Visualizing Data

Ismail Fadeli

August 19, 2021

Data Visualization in R

In order to be able to visualize data, we have to install **tidyverse** package in R Studio. Here's the code for that:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Viewing Dataset

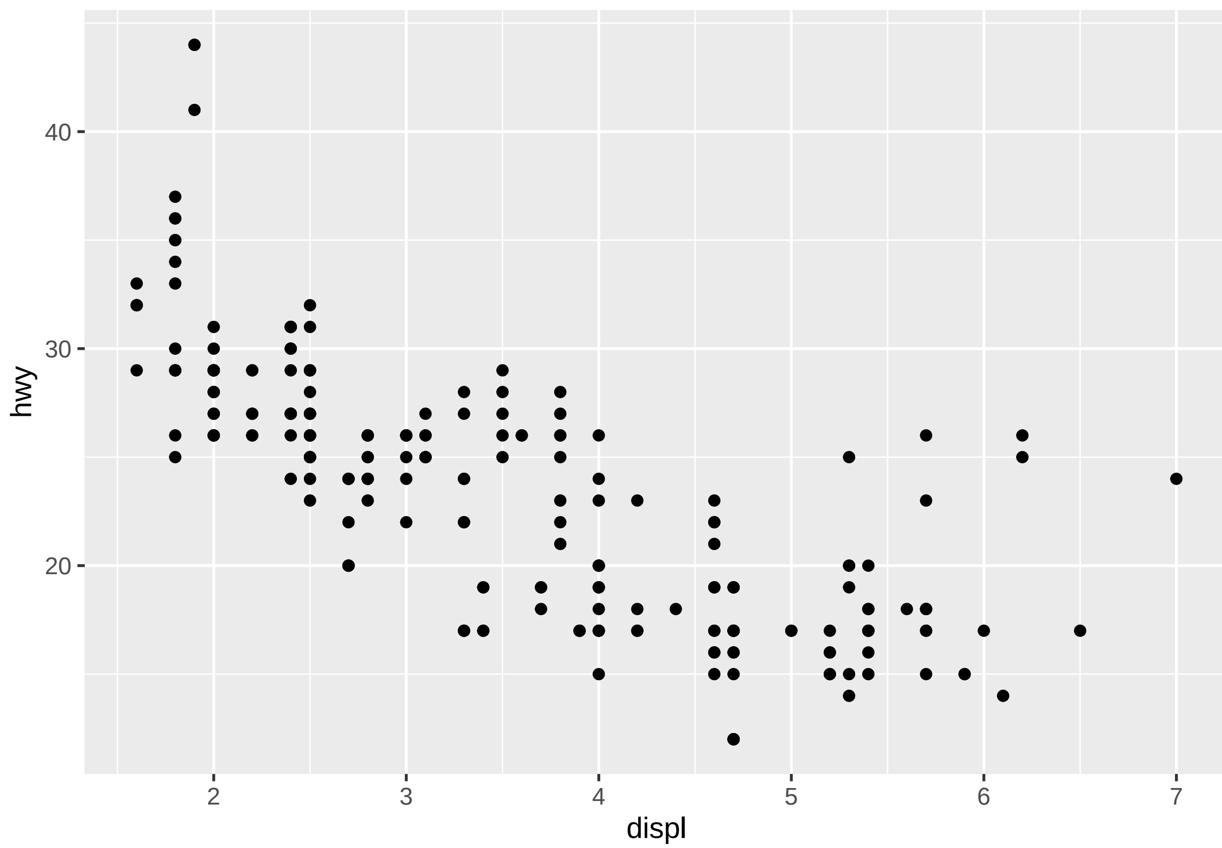
To visualize cars miles per gallon dataset from the USA datacenter:

```
mpg
```

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

To plot **mpg** data, we need to run this code to put **displ** into x-axis and **hwy** into the y-axis.

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



The visualized graph shows a negative relationship between the engine size and fuel efficiency. Bigger engine size uses more fuel than lower engine sizes to travel the same distance.

With **ggplot**, you begin a plot with the function **ggplot()** creates a coordinate system that you can add layers to. The first argument of **ggplot** is the dataset to use in the graph. So **ggplot(data = mpg)** creates an empty graph. We can complete the graph by adding more layers to **ggplot()**. The function **geompoint()** creates a layer to your plot, which creates a scatterplot. The mapping argument is always paired with **aes()**, and the x and y arguments of aes() specify which variables to map to the x-axes and y-axes.

Exercice

1. When running the code **ggplot(data = mpg)**, we see an empty graph:
2. The **mtcars** dataset has 32 rows and 11 columns.

```
mtcars
```

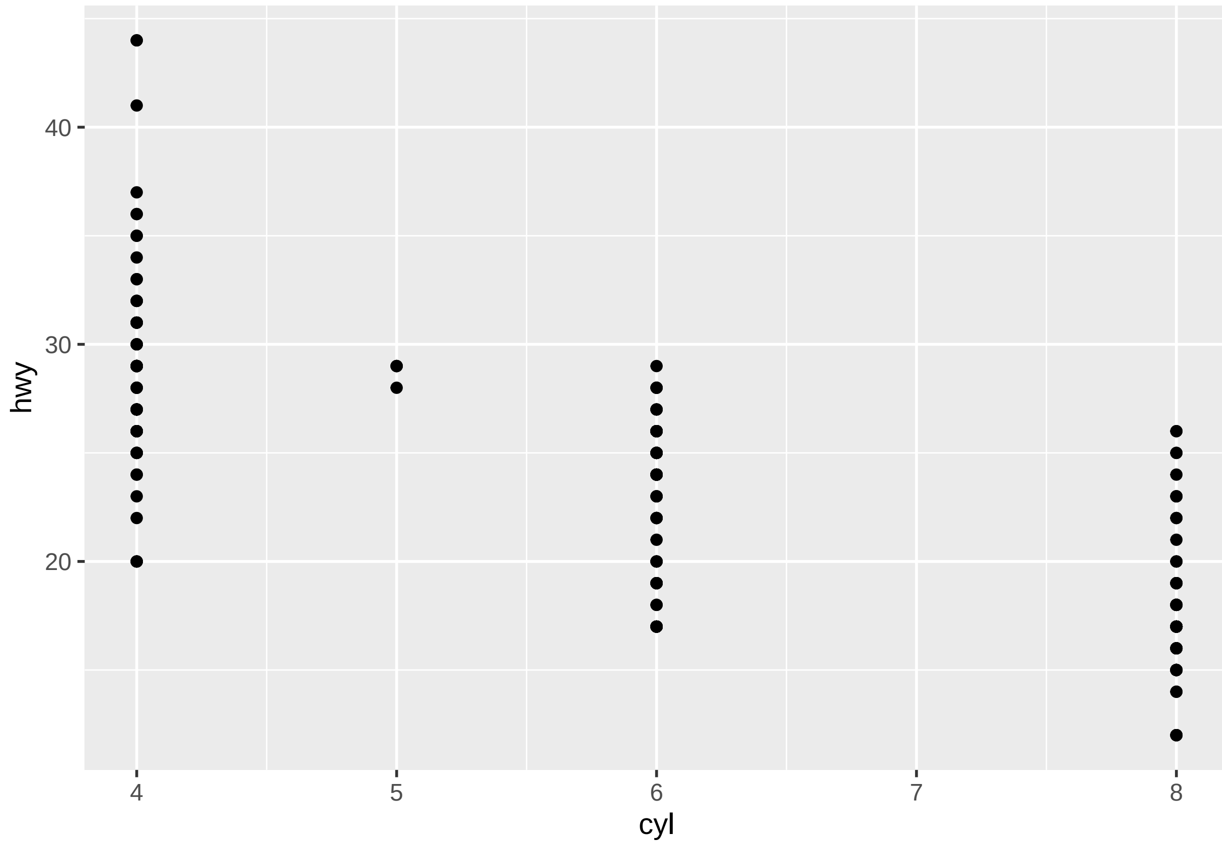
```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
```

```
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

3. The **drv** variable describes the type of the car, meaning it is either a front wheel drive, rear wheel, or four wheel drive.

4. The code for a scatter plot of **hwy** versus **cyl**:

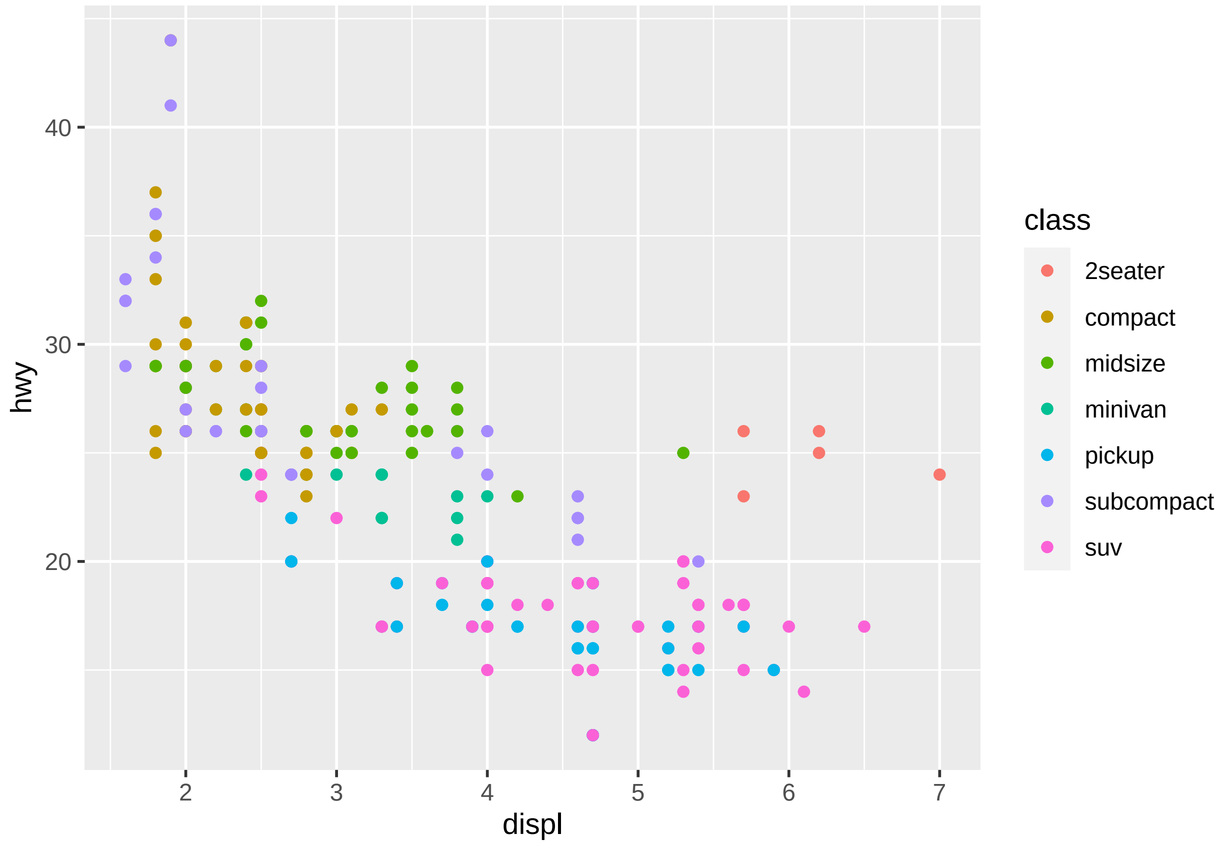
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = cyl, y = hwy))
```



Aesthetic Mappings

You can show information about your data by mapping the aesthetics in your plot to the variables in your dataset. For example, you can map the colors of your points to the class variable to reveal the class of each car:

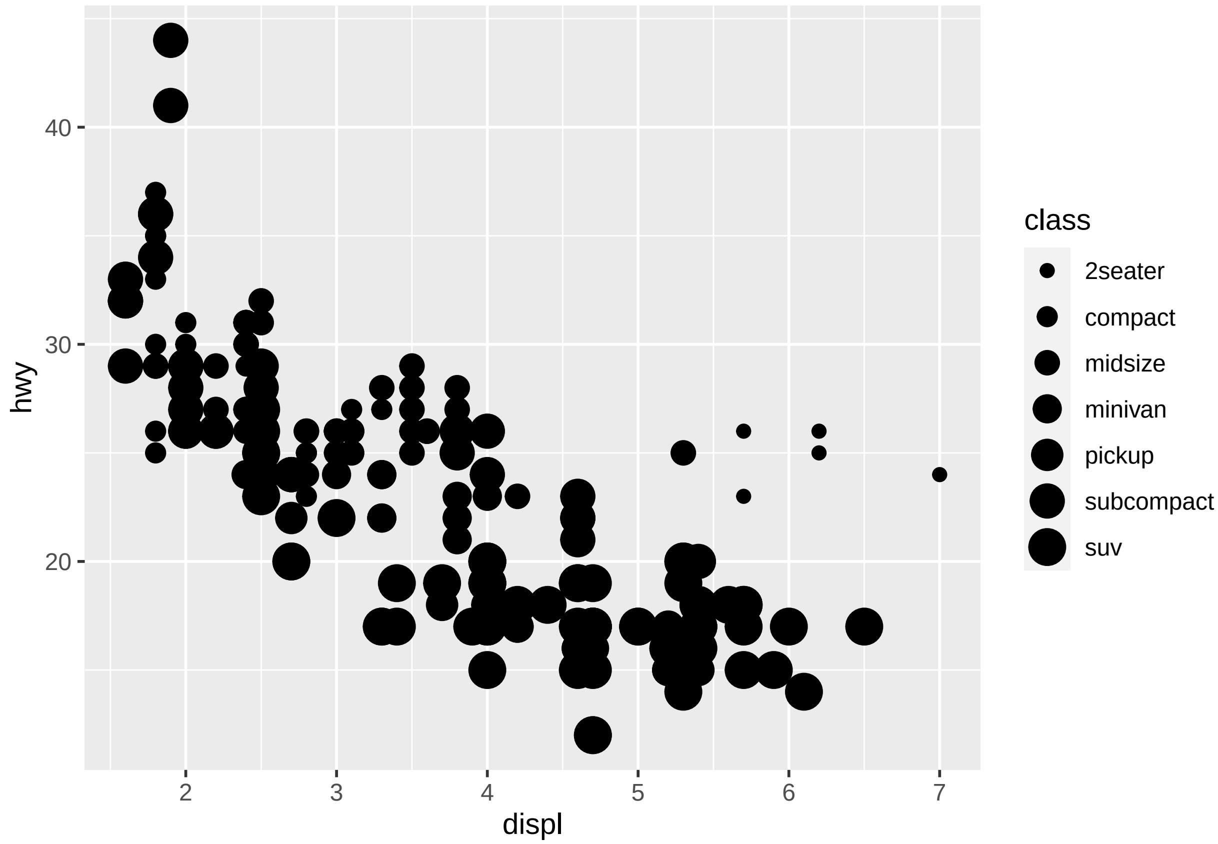
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



In the previous example, we mapped the class to the color aesthetic, but we could have mapped class to the size aesthetic in the same way. In this case, the exact size of each point would reveal its class affiliation. We get a warning here, because mapping an unordered variable (class) to an ordered variable (size) is not a good idea:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
```

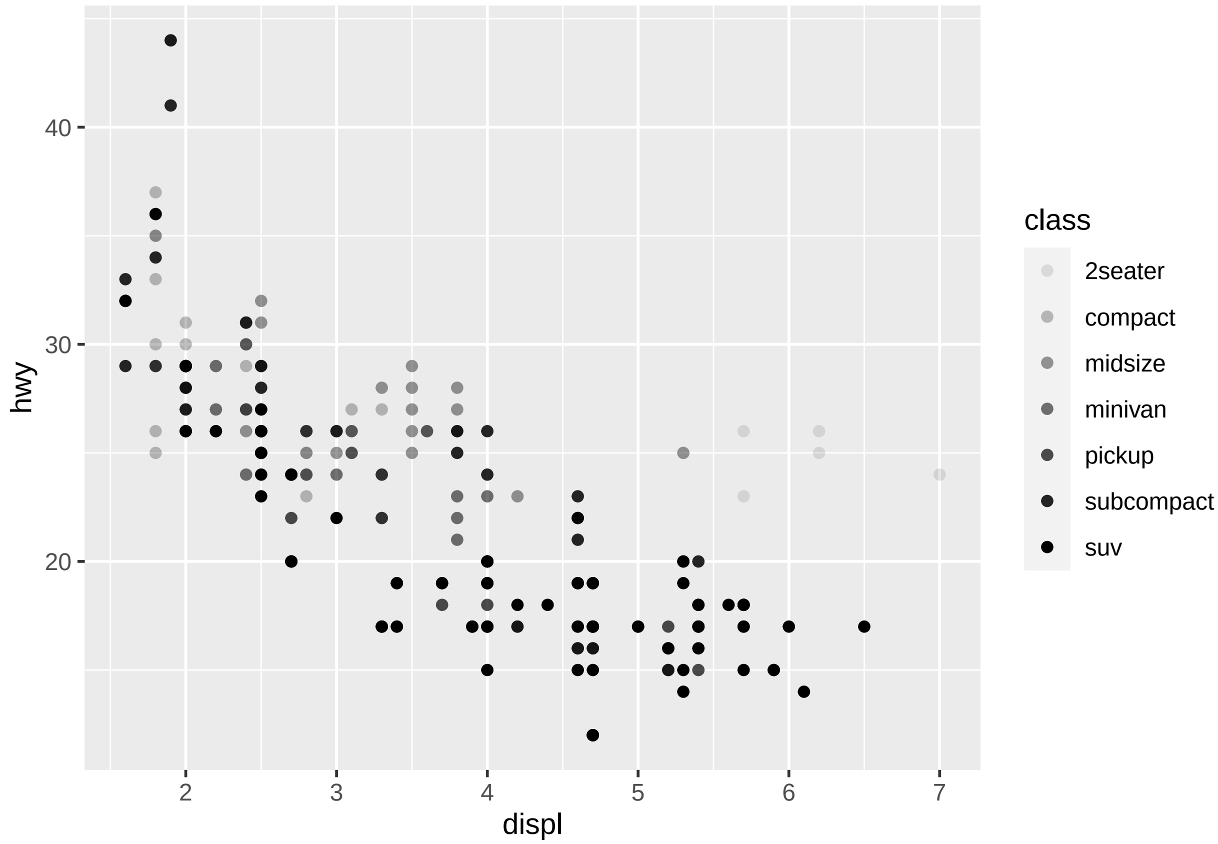
```
## Warning: Using size for a discrete variable is not advised.
```



Or we could have mapped class to the **alpha** aesthetic, which controls transparency of the points, or the shape of the points:

```
# Top
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```

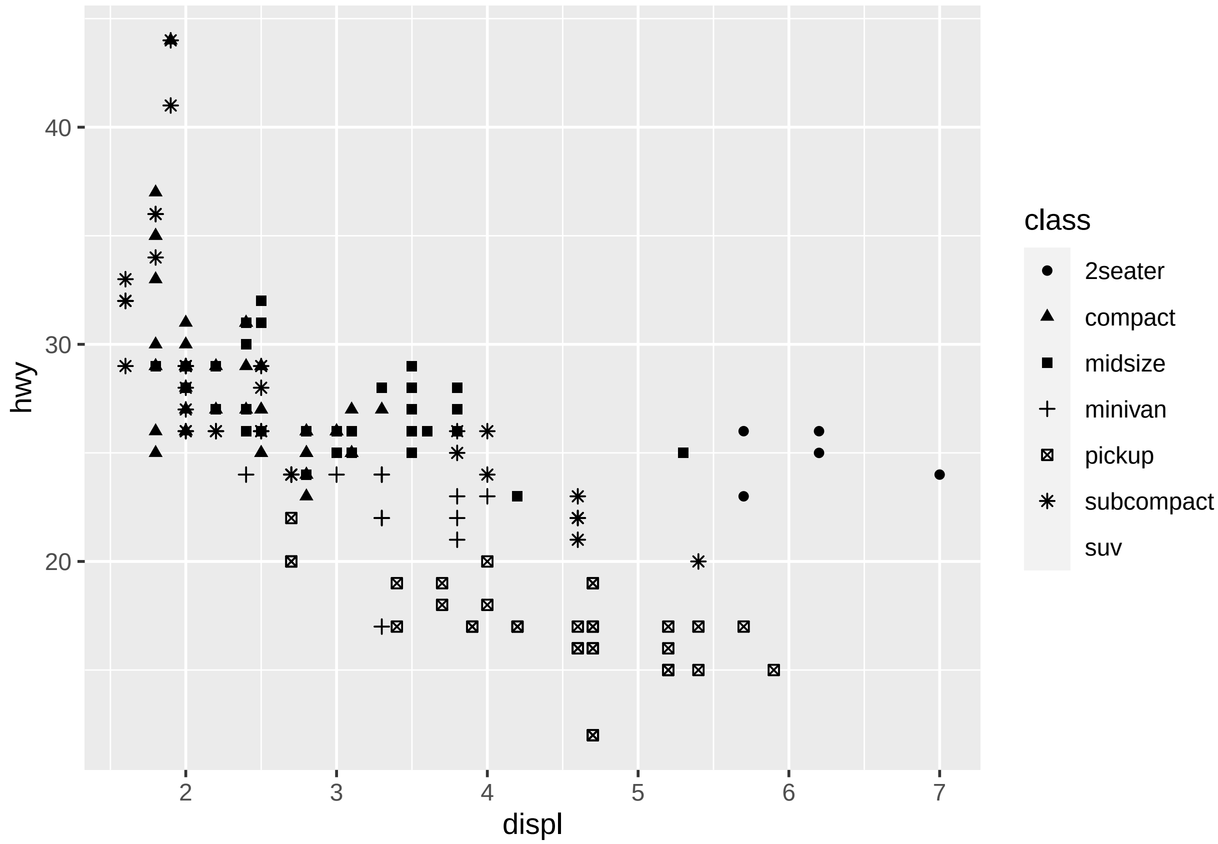
```
## Warning: Using alpha for a discrete variable is not advised.
```




```r
# Buttom
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```

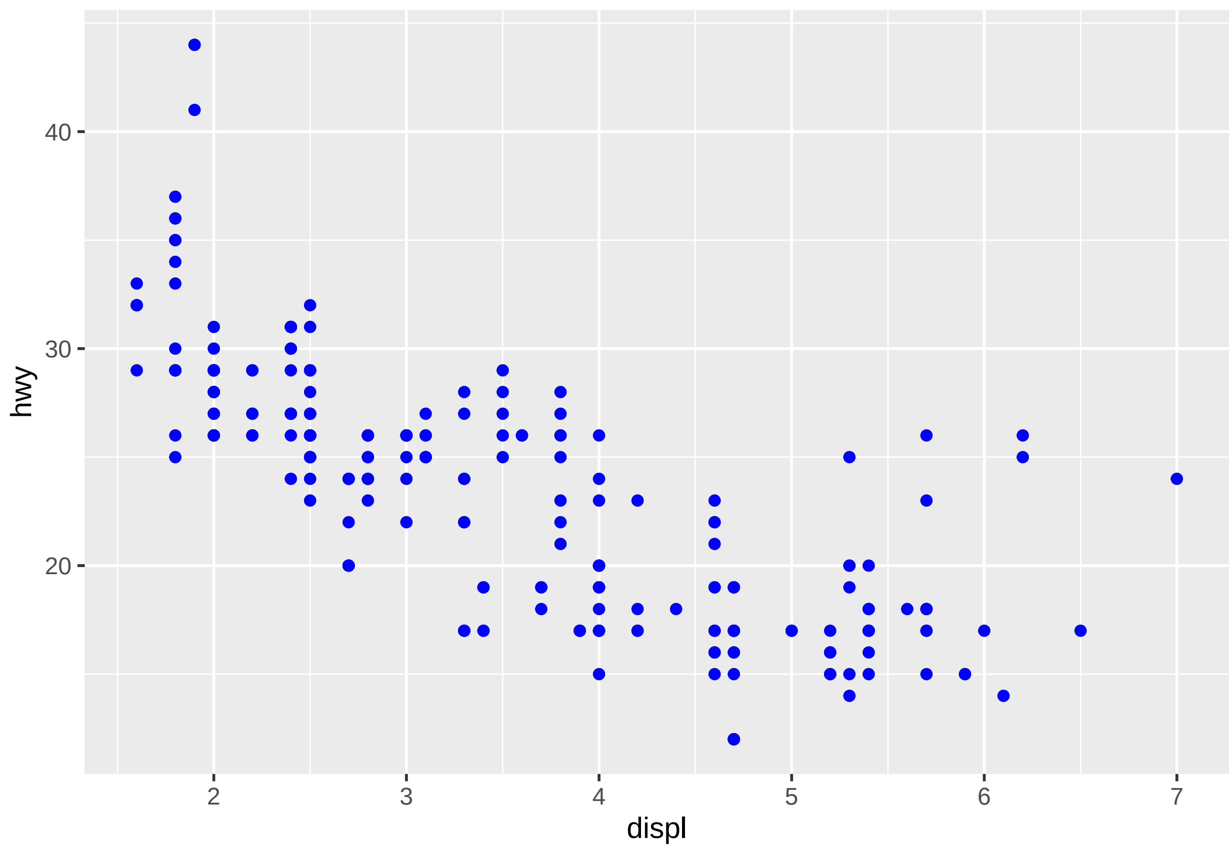
```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.

## Warning: Removed 62 rows containing missing values (geom_point).
```



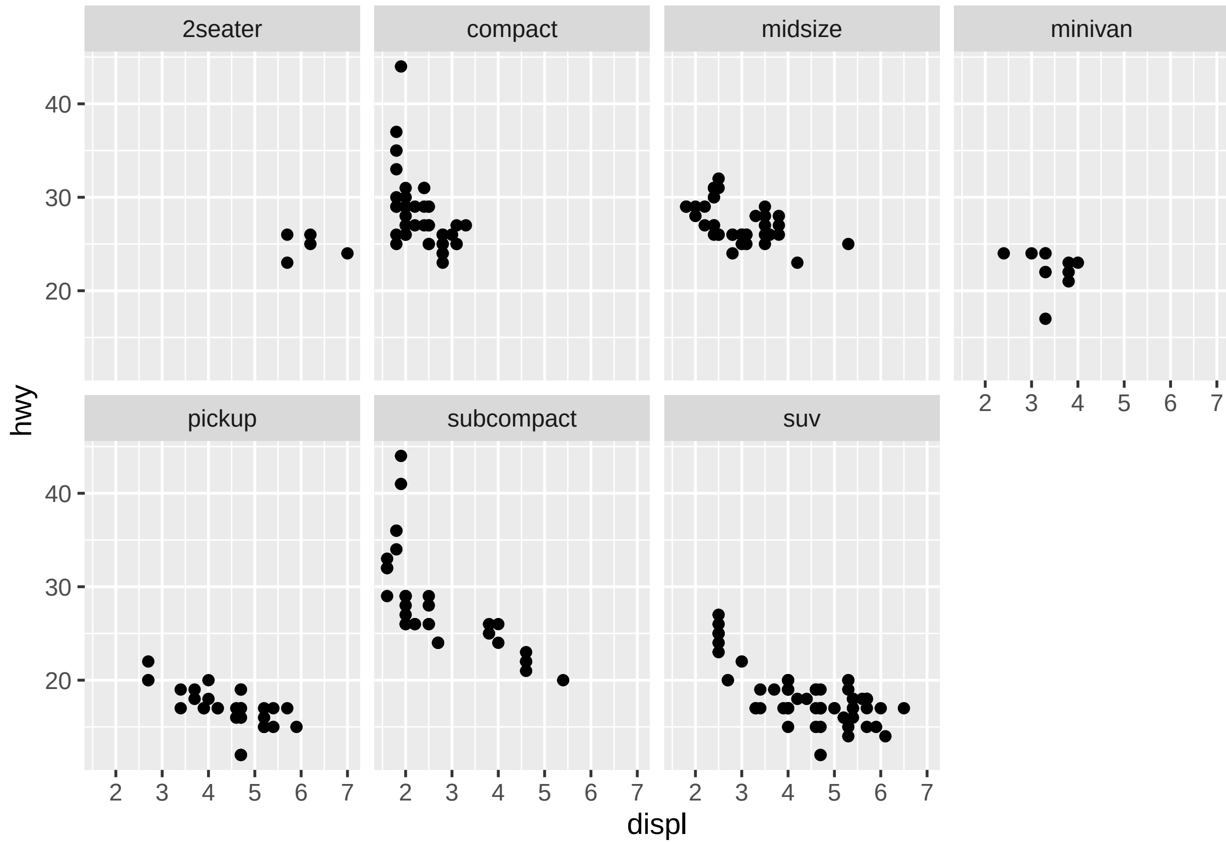
You can also set the aesthetic properties of your geom manually. For example, we can make all of the points in our plot blue:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



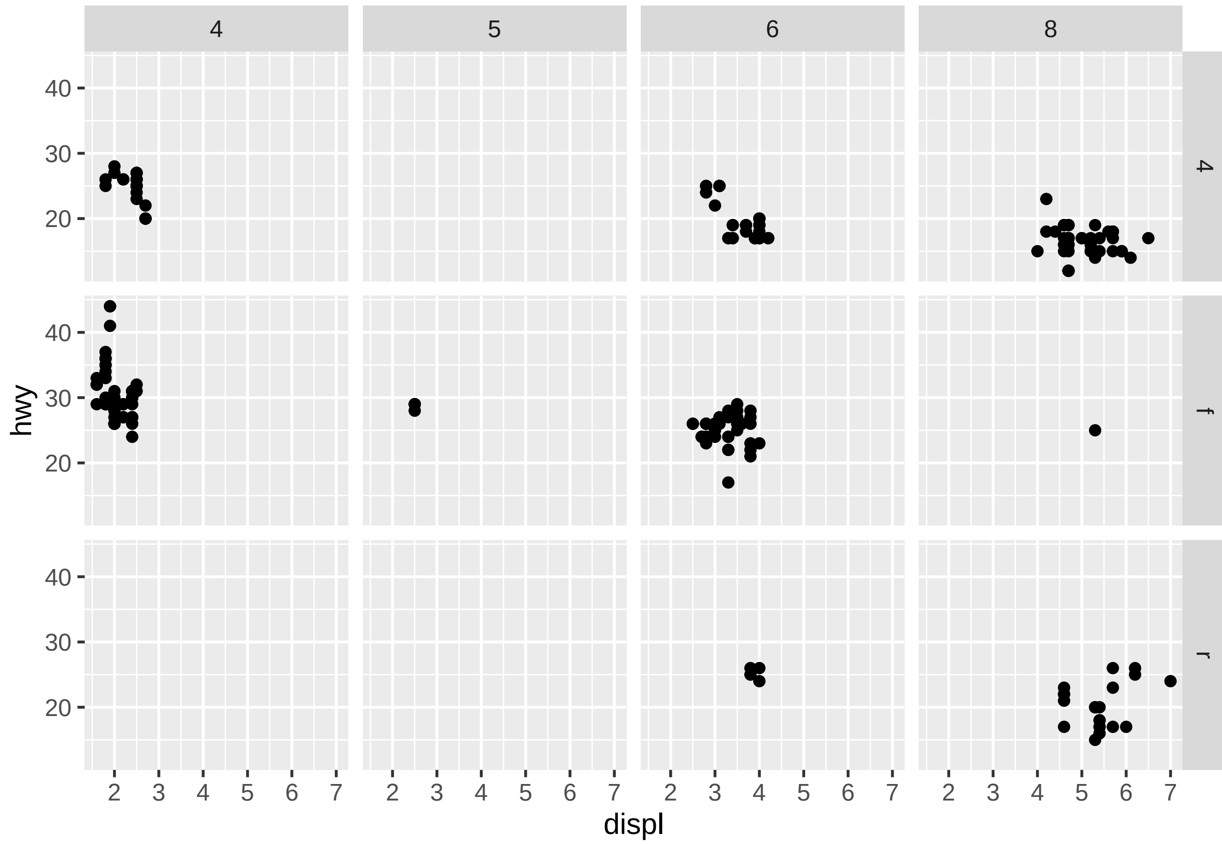
Facets one way to split categorical variables is to plot into facets, subplots that each display one subset of the data.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~class, nrow = 2)
```



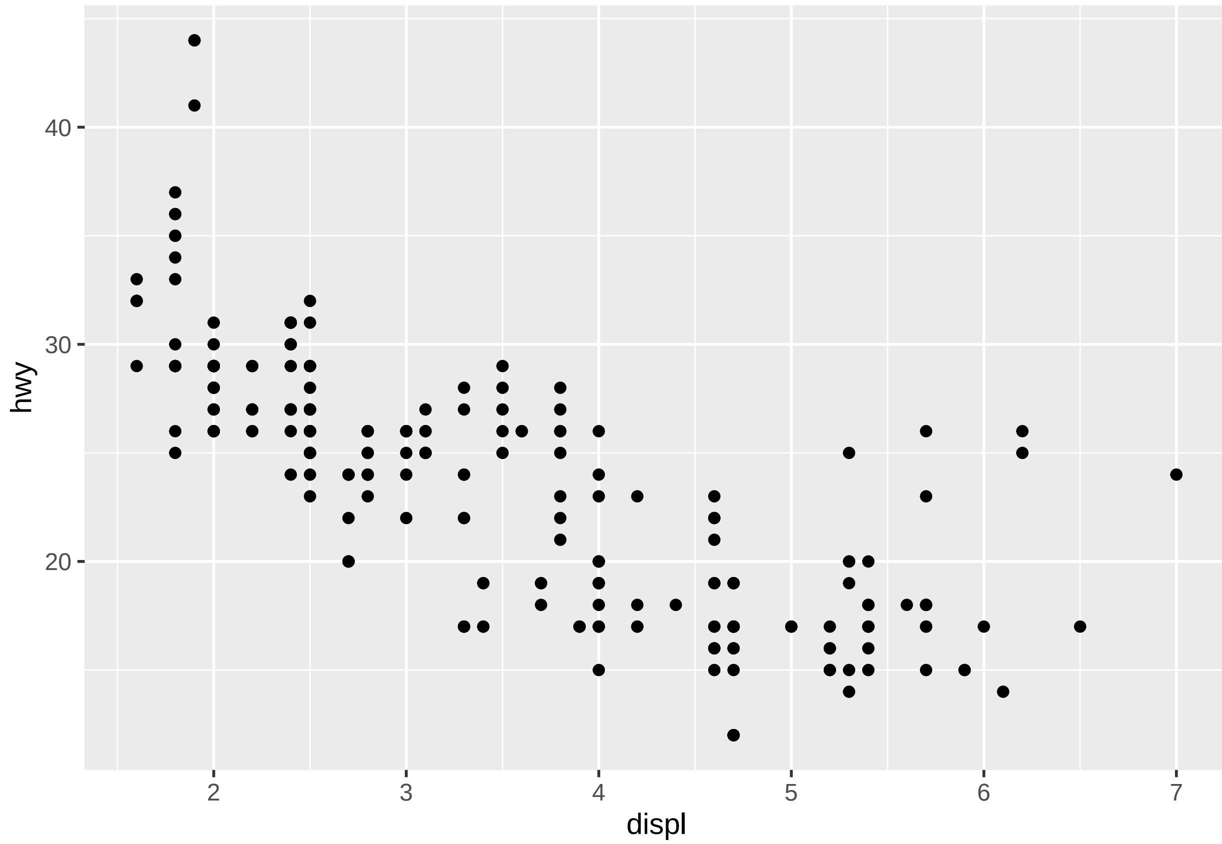
To facet the plot on combination of two variables, add **facetgrid** to your plot call. The first argument of **facetgrid** is also a formula. This time the formula should contain two variable names separated by a ~:

```
ggplot(data = mpg)+
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)
```



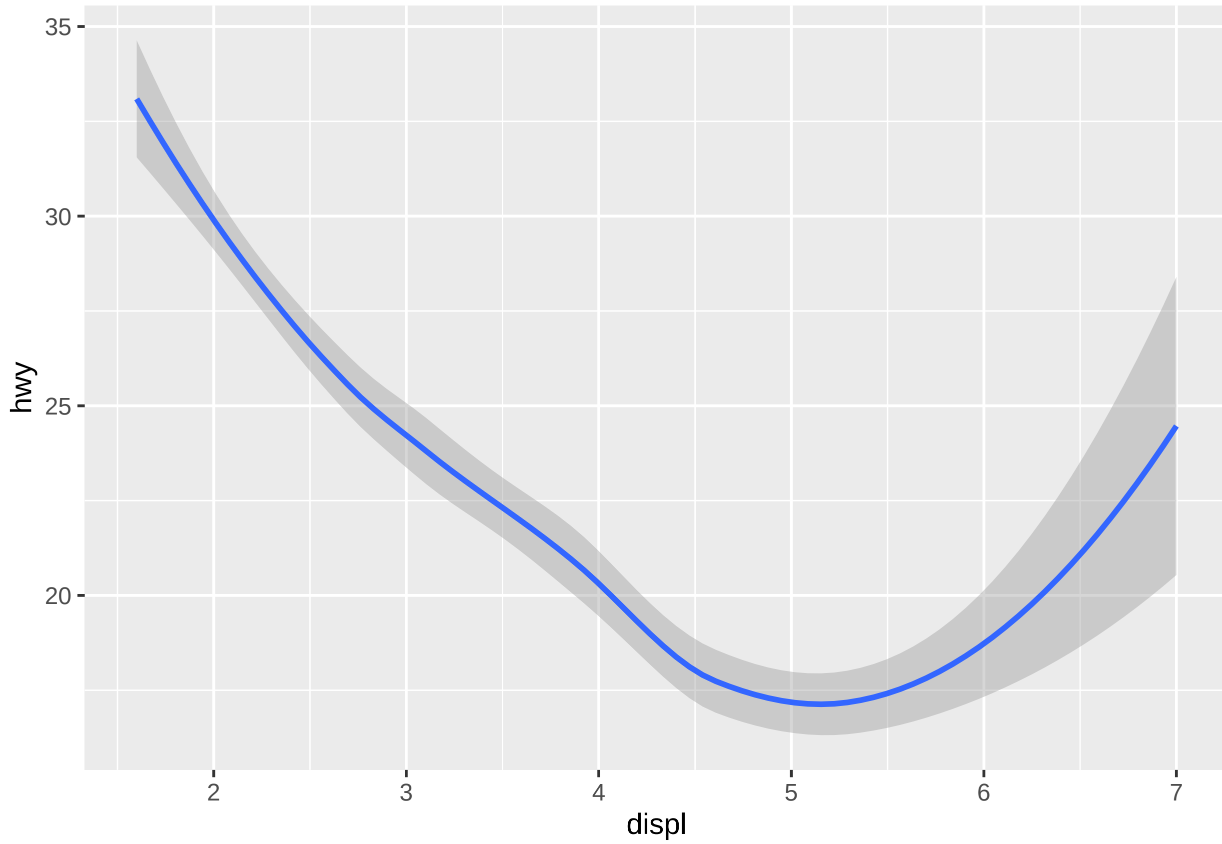
Geometric Objects

```
# Left
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



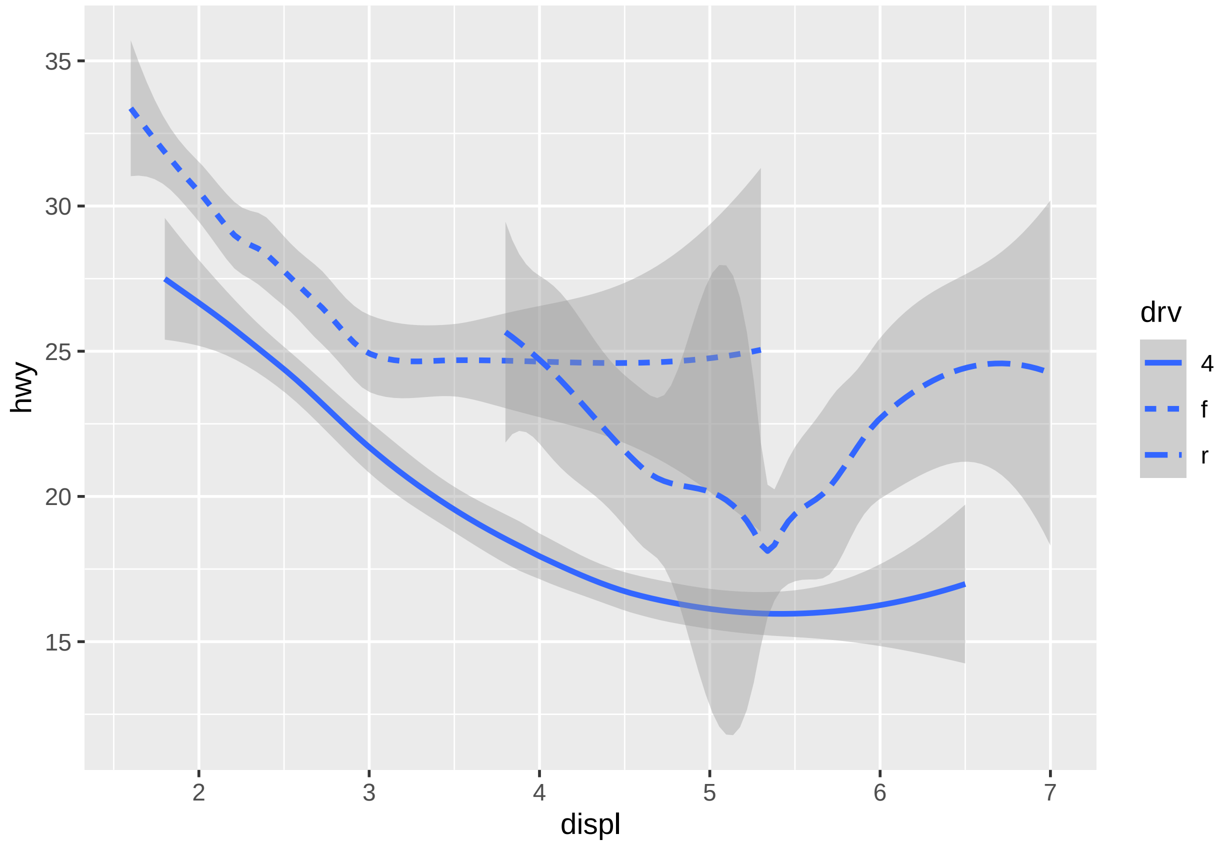

```r
# Right
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



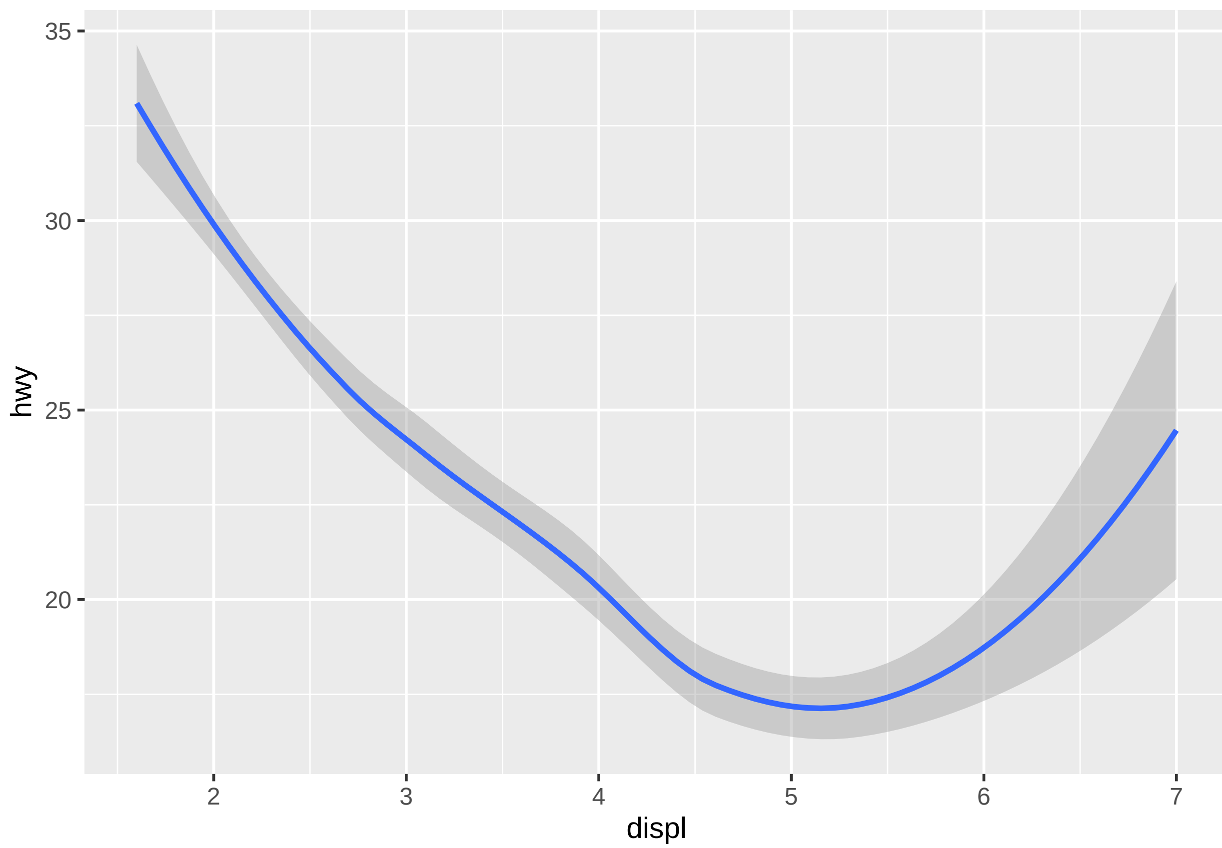

```r
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



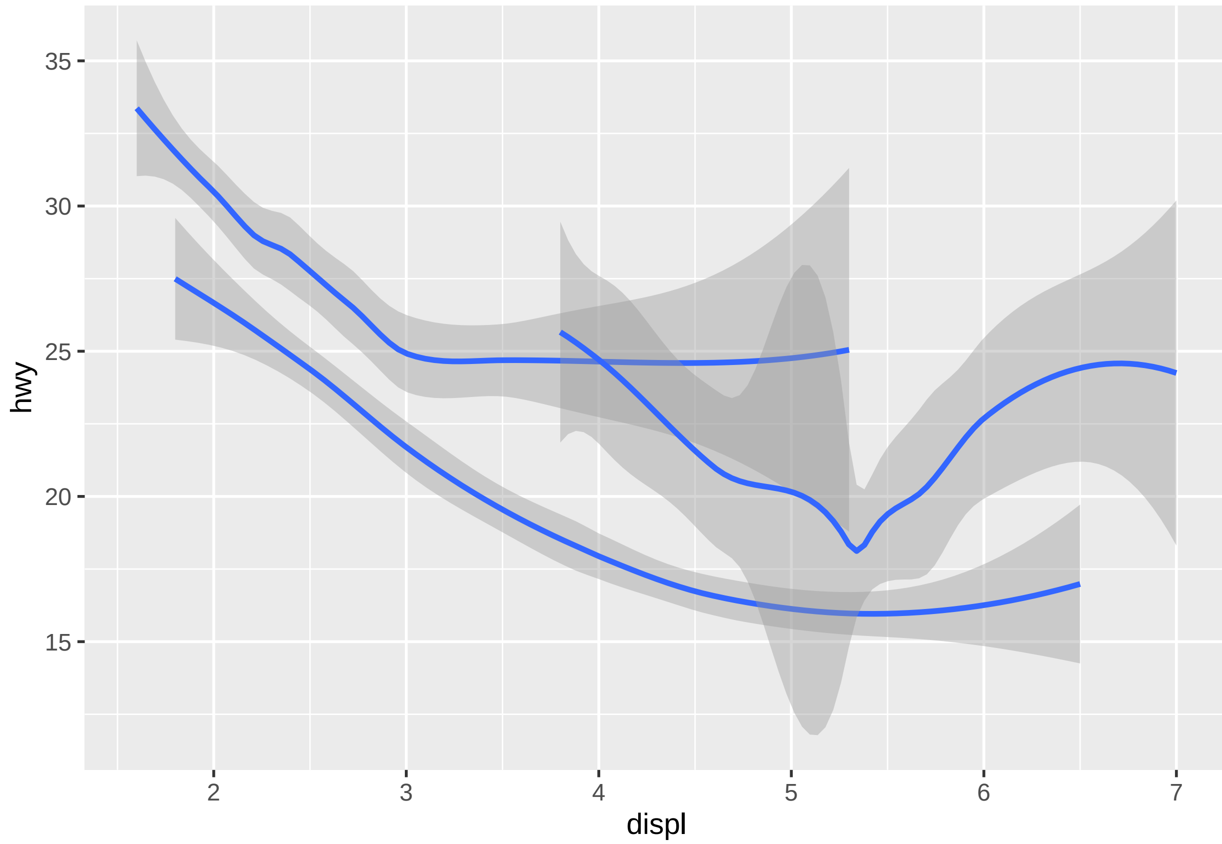

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



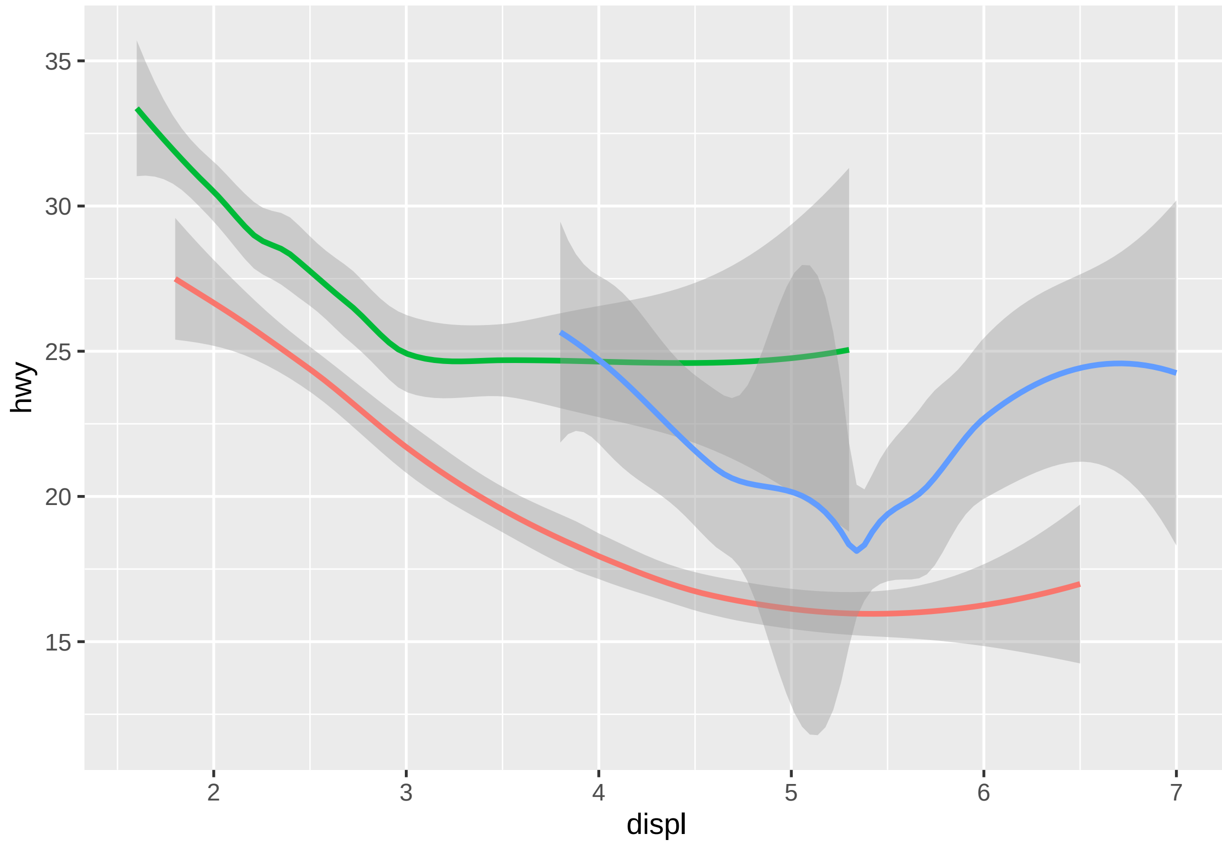

```r
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, group = drv))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

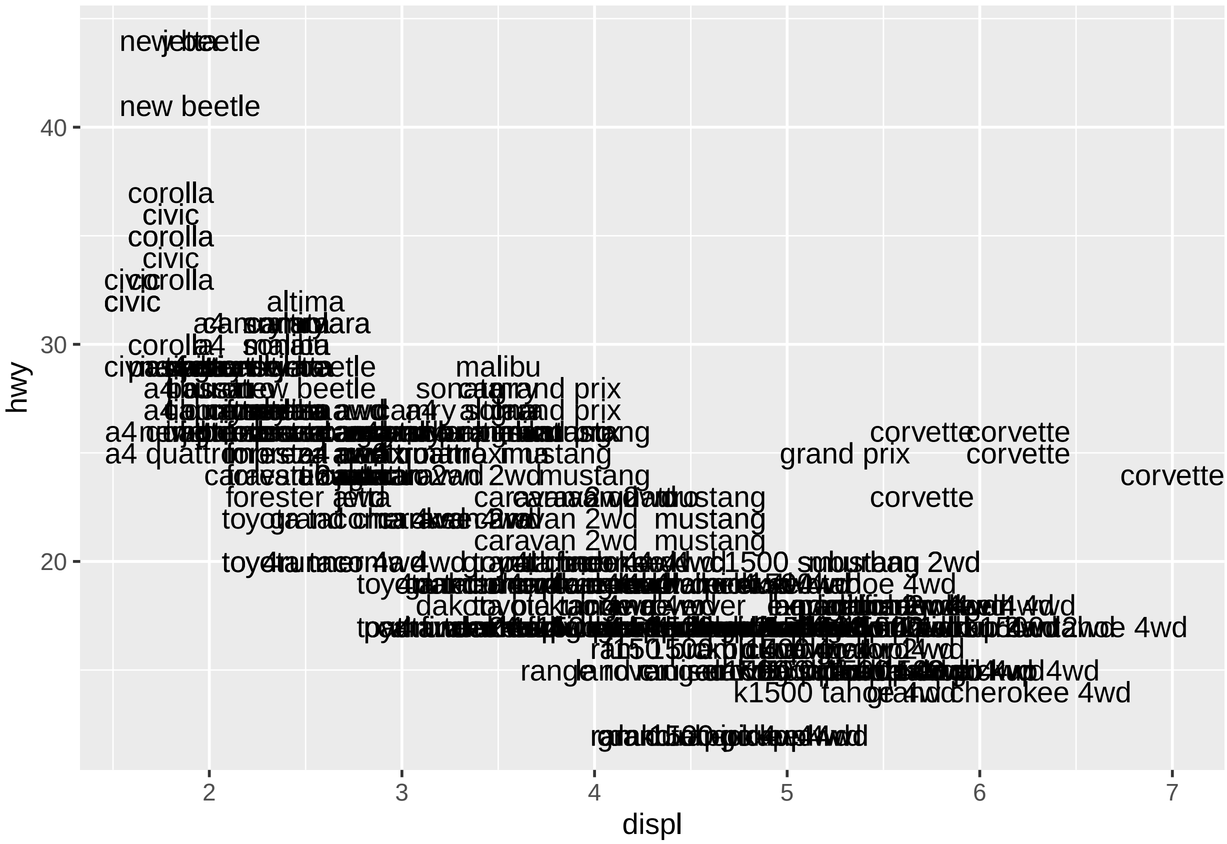



```
ggplot(data = mpg) +
  geom_smooth(
    mapping = aes(x = displ, y = hwy, color = drv),
    show.legend = FALSE
  )
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



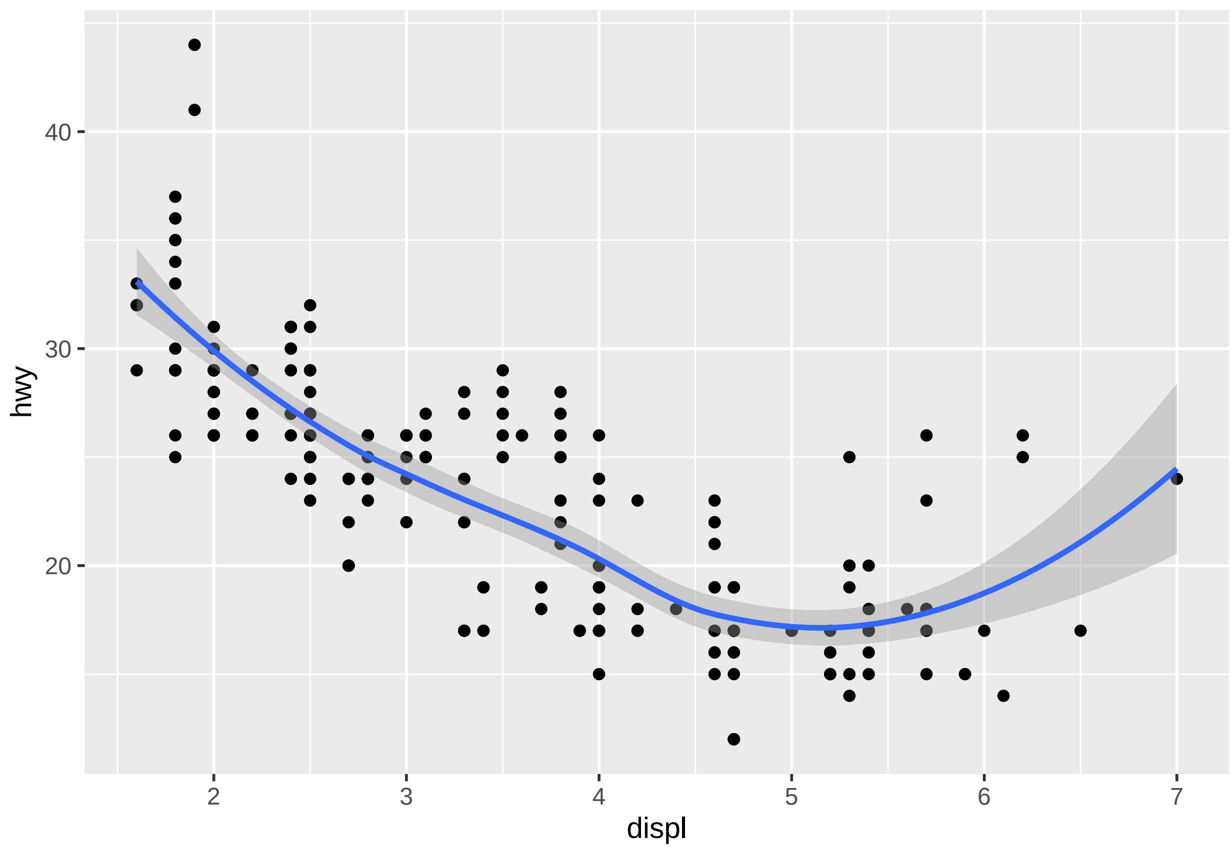

```
ggplot(mpg) +
  geom_text(mapping = aes(x = displ, y = hwy, label = model))
```



In order to display multiple geoms in the same plot, add multiple geom functions to **ggplot()**:

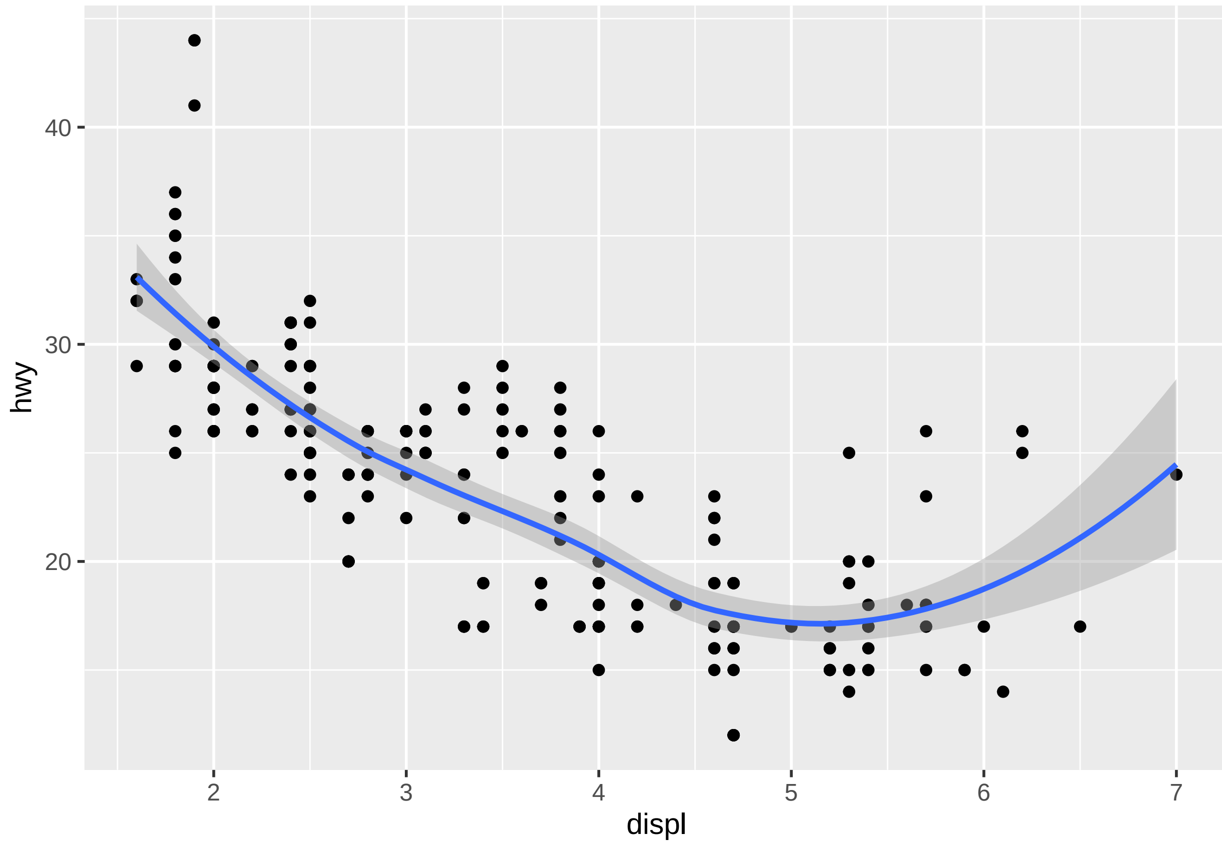
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



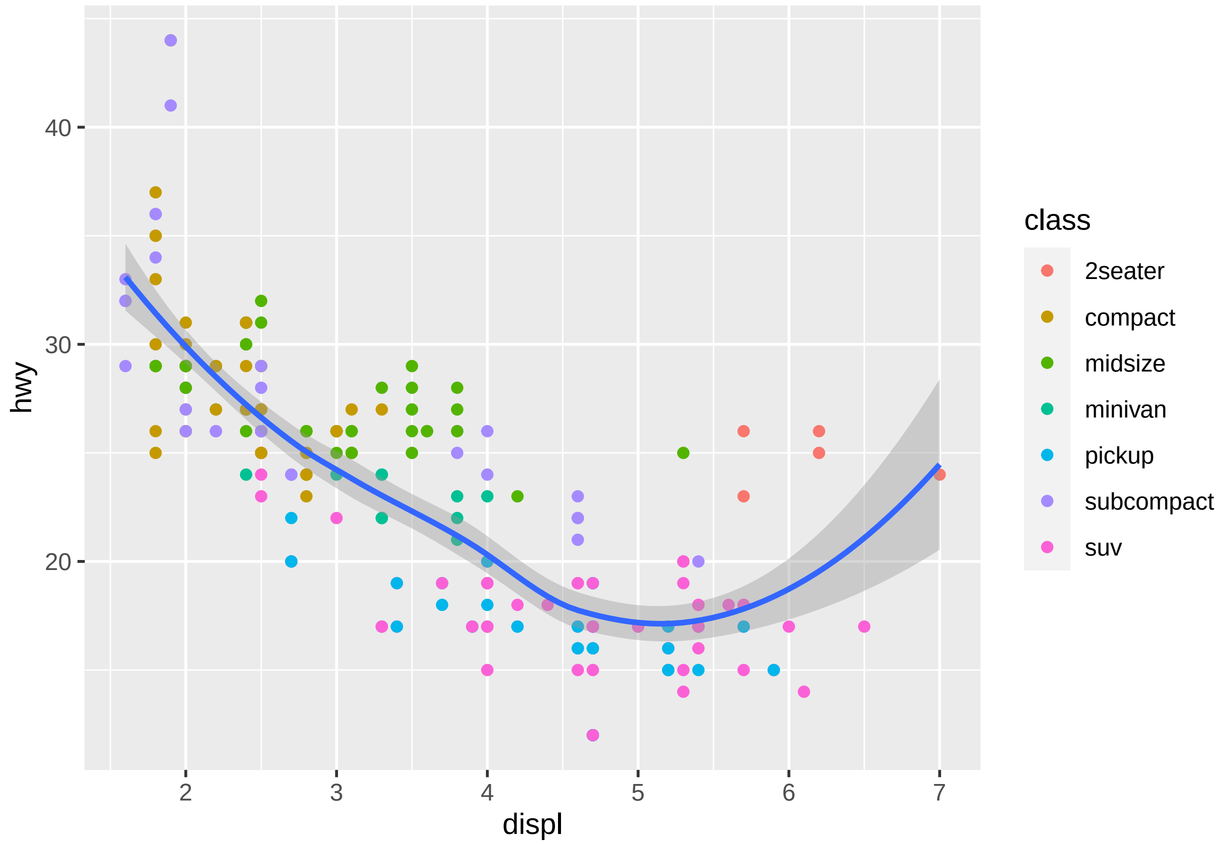

```
ggplot(data = mpg, mapping = aes( x = displ, y = hwy)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



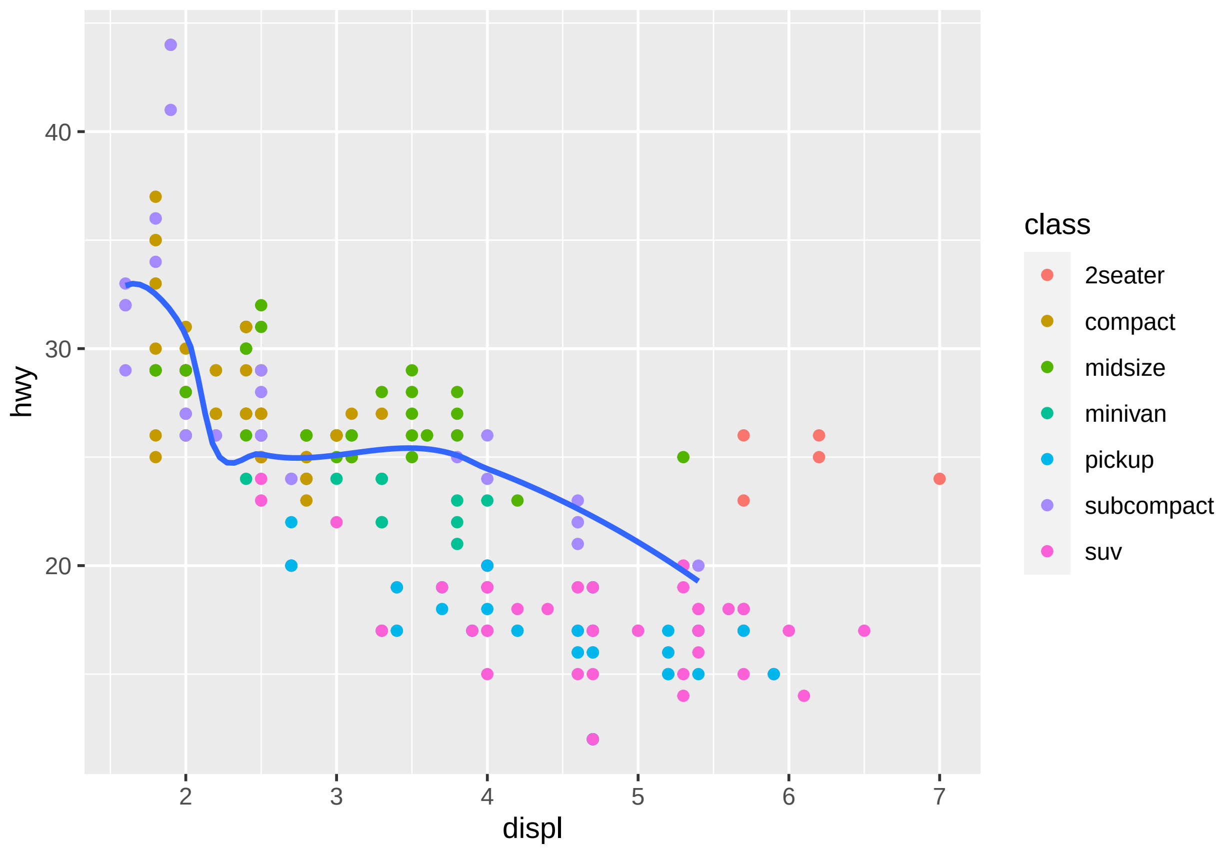

```
ggplot(data = mpg, mapping = aes( x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(
    data = filter(mpg, class == "subcompact"),
    se = FALSE
  )
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

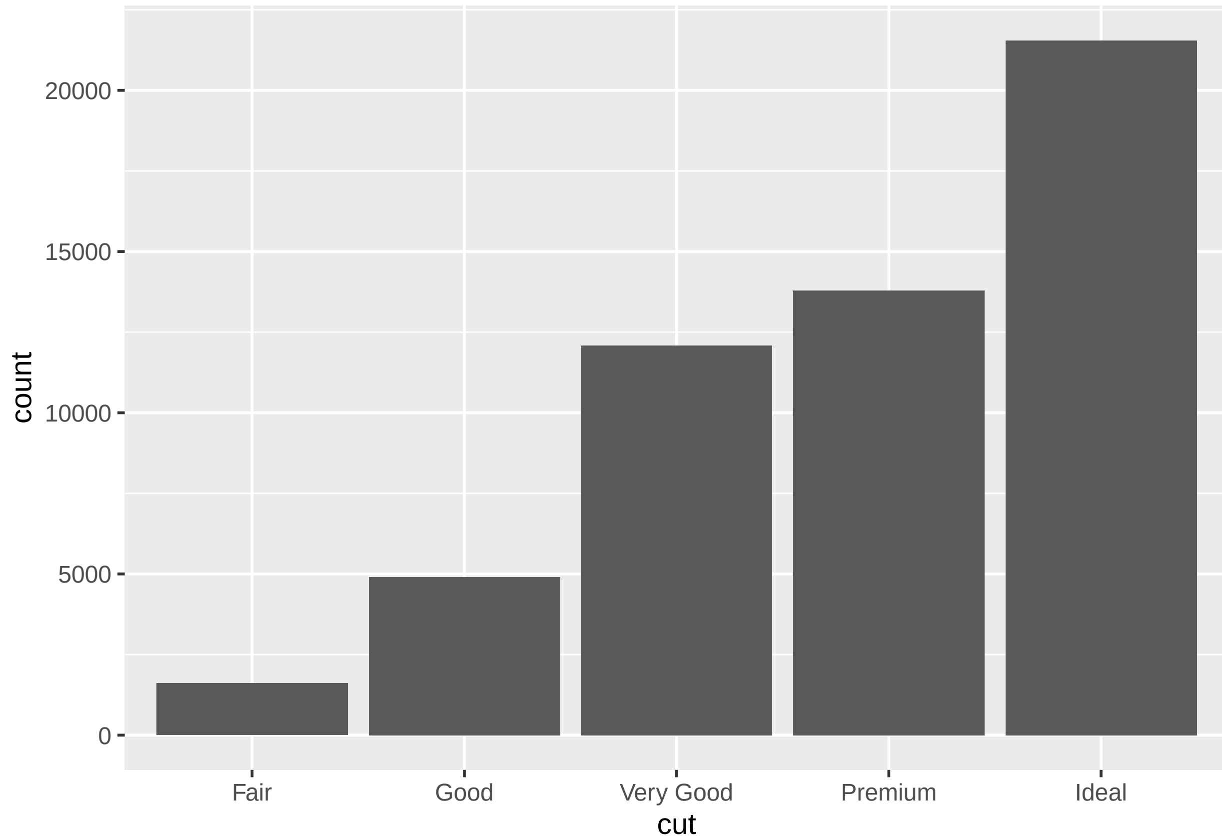



```
diamonds
```

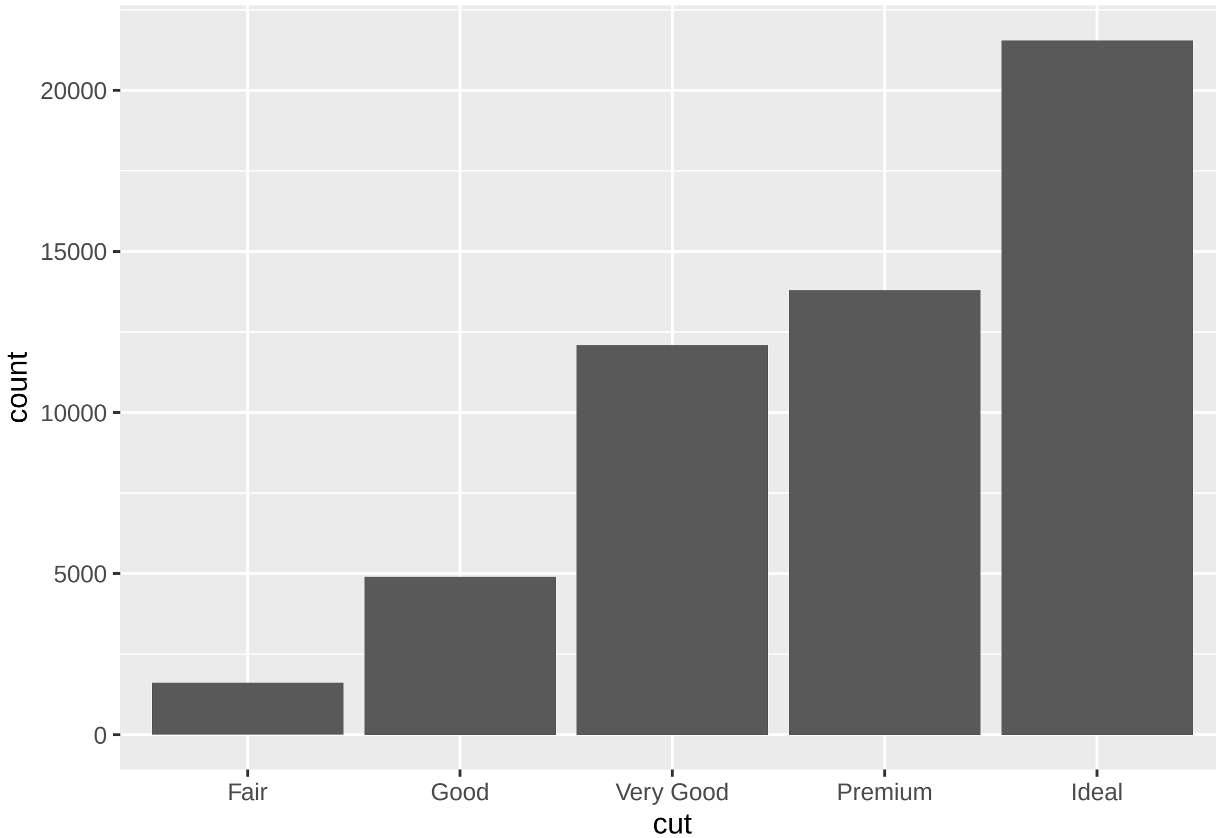
```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

Statistical Transformation

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```

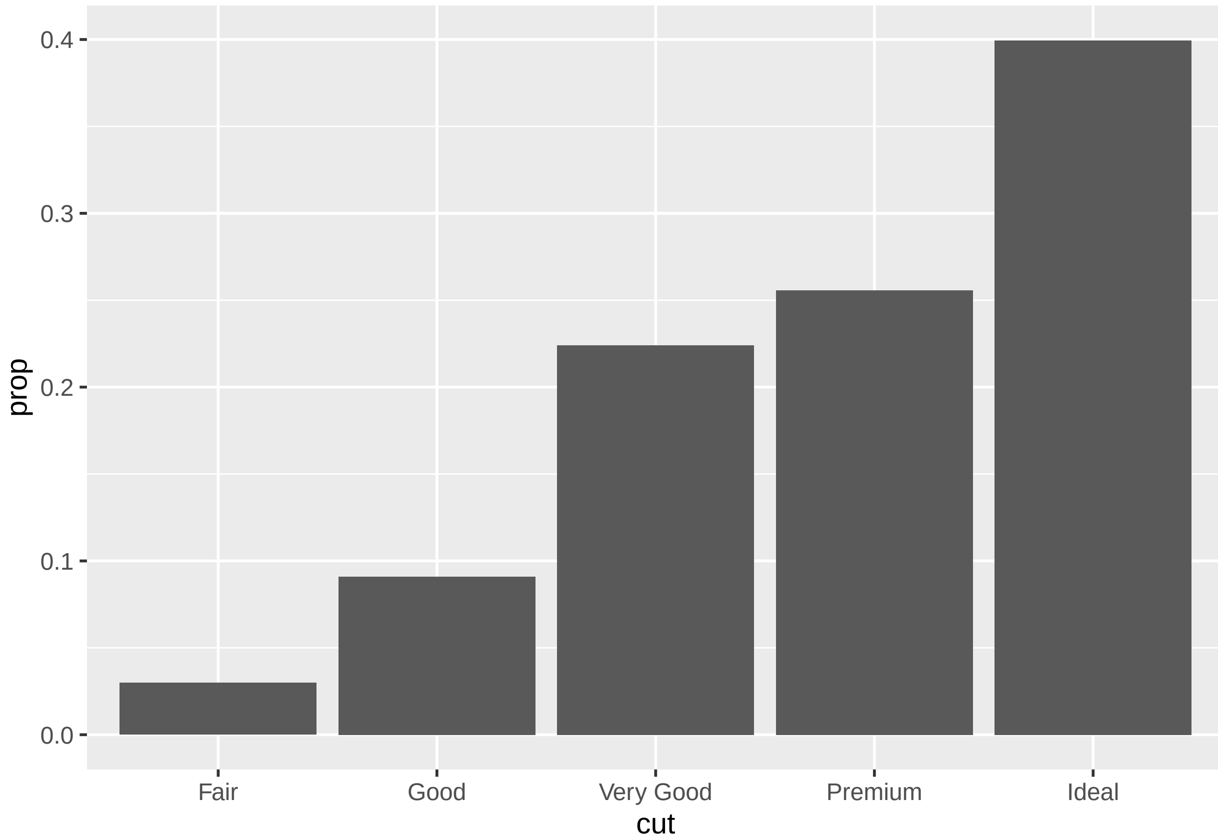



```
ggplot(data = diamonds) +
  stat_count(mapping = aes(x = cut))
```



You might want to override the default mapping from trans- formed variables to aesthetics. For example, you might want to display a bar chart of proportion, rather than count:

```
ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = cut, y = ..prop.., group = 1)
  )
```



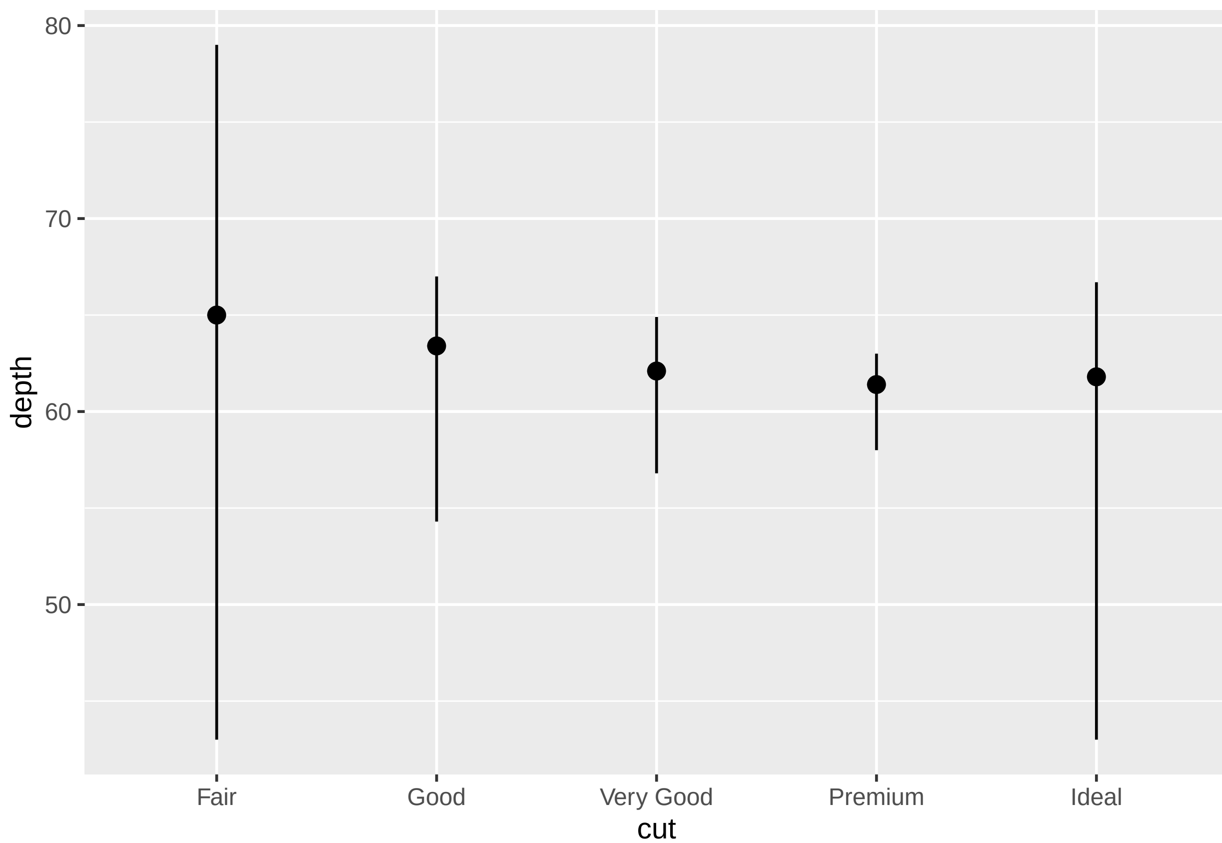
You might want to draw greater attention to the statistical trans- formation in your code. For example, you might use stat_sum mary(), which summarizes the y values for each unique x value, to draw attention to the summary that you're computing

```
ggplot(data = diamonds) +
  stat_summary(
    mapping = aes(x = cut, y = depth),
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
  )
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

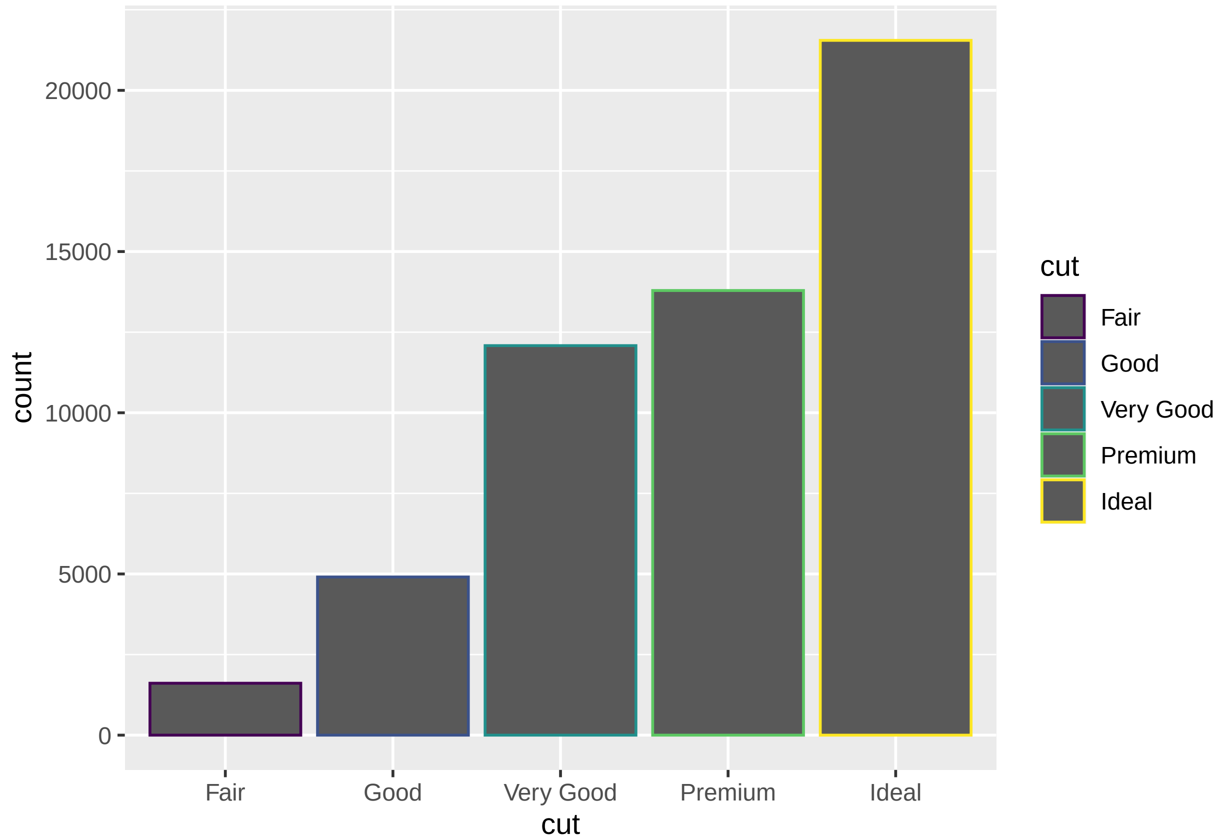
```
## Warning: `fun.ymin` is deprecated. Use `fun.min` instead.
```

```
## Warning: `fun.ymax` is deprecated. Use `fun.max` instead.
```

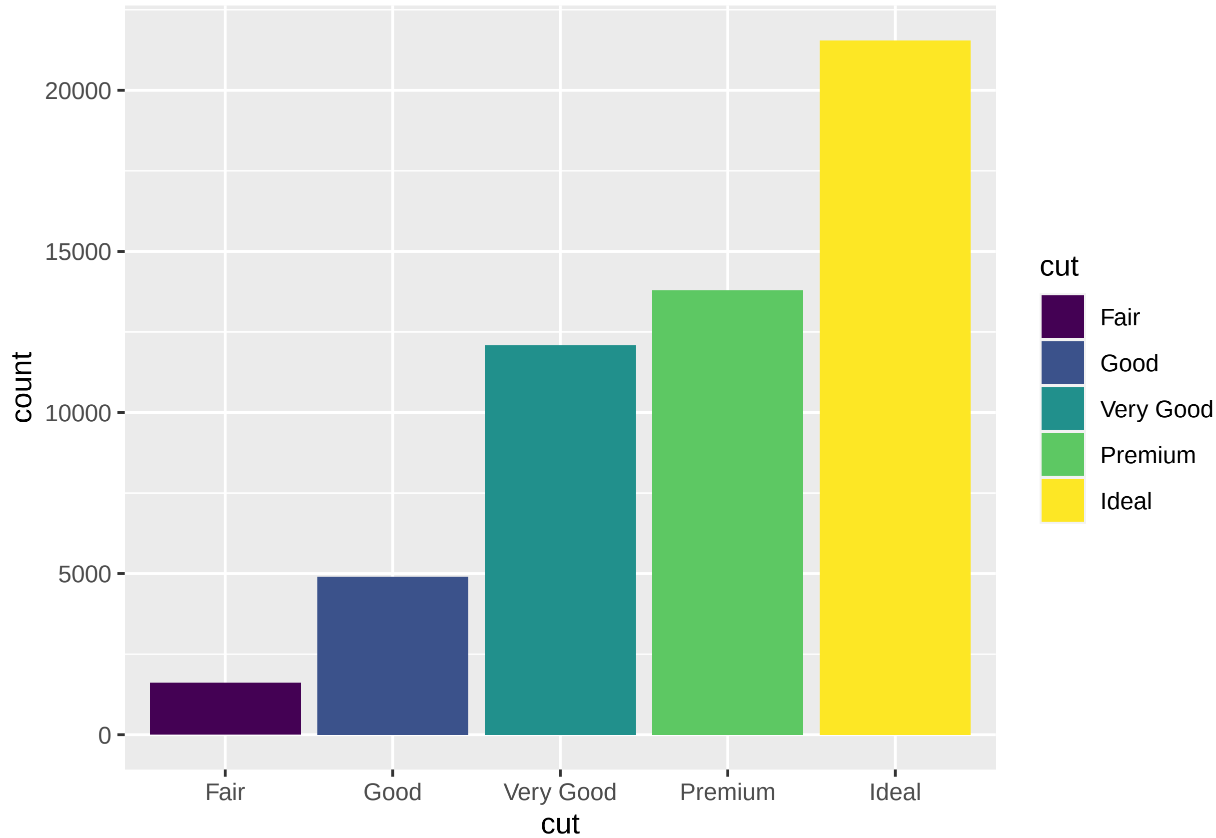


##Position Adjustments You can color a bar chart using either the color aesthetic, or more usefully, fill:

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, color = cut))
```

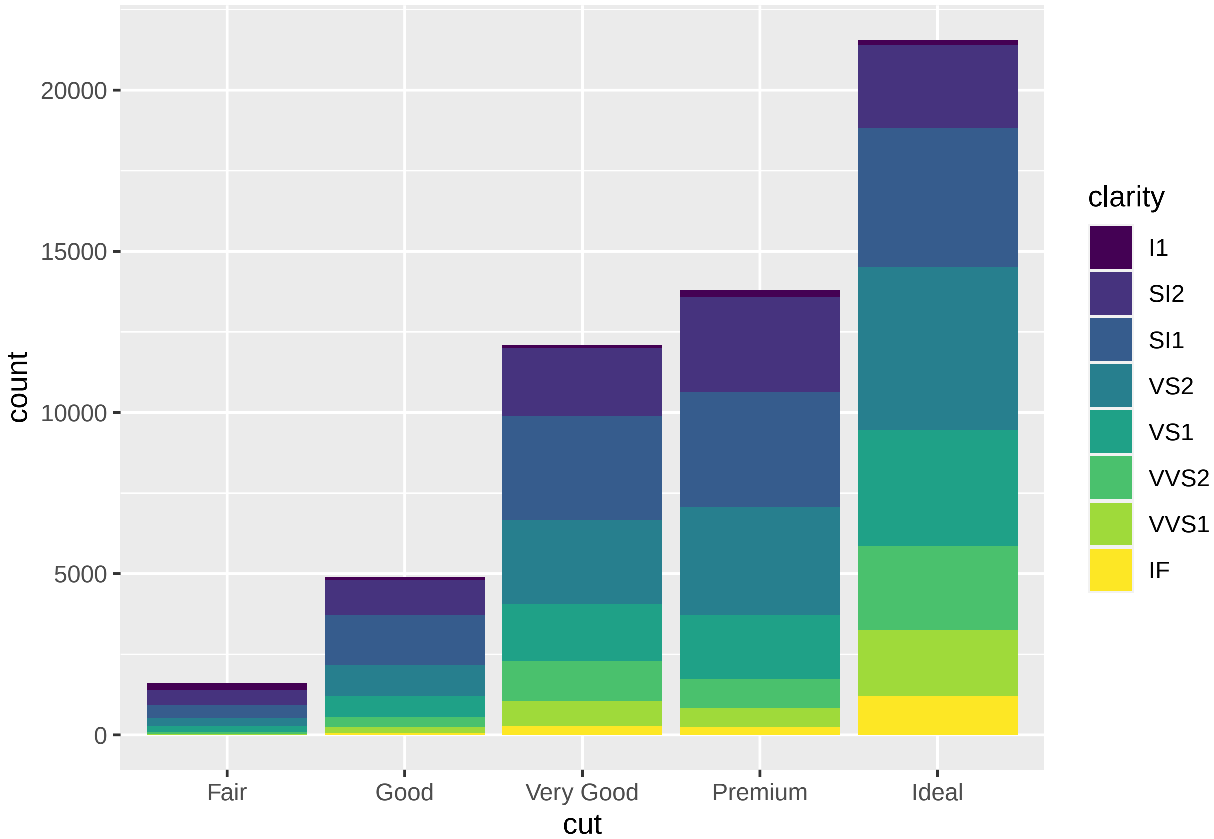



```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```



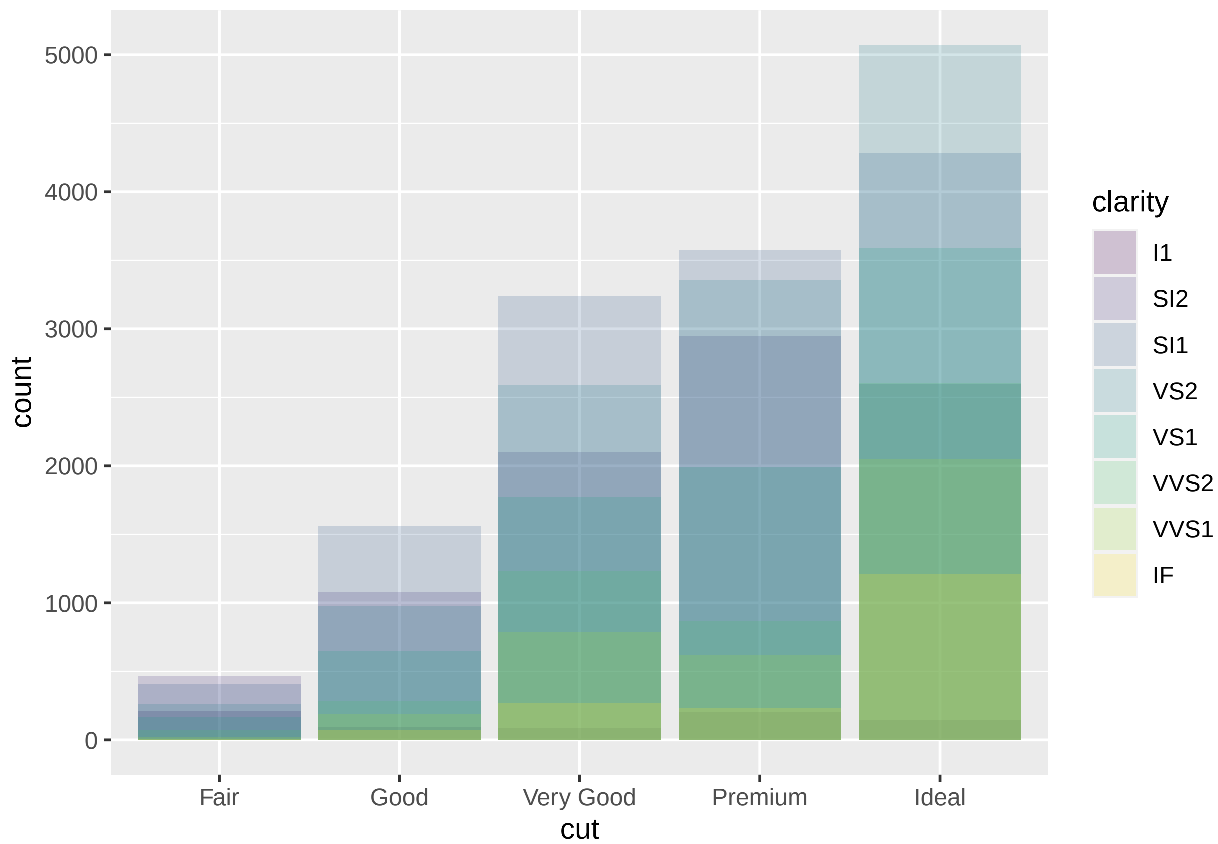
Note what happens if you map the fill aesthetic to another vari- able, like clarity: the bars are automatically stacked. Each colored rectangle represents a combination of cut and clarity:

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```

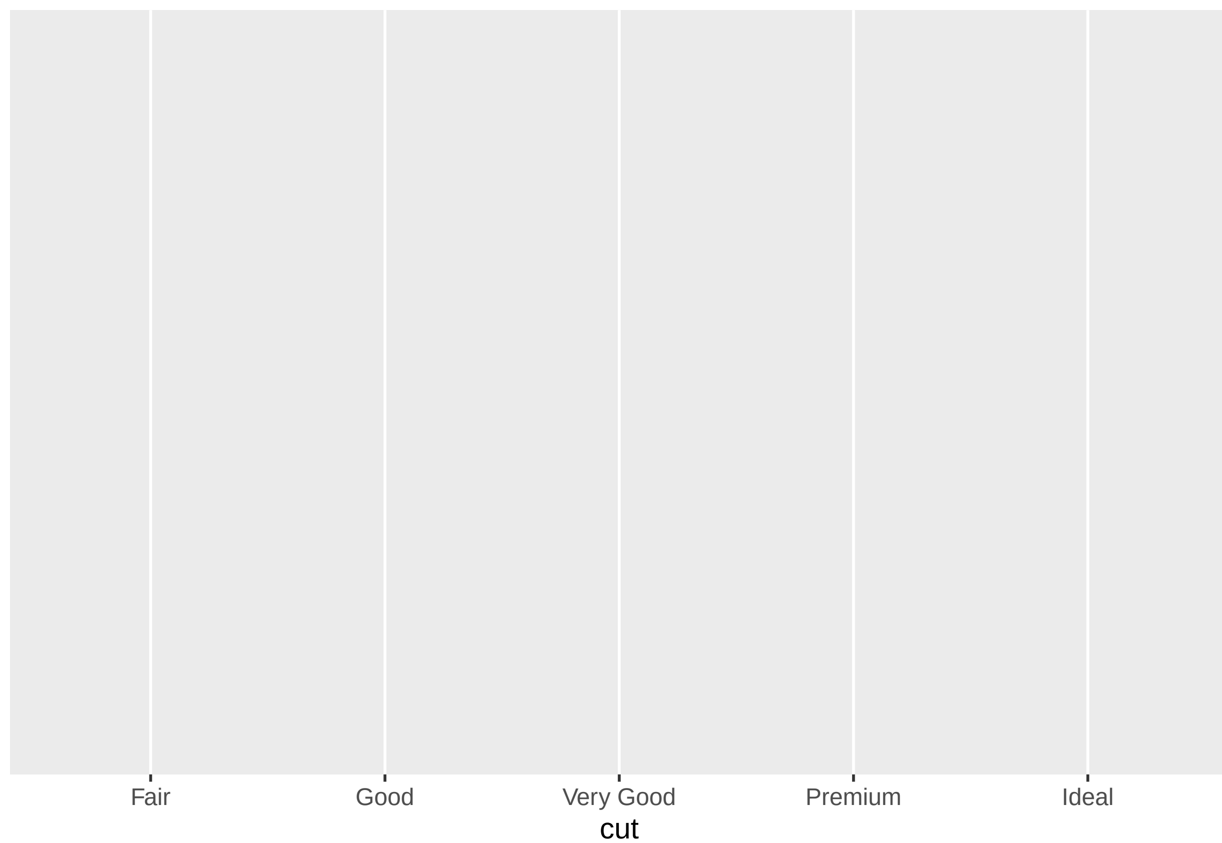


position = "identity" will place each object exactly where it falls in the context of the graph. This is not very useful for bars, because it overlaps them. To see that overlapping we either need to make the bars slightly transparent by setting alpha to a small value, or completely transparent by setting fill = NA:

```
ggplot(
  data = diamonds,
  mapping = aes(x = cut, fill = clarity)
) +
  geom_bar(alpha = 1/5, position = "identity")
```

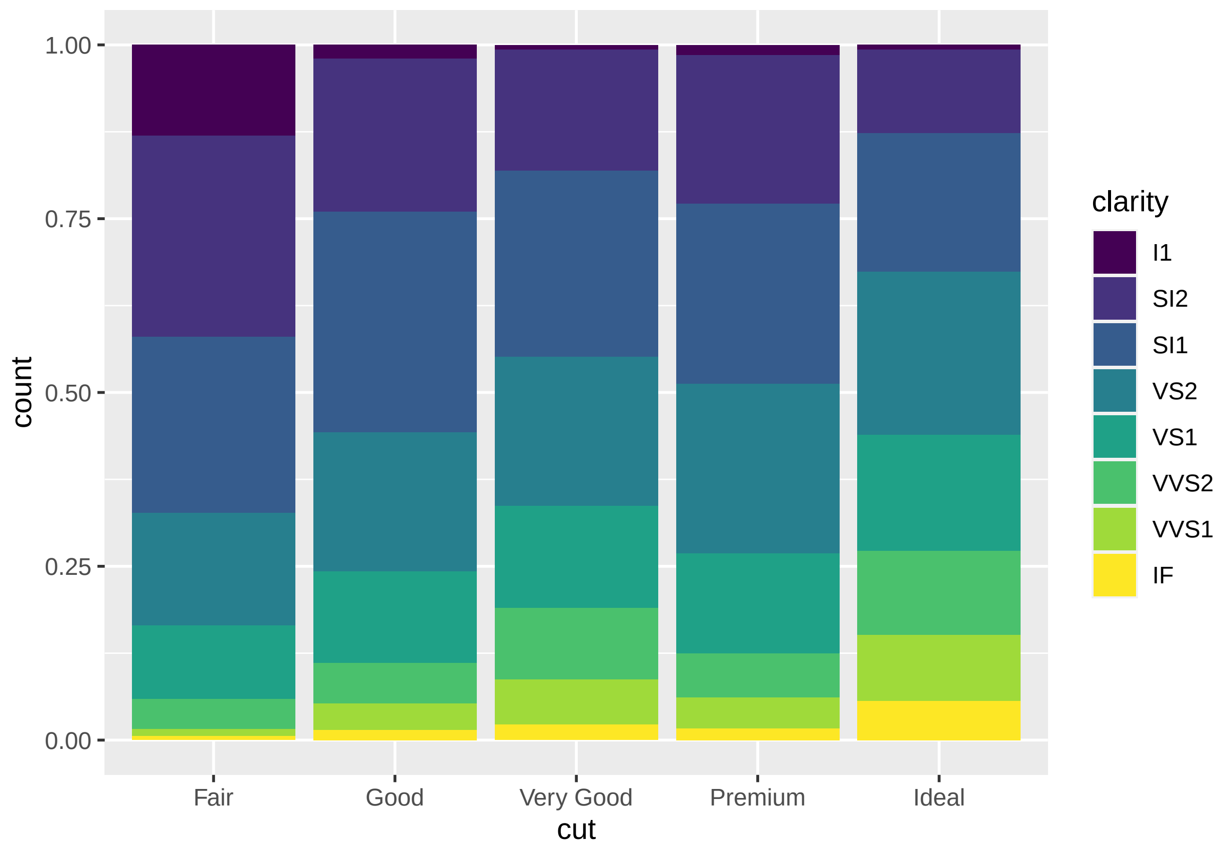



```
ggplot(
  data = diamonds,
  mapping = aes(x = cut, color = clarity)
)
```



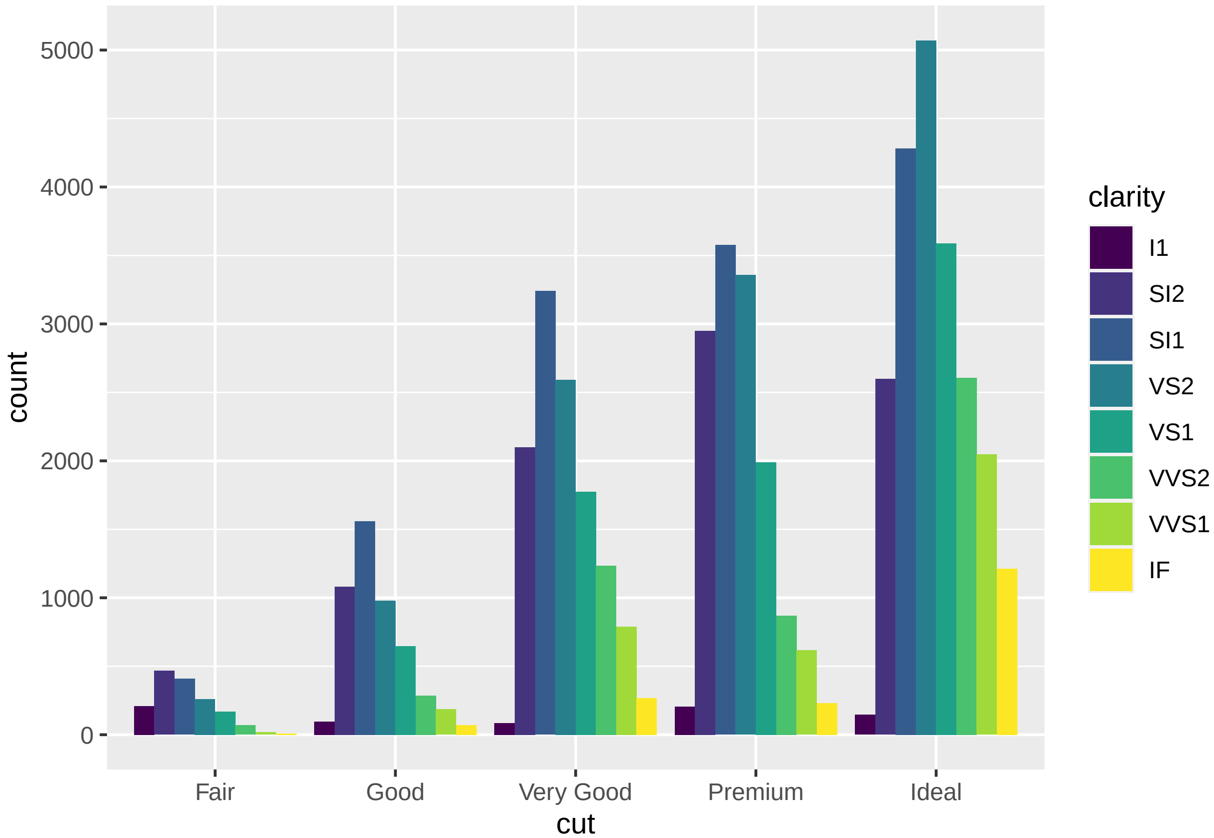
position = "fill" works like stacking, but makes each set of stacked bars the same height. This makes it easier to compare proportions across groups:

```
ggplot(data = diamonds) +
geom_bar(
  mapping = aes(x = cut, fill = clarity),
  position = "fill"
)
```



position = "dodge" places overlapping objects directly beside one another. This makes it easier to compare individual values:

```
ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = cut, fill = clarity),
    position = "dodge"
  )
```

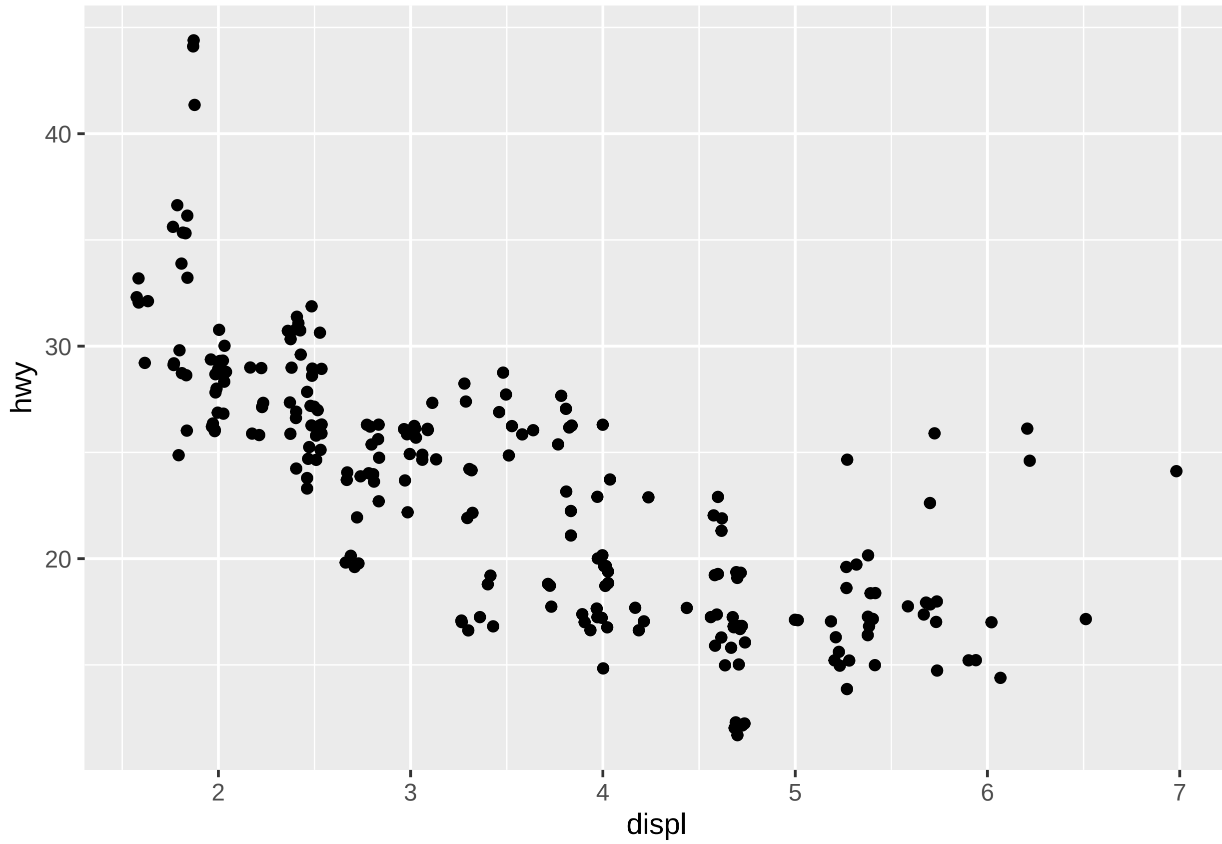


here's one other type of adjustment that's not useful for bar charts, but it can be very useful for scatterplots. Recall our first scatterplot. Did you notice that the plot displays only 126 points, even though there are 234 observations in the dataset?

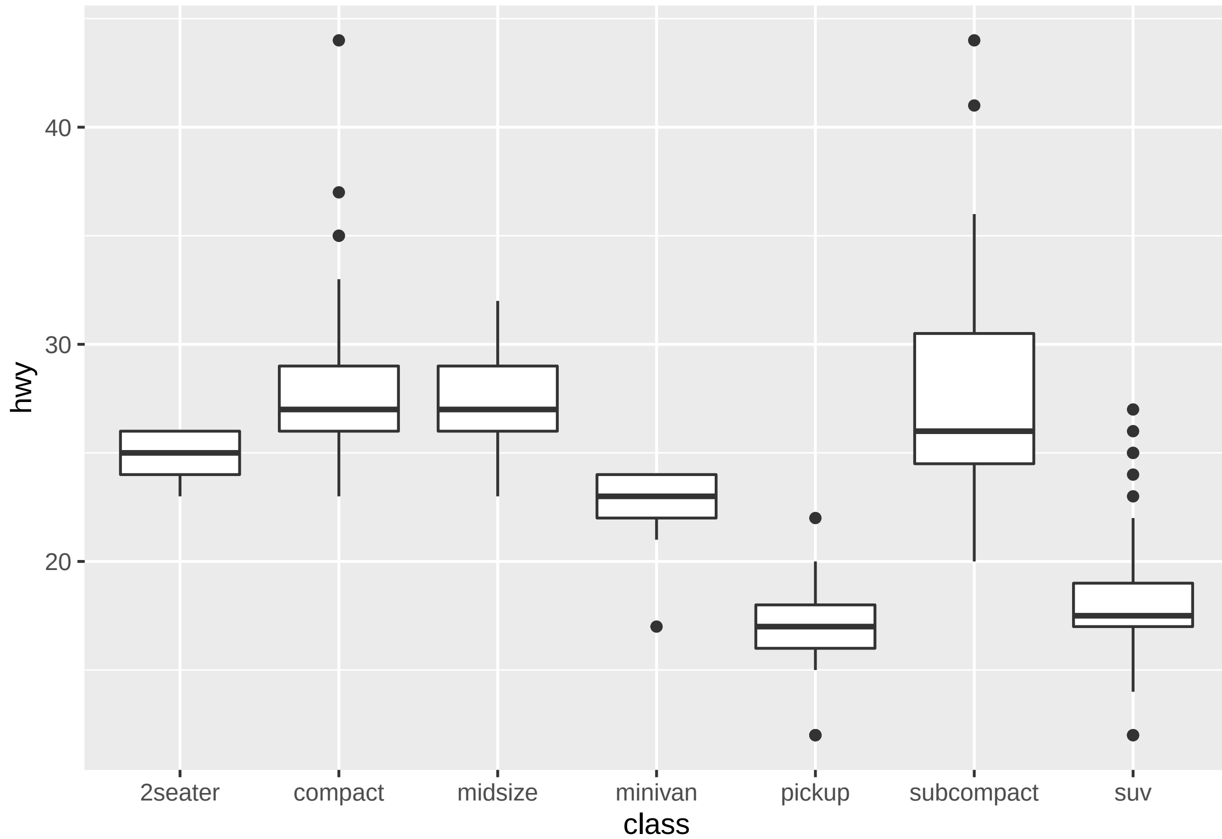
The values of hwy and displ are rounded so the points appear on a grid and many points overlap each other. This problem is known as overplotting. This arrangement makes it hard to see where the mass of the data is. Are the data points spread equally throughout the graph, or is there one special combination of hwy and displ that contains 109 values

You can avoid this gridding by setting the position adjustment to "jitter." position = "jitter" adds a small amount of random noise to each point. This spreads the points out because no two points are likely to receive the same amount of random noise:

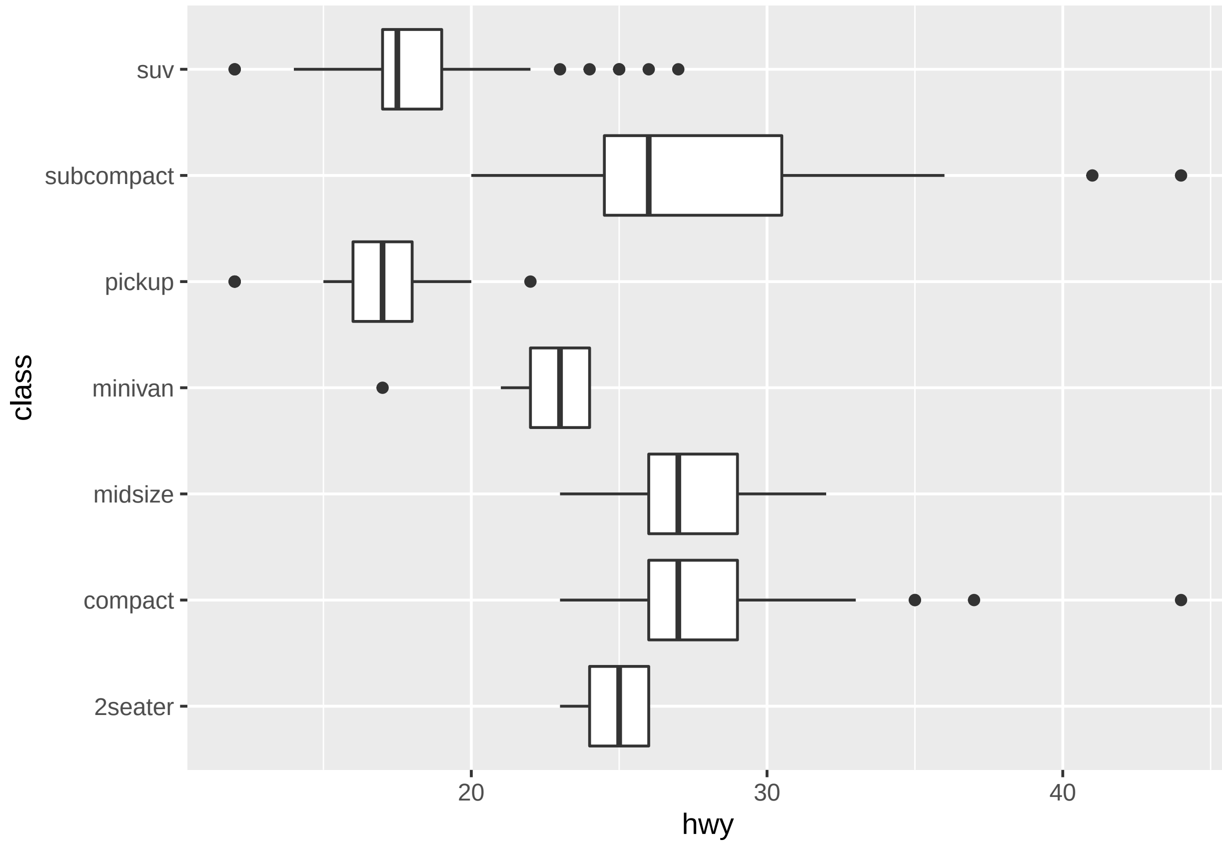
```
ggplot(data = mpg) +
  geom_point(
    mapping = aes(x = displ, y = hwy),
    position = "jitter"
    )
```




```
ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
  geom_boxplot()
```

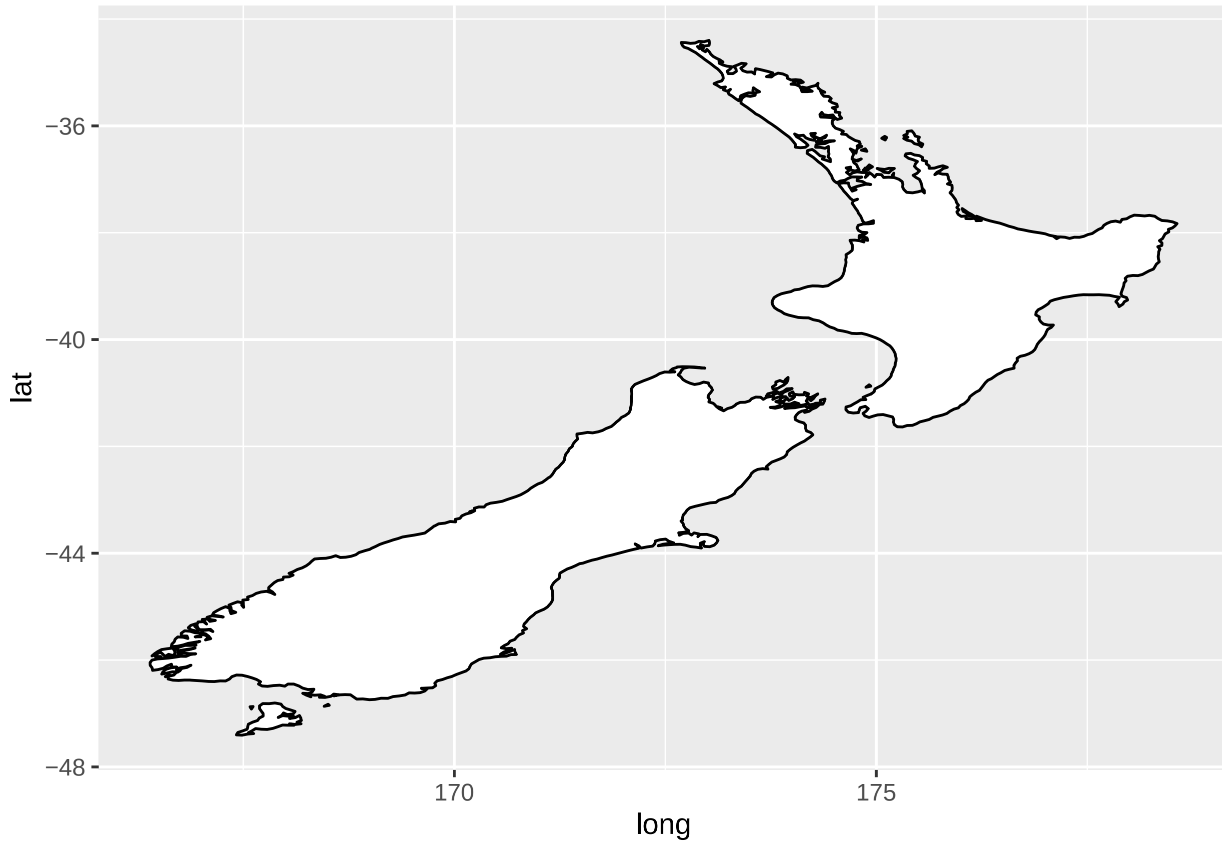



```r
ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
  geom_boxplot() +
  coord_flip()
```



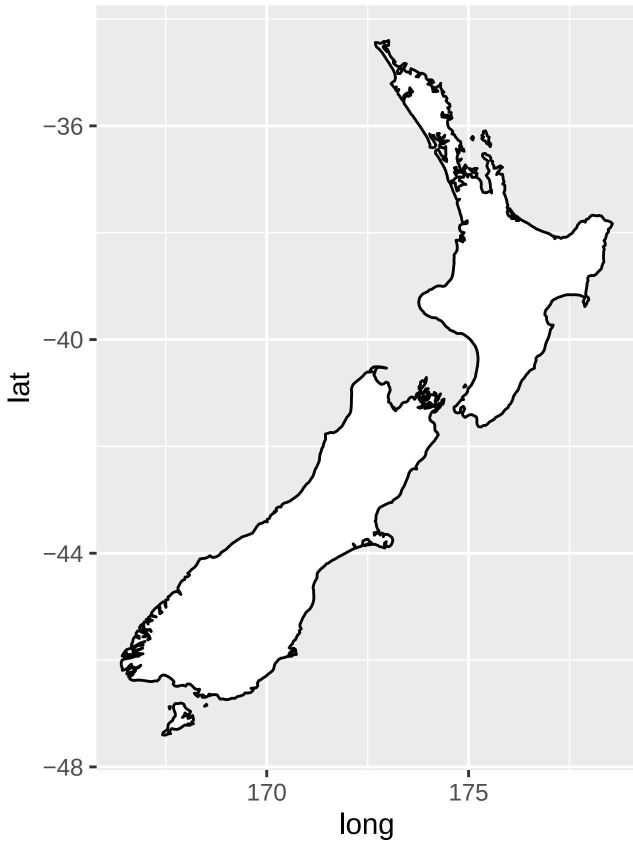

```
nz <- map_data("nz")

ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", color = "black")
```



coord_quickmap() sets the aspect ratio correctly for maps. This is very important if you're plotting spatial data with ggplot2 (which unfortunately we don't have the space to cover in this book):

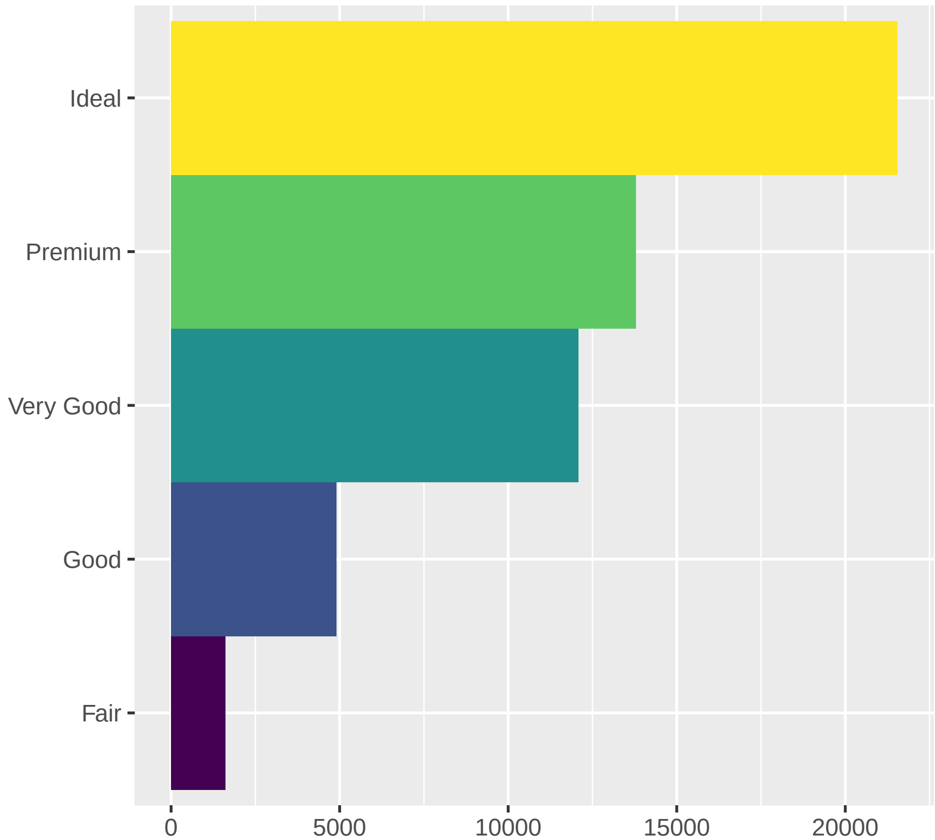
```
ggplot(nz, aes(long, lat, group = group))+
  geom_polygon(fill = "white", color = "black") +
  coord_quickmap()
```



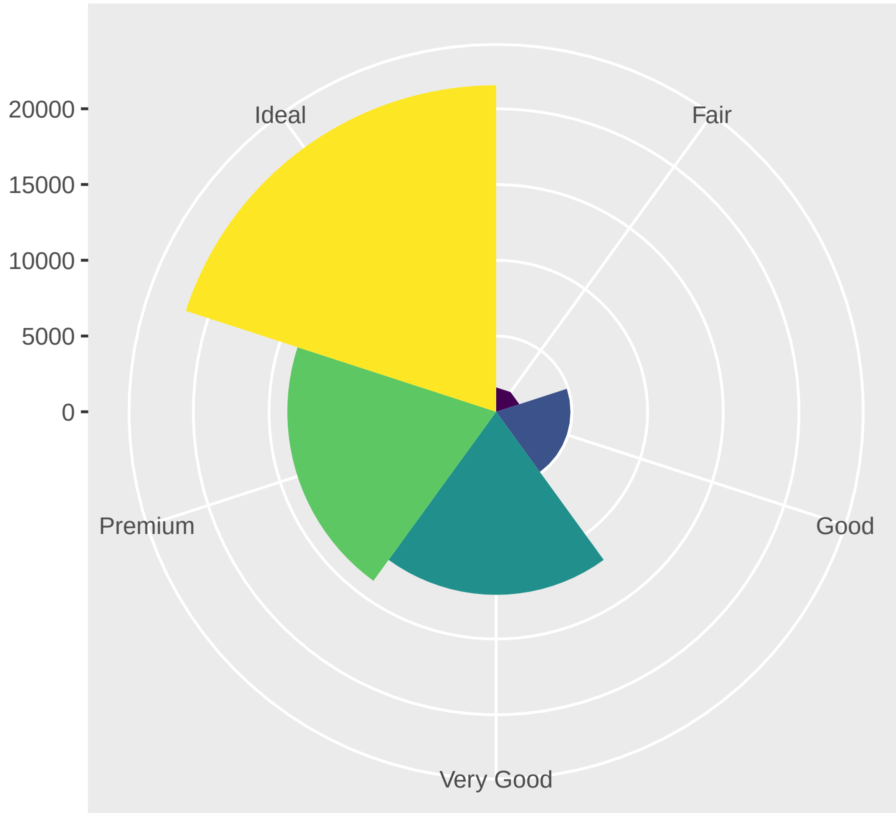
coord_polar() uses polar coordinates. Polar coordinates reveal an interesting connection between a bar chart and a Coxcomb chart:

```
bar <- ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = cut, fill = cut),
    show.legend = FALSE,
    width = 1
  ) +
  theme(aspect.ratio = 1) +
  labs(x = NULL, y = NULL)

  bar + coord_flip()
```




```
bar + coord_polar()
```




```
my_val <- 10
my_val
```

```
## [1] 10
```