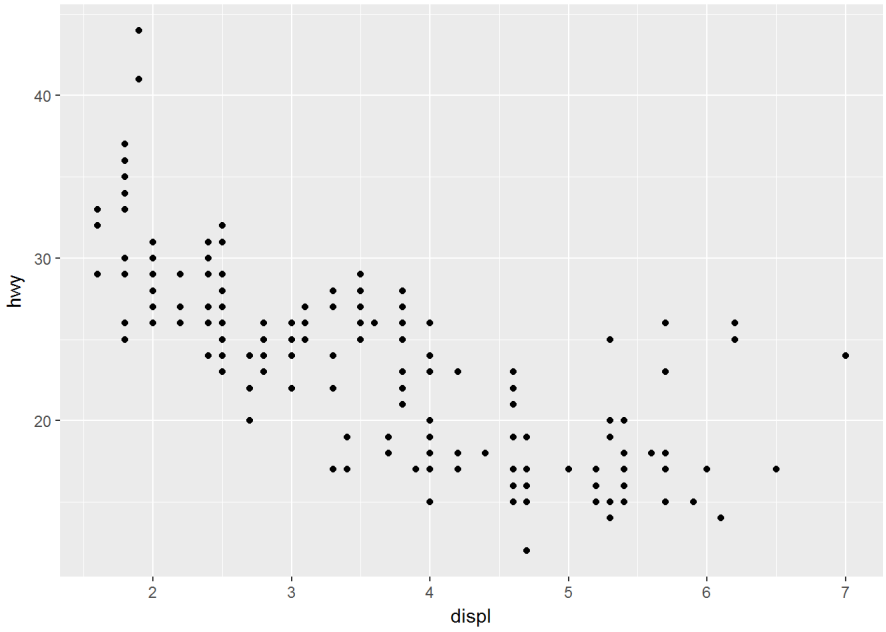
```
## # A tibble: 234 x 11
##    manufacturer      model displ  year   cyl      trans   drv   cty   hwy
##           <chr>      <chr> <dbl> <int> <int>      <chr> <chr> <int> <int>
##  1         audi         a4   1.8  1999     4   auto(l5)     f    18    29
##  2         audi         a4   1.8  1999     4 manual(m5)     f    21    29
##  3         audi         a4   2.0  2008     4 manual(m6)     f    20    31
##  4         audi         a4   2.0  2008     4   auto(av)     f    21    30
##  5         audi         a4   2.8  1999     6   auto(l5)     f    16    26
##  6         audi         a4   2.8  1999     6 manual(m5)     f    18    26
##  7         audi         a4   3.1  2008     6   auto(av)     f    18    27
##  8         audi a4 quattro   1.8  1999     4 manual(m5)     4    18    26
##  9         audi a4 quattro   1.8  1999     4   auto(l5)     4    16    25
## 10         audi a4 quattro   2.0  2008     4 manual(m6)     4    20    28
## # ... with 224 more rows, and 2 more variables: fl <chr>, class <chr>
```

The variables are mostly self-explanatory:

- cty and hwy record miles per gallon (mpg) for city and highway driving.
- displ is the engine displacement in liters.
- drv is the drivetrain: front wheel (f), rear wheel (r) or four wheel (4).
- model is the model of car. There are 38 models, selected because they had a new edition every year between 1999 and 2008.
- class (not shown), is a categorical variable describing the “type” of car: two-seater, SUV, compact, etc.

```
#This would be a basic exploratory graph

ggplot(mpg, aes(x = displ, y = hwy)) +
geom_point()
```



ggplot2 Settings

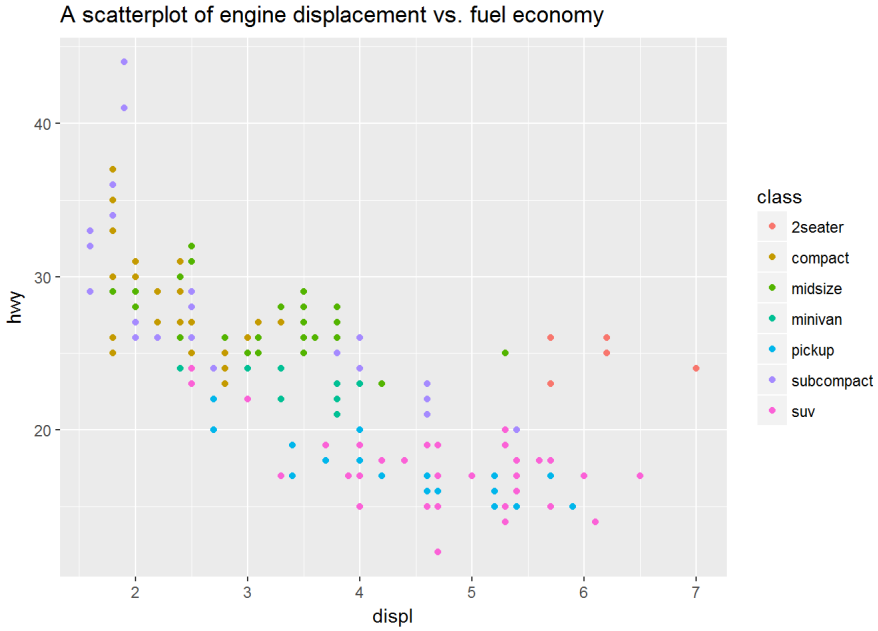
> I will not explore every option of every geom, but instead to show the most important tools for a given task. If you need more information about individual geoms, please see the documentation in Help.

Labels

> The easiest step with improving a graph is to add labels with the `labs()` function.

Here is an example adding a graph title:

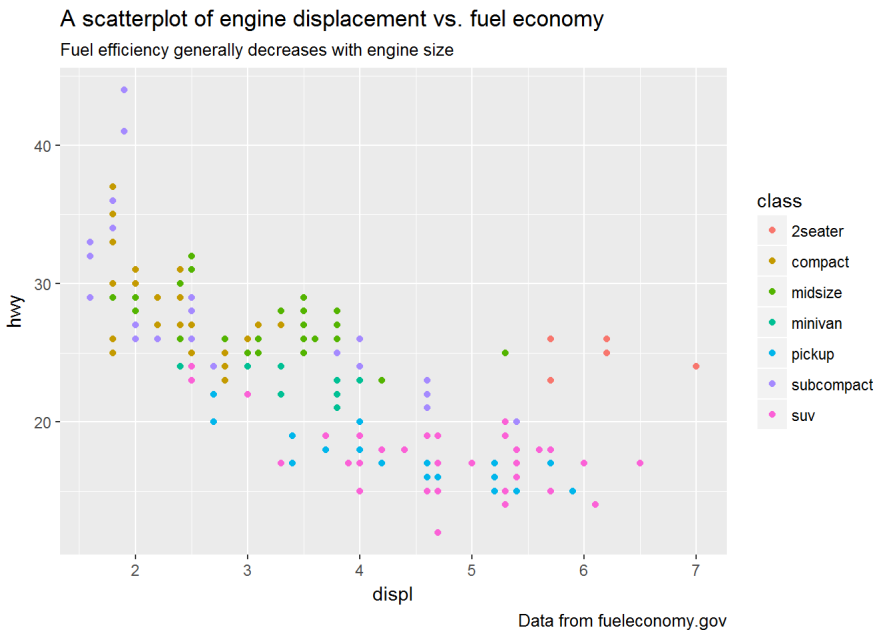
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  labs(title = "A scatterplot of engine displacement vs. fuel economy")
```



Often, we need more information to be shown in addition to the title, so we need to add more labels:

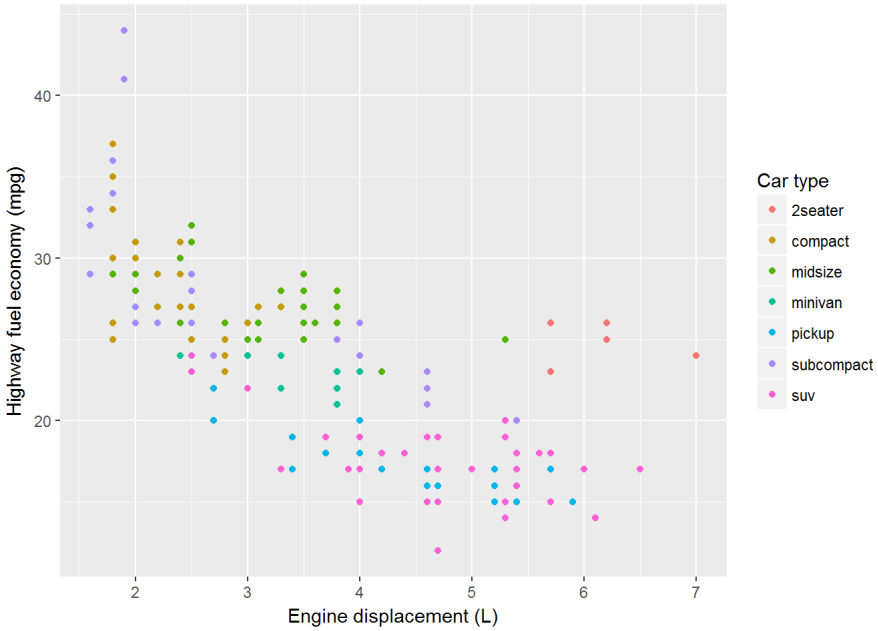
- `subtitle` adds additional detail in a smaller font beneath the title.
- `caption` adds text at the bottom right of the plot, often used to describe the source of the data.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  labs(
    title = "A scatterplot of engine displacement vs. fuel economy",
    subtitle = "Fuel efficiency generally decreases with engine size",
    caption = "Data from fueleconomy.gov"
  )
```



Next, we can replace the axis and legend titles to be more informative. One tip is to replace short variable names with more detailed descriptions as well as the units.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  labs(
    x = "Engine displacement (L)",
    y = "Highway fuel economy (mpg)",
    colour = "Car type"
  )
```



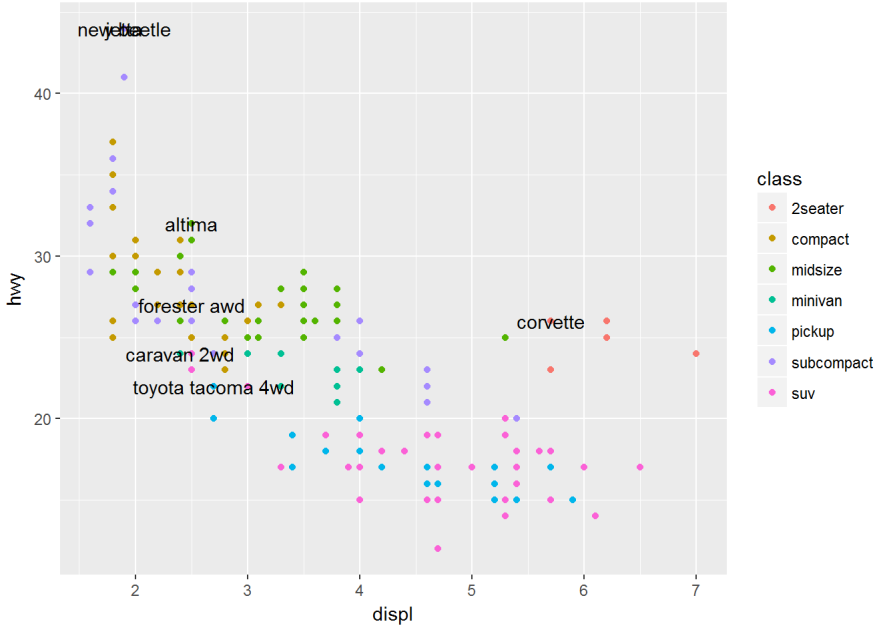
Annotations

> All the plot points in the graph can seem a bit overwhelming as there is no center of attention. We can then add annotations to the graph with the `geom_text()` setting. This adds labels at the specified x and y positions to the plot and makes the audience focus less of each individual points.

```
best_in_class <- mpg %>%
  group_by(class) %>%
  filter(row_number(desc(hwy)) == 1)
```

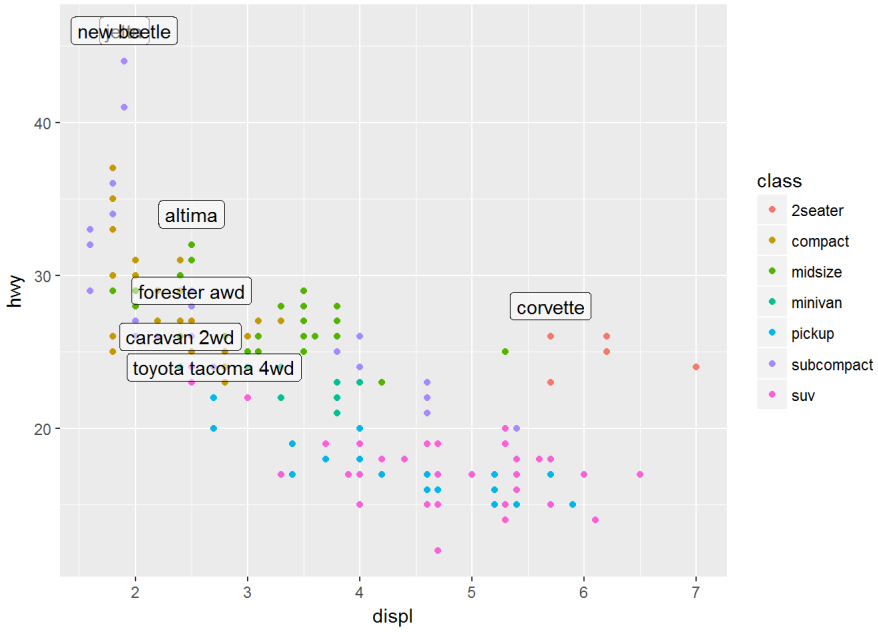
```
## Warning: package 'bindrcpp' was built under R version 3.3.3
```

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_text(aes(label = model), data = best_in_class)
```



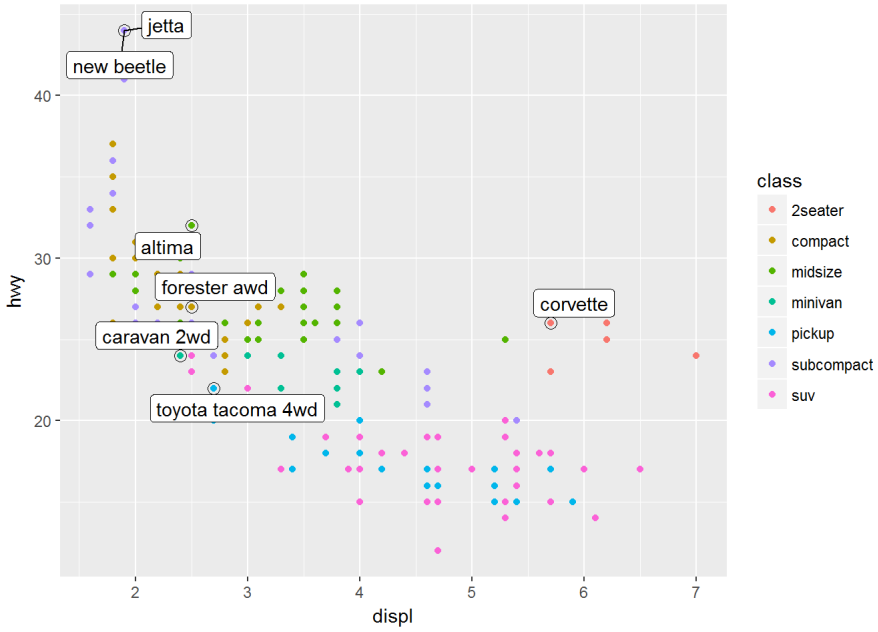
Note that the labels did not come out very clean. They overlap with the points and with one another. We can fix this by switching to another setting, `geom_label()`, which draws a rectangle around the label. I also used `nudge_y` to move the labels so that they do not sit directly on top of the points (`nudge_x` works just the same, on the x scale).

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_label(aes(label = model), data = best_in_class, nudge_y = 2, alpha = 0.5)
```



In addition, we can use the help of a ggplot2 extension package, ggrepel by Kamil Slowikowski, to resolve the problem. This package will automatically adjust labels so that they don't overlap.

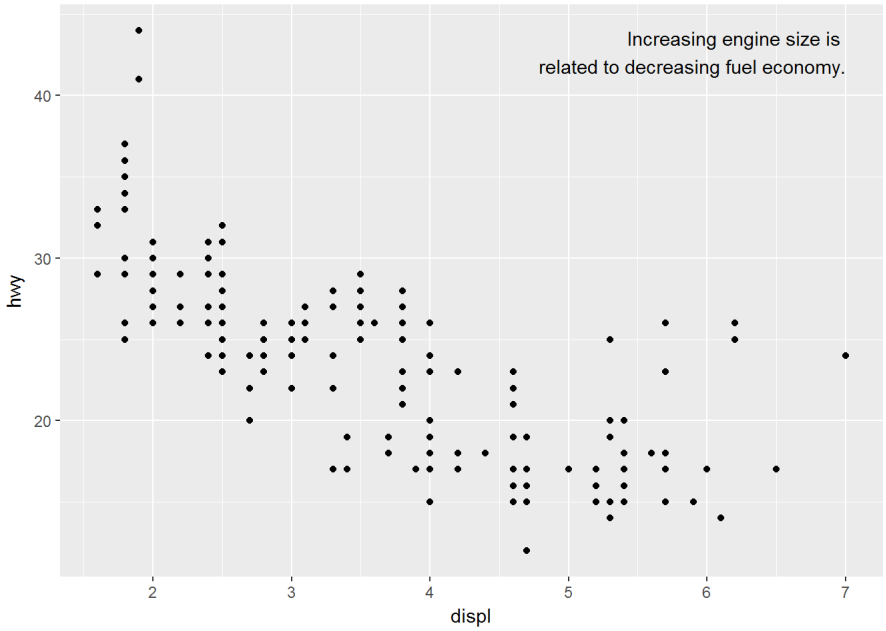
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_point(size = 3, shape = 1, data = best_in_class) +
  ggrepel::geom_label_repel(aes(label = model), data = best_in_class)
```



`geom_text` is still useful in adding text lines on the graph. Because you can control the x and y positions of the text, you can place anywhere on the graph you see fit. For example, I will add a text to serve as the title of the graph. I want to put it on the top right corner of the graph since there is an opening there, but I need to find out the x and y positions first. Another trick is to use check `overlap = TRUE` in the aesthetics, as this will omit a label that overlaps with an existing label.

```
label <- mpg %>% #This computes the maximum values of x and y.
  summarise(
    displ = max(displ),
    hwy = max(hwy),
    label = "Increasing engine size is \nrelated to decreasing fuel economy."
  )

ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_text(aes(label = label), data = label, vjust = "top", hjust = "right")
```



You can use `hjust` and `vjust` to control the alignment of the label by using any combination of “top”, “bottom”, “left”, “right”, and “center”.

Scales

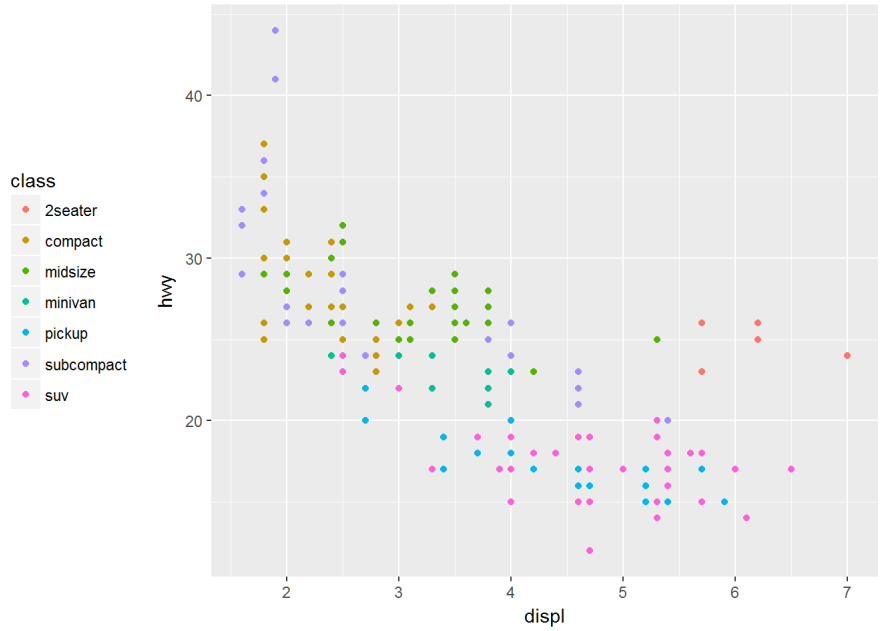
> The scales map values in the data space to values in an aesthetic space, that includes color, size, and shape. The default scales are named according to the type of variable they align with: continuous, discrete, datetime, or date. Depending on your specific data type, you may want to change some of the parameters of the default scale. This allows you to do things like change the breaks on the axes, or the key labels on the legend. You might want to replace the scale altogether if you know more about the data.

One way that you can improve the appearance of the graph is the edit the legend layout. To control the overall position of the legend, you need

to use `legend.position` under the `theme()` setting because this is the setting that controls the non-data parts of the plot. Here is an example with different ways you can place the legend:

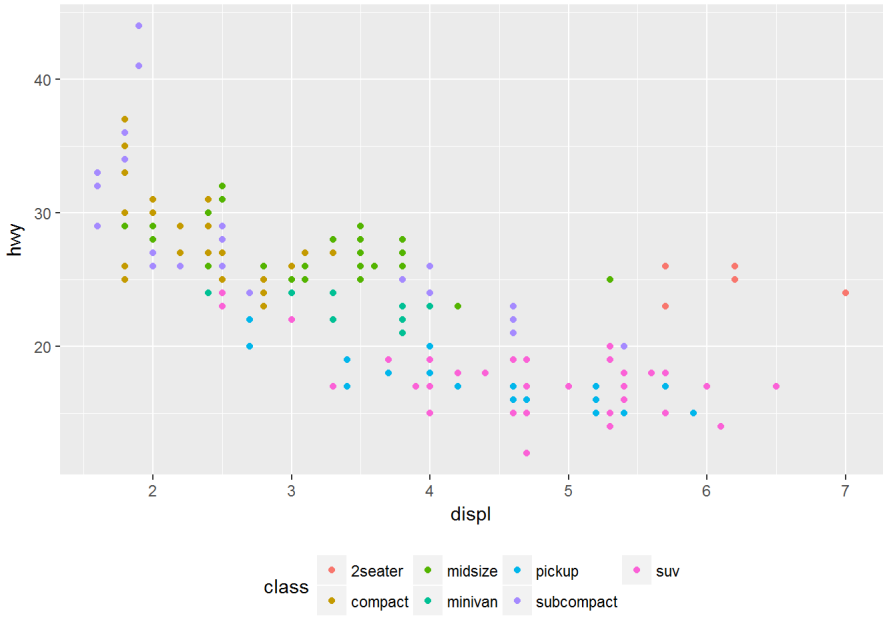
```
base <- ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class))

base + theme(legend.position = "left")
```

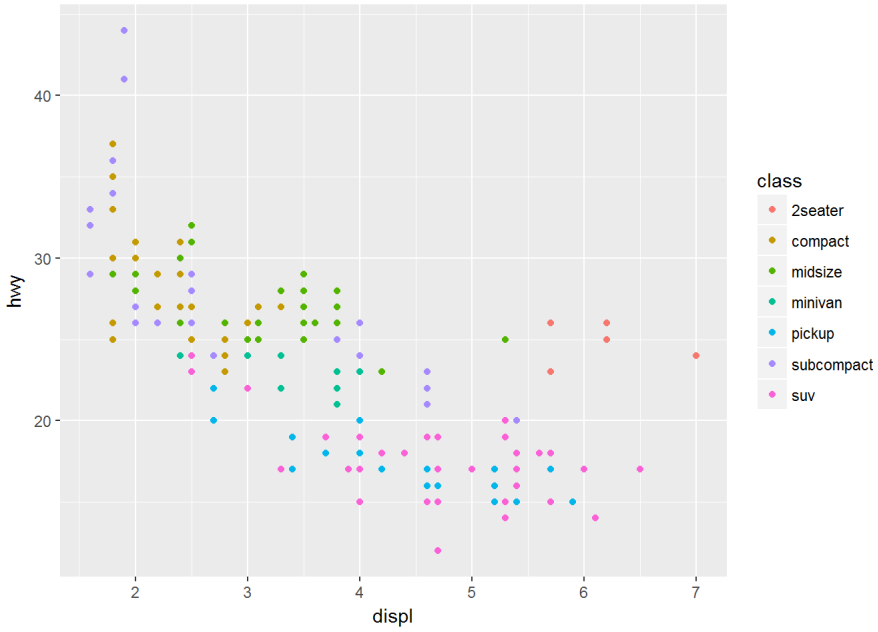



```
base + theme(legend.position = "top")
```


```
base + theme(legend.position = "bottom")
```



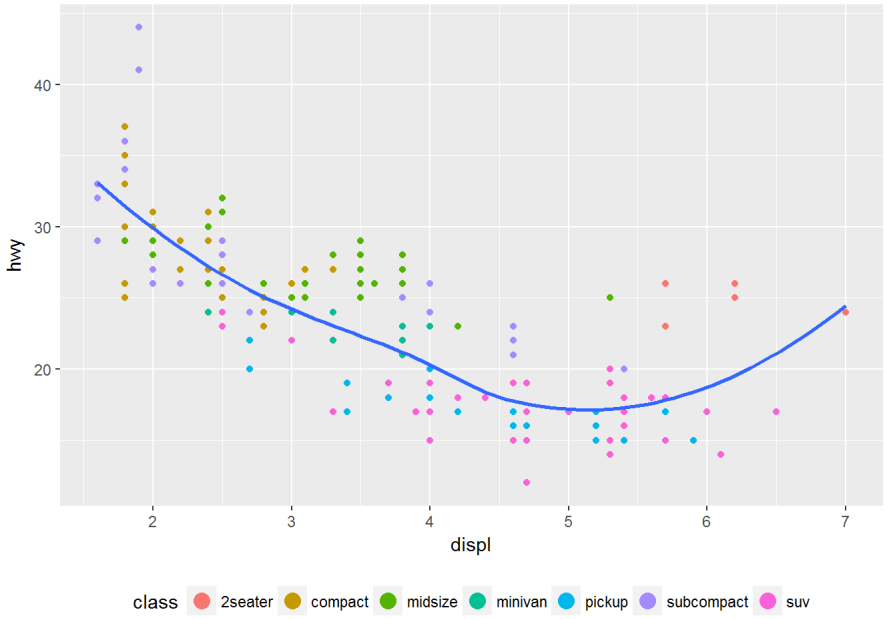

```
base + theme(legend.position = "right") # the default
```



There is no “best” position for the legend, it really depends on how your data look. If you think that your graph looks best without a legend, you can use `legend.position = "none"`. In addition, you can control the display of individual legends: use `guides()` along with `guide_legend()`. In the following example, `nrow` changes the number of rows in the legend and `override.aes` overrides one of the aesthetics to make the points bigger.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_smooth(se = FALSE) +
  theme(legend.position = "bottom") +
  guides(colour = guide_legend(nrow = 1, override.aes = list(size = 4)))
```

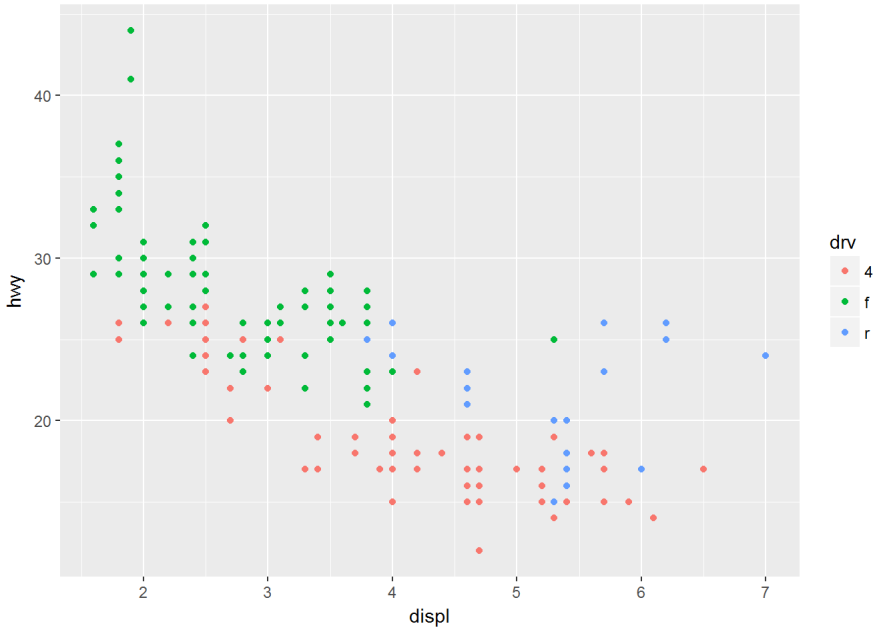
```
## `geom_smooth()` using method = 'loess'
```



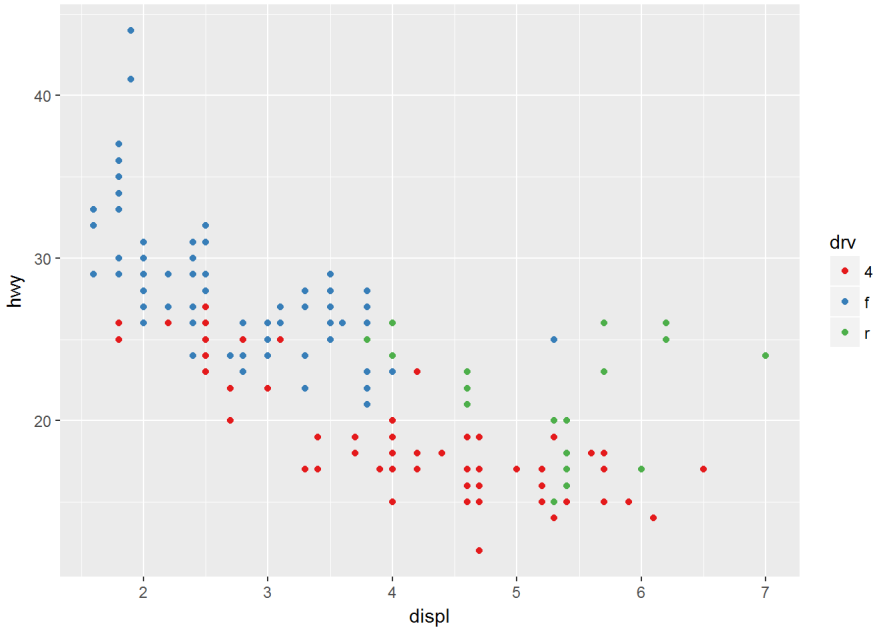
Colors

> ggplot2 by default selects colors using the `scale_color_hue()`, which selects the most distinct colors possible while keeping lightness constant. In order to make a graph stand out, you may want to specify a different color selection to add distinctness to your graph. The ColorBrewer scales is a highly recommended alternative because it works better for people with common types of color blindness. The two plots below look similar, but the second plot has a strong contrast between the shades of red and green that the points on the right can be distinguished even by people with red-green color blindness.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv))
```



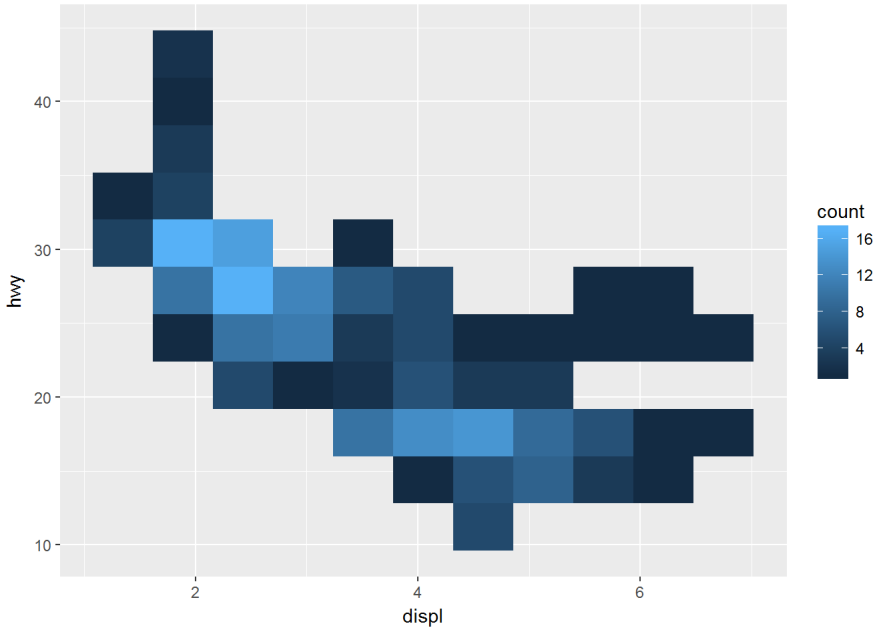

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv)) +
  scale_colour_brewer(palette = "Set1")
```



The ColorBrewer scales are documented online at http://colorbrewer2.org/ and made available in R via the RColorBrewer package, by Erich Neuwirth.

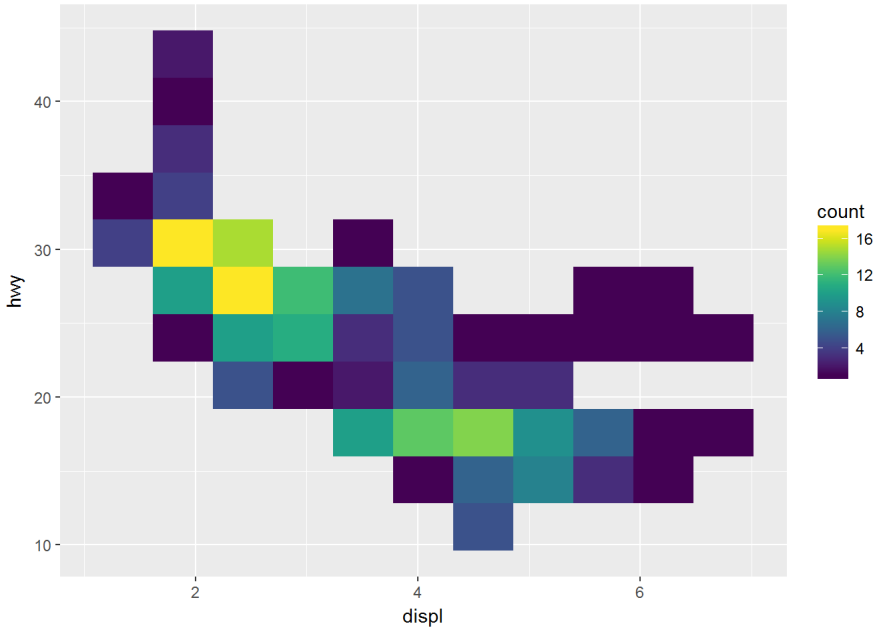
The color of points in a plot may not be that big of a deal, but in other types of graph it has a more important role than just giving aesthetics. Let's change it up use the the geom `geom2d_bin()` instead of `geom_point`. `geom2d_bin()` counts the number of points in a given 2d spatial area and this is represented by the color intensity, much like a thematic map.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_bin2d(bins=10)
```

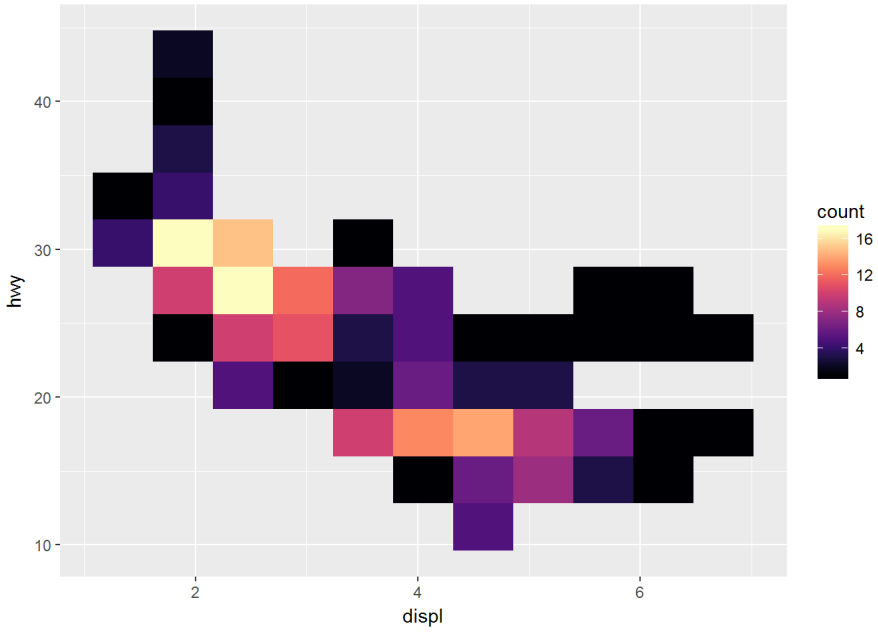


This graph does the job of showing a high concentration of points around (2, 30). However, as noted, the default ggplot2 color palette is boring. Here, we can use the viridis package, which provides a set of 4 high-contrast palettes, to change up the colors.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_bin2d(bins=10) +
  scale_fill_viridis(option="viridis") #the default "viridis" palette
```




```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_bin2d(bins=10) +
  scale_fill_viridis(option="magma")
```

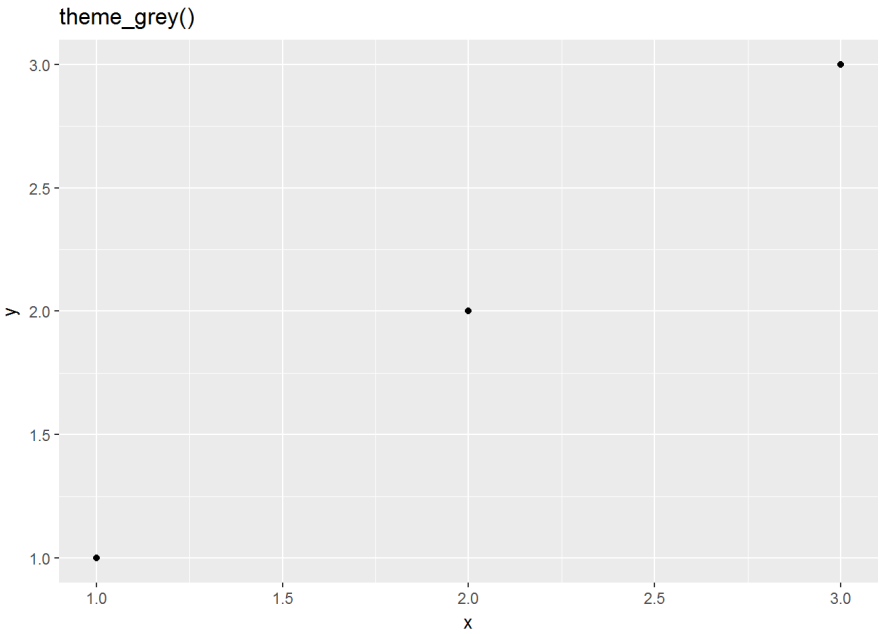


Themes

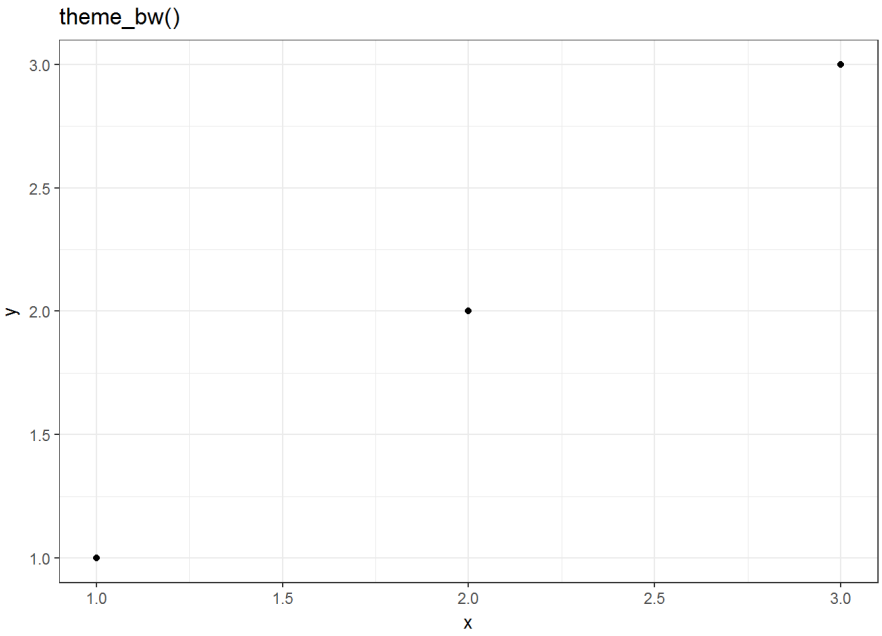
> Finally, you can customize the non-data elements of your plot with the `theme()` setting. Themes don't change the perceptual properties of the plot, but they do help you make the plot aesthetically pleasing or match an existing style guide. You can customize things like fonts, ticks, panel strips, and backgrounds, or you can use the default themes in ggplot2. There are only eight themes, but you can get more themes by installing add-on packages like [ggthemes] (https://github.com/jrnold/ggthemes) . As Wickham points out, the theming system is composed of four main components:

- Theme elements specify the non-data elements that you can customize. For example, the `plot.title` element changes the appearance of the plot title; `axis.ticks.x` , the ticks on the x axis; `legend.key.height` , the height of the keys in the legend.
- Each element is associated with an element function, which describes the visual properties of the element. For example, element `text()` sets the font size, color, and face of a text element like `plot.title` .
- The `theme()` function, like `theme(plot.title = element text(colour = "red"))` , allows you to override the default theme elements.
- Complete themes, like theme `grey()` , changes all of the theme elements to fit with the theme.

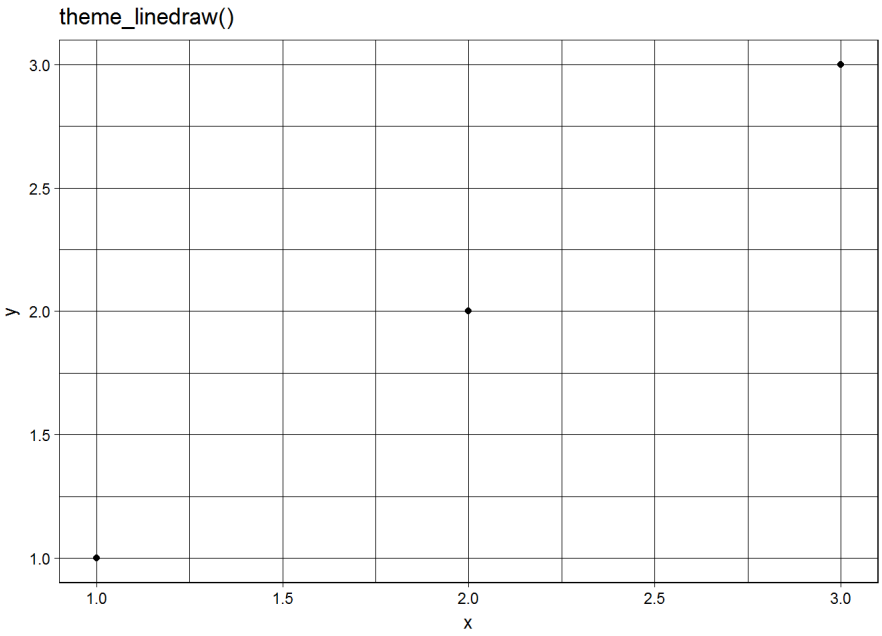
```
df <- data.frame(x = 1:3, y = 1:3)
base <- ggplot(df, aes(x, y)) + geom_point()
base + theme_grey() + ggtitle("theme_grey()")
```



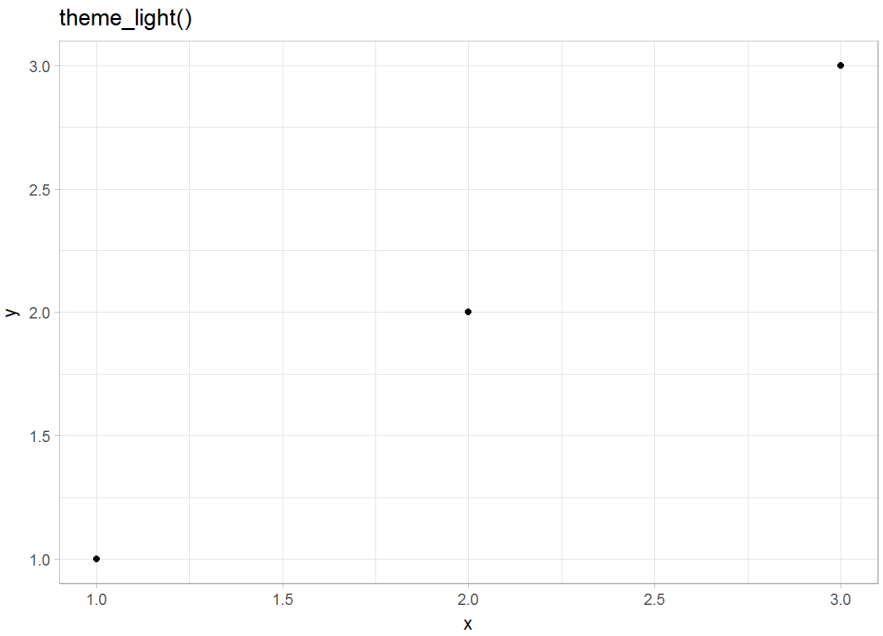

```
base + theme_bw() + ggtitle("theme_bw()")
```



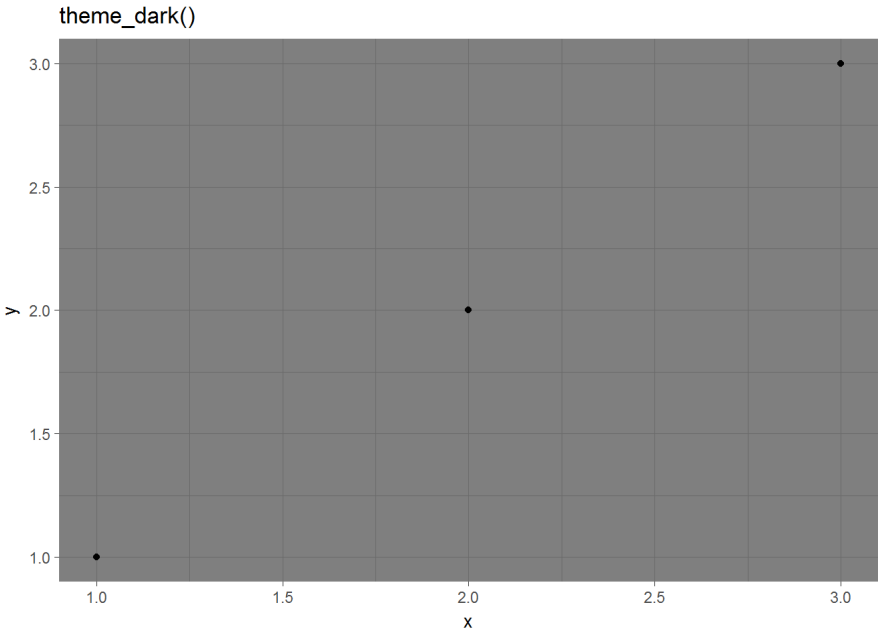

```
base + theme_linedraw() + ggtitle("theme_linedraw()")
```



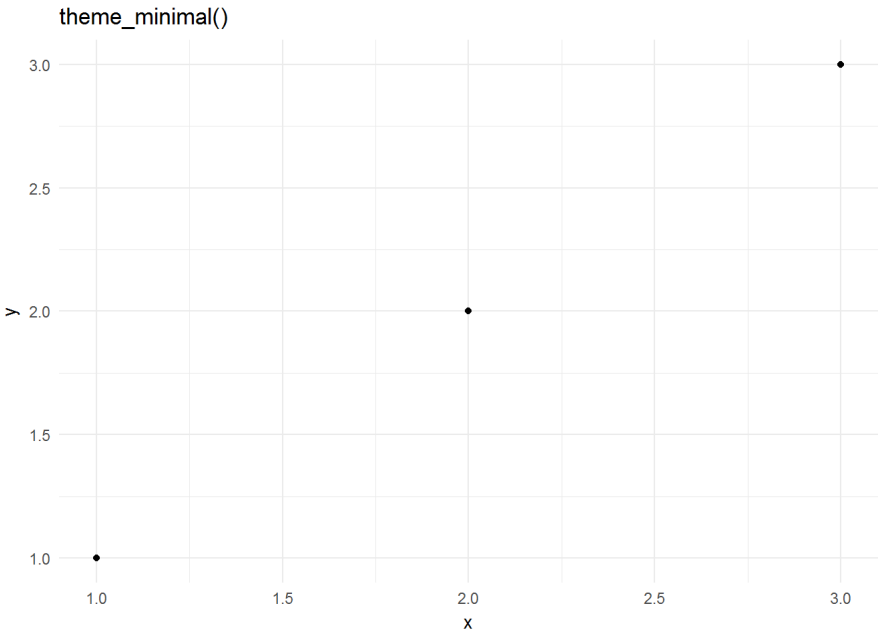
```
base + theme_light() + ggtitle("theme_light()")
```




```
base + theme_dark() + ggtitle("theme_dark()")
```




```
base + theme_minimal() + ggtitle("theme_minimal()")
```



In the End

> This is just the basics ways to make a beautiful graph. There are so many controls that you can play around with in ggplot2 that it may take a while to create a graph that best represents your data. However, you can follow guidlines and listen to critiques to improve, just never use the `qplot()` function. Also check out other packages to use alongside ggplot2 for even cooler graphs.

References:

DataCamp, director. Learn R: An Introduction to ggplot2. Youtube, 9 Nov. 2016. http://www.youtube.com/watch?v=YxKr2a-Y1WE&list=PLjgj6kdf_snaBCTJEi53DvRVgOuVbzyk

Prabhakaran, Selva. “The Complete ggplot2 Tutorial.” r-Statistics.co. http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html

Smith, David. “10 Tips for Making Your R Graphics Look Their Best.” Revolutions. http://blog.revolutionanalytics.com/2009/01/10-tips-for-making-your-r-graphics-look-their-best.html

Wickham, Hadley, and Carson Sievert. ggplot2: Elegant Graphics for Data Analysis. Springer, 2016.

Wickham , Hadley. “Create Elegant Data Visualisations Using the Grammar of Graphics . ggplot2.” Create Elegant Data Visualisations Using the Grammar of Graphics . ggplot2, Tidyverse. <ggplot2.tidyverse.org> http://ggplot2.tidyverse.org/

Wickham, Hadley. “R Graph Catalog.” R Graph Catalog. http://shiny.stat.ubc.ca/r-graph-catalog/

Woolf, Max. “How to Make High Quality Data Visualizations for Websites With R and ggplot2.” Minimaxir | Max Woolf’s Blog. http://minimaxir.com/2017/08/ggplot2-web/