Graphics

DJM

11 April 2017

Making better graphics

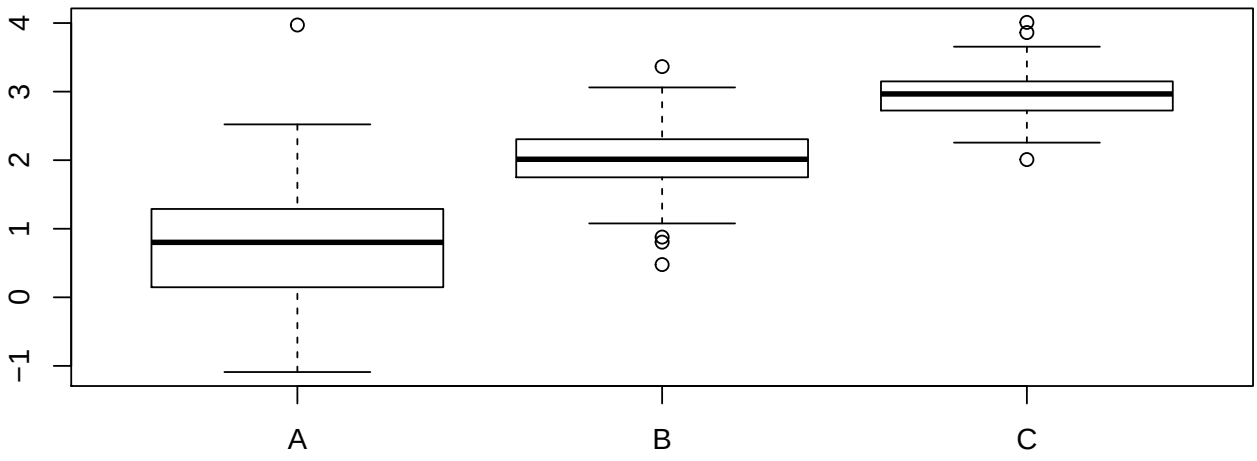
> "The simple graph has brought more information to the data analyst's mind than any other device." — John Tukey

- This week we're going to learn how to use `ggplot2`, Hadley Wickham's package for making graphics.
- This material is based on Chapter 3 of his book "`R` for data science".
- That is your reading for this week (link on syllabus page).

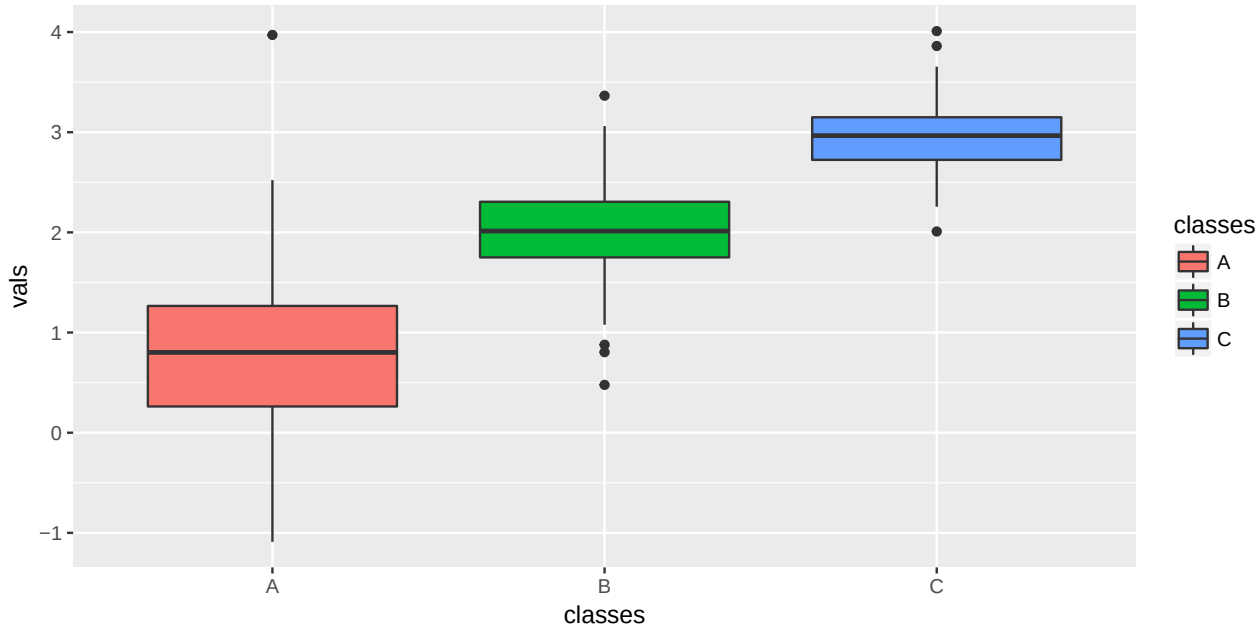
Why ggplot?

- `ggplot2` is a bit hard to get used to.
- But it's figures are much better looking than `R`'s `plot` functions.
- Compare:

```
boxplot(vals~classes, data=df)
```



```
ggplot(df,mapping=aes(x=classes,y=vals,fill=classes)) + geom_boxplot()
```



Graphics are for addressing questions

- Do cars with big engines use more fuel than cars with small engines?
- You probably already have an answer, but try to make your answer precise.
- What does the relationship between engine size and fuel efficiency look like?
- Is it positive? Negative? Linear? Nonlinear?

Some data

```
mpg
```

```
## # A tibble: 234 × 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>
```

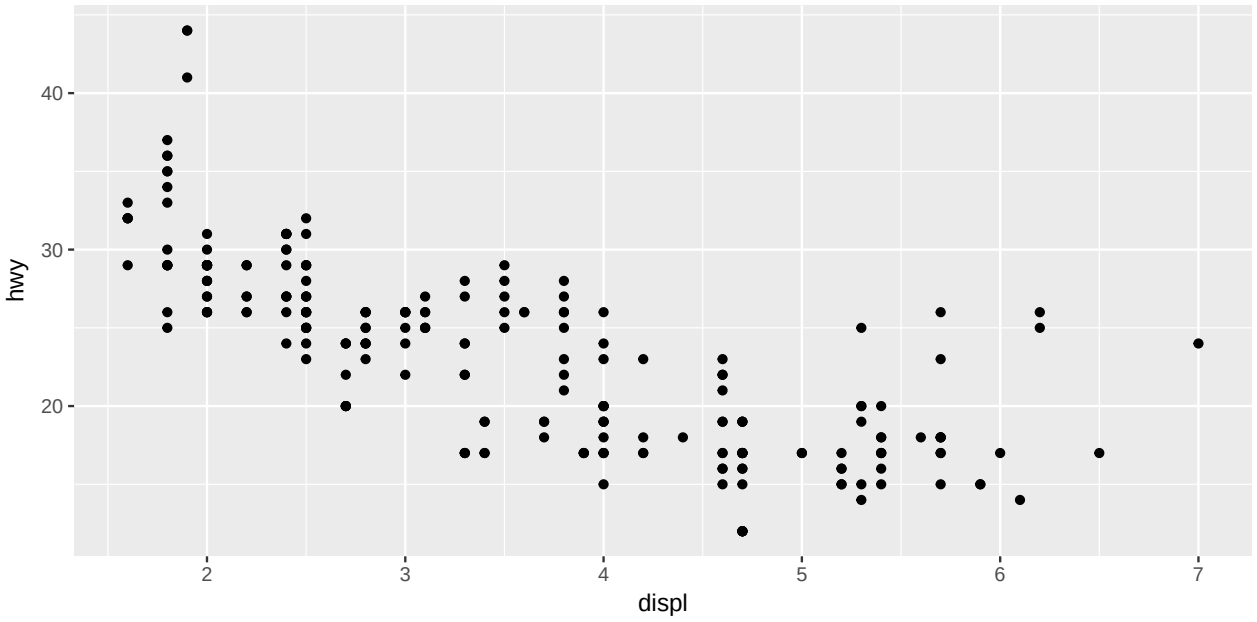
Among the variables in `mpg` are:

1. `displ`, a car's engine size, in litres.

2. `hwy`, a car's fuel efficiency on the highway, in miles per gallon (mpg). A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance.

First plot

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

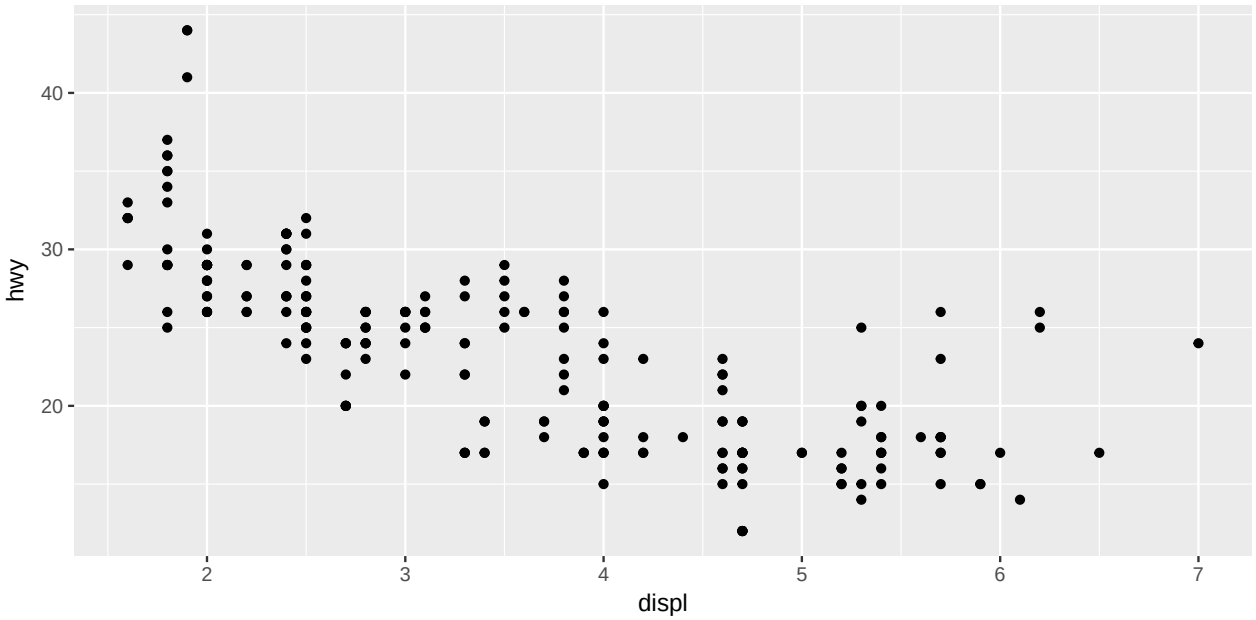


- `ggplot()` creates a coordinate system.
- The first argument of `ggplot()` is the dataset to use in the graph. So `ggplot(data = mpg)` creates an empty graph, but it's not very interesting.
- `geom_point()` adds a layer of points to your plot.
- Each geom function in ggplot2 takes a `mapping` argument.
- This defines how variables in your dataset are mapped to visual properties.
- The `mapping` argument is always paired with `aes()`, and the `x` and `y` arguments of `aes()` specify which variables to map to the x and y axes.
- ggplot2 looks for the mapped variable in the `data` argument, in this case, `mpg`.

Again...

Note:

```
ggplot(mpg) + geom_point(aes(displ,hwy))
```

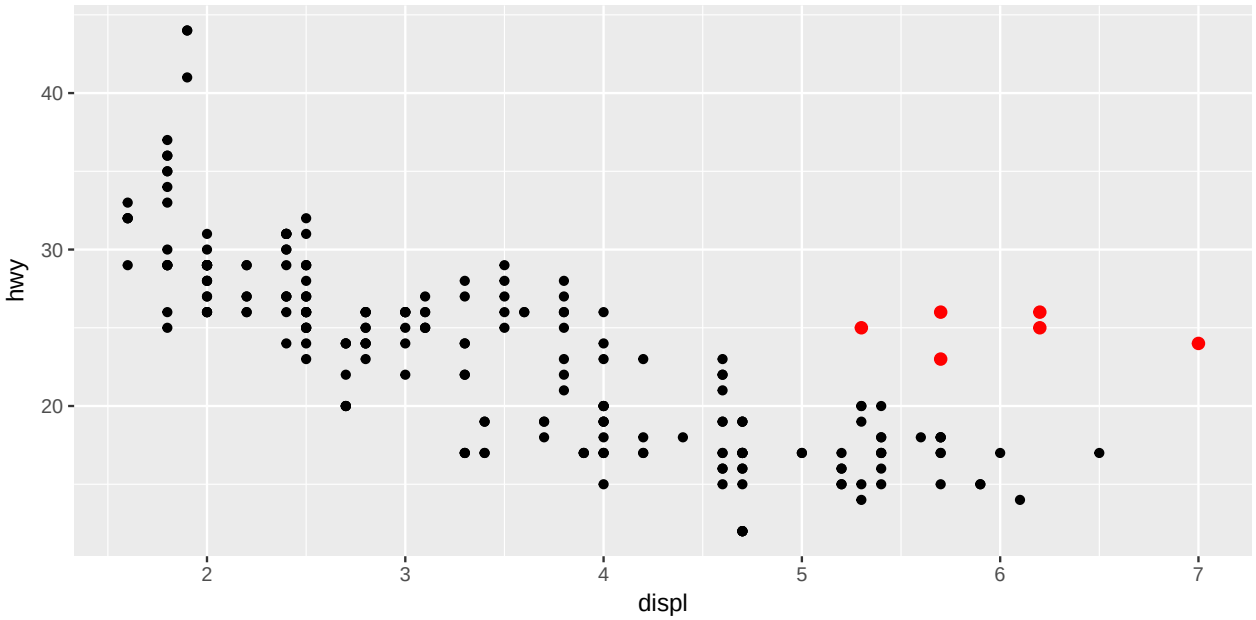


In general:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Aesthetics

> "The greatest value of a picture is when it forces us to notice what we never expected to see." — John Tukey

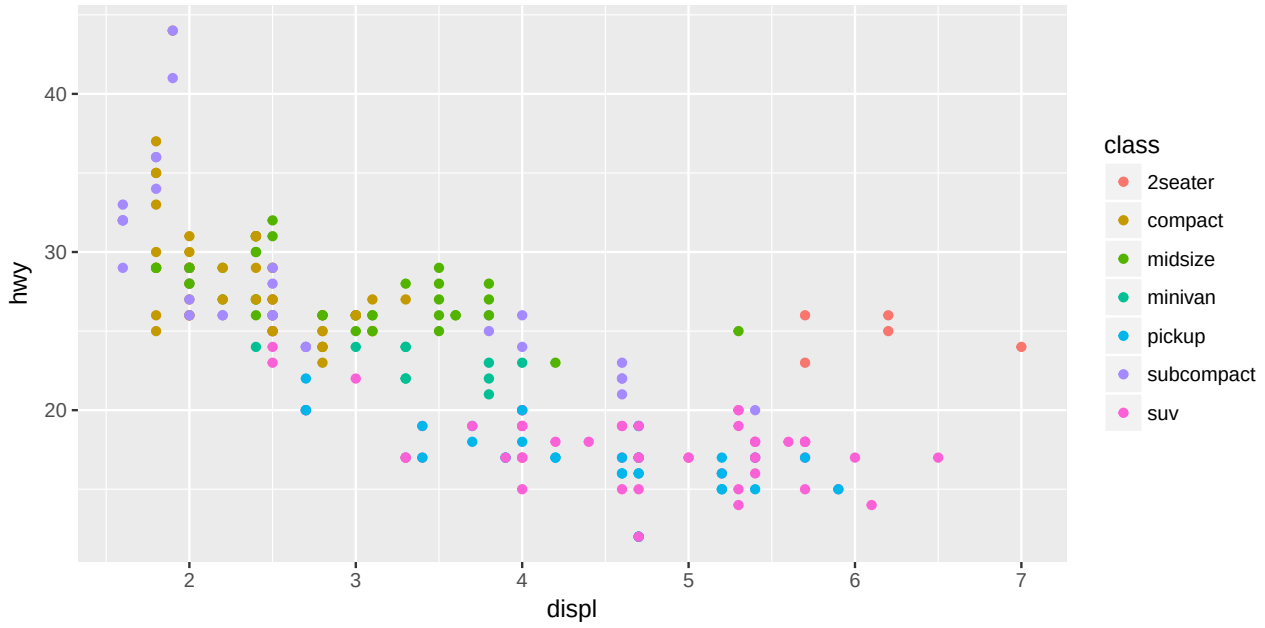


- The red points seem to be outliers.

Coloring

- Perhaps another variable explains them (say, hybrid SUVs)

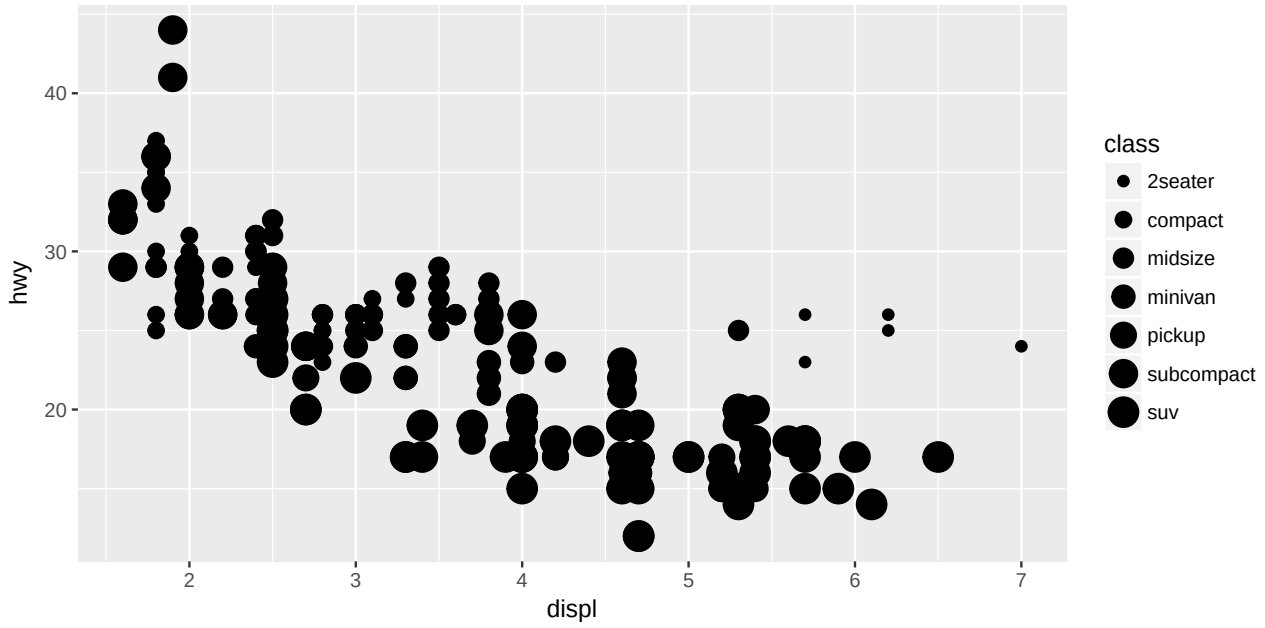
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

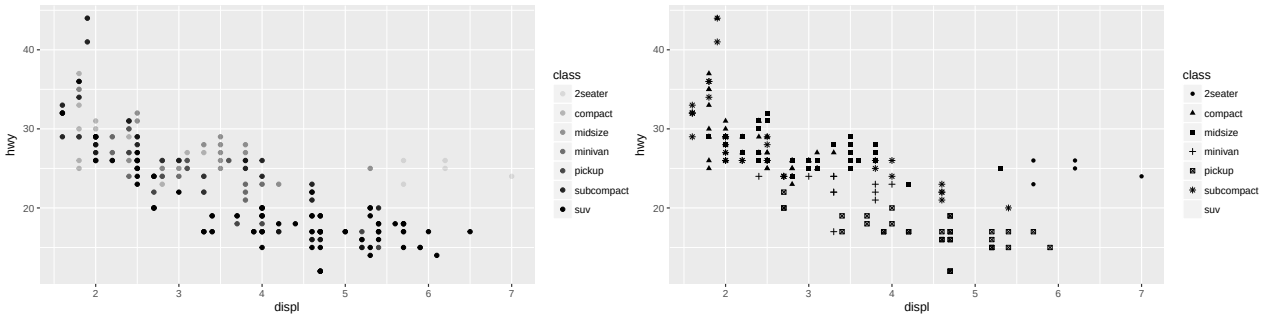


Size

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

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





Transparency and shape

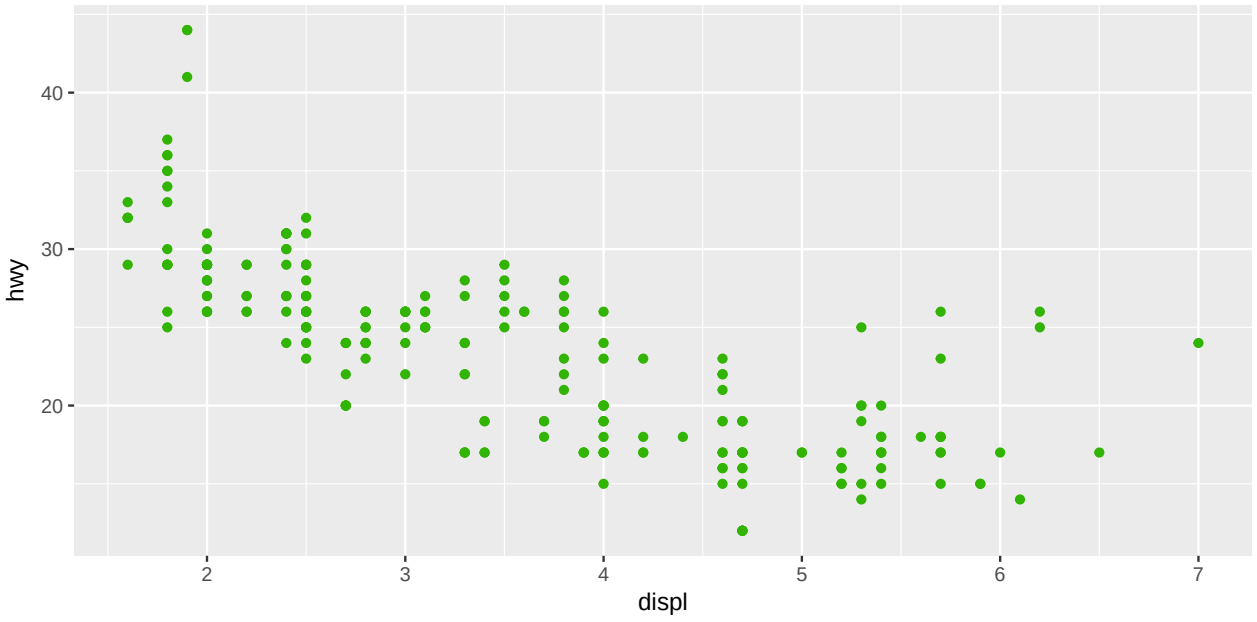
```
# Left
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))

# Right
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).
```

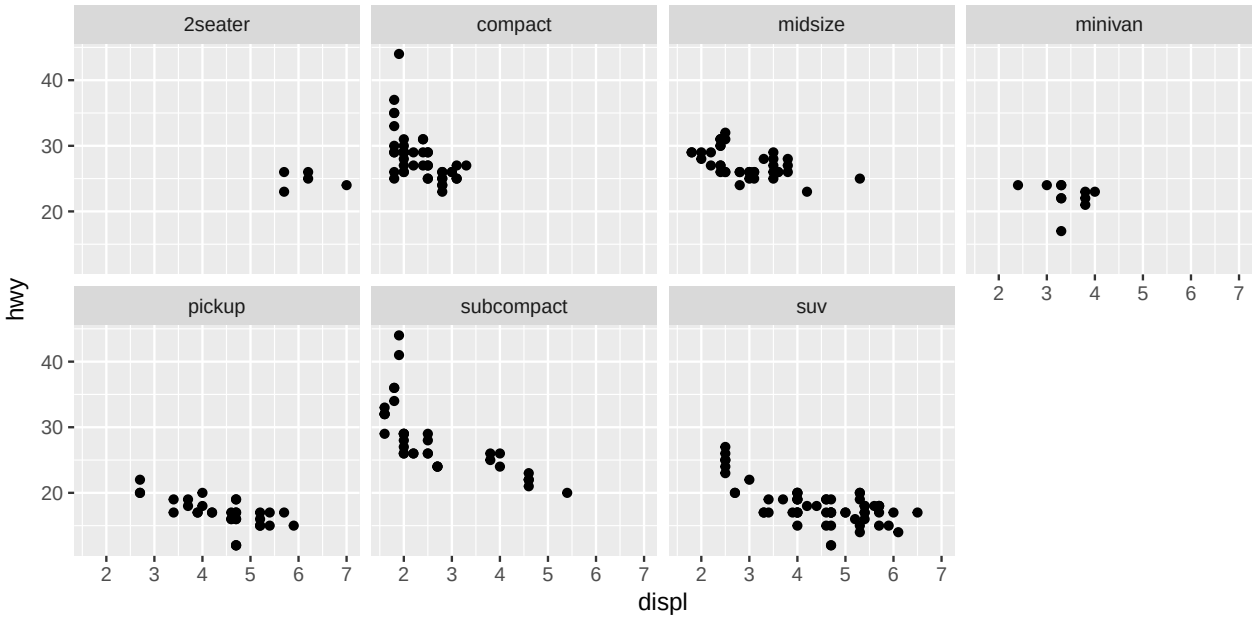
Just like some color

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "#31B404")
```

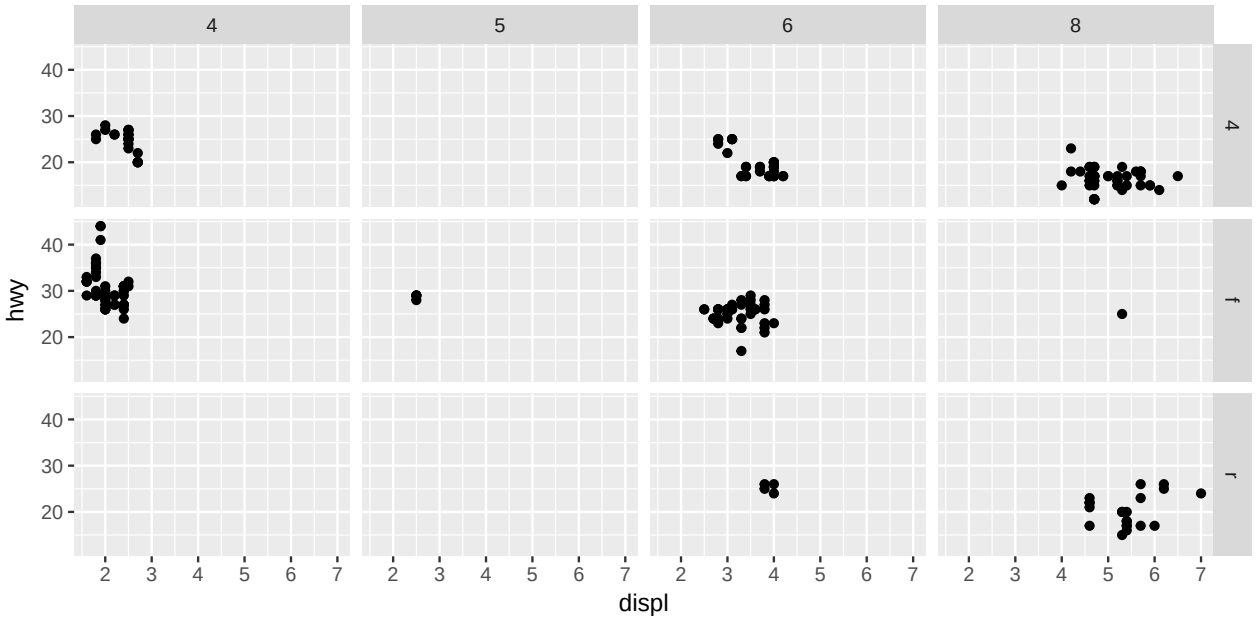


Facets

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

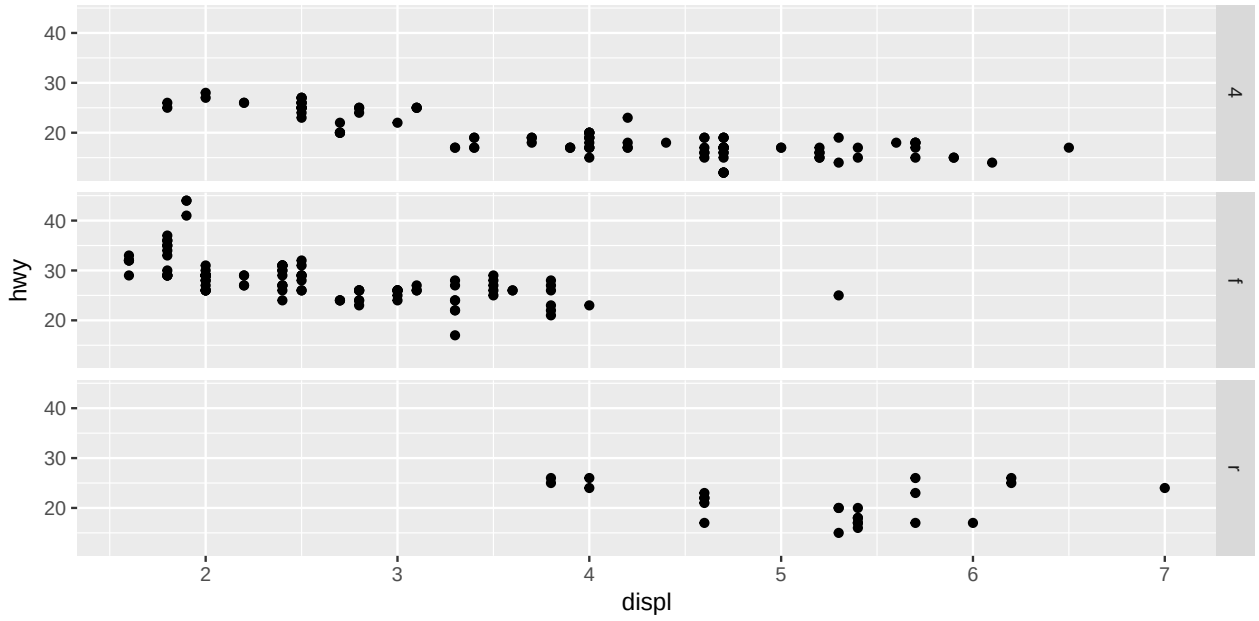



```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)
```

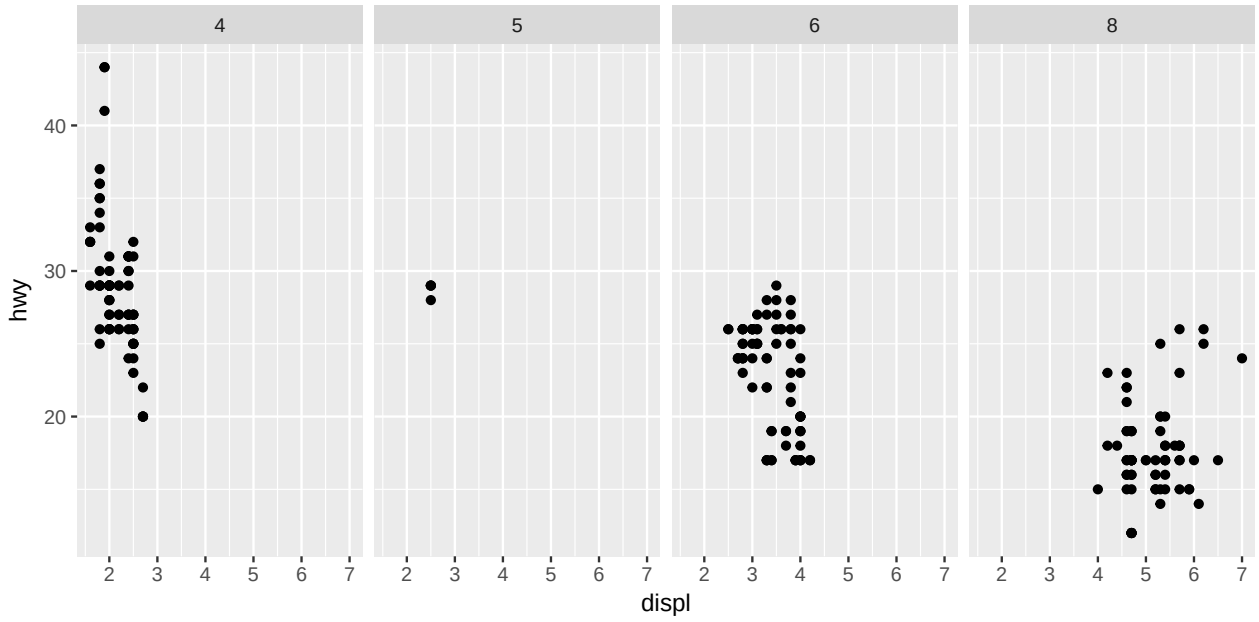


Rows or columns

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

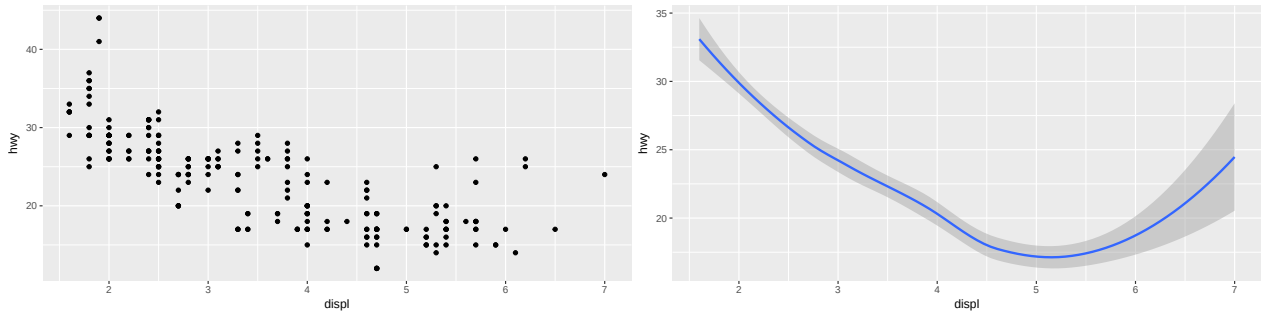



```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(. ~ cyl)
```



What are geoms?

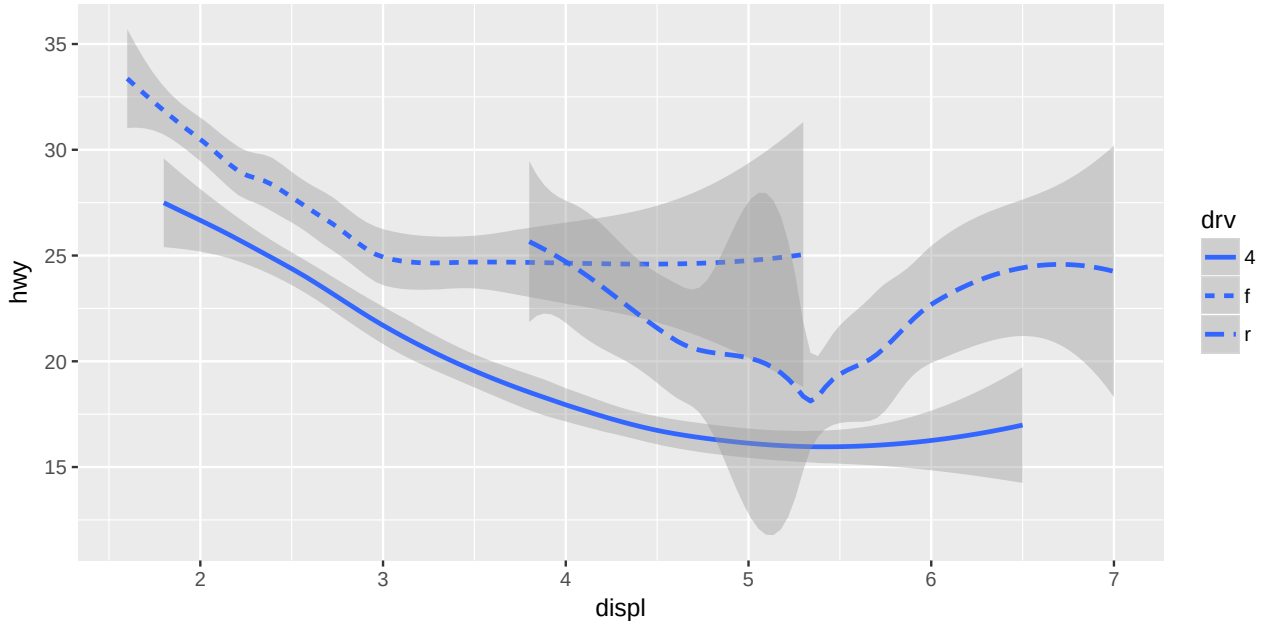
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))

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



- Different **geoms** lead to different kinds of geometric objects.
- bar charts use bar geoms,
- line charts use line geoms,
- boxplots use boxplot geoms
- Scatterplots break the trend; they use the point geom.

Line types

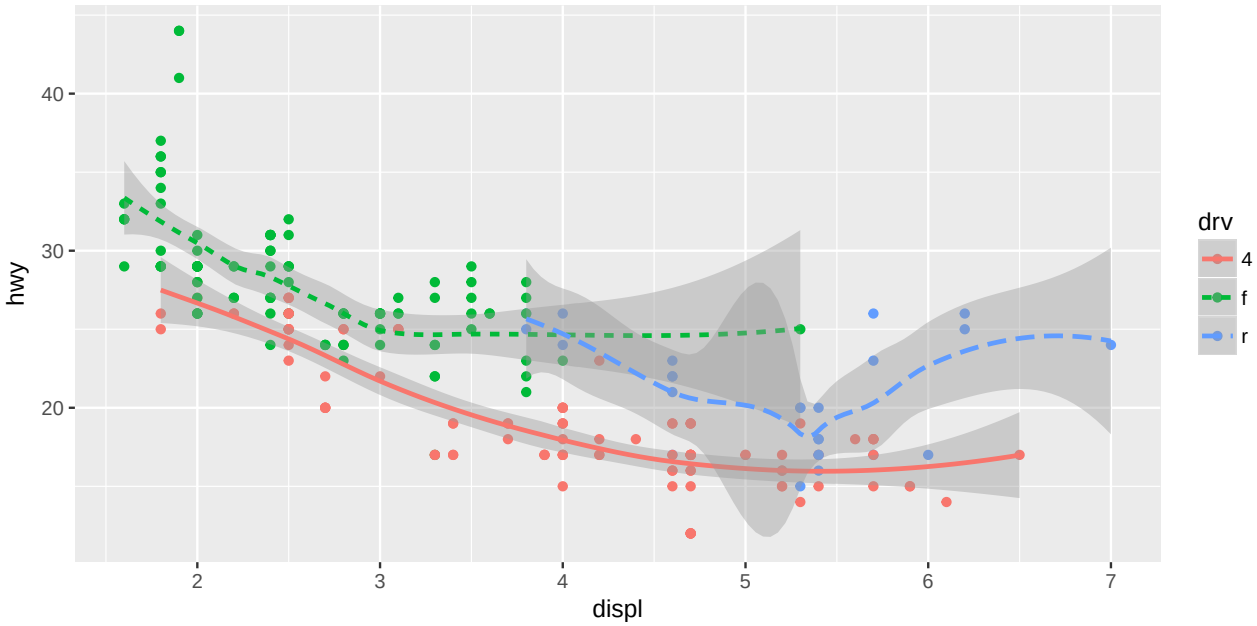
```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```



- One line describes all of the points with a `4` value,

- one line describes all of the points with an `f` value,
- and one line describes all of the points with an `r` value.

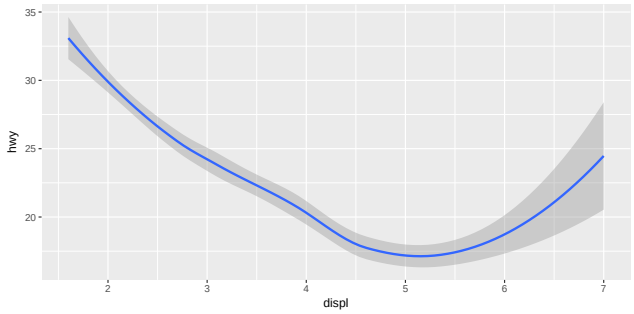
More clear

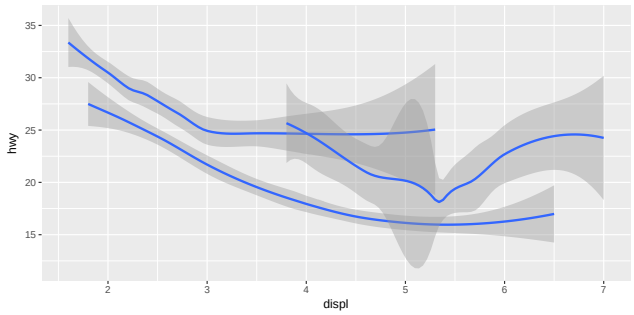


Grouping

- Many geoms use a single geometric object to display multiple rows of data.
- you can set the `group` aesthetic to a categorical variable to draw multiple objects.
- ggplot2 will draw a separate object for each unique value of the grouping variable.
- the `group` aesthetic by itself does not add a legend or distinguishing features to the geoms (`linetype` or `color` do).

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

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

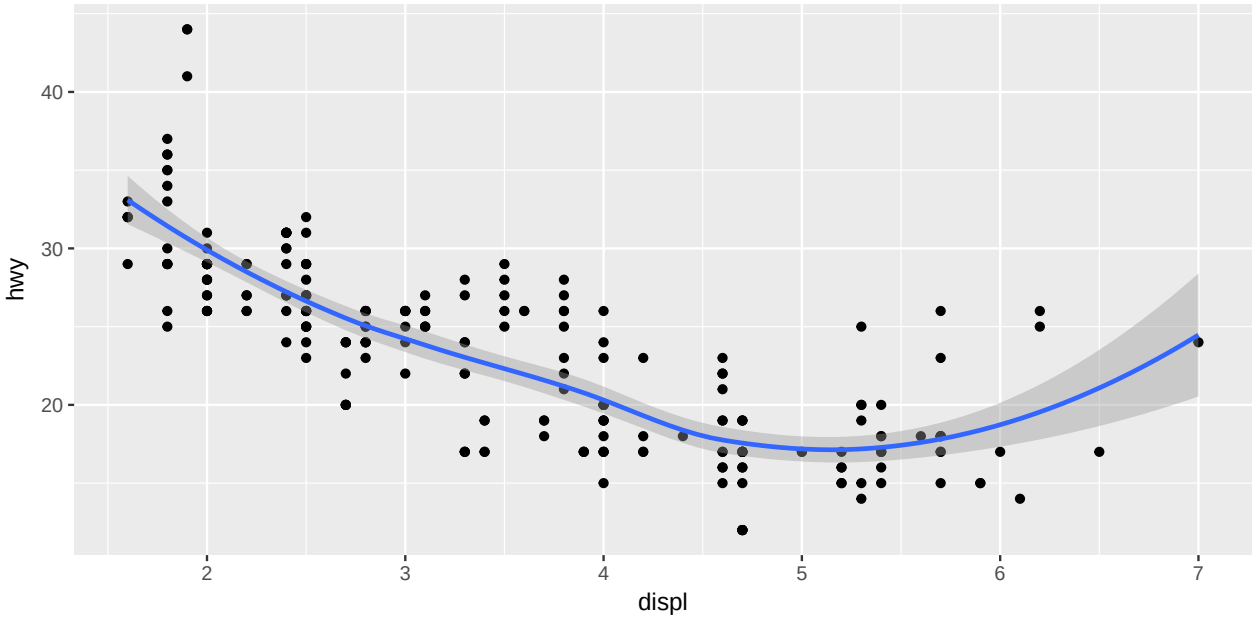




Multiple geoms

- 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))
```



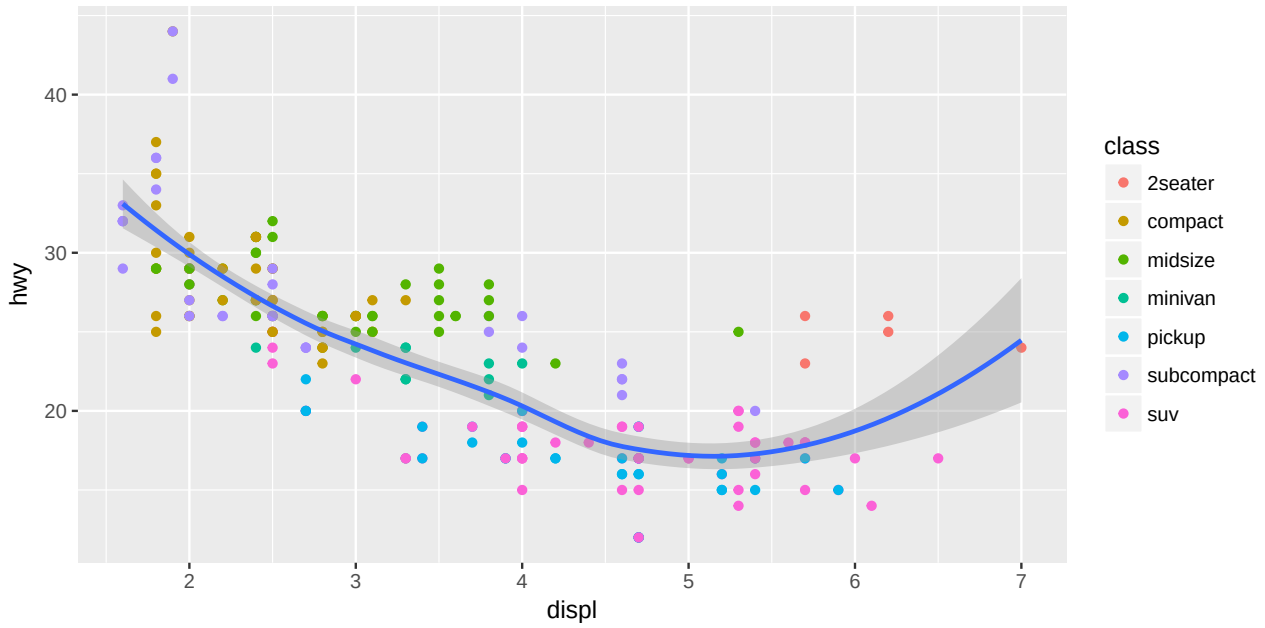
- Creates duplication, annoying to type `aes(x = displ, y = hwy)`.

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

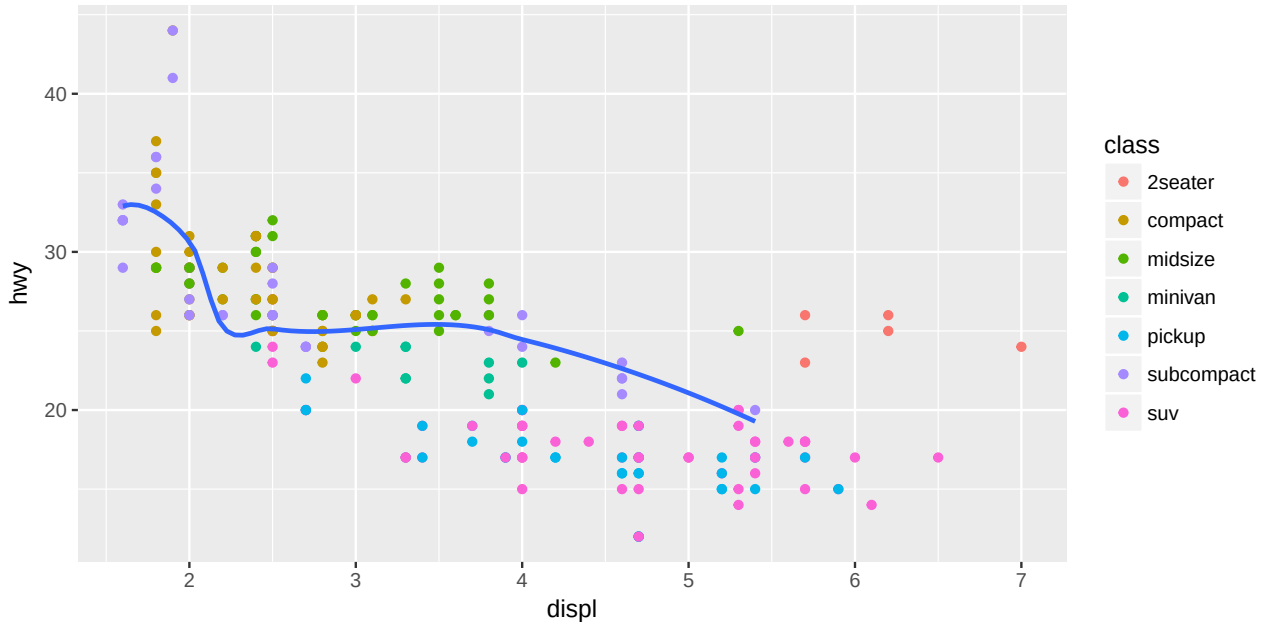
- mappings in `ggplot` are *global*, in a geom, they are *local* to the geom.

Ex

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



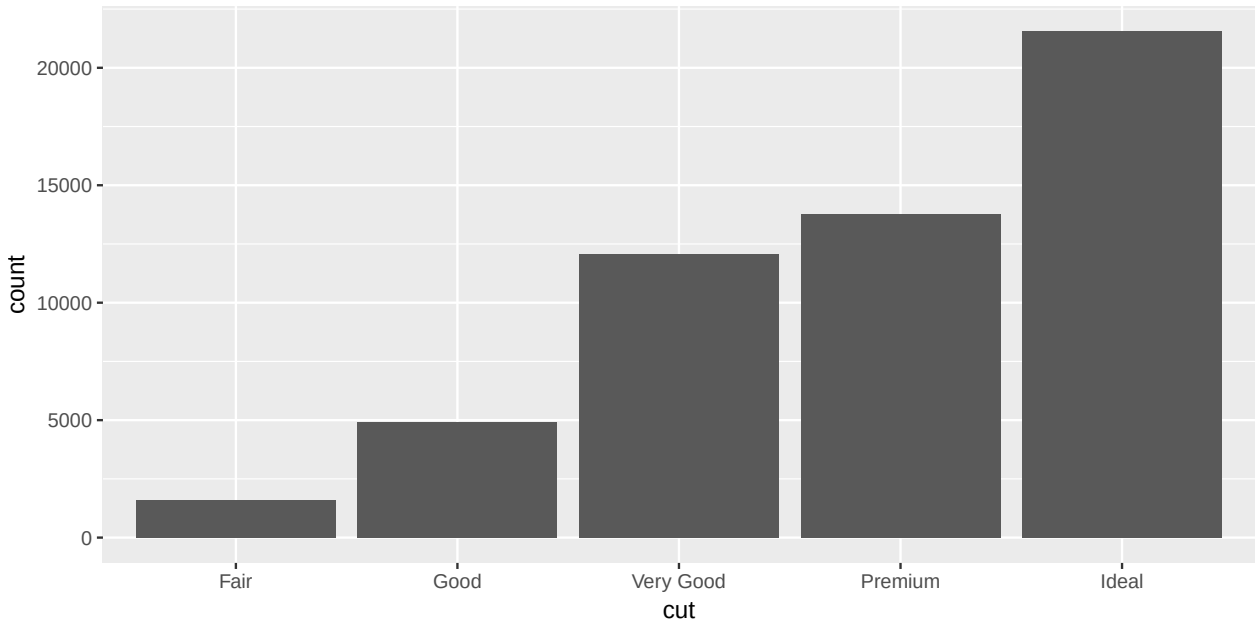

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(data = filter(mpg, class == "subcompact"), se = FALSE)
```



Bar charts

- The `diamonds` dataset comes in ggplot2 and contains information about ~54,000 diamonds,
- Information about the `price`, `carat`, `color`, `clarity`, and `cut` of each diamond.

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



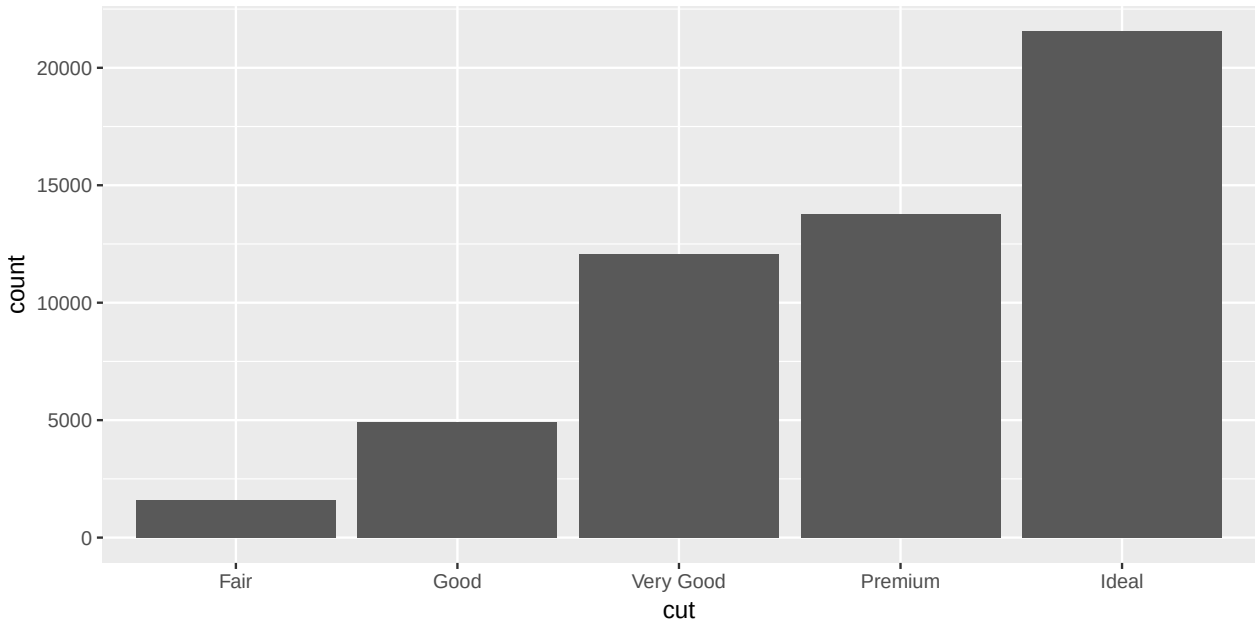
Some graphs, like bar charts, calculate new values to plot:

- bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- smoothers fit a model to your data and then plot predictions from the model.
- boxplots compute a robust summary of the distribution and then display a specially formatted box.

The algorithm used to calculate new values for a graph is called a **stat**, short for statistical transformation.

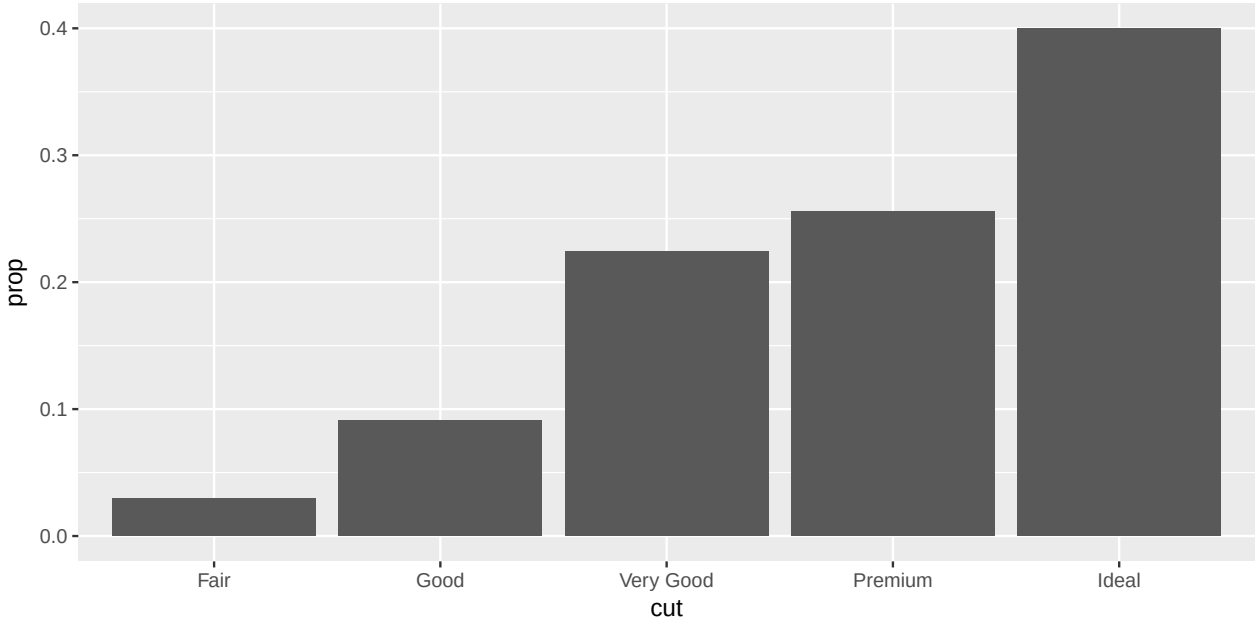
Some stats

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

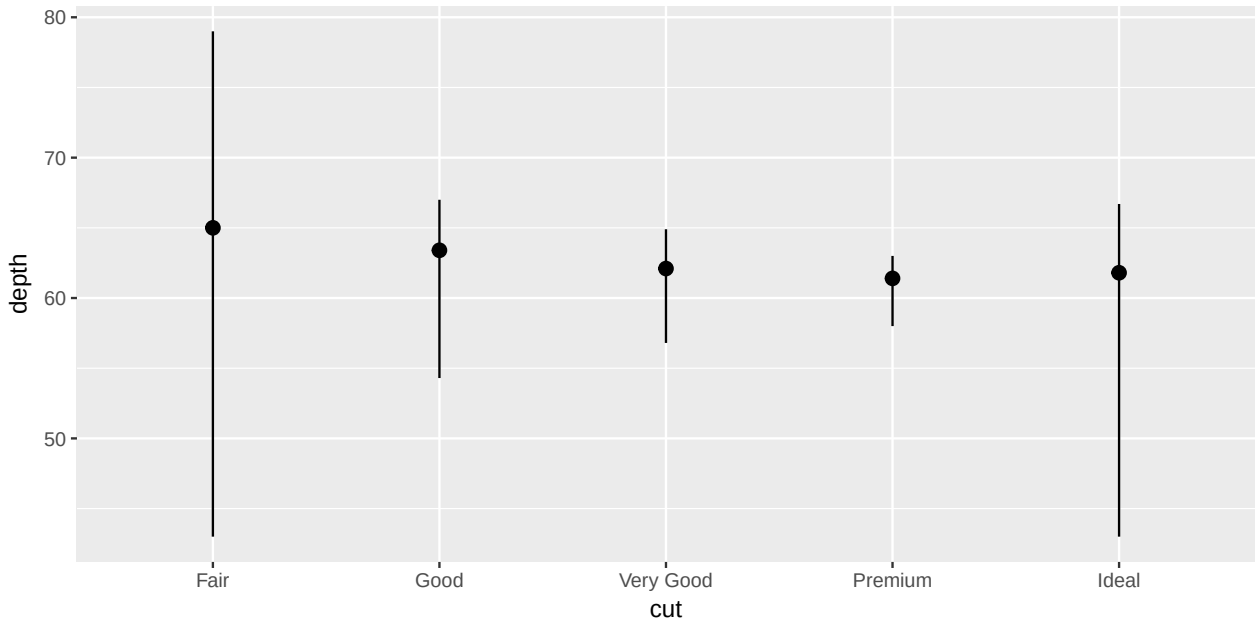


- Every geom has a default stat.

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



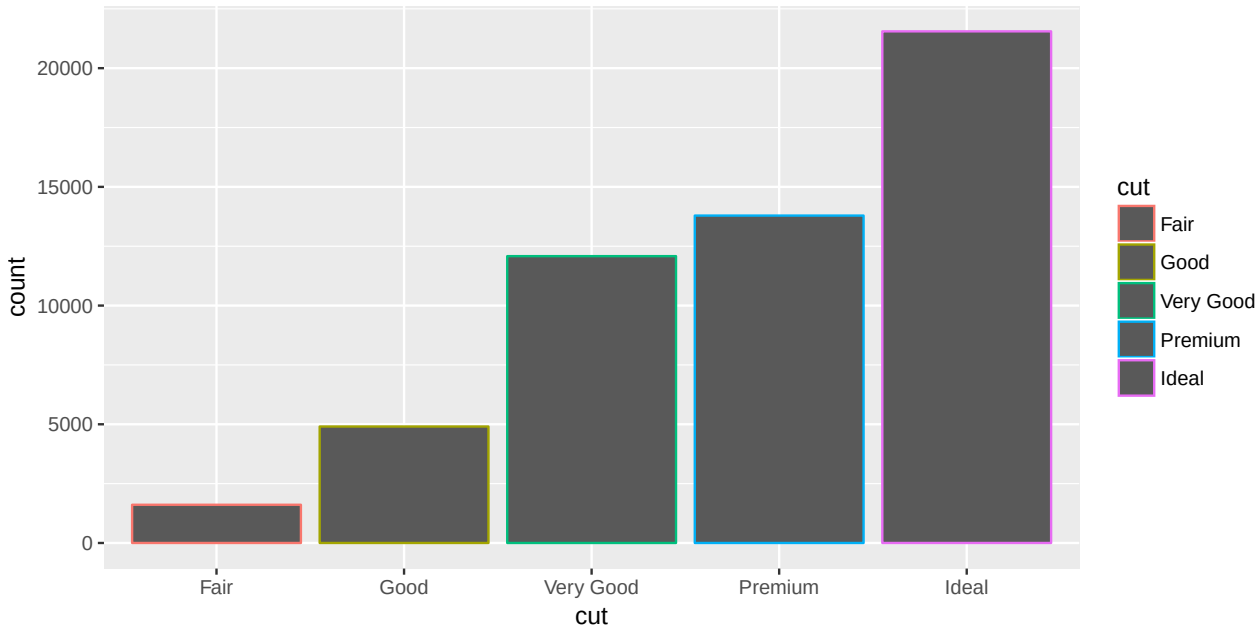

```
ggplot(data = diamonds) +
  stat_summary(
    mapping = aes(x = cut, y = depth),
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
    )
```

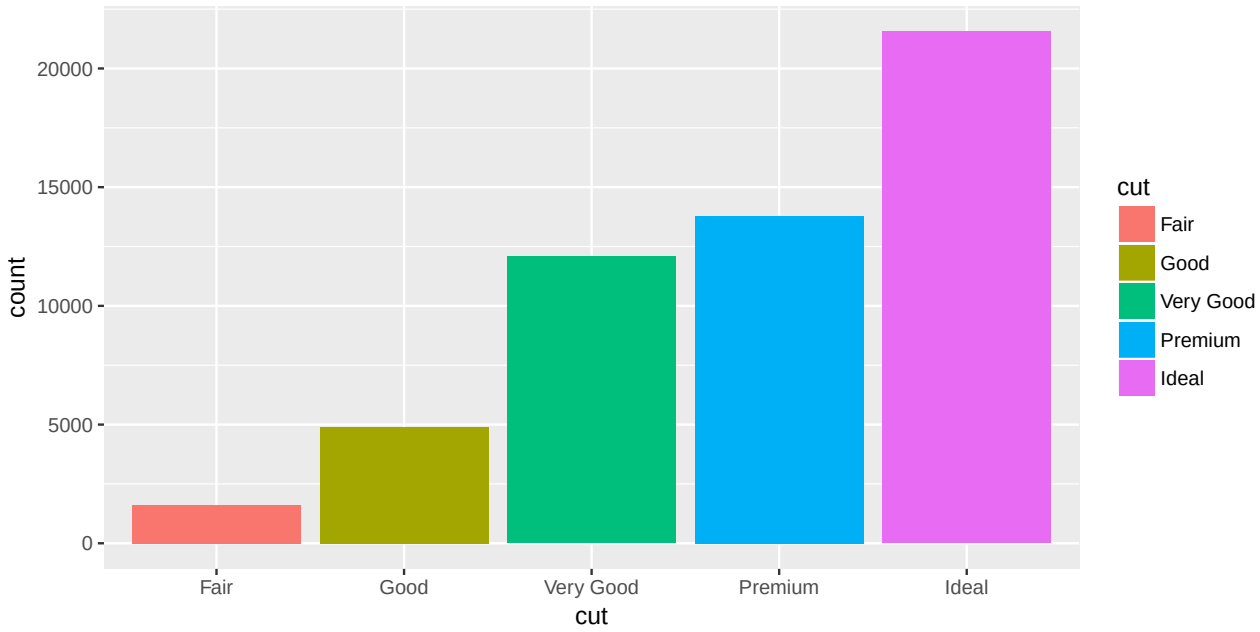


- ggplot2 provides over 20 stats for you to use.

Looking better

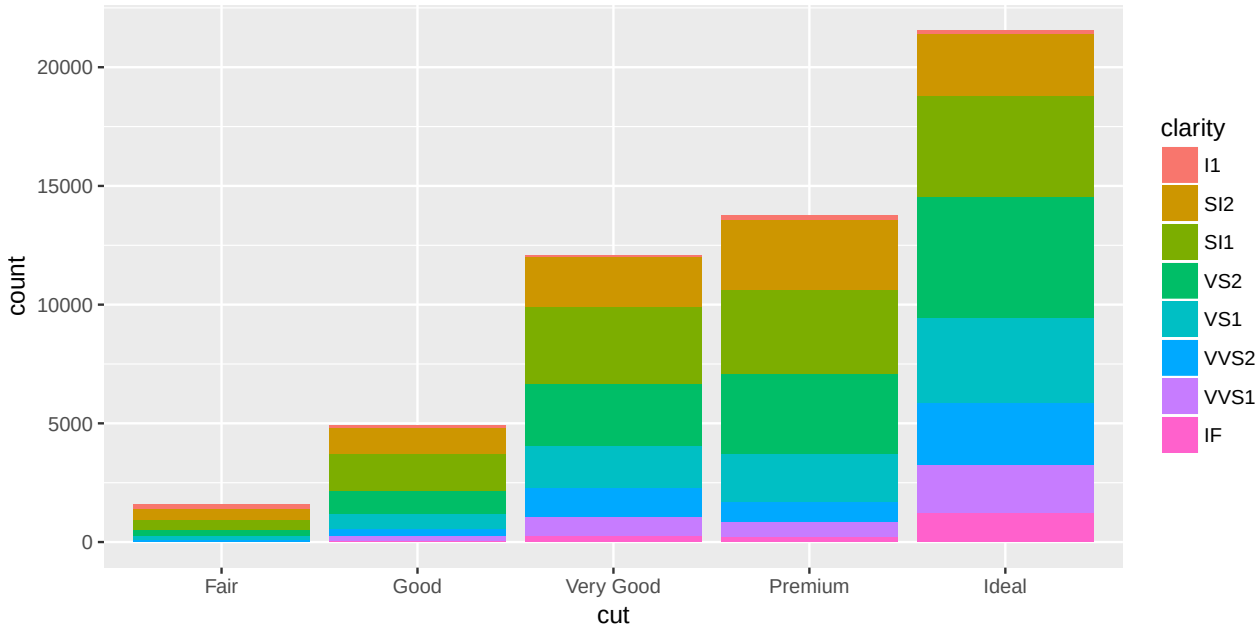
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, colour = cut))
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```





Stacking

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

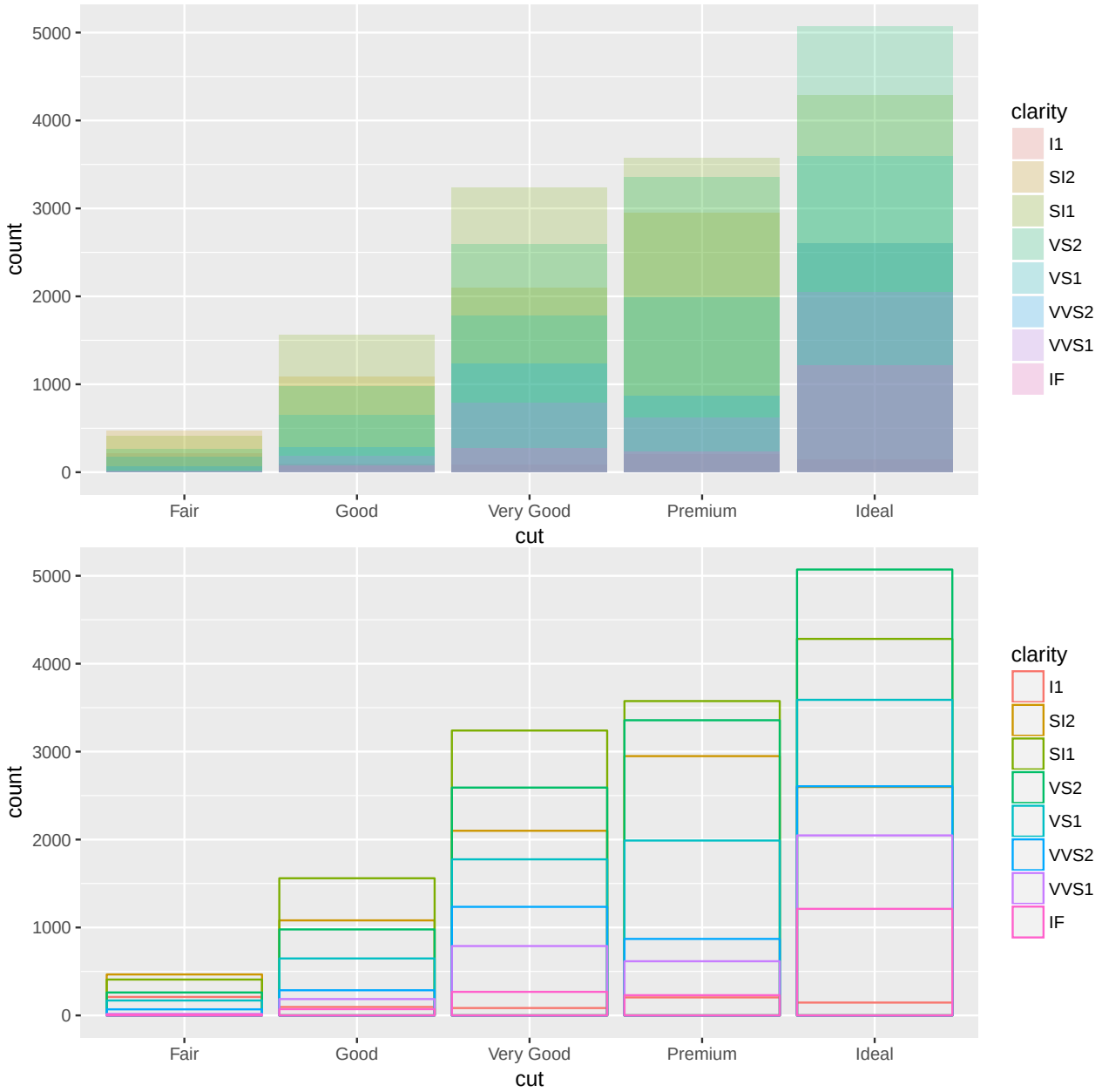


- The stacking is performed automatically by the **position adjustment** specified by the `position` argument.
- If you don't want a stacked bar chart, you can use one of three other options: `"identity"`, `"dodge"` or `"fill"`.

Identity

- `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, colour = clarity)) +
  geom_bar(fill = NA, position = "identity")
```

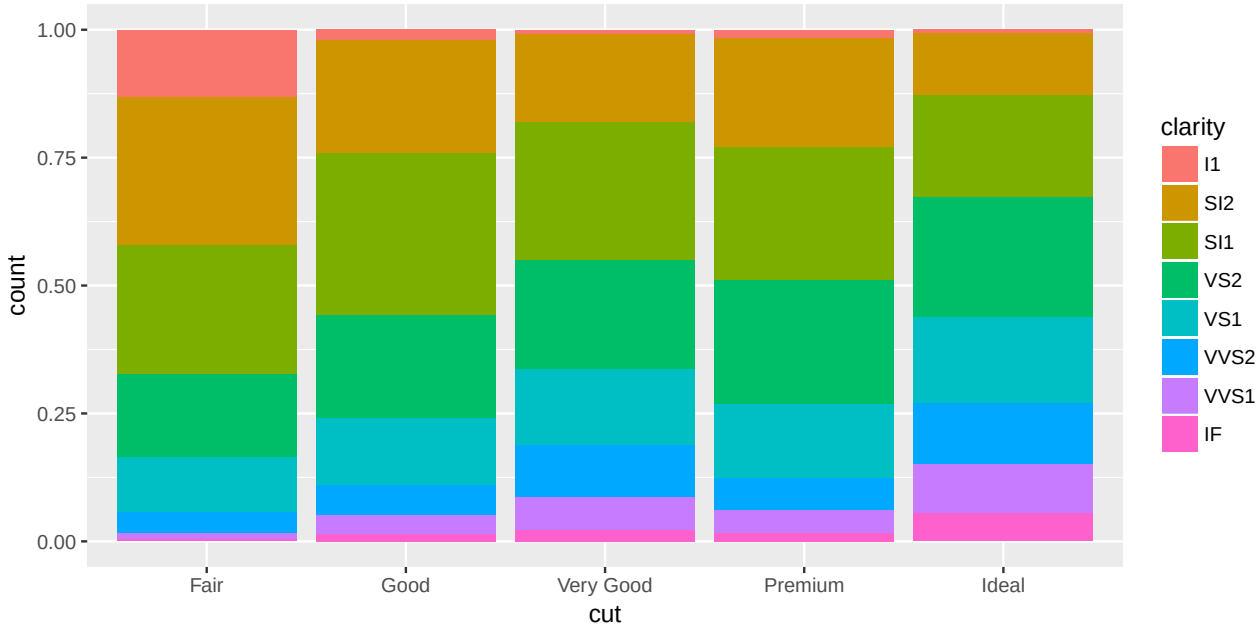


- The identity position adjustment is more useful for 2d geoms, like points, where it is the default.

Fill

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

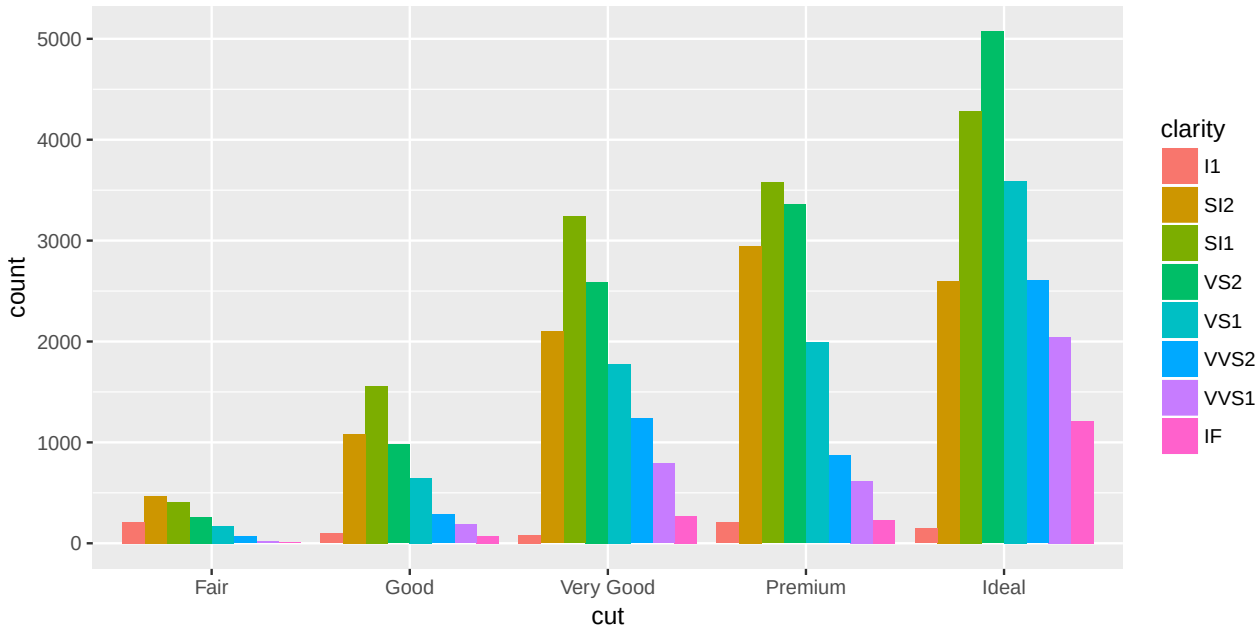
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



Dodge

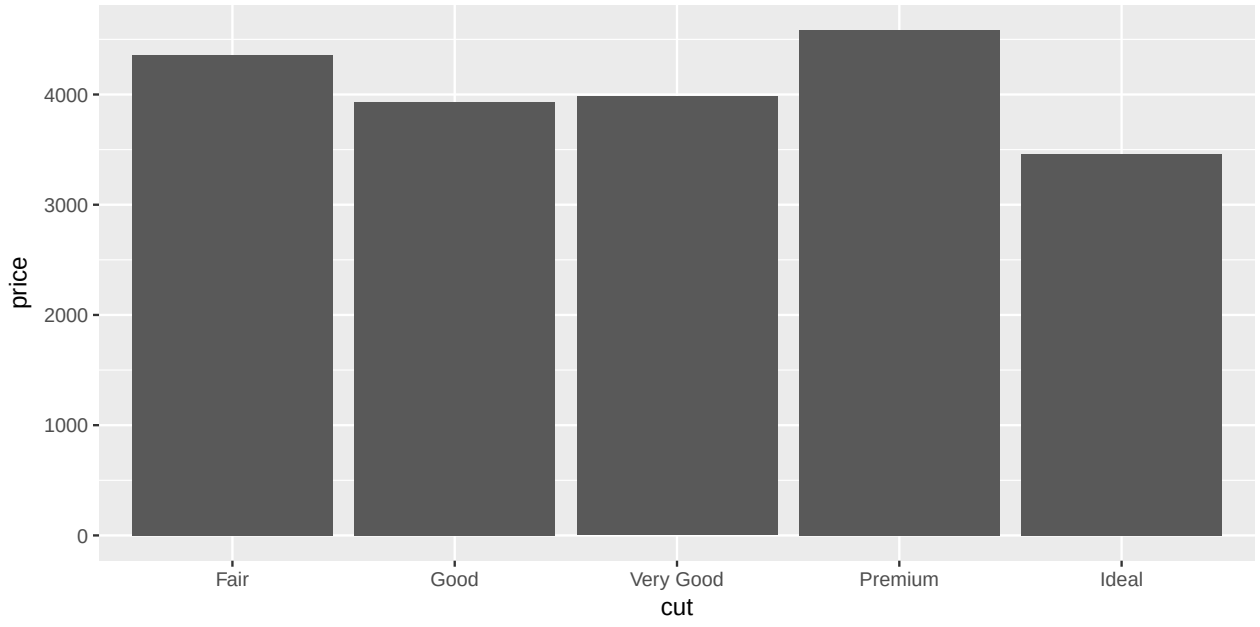
- `position = "dodge"` places overlapping objects directly *beside* one another.
- This makes it easier to compare individual values.

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

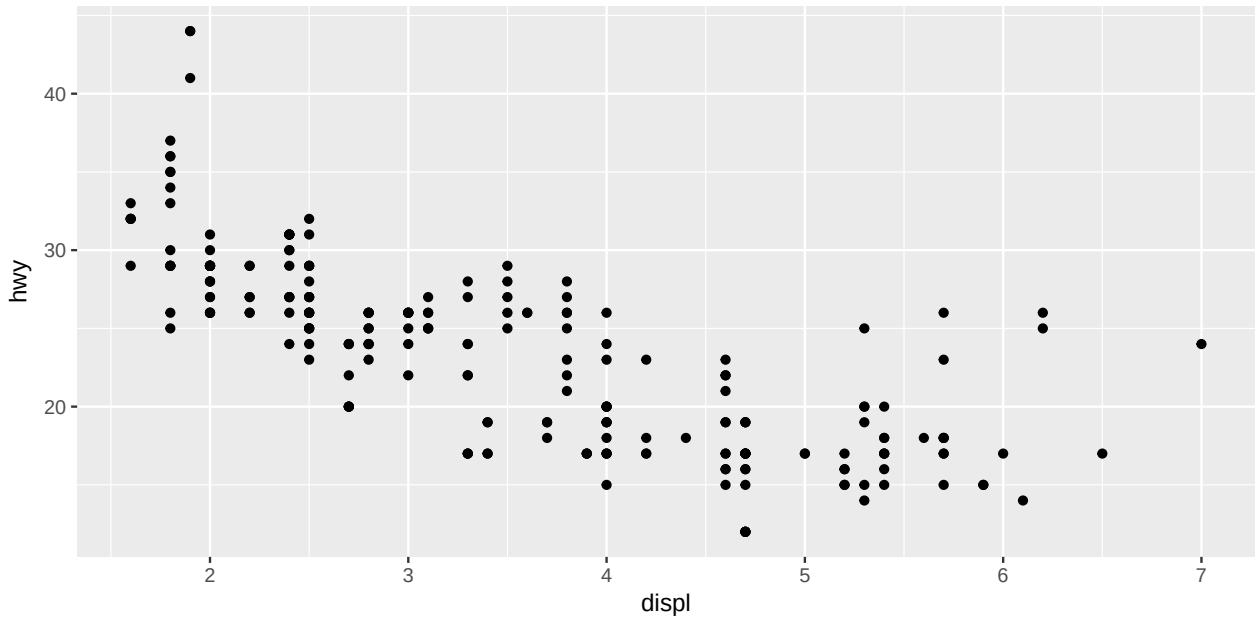


Bar charts with mean

```
ggplot(diamonds)+geom_bar(aes(x=cut,y=price),stat="summary",fun.y="mean")
```

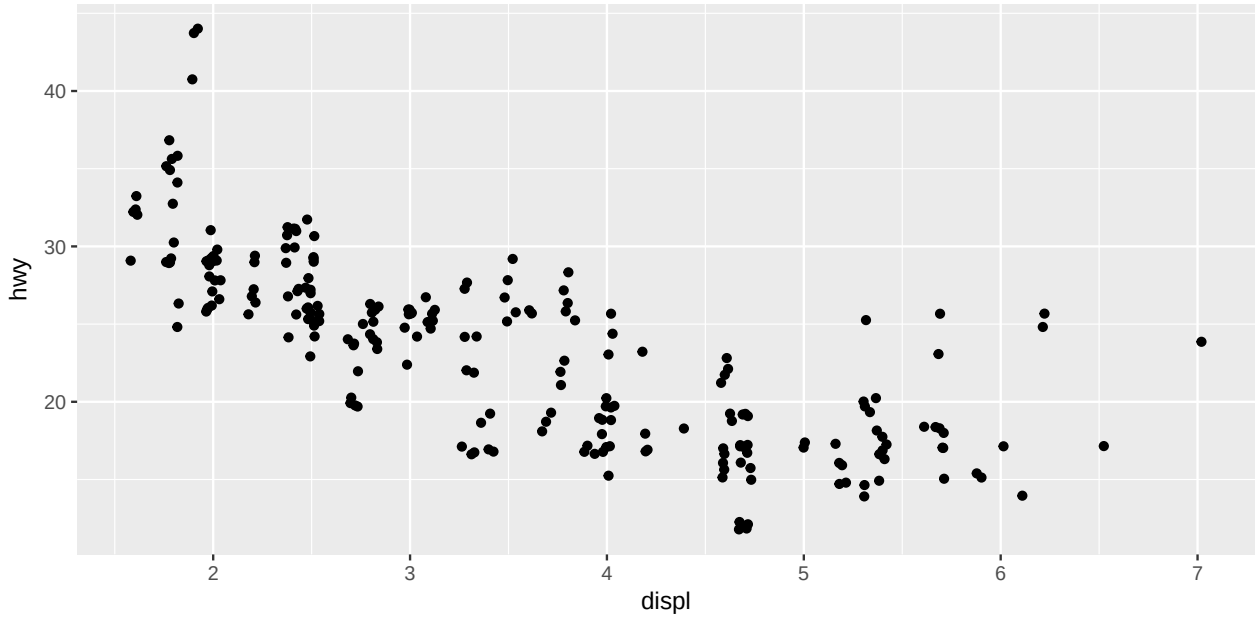


Jitter



- 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.
- `position = "jitter"` adds a small amount of random noise to each point.

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



Interactive plots

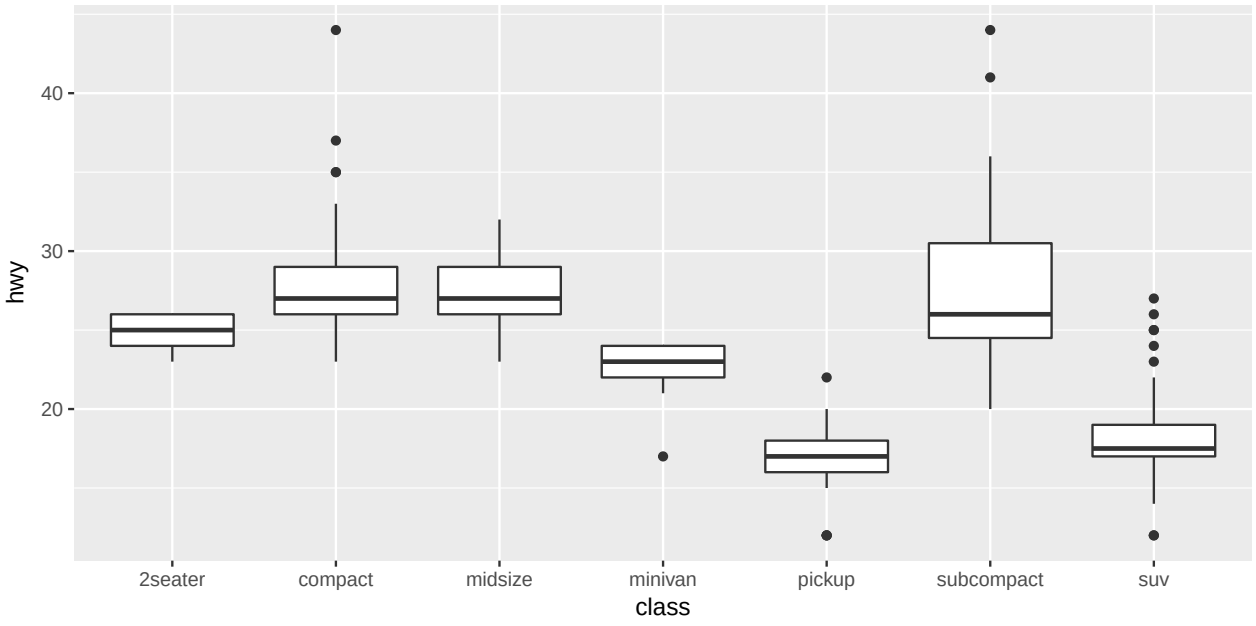
```
library(plotly)
g <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(aes(color=class), position = "jitter") +
  geom_smooth()
ggplotly(g)
```

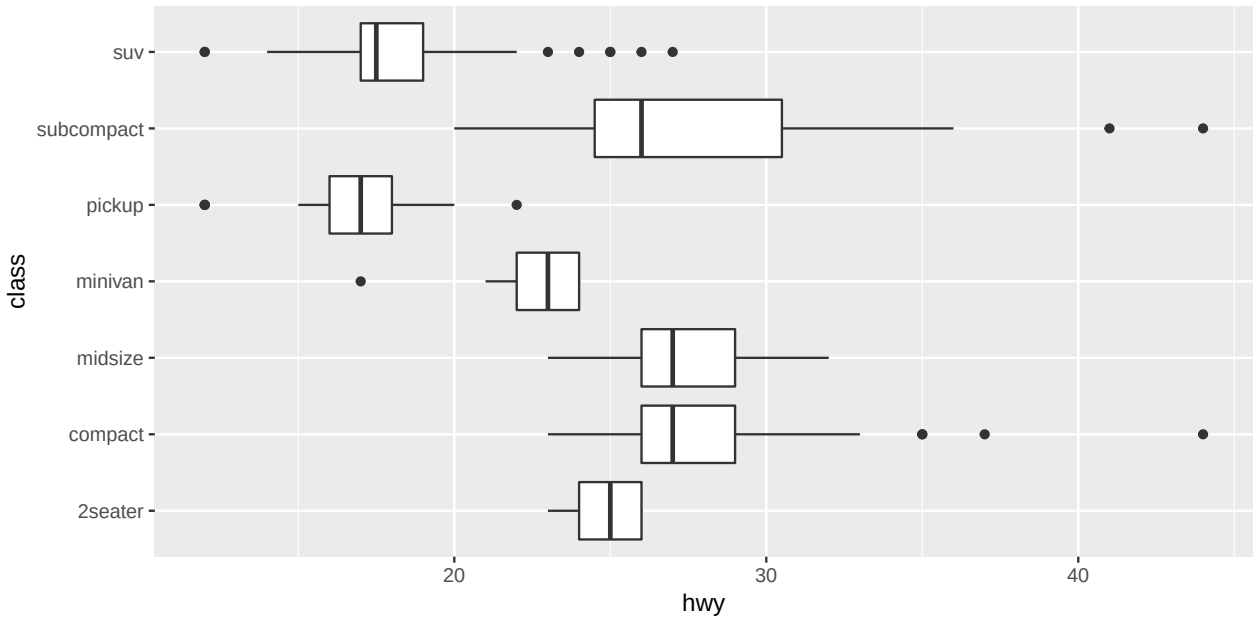
- This can do lots of other stuff, but I'm no expert.

Coordinate systems

- Coordinate systems are probably the most complicated part of ggplot2.
- The default coordinate system is the Cartesian coordinate system where the x and y position act independently to find the location of each point.
- There are a number of other coordinate systems that are occasionally helpful.
- `coord_flip()` switches the x and y axes.
- This is useful (for example), if you want horizontal boxplots.
- It's also useful for long labels: it's hard to get them to fit without overlapping on the x-axis.

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

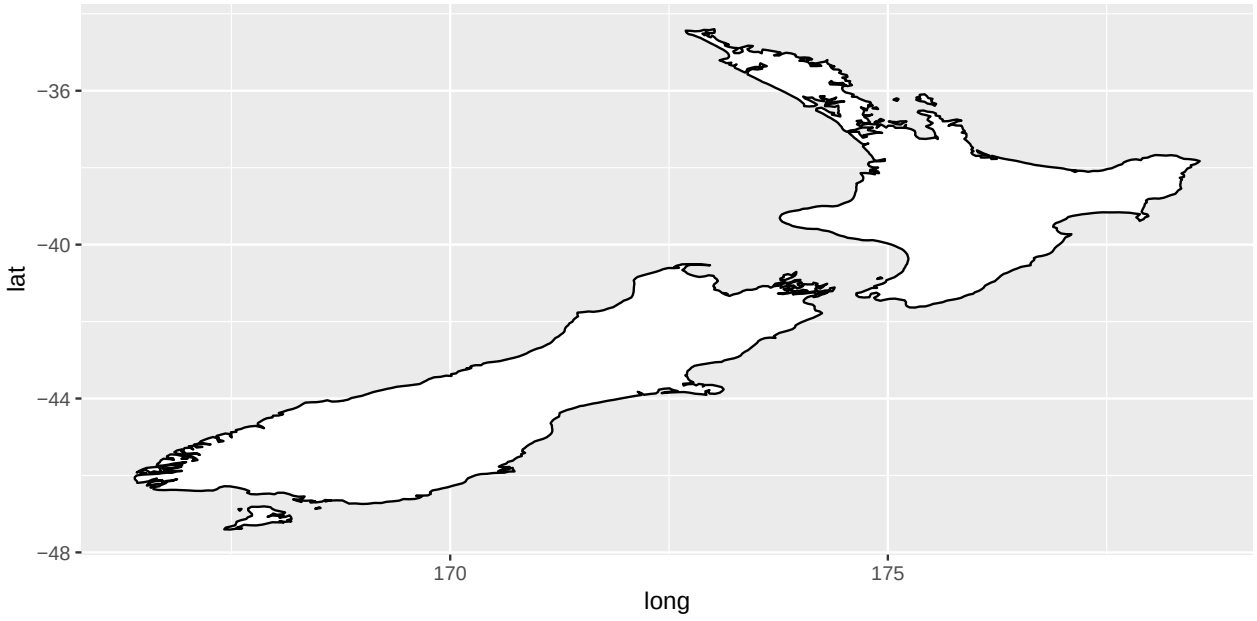


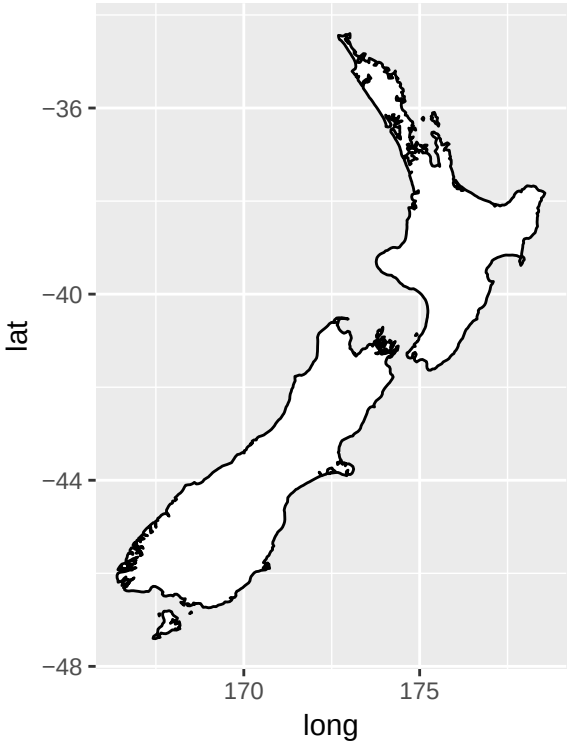


Quickmap

- `coord_quickmap()` sets the aspect ratio correctly for maps.
- important if you're plotting spatial data

```
nz <- map_data("nz")
ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black")
ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black") +
  coord_quickmap()
```

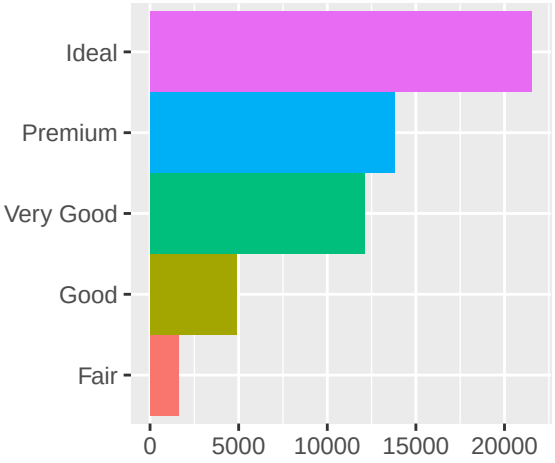




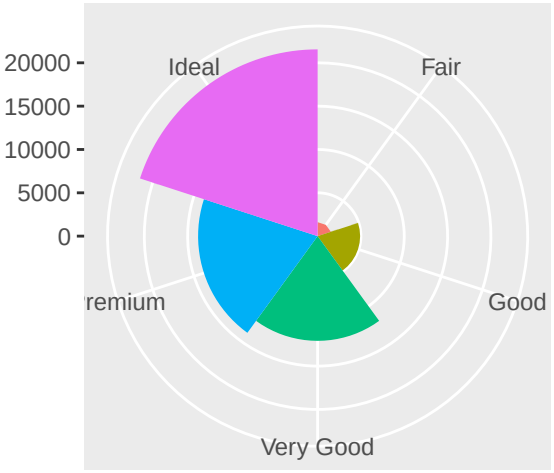
Polar

- `coord_polar()` uses polar coordinates.
- Polar coordinates reveal an interesting connection between a bar chart and a Coxcomb chart.
- I find this figure stupid, but some people like it.

```
## ggplot actually returns "invisibly"
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)

## Note this odd syntax
bar + coord_flip()
```




```
bar + coord_polar()
```

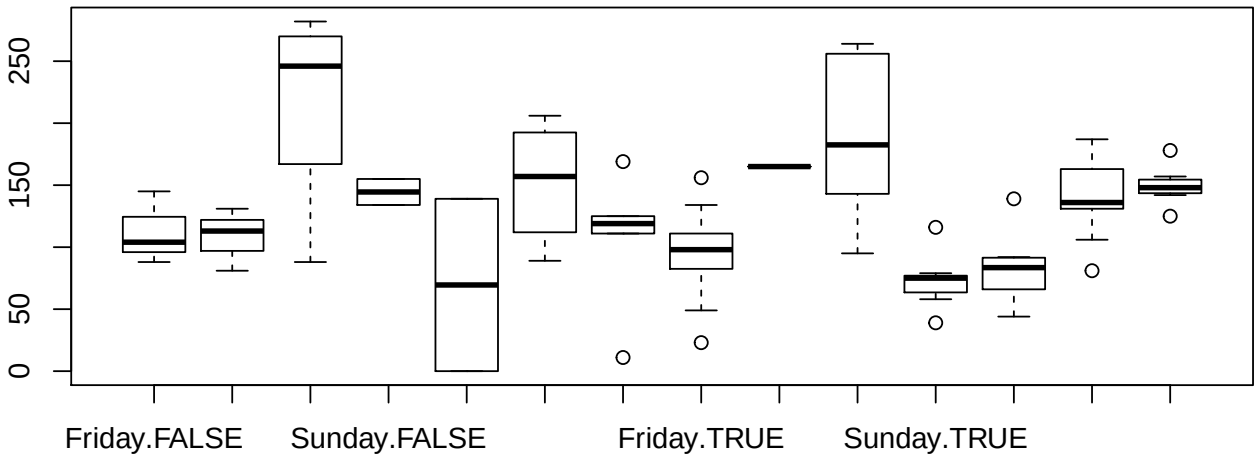


Summary

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

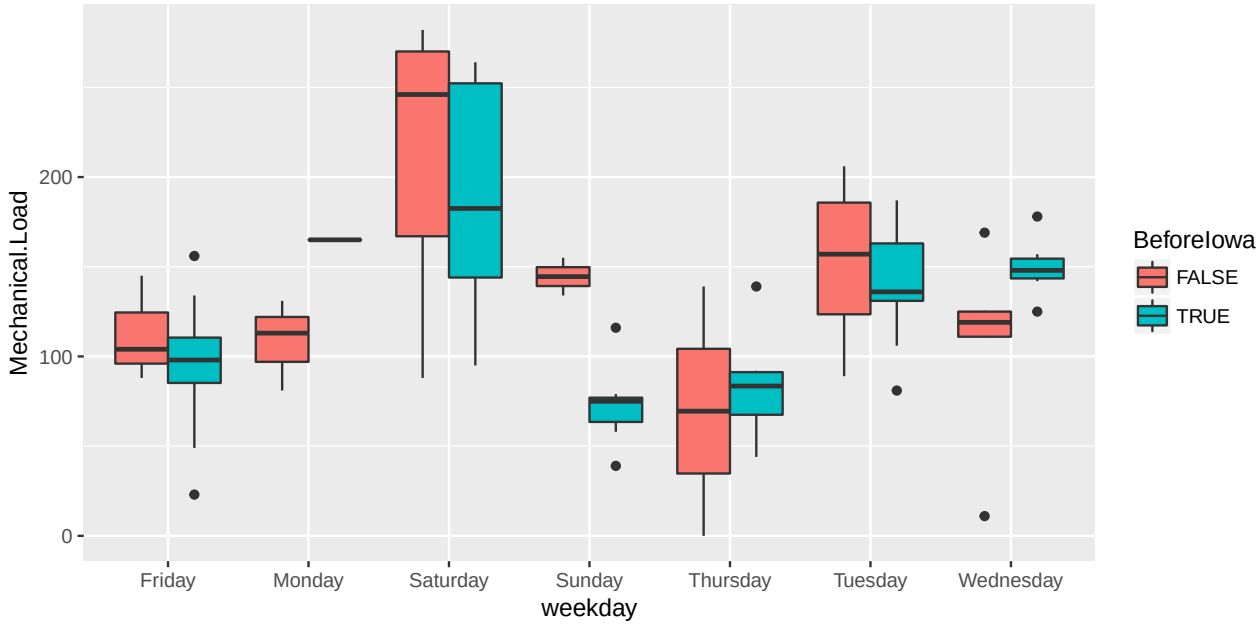
Another example

```
boxplot(Mechanical.Load~weekday+BeforeIowa, data=db1) # ugly, hard to read
```



ggplot

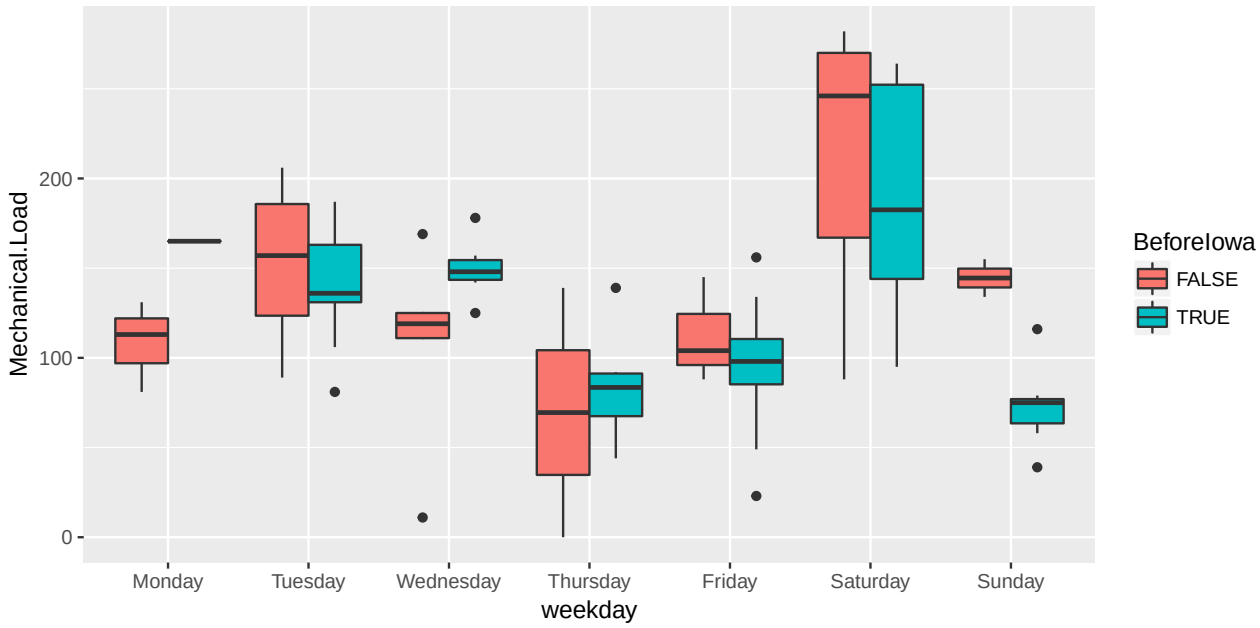
```
ggplot(db1, mapping=aes(x=weekday, y=Mechanical.Load)) +
  geom_boxplot(aes(fill=BeforeIowa))
```



Make weekdays in the correct order

I googled `R reorder factor levels`

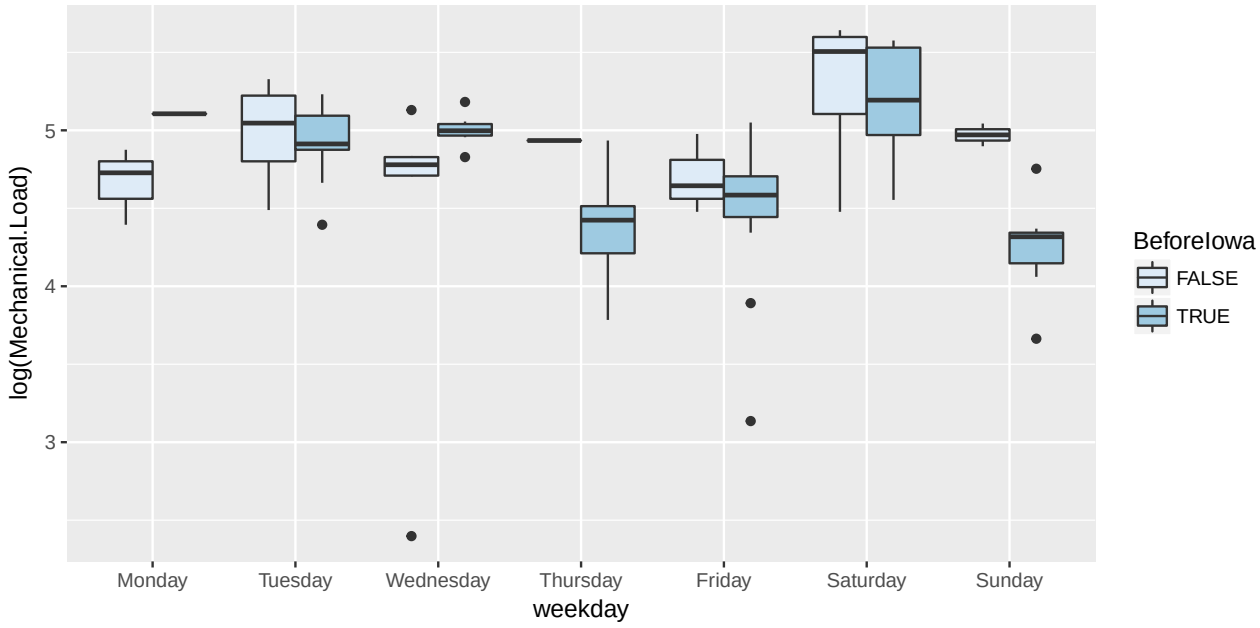
```
db1$weekday = factor(db1$weekday, c('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunda
ggplot(db1, mapping=aes(x=weekday, y=Mechanical.Load)) +
  geom_boxplot(aes(fill=BeforeIowa))
```



these colors are ugly (log scale?)

I googled `R ggplot2 fill colors`

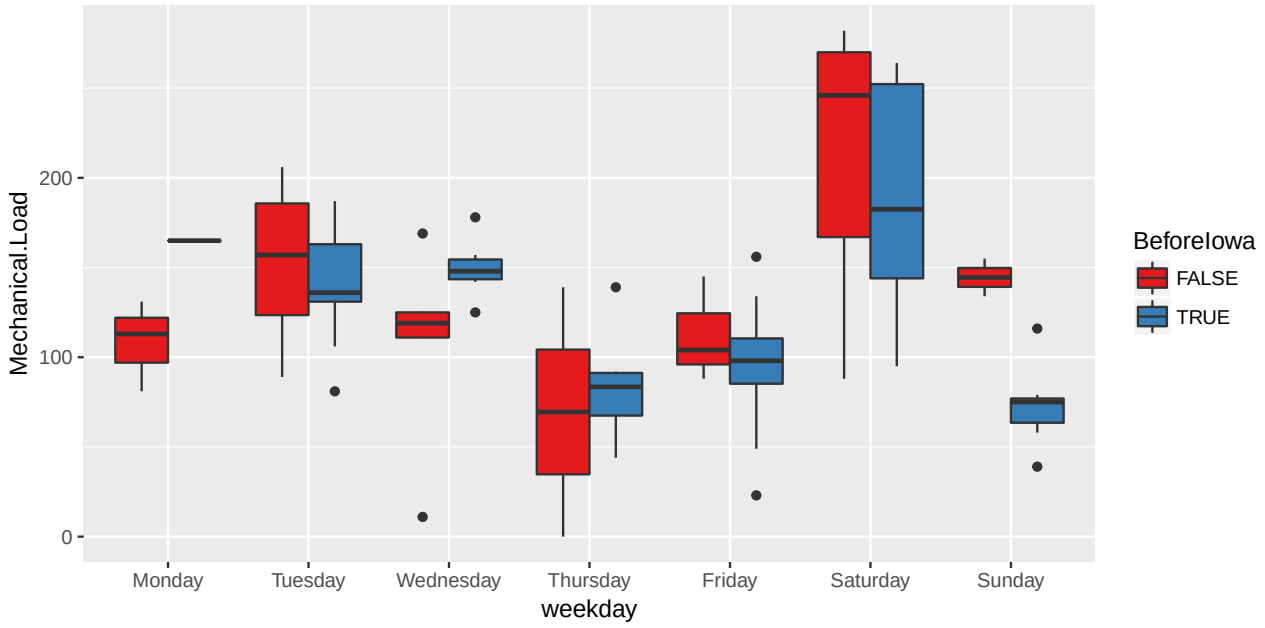
```
ggplot(db1, mapping=aes(x=weekday, y=log(Mechanical.Load))) + geom_boxplot(aes(fill=BeforeIowa)) +
  scale_fill_brewer()
```



still ugly, log is dumb

Try `?scale_fill_brewer` to see options, pretty opaque except for the examples

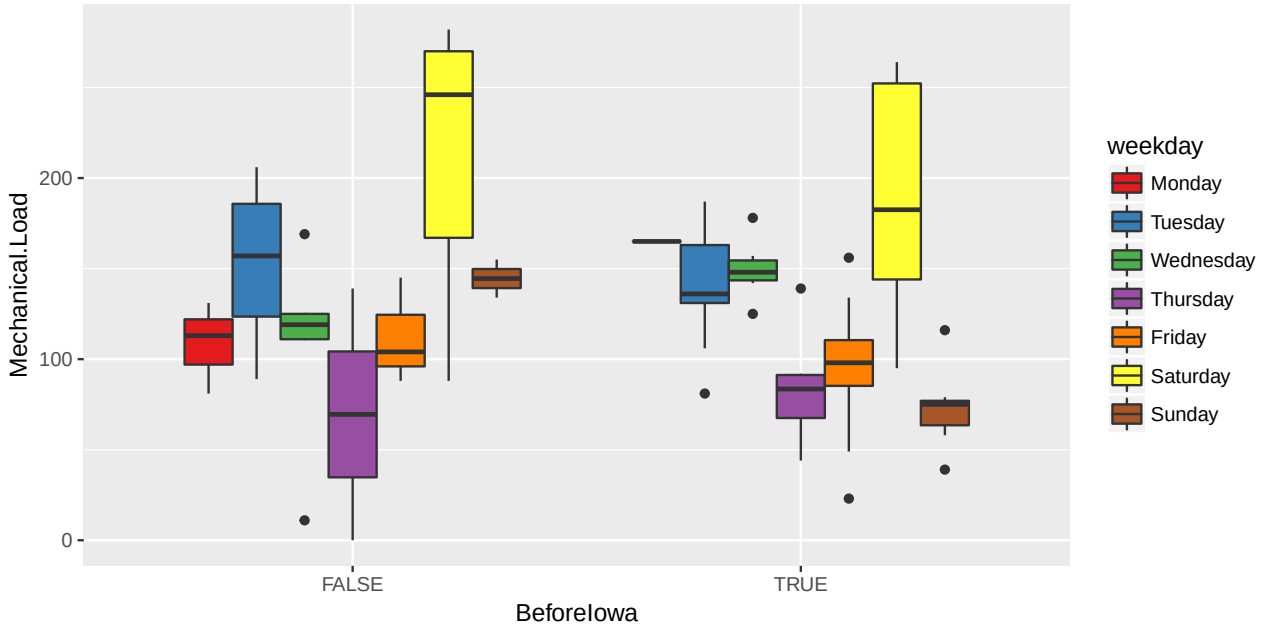
```
ggplot(db1, mapping=aes(x=weekday, y=Mechanical.Load)) + geom_boxplot(aes(fill=BeforeIowa)) +
  scale_fill_brewer(palette='Set1')
```



Not so bad

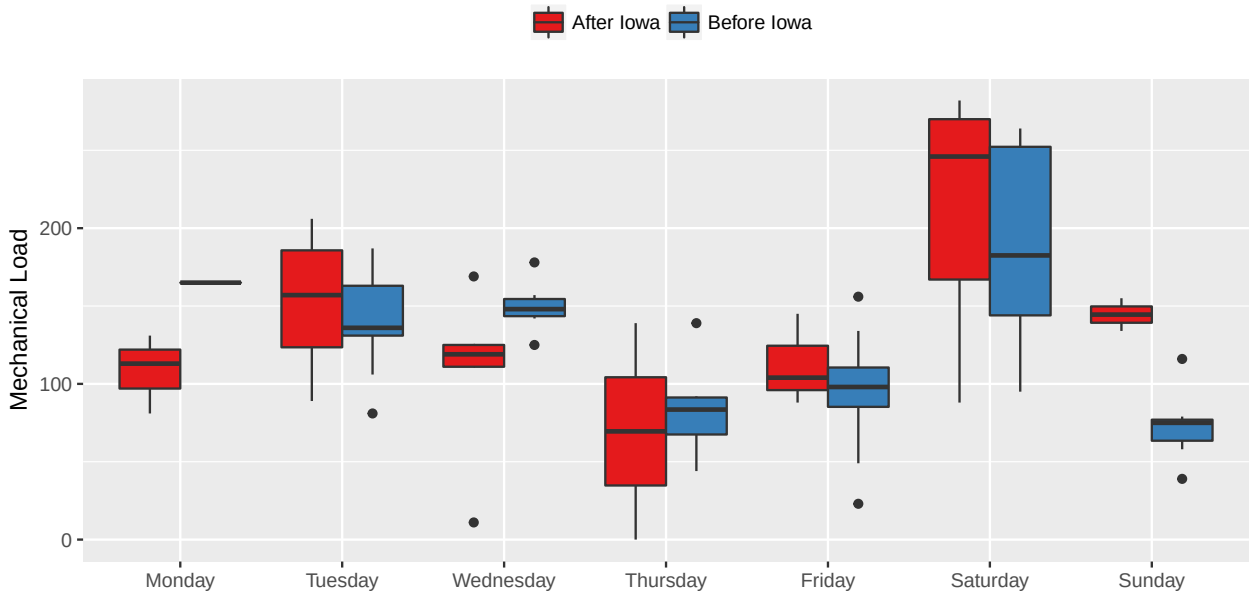
What if I made `weekday` the colors and `BeforeIowa` on the `x`-axis?

```
ggplot(db1, mapping=aes(x=BeforeIowa, y=Mechanical.Load)) + geom_boxplot(aes(fill=weekday)) +
  scale_fill_brewer(palette='Set1')
```



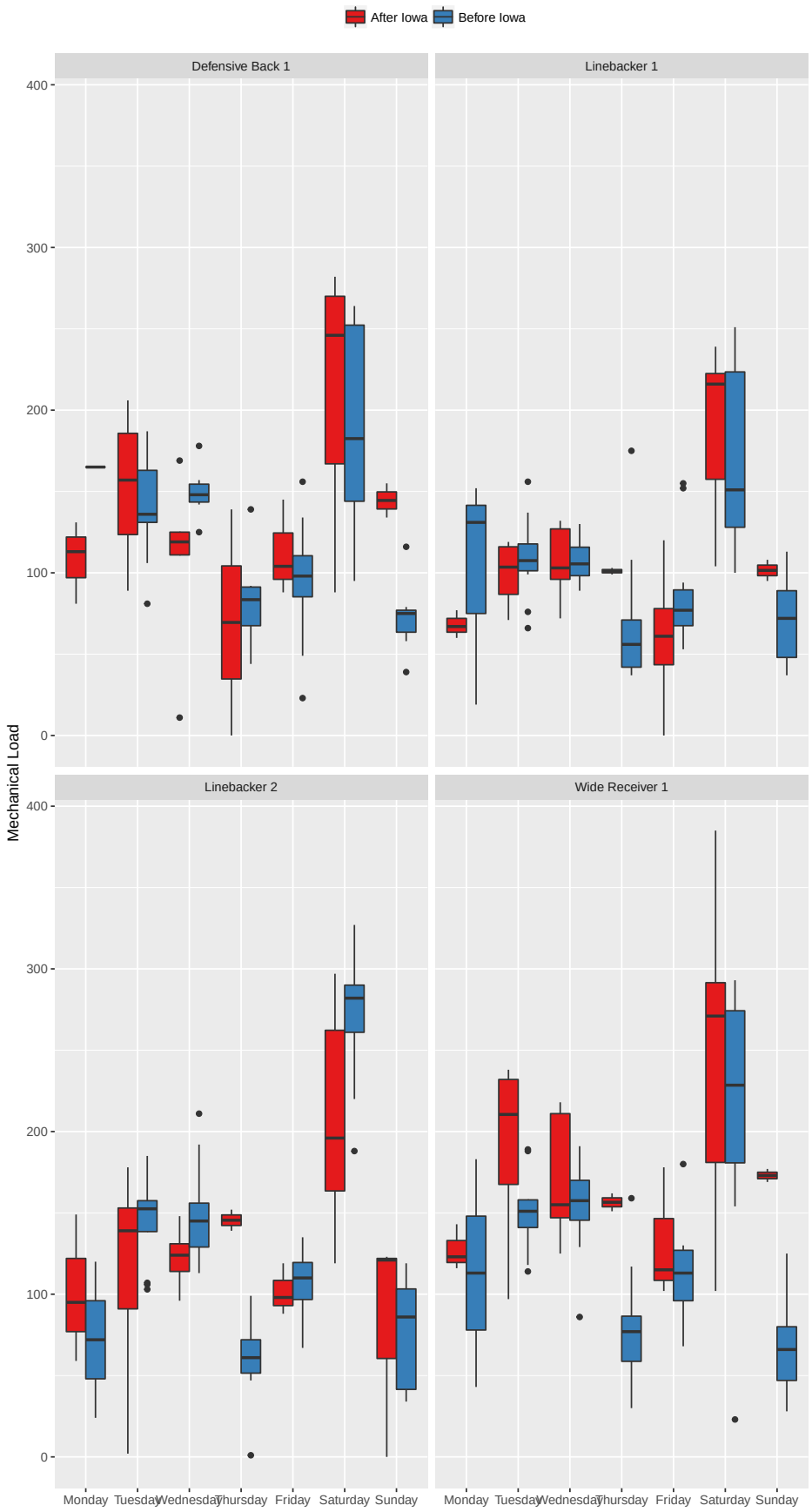
Stick with the previous, rename labels

```
db1$BeforeIowa = factor(db1$BeforeIowa, labels=c('After Iowa','Before Iowa'))
ggplot(db1, mapping=aes(x=weekday, y=Mechanical.Load)) + geom_boxplot(aes(fill=BeforeIowa)) +
  scale_fill_brewer(palette='Set1') +
  labs(y='Mechanical Load', x=element_blank(), fill=element_blank()) +
  theme(legend.position='top')
```



All the players in one dataset

```
ggplot(workout, mapping=aes(x=weekday, y=Mechanical.Load)) + geom_boxplot(aes(fill=BeforeIowa)) +
  scale_fill_brewer(palette='Set1') +
  labs(y='Mechanical Load', x=element_blank(), fill=element_blank()) +
  theme(legend.position='top') + facet_wrap(~Subject.Name, nrow=2)
```


After Iowa
Before Iowa
Defensive Back 1
Linebacker 1
Linebacker 2
Wide Receiver 1
Mechanical Load
0
100
200
300
400
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday