COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Data Science for Business Analytics

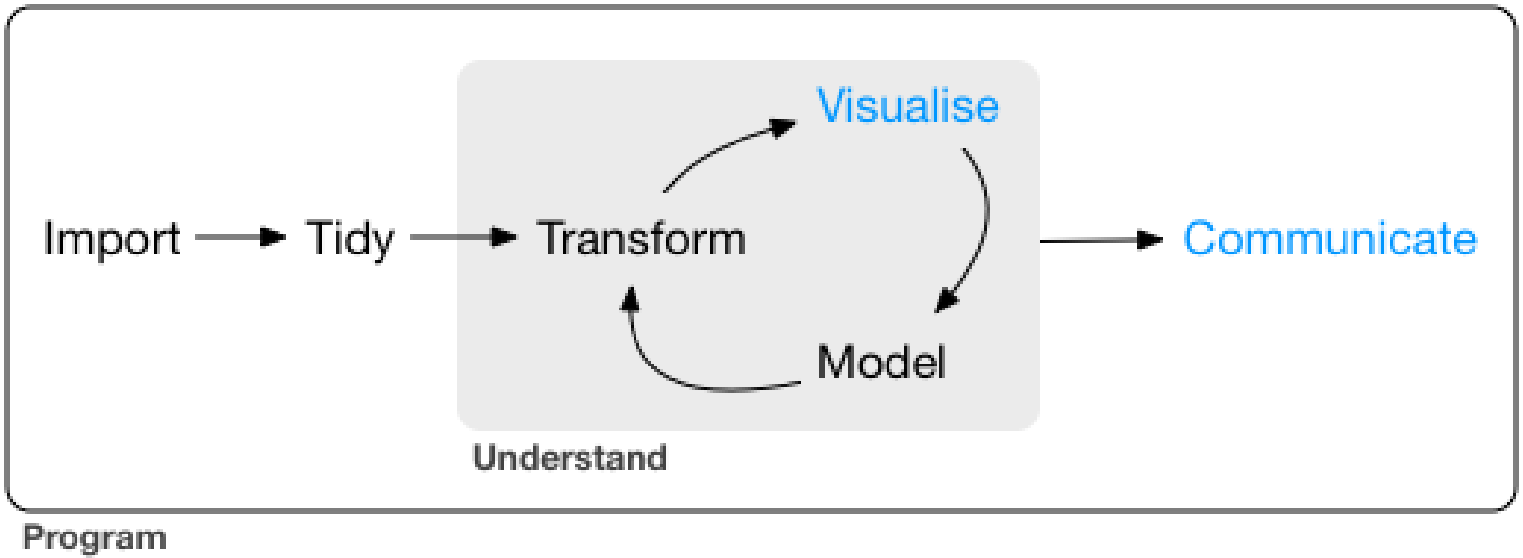
Lecture 3

Thibault Vatter

Department of Statistics, Columbia University

03/02/2020

Today



Most of the material (e.g., the picture above) is borrowed from

R for data science

Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

4 Coordinate systems

5 The layered grammar of graphics

6 Tibbles

Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

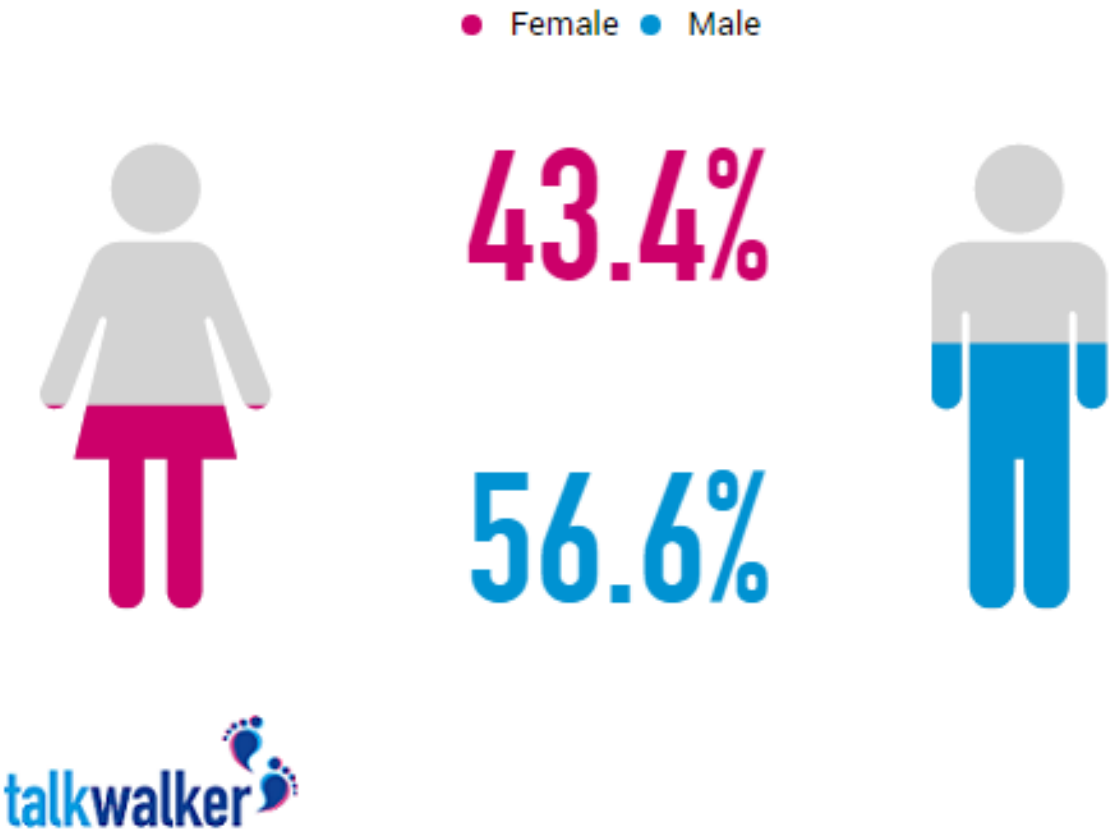
4 Coordinate systems

5 The layered grammar of graphics

6 Tibbles

Data content

- Makes no sense to use graphs for very small amounts of data.
- The human brain is capable of grasping a few values.



source: talkwalker.com

Data relevance

- Graphs are only as good as the data they display.
- No creativity can produce a good graph from poor data.

- Leinweber (author of *Nerds on Wall Street*):
  - The S&P500 could be "predicted"" at 75% by the butter production in Bangladesh.
  - ... Or 99% when adding cheese production in the USA, and the population of sheep.

Complexity

- Graphs shouldn't be more complex than the data they portray.
- Unnecessary complexity can be introduced by irrelevant
  - decoration
  - color
  - 3d effects
- . . . Collectively known as "chartjunk"!

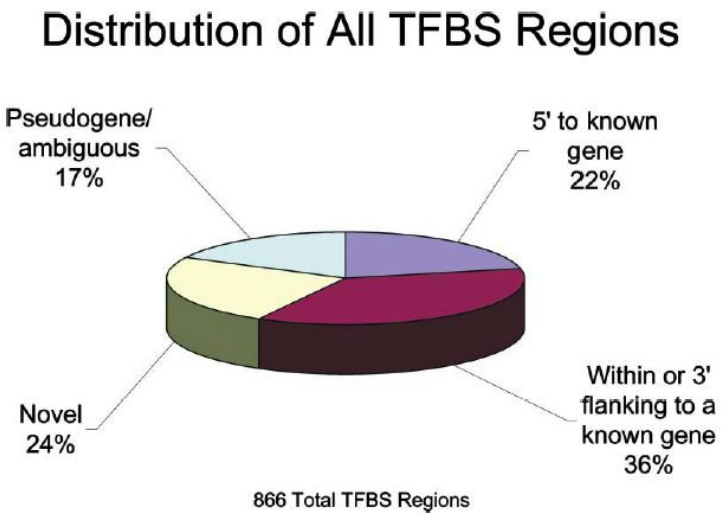


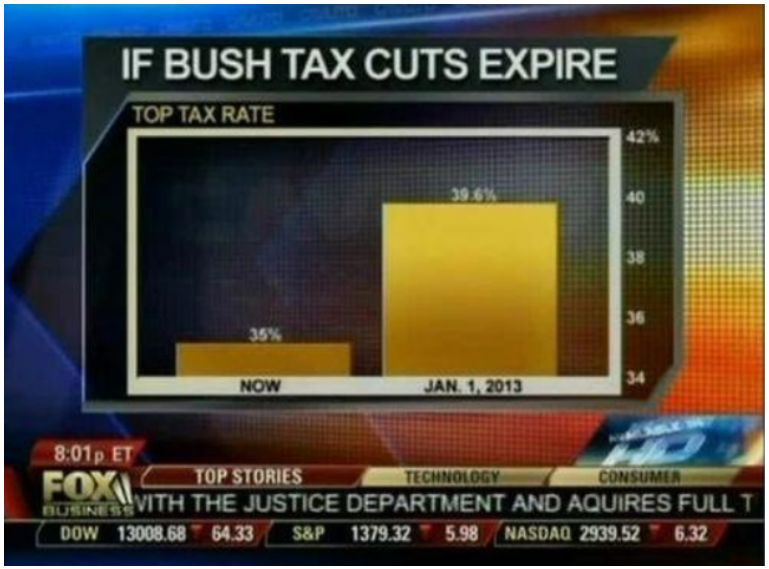
Figure 1. Classification of TFBS Regions

TFBS regions for Sp1, cMyc, and p53 were classified based upon proximity to annotations (RefSeq, Sanger hand-curated annotations, GenBank full-length mRNAs, and Ensembl predicted genes). The proximity was calculated from the center of each TFBS region. TFBS regions were classified as follows: within 5 kb of the 5′ most exon of a gene, within 5 kb of the 3′ terminal exon, or within a gene, novel or outside of any annotation, and pseudogene/ambiguous (TFBS overlapping or flanking pseudogene annotations, limited to chromosome 22, or TFBS regions falling into more than one of the above categories).

source: Cawley S, et al. (2004), Cell 116:499-509, Figure 1

Distorsion

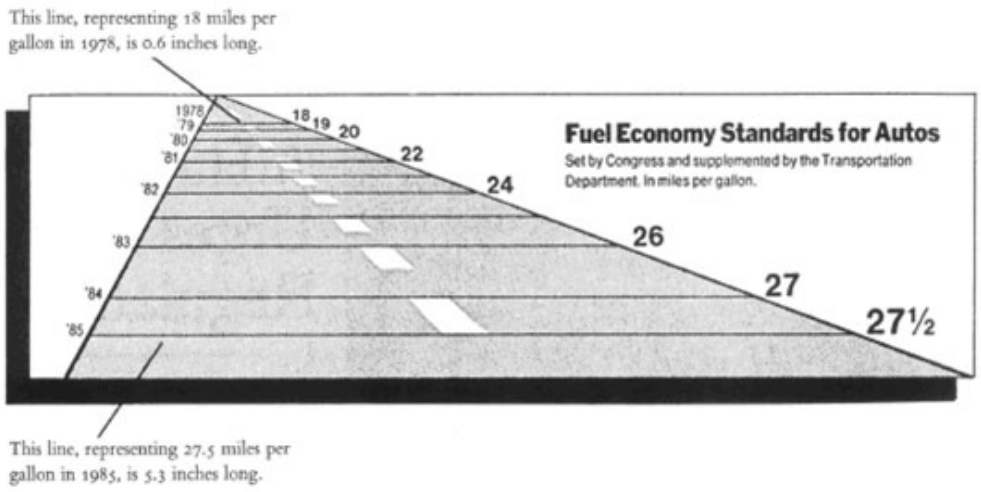
- Graphs shouldn't be distorted pictures of the portrayed values:
  - Can be either deliberate or accidental.
  - Useful to know how to produce truth bending graphs.
  - Misleading often used as a synonym of distorted.



source: statisticshowto.com/misleading-graphs/

More on distorsion

- Common sources of distortion:
  - 3 dimensional "effects".
  - linear scaling when using area or volume to represent values.
- The "lie factor":
  - Measure of the amount of distortion in a graph.
    - lie factor $= \frac{\text{size of effect shown in graphic}}{\text{size of effect shown in data}}$
    - Don't take this too seriously.
    - Defined by Ed Tufte of Yale.
  - If lie factor is > 1, the graph is exaggerating the effect.



Drawing good graphs

- The three main rules:
  - If the "story" is simple, keep it simple.
  - If the "story" is complex, make it look simple.
  - Tell the truth – do not distort the data.
- Specifically:
  - There should be a high data to chart ratio.
  - Use the appropriate graph for the appropriate purpose.
    - Most graphs presented in Excel are POOR CHOICES!
    - In particular, never use a pie chart!
  - Make sure that the graph is complete:
    - All axes must be labeled.
    - The units should be indicated.
    - There should be a title.
    - A legend can provide needed additional information (e.g., for colors or line types).

A grammar of graphics

"A grammar of graphics is a tool that enables us to concisely describe the components of a graphic. Such a grammar allows us to move beyond named graphics (e.g., the"scatterplot") and gain insight into the deep structure that underlies statistical graphics." — *Hadley Wickham*

- `ggplot2` is an R implementation of the concept:
  - A coherent system for describing and creating graphs.
  - Based on The Grammar of Graphics.
  - Learn one system and apply it in many places.
  - The equivalent of `dplyr` for graphs.
- To learn more, read The Layered Grammar of Graphics.
- Implementations exist in other languages (e.g., Python)

The mpg data frame

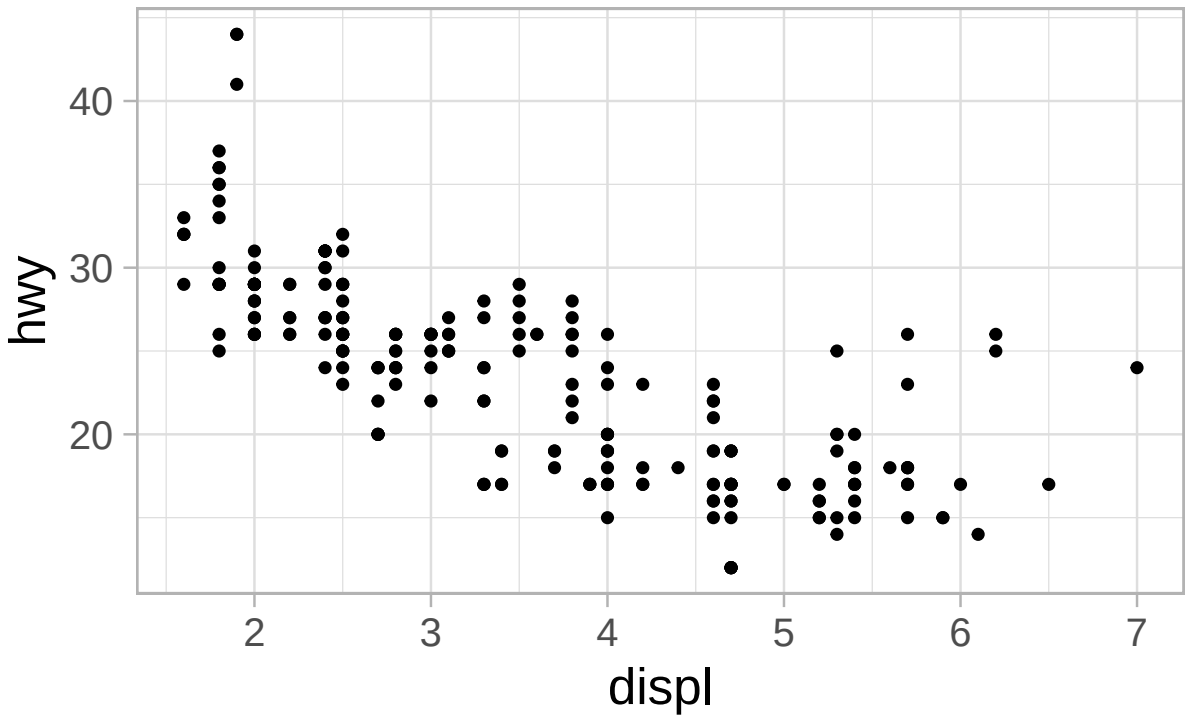
- Data from the US EPA on 38 models of car:

```
mpg %>% print(n = 5)
#> # A tibble: 234 x 11
#>   manufacturer model displ  year   cyl trans drv     cty   hwy fl
#>   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr>
#> 1 audi         a4      1.8  1999     4 auto~ f        18    29 p
#> 2 audi         a4      1.8  1999     4 manu~ f        21    29 p
#> 3 audi         a4      2    2008     4 manu~ f        20    31 p
#> 4 audi         a4      2    2008     4 auto~ f        21    30 p
#> 5 audi         a4      2.8  1999     6 auto~ f        16    26 p
#> # ... with 229 more rows, and 1 more variable: class <chr>
```

- Among the variables in mpg are:
  - displ, a car's engine size, in litres.
  - hwy, a car's fuel efficiency on the highway (in miles per gallon).
- A few questions
  - Do cars with big engines use more fuel ?
  - What does the relationship between engine size and fuel efficiency look like? Positive? Negative? Linear? Nonlinear?

Creating a plot


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



A graphing template


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

Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

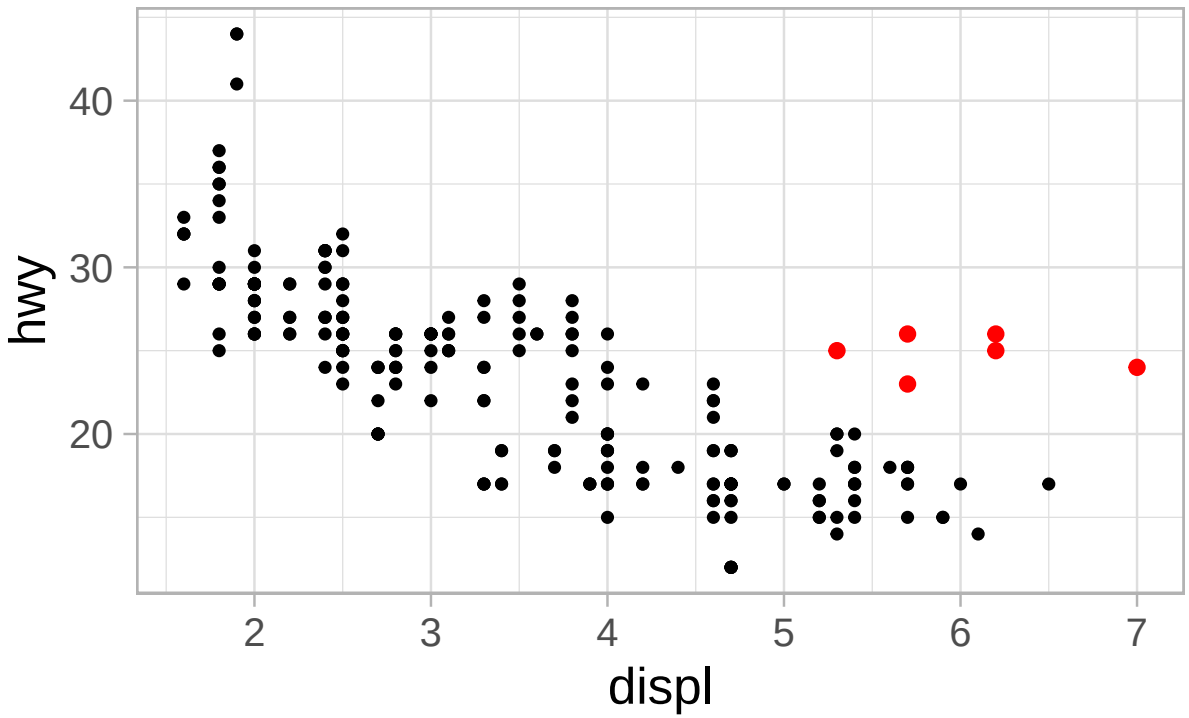
4 Coordinate systems

5 The layered grammar of graphics

6 Tibbles

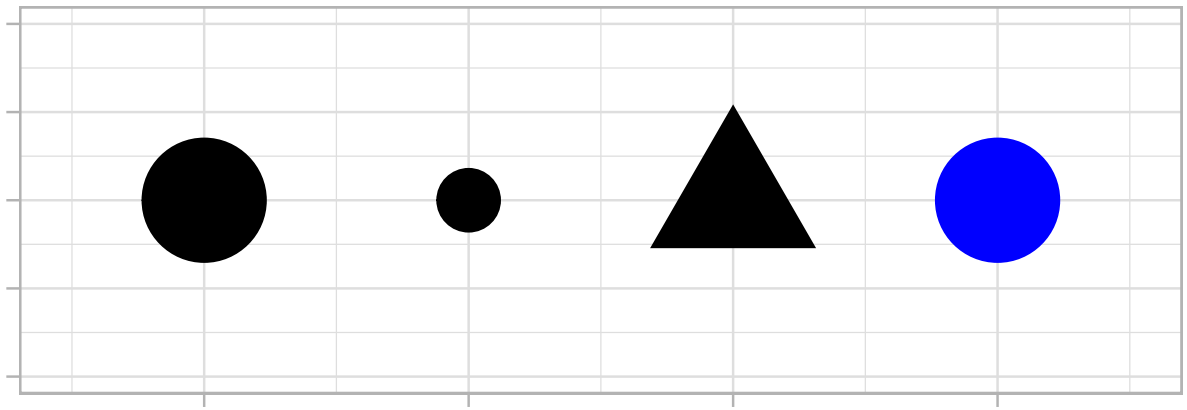
Aesthetic mappings

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



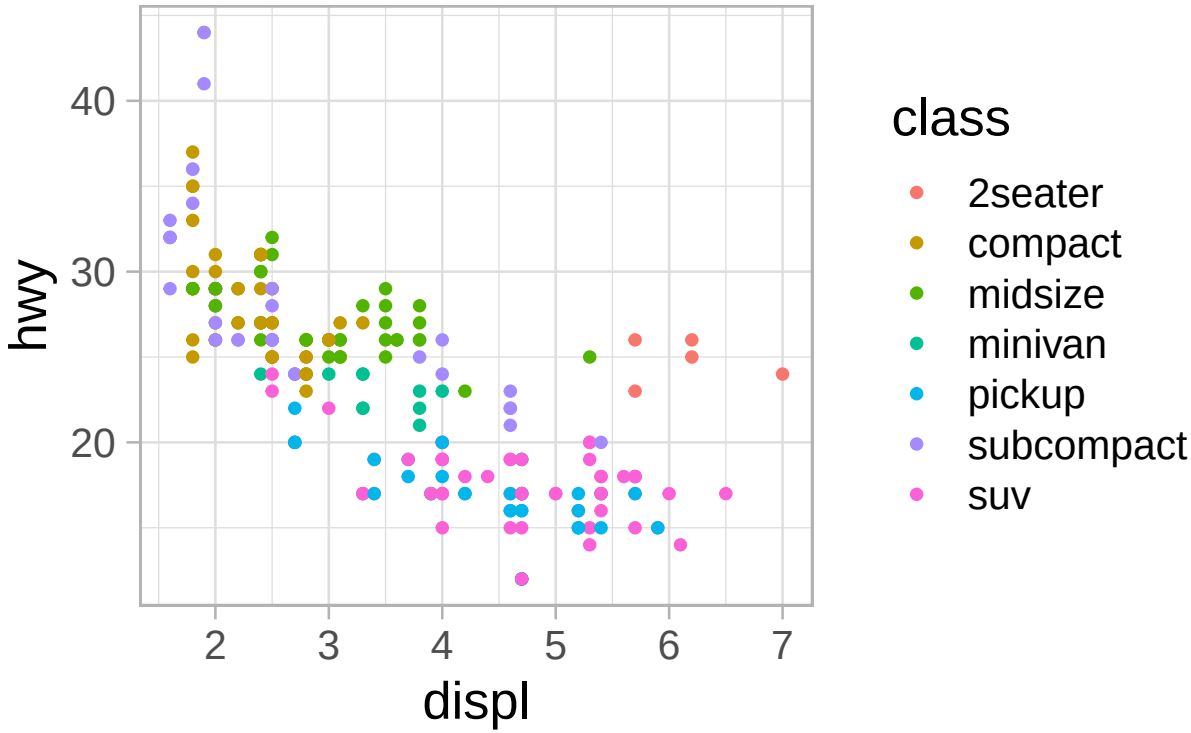
Aesthetic

- How to add a third variable to a two dimensional scatterplot?
- By mapping it to an **aesthetic**:
  - A visual property of the objects in your plot.
  - Include the size, the shape, or the color of the points.
- We use the words
  - **"value"** to describe data,
  - and **"level"** to describe aesthetic properties.



Adding classes to your plot

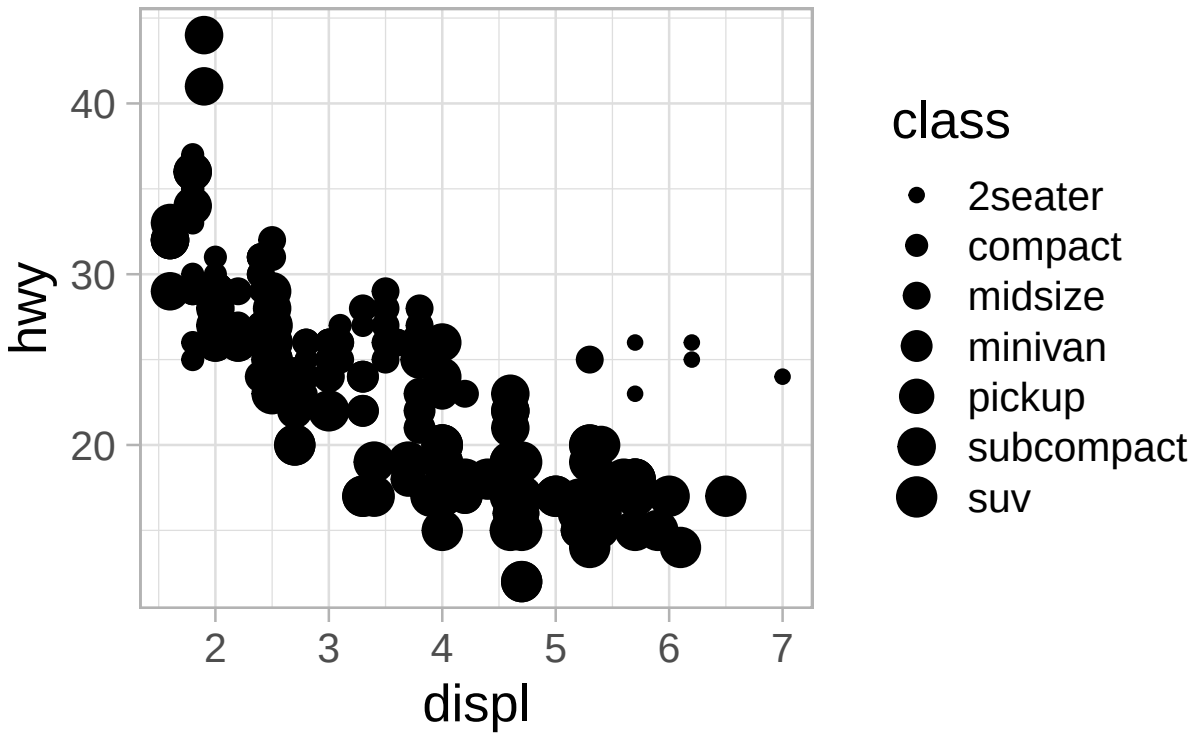

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



- If you prefer British English, use `colour` instead of `color`.

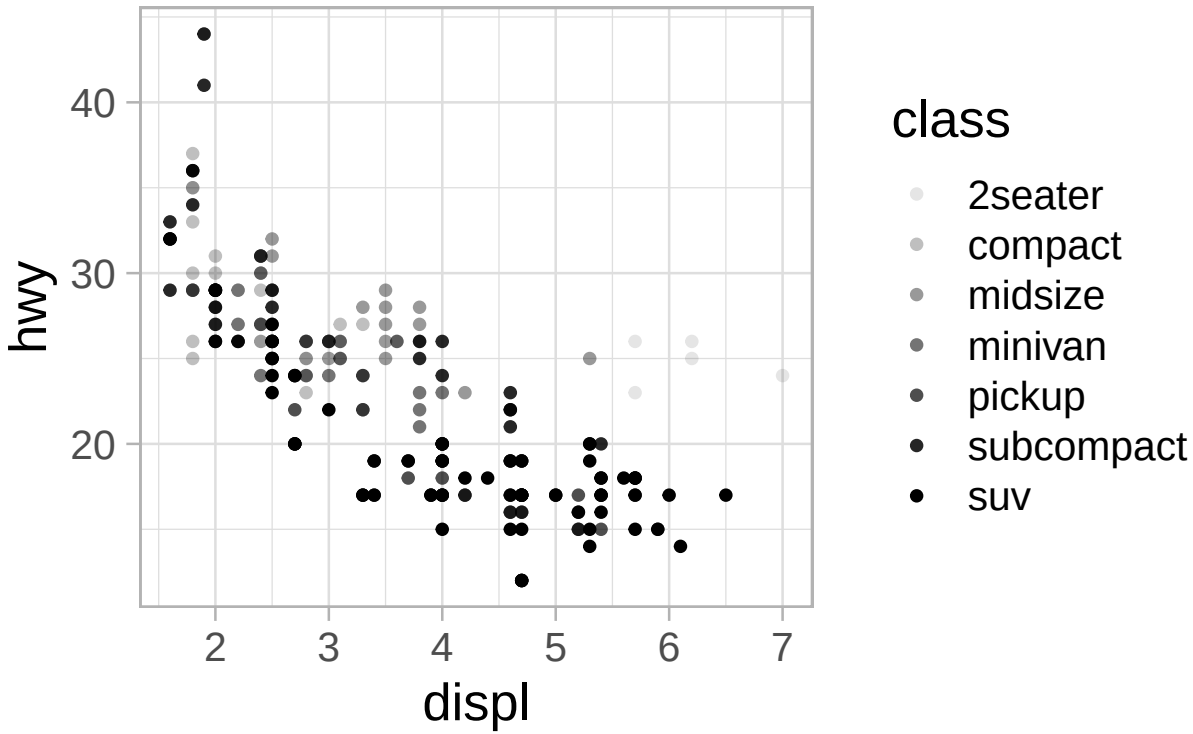
The size aesthetic


```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```



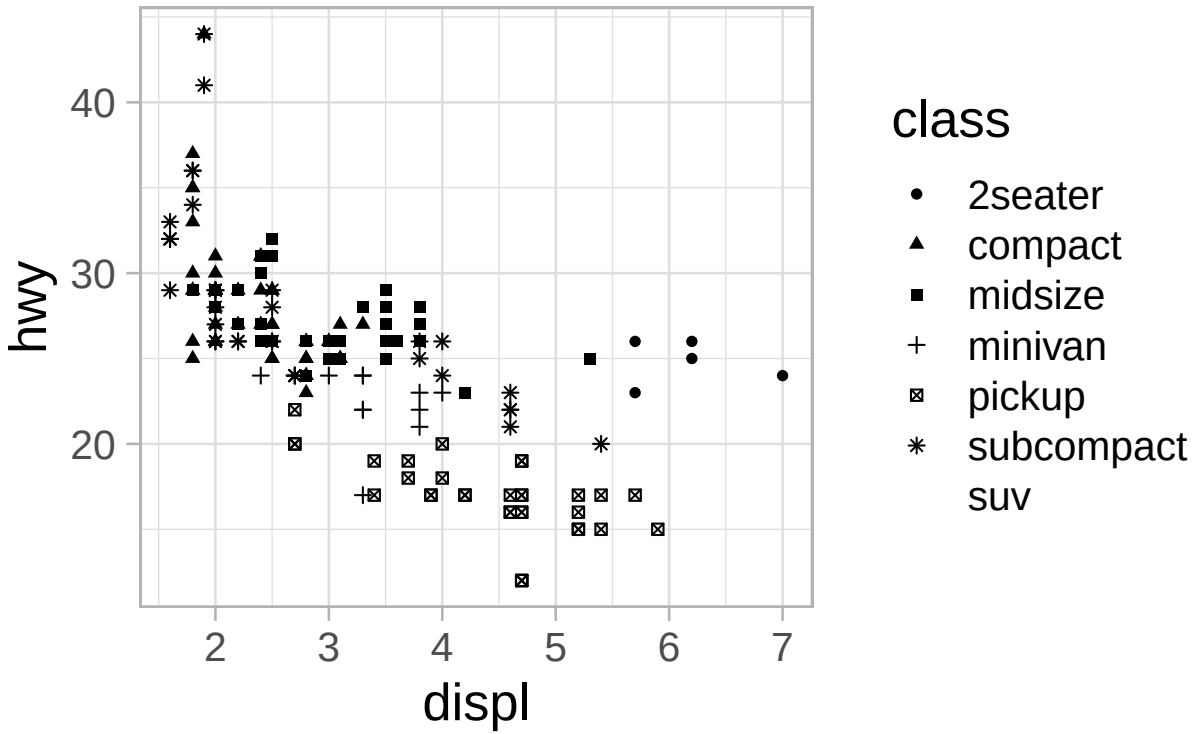
The alpha aesthetic


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



The shape aesthetic

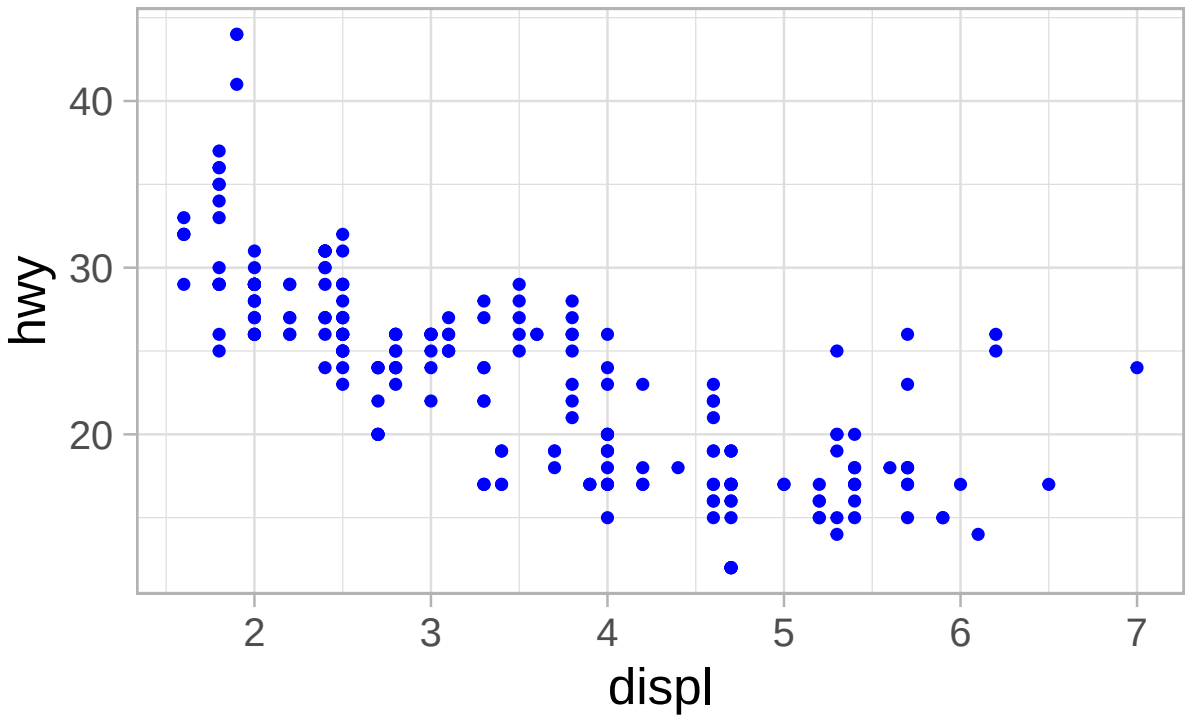

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```



Set the aesthetics manually

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

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



Set the aesthetics manually cont'd

- Need values that make sense for that aesthetic:
  - The name of a color as a character string.
  - The size of a point in mm.
  - The shape of a point as a number.

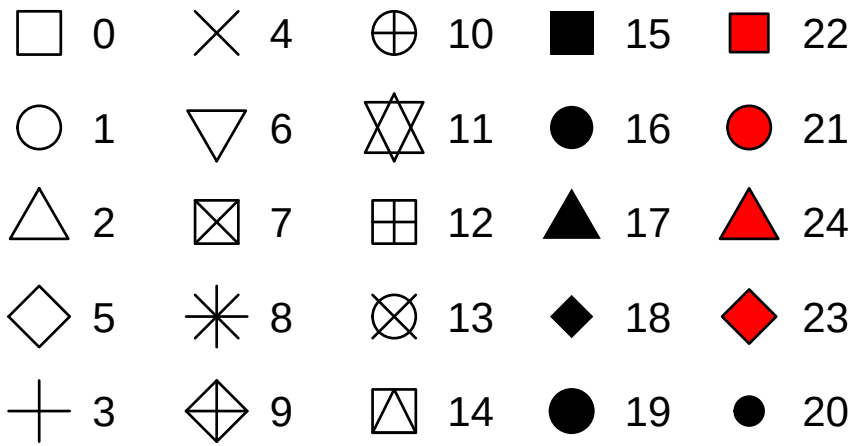
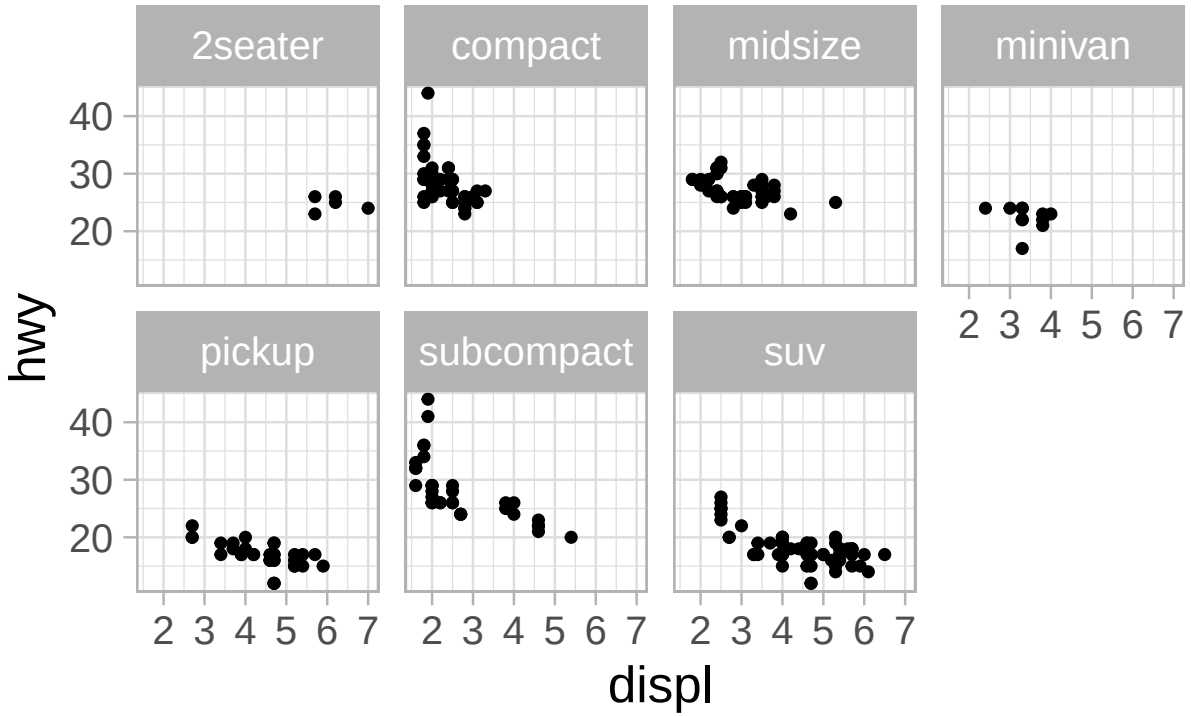


Figure 1: The hollow shapes (0–14) have a border determined by 'color'; the solid shapes (15–18) are filled with 'color'; the filled shapes (21–24) have a border of 'color' and are filled with 'fill'.

Facets wrap

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

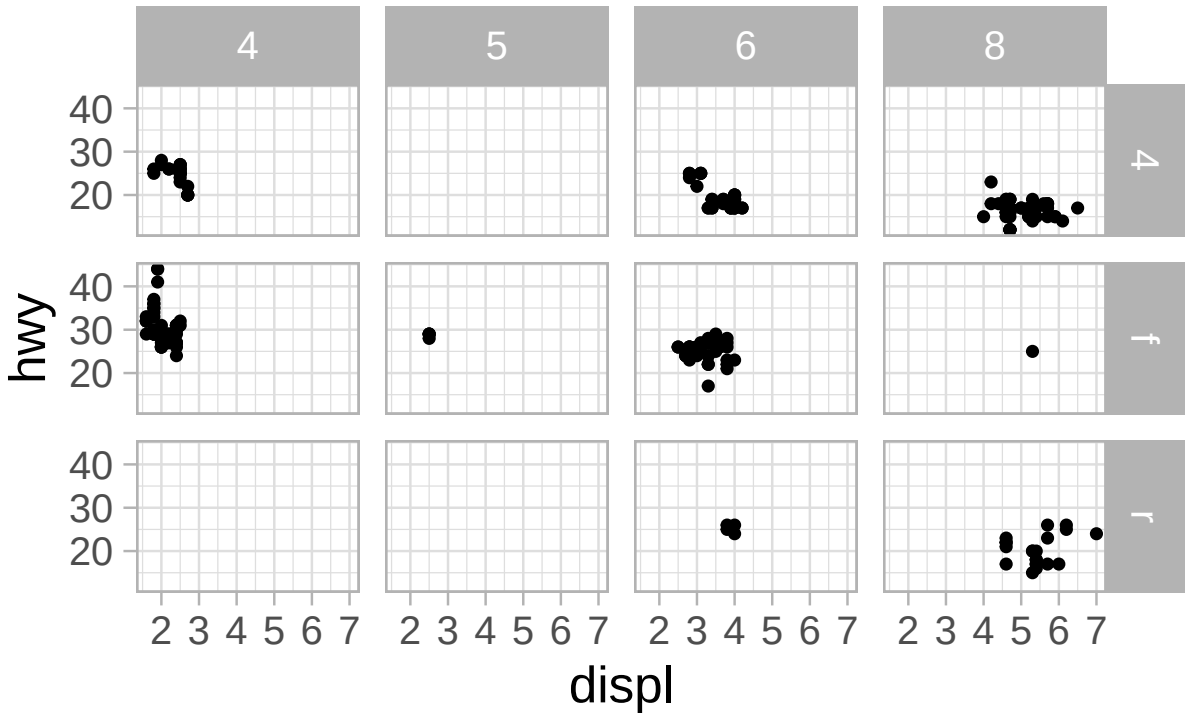
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```



Facets grid

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

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



Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

4 Coordinate systems

5 The layered grammar of graphics

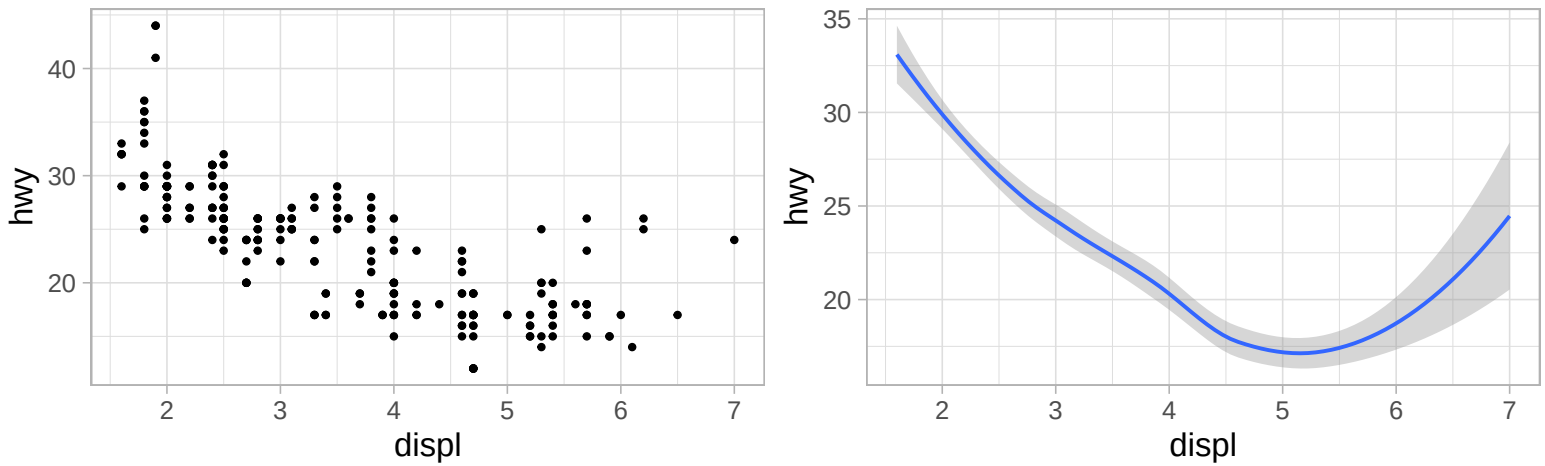
6 Tibbles

How are these two plots similar?

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

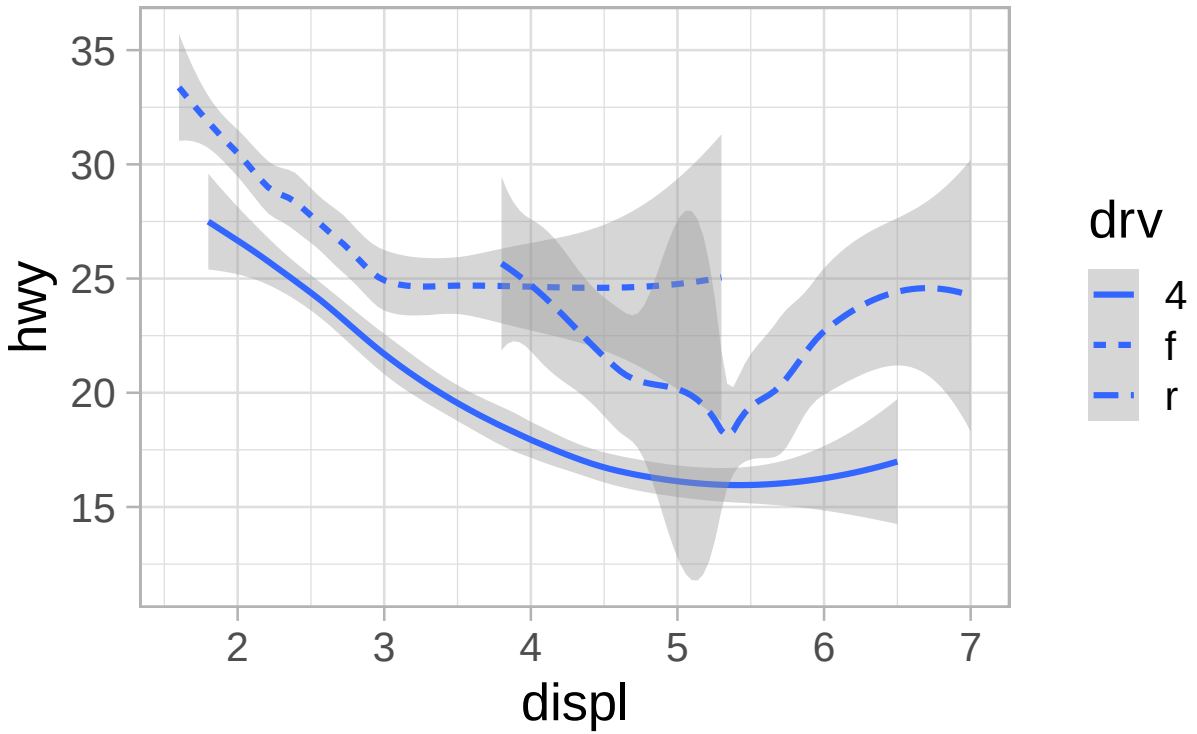
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))

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



The linetype aesthetic


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



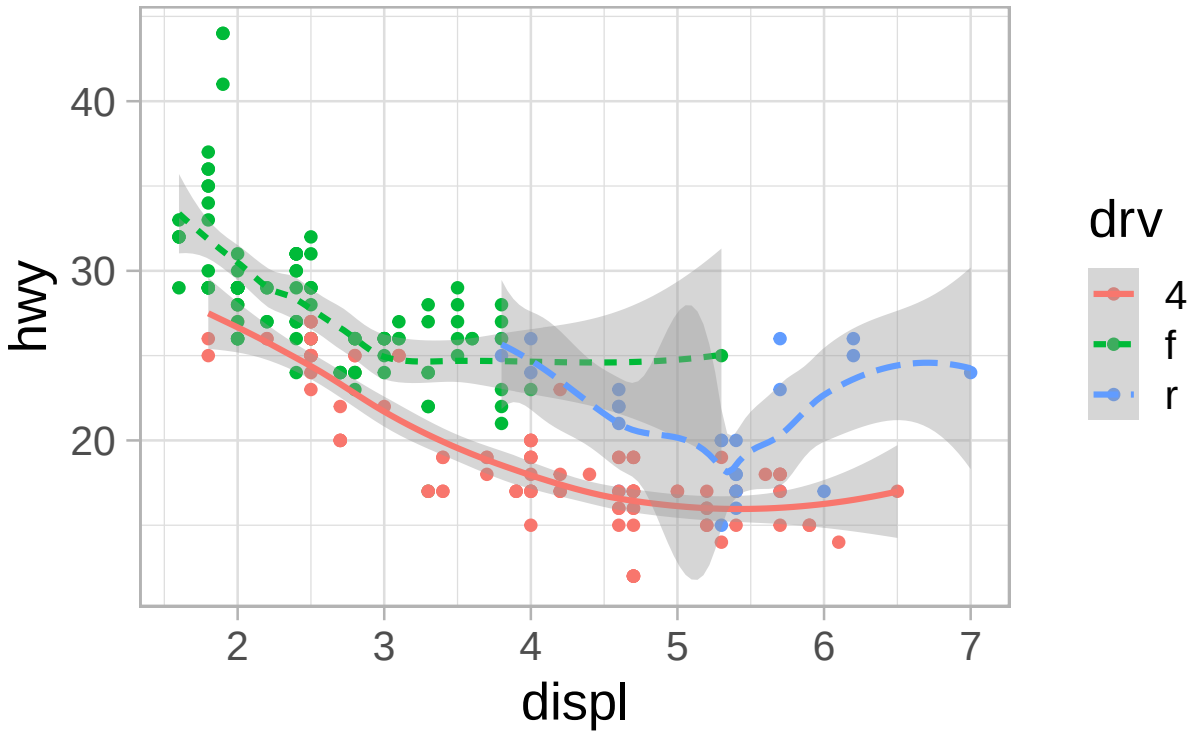
Geometric objects

- A **geom**:
  - The object that a plot uses to represent data.
  - Plots often describeds by the geom type:
    - Bar charts use bar geoms.
    - Line charts use line geoms.
    - Boxplots use boxplot geoms.
  - An exception:
    - Scatterplots use the point geom.
- Every **geom** function takes a `mapping` argument.
- But **not every aesthetic works with every geom**:
  - **shape** exists for `geom_point` but not for `geom_line`,
  - and conversely for **linetype**.

Combining two geoms

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

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

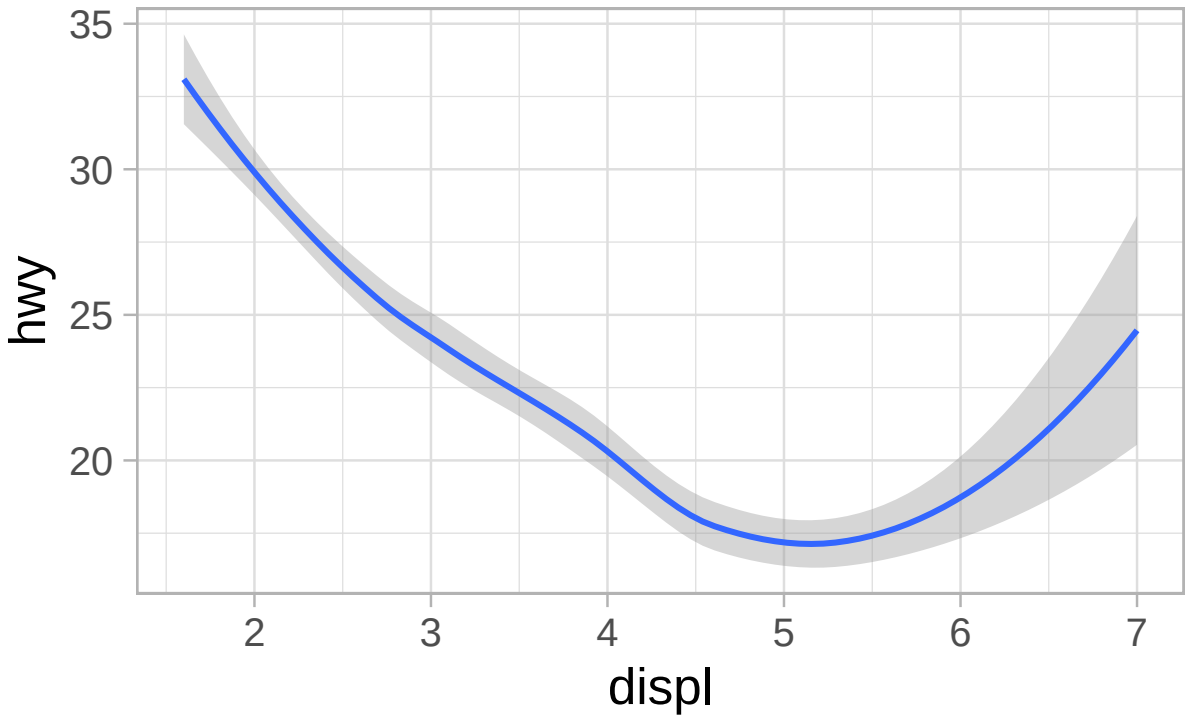


More on geoms

- `ggplot2` provides over 30 geoms.
- extension packages provide even more.
- Use RStudio's data visualization cheatsheet.
- To learn more about any single geom, use help: `?geom_smooth`.

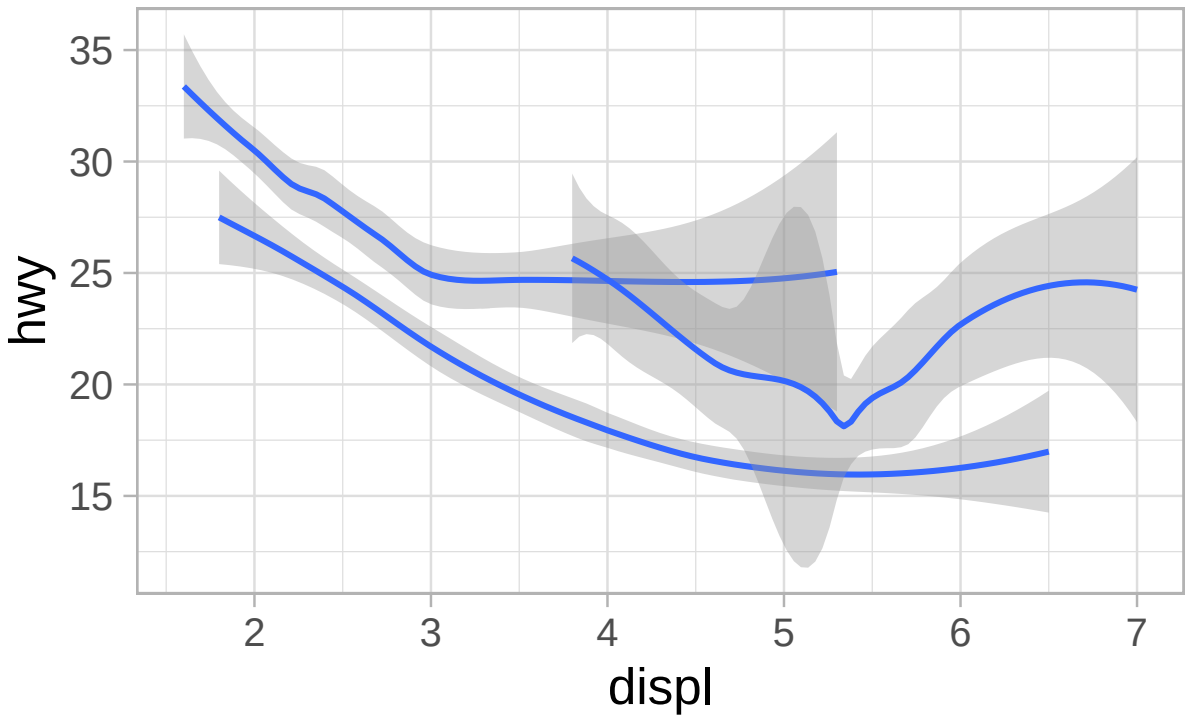
Geoms and legends


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



Geoms and legends

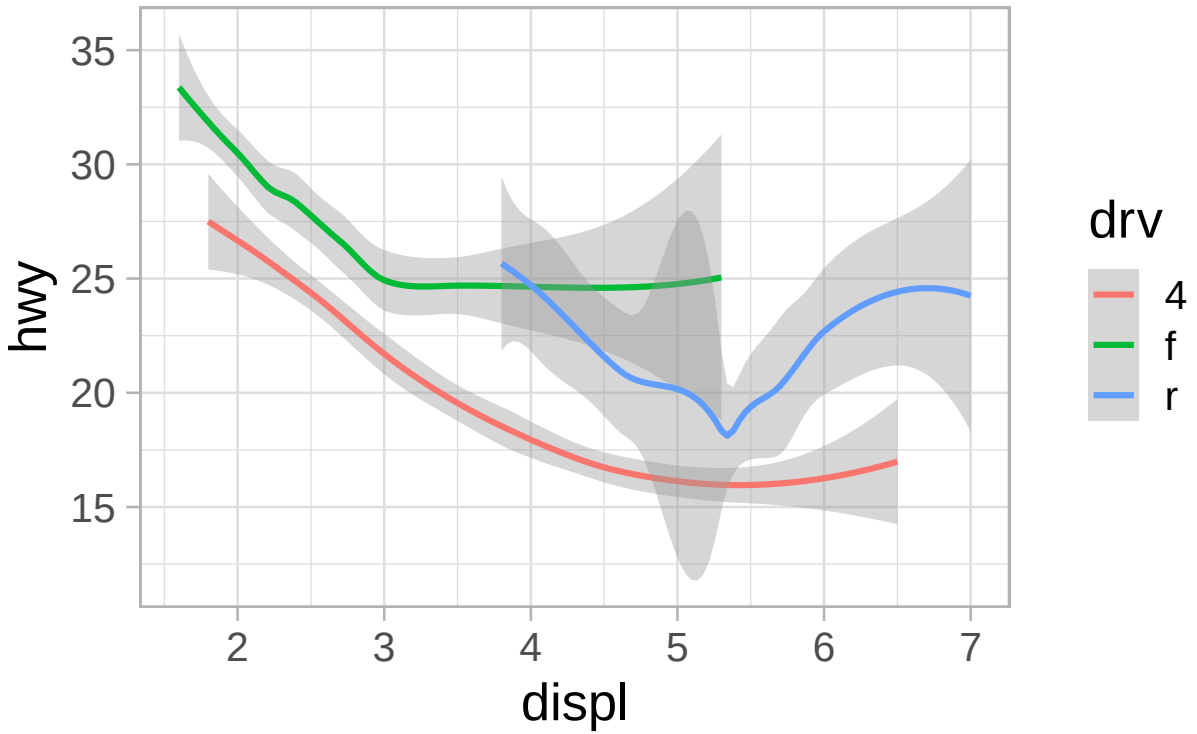

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, group = drv))
```



Geoms and legends

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

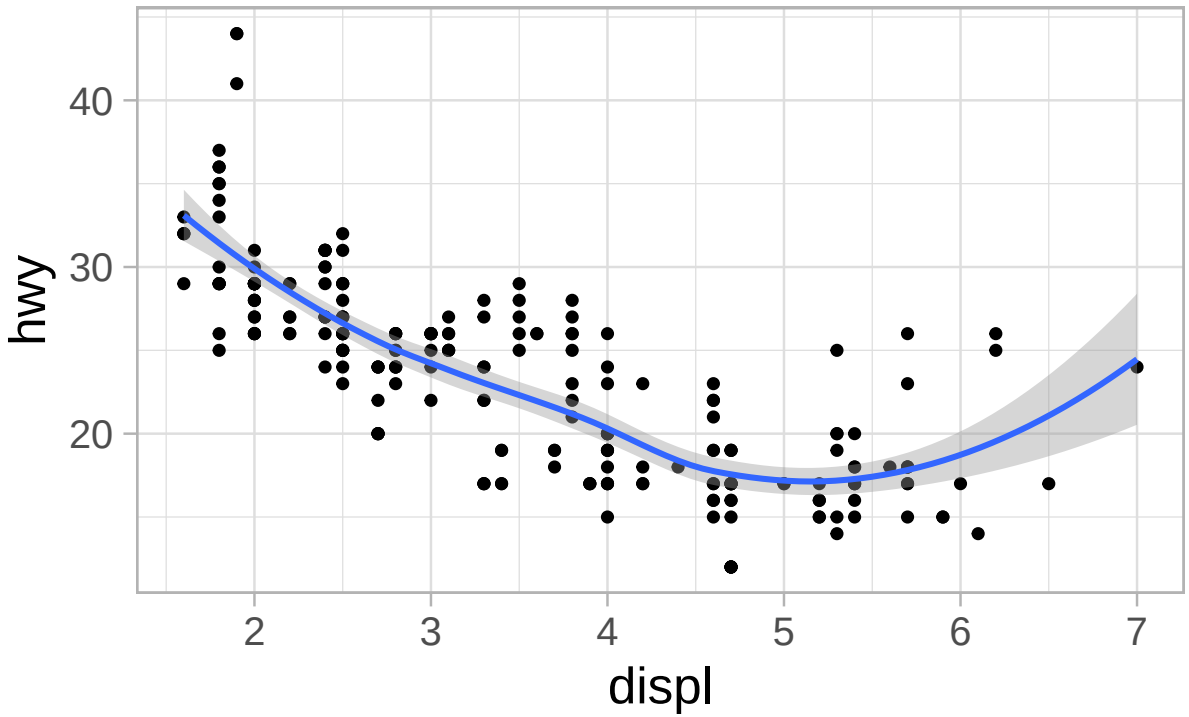
```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, color = drv))
```



Multiple geoms in the same plot

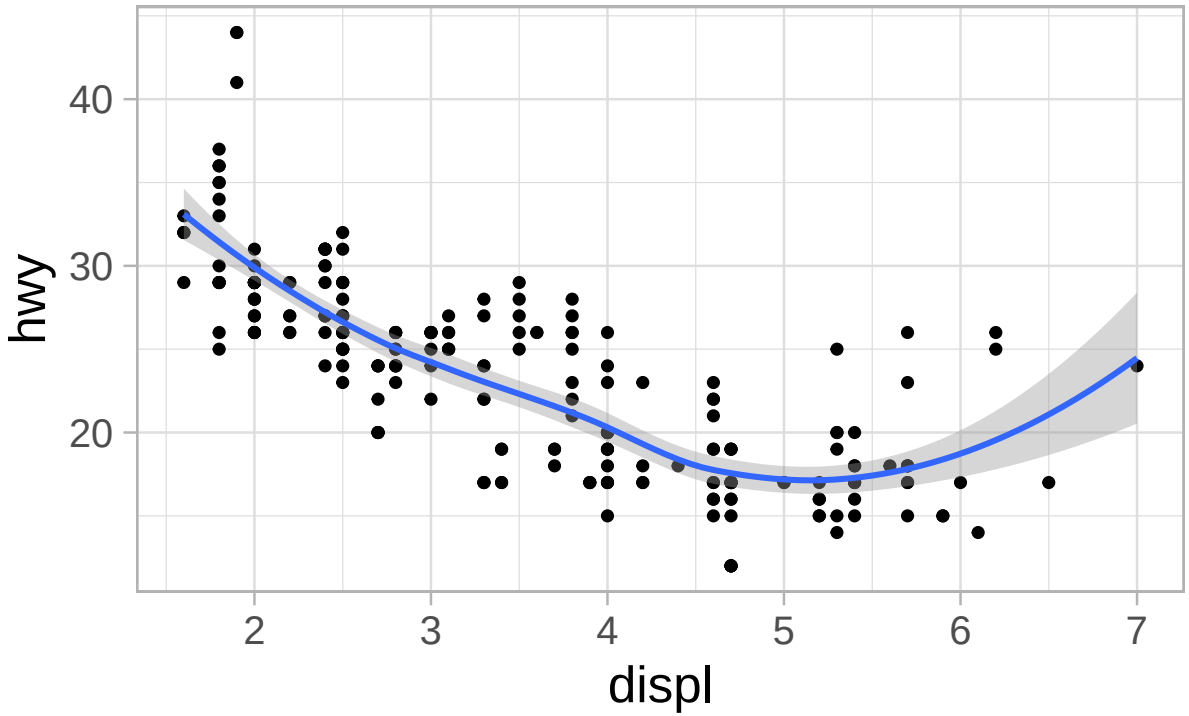
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

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



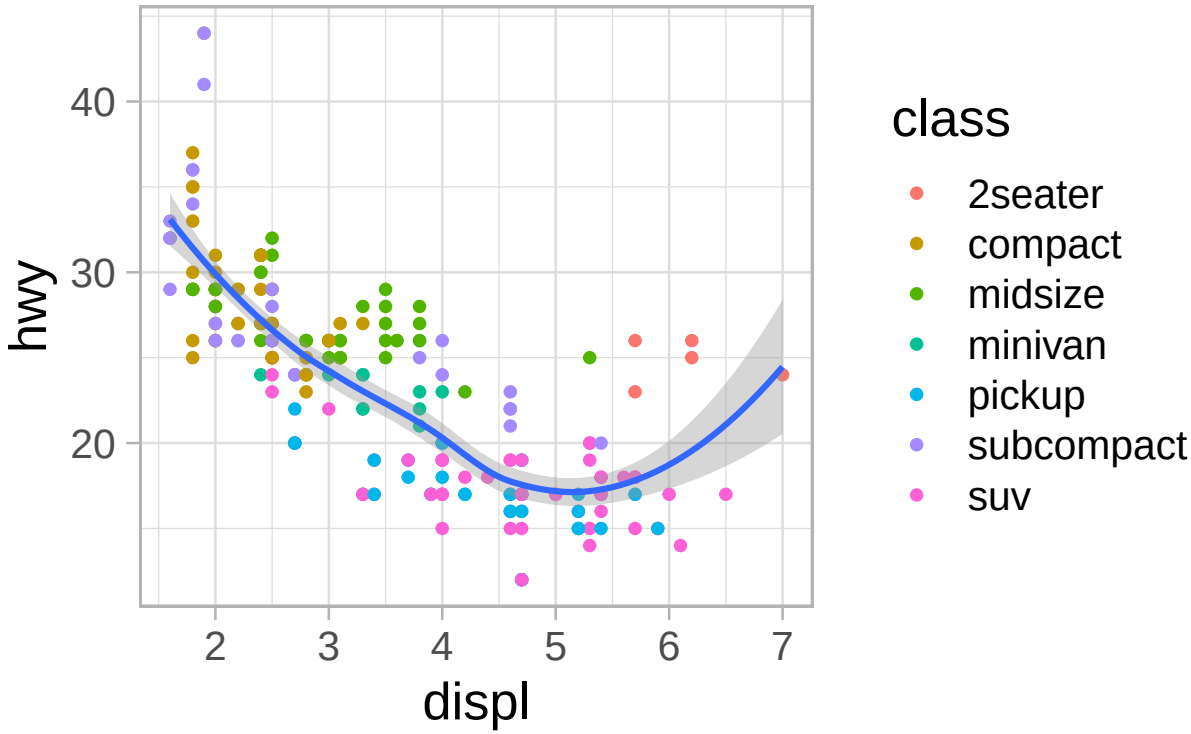
A better way


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



Local vs global mappings

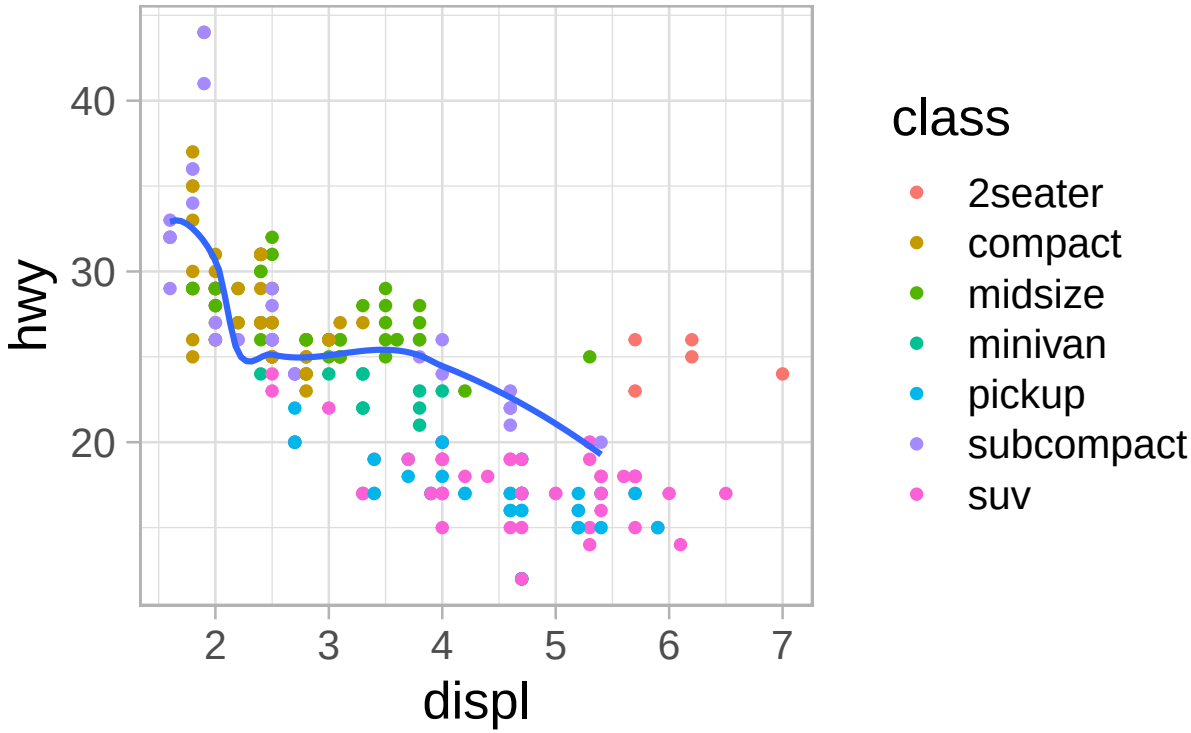

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth()
```



Layer dependent `data`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

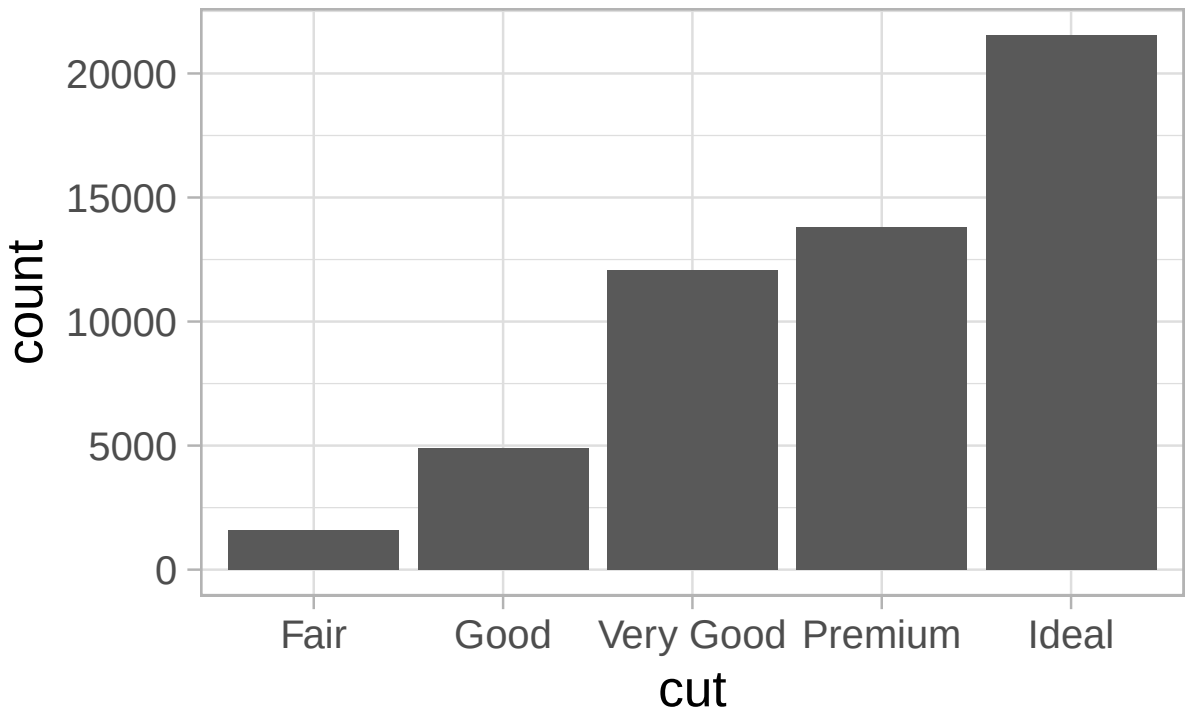
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(data = filter(mpg, class == "subcompact"), se = FALSE)
```



Bar charts

- The `diamonds` dataset:
  - About 54,000 diamonds.
  - Information about `price`, `carat`, `color`, `clarity`, and `cut` for each.

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



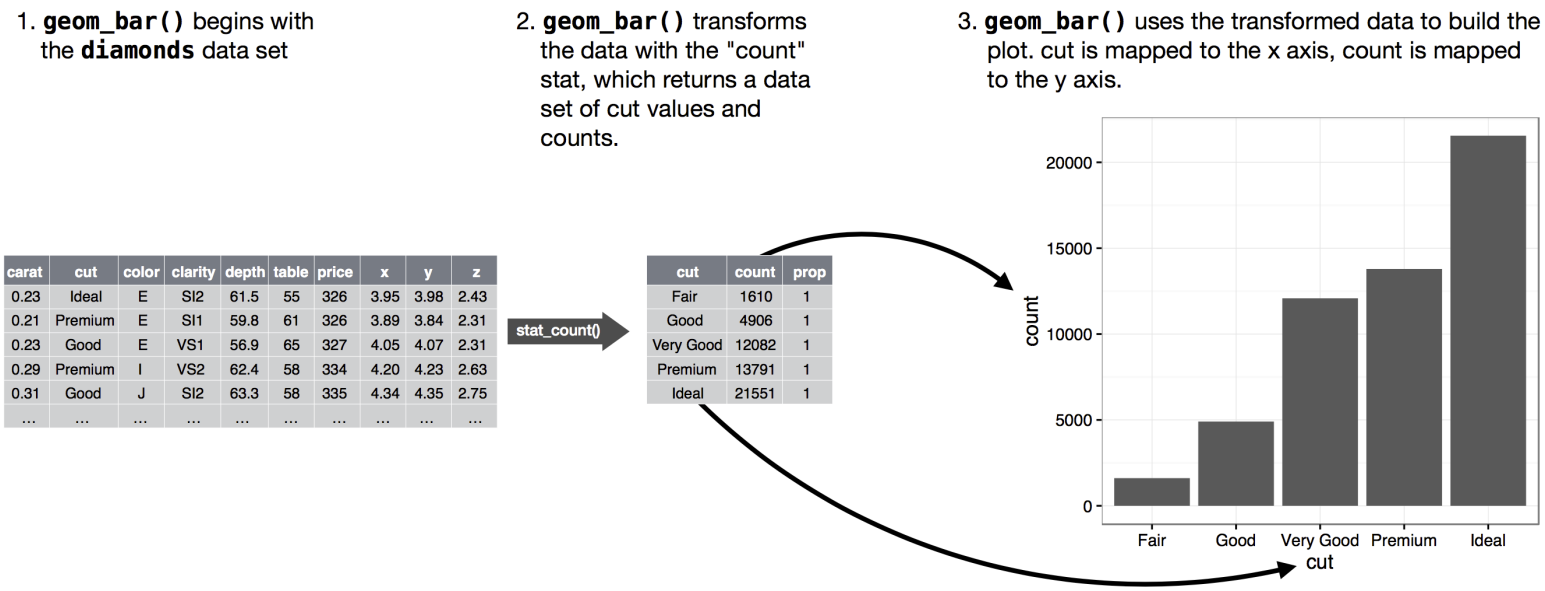
Beyond scatterplots

- Other graphs, like bar charts, calculate new values to plot.
  - Bar charts, histograms, and frequency polygons:
    - Bin data.
    - Plot bin counts (number of points falling in each bin).
  - Smoothers:
    - Fit a model to your data.
    - Plot predictions from the model.
  - Boxplots:
    - Compute a robust summary of the distribution.
    - Display a specially formatted box.

Statistical transformations

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- A **stat**:
  - The algorithm used to calculate new values for a graph.
  - Short for statistical transformation.



- `ggplot2` provides over 20 stats.
- Each stat is a function, get help as usual, e.g. `?stat_bin`.
- Use RStudio's data visualization cheatsheet for a complete list.

Geom and stat

- Every geom has a default stat and conversely.
  - `?geom_bar` shows that the default value for `stat` is "count".
  - Means that `geom_bar()` uses `stat_count()`.
  - `?stat_count` has a section called "Computed variables" with two new variables: `count` and `prop`.
- You can generally use geoms and stats interchangeably!

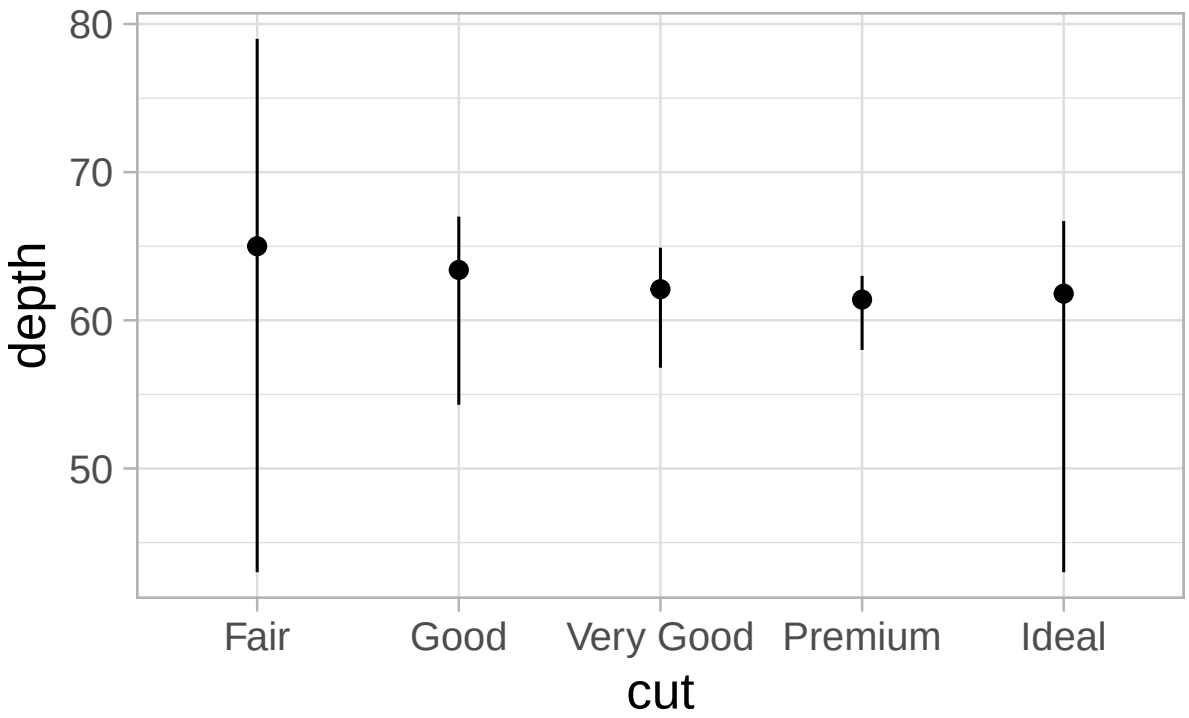
```
ggplot(data = diamonds) +
  stat_count(mapping = aes(x = cut))
```

- Typically, use geoms without worrying about the stat.
- Three reasons to use a stat explicitly:
  - To override the default stat.
  - To override the default mapping from transformed variables to aesthetics.
  - To draw greater attention to the stat in your code.

Use a stat explicitely I

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

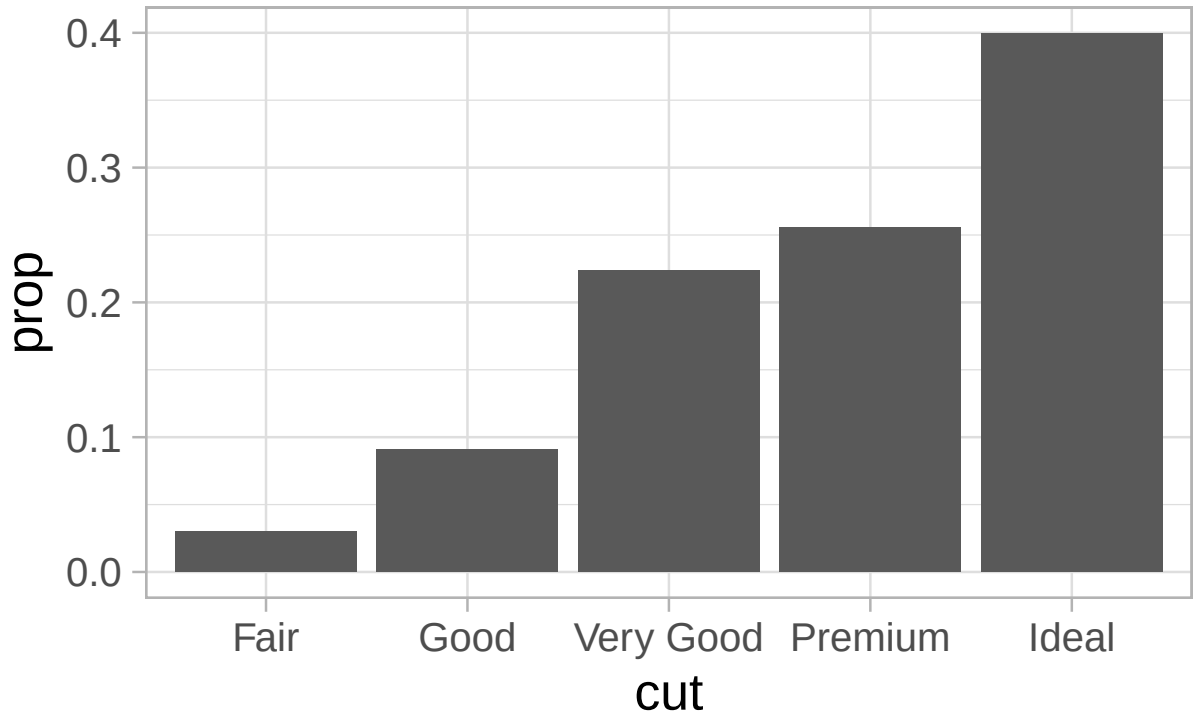
```
ggplot(data = diamonds) +
      stat_summary(
        mapping = aes(x = cut, y = depth),
        fun.ymin = min,
        fun.ymax = max,
        fun.y = median
        )
```



Use a stat explicitely II

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

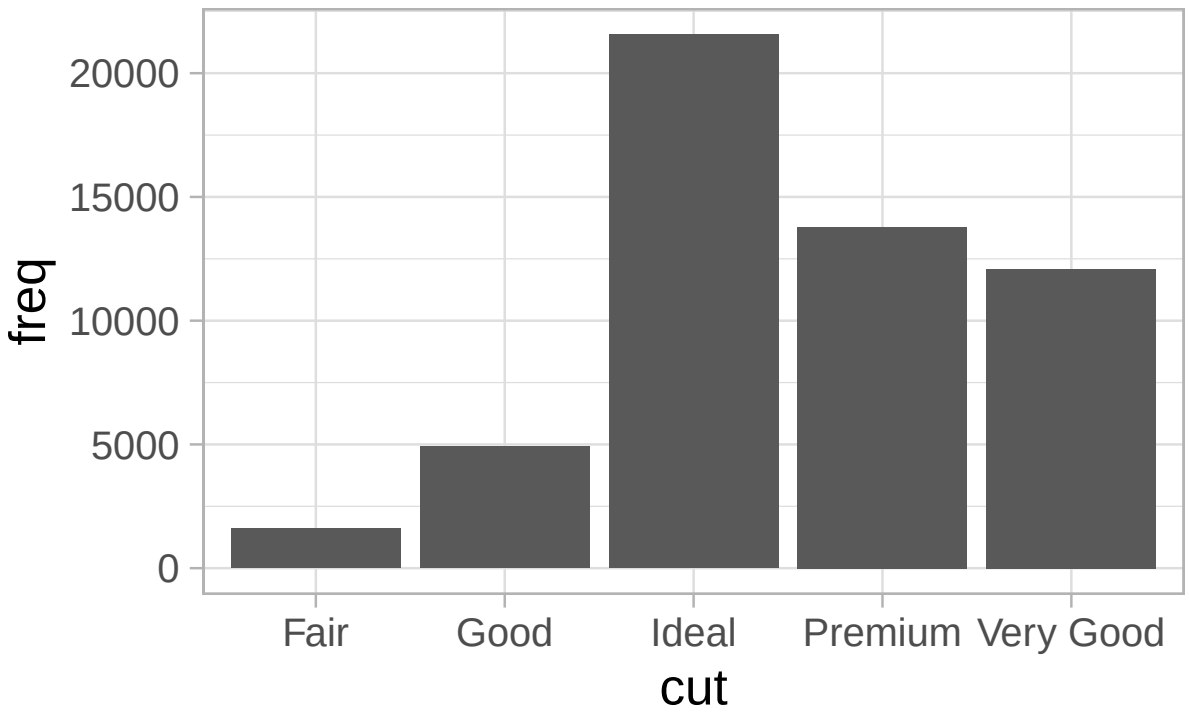
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y = ..prop.., group = 1))
```



Use a stat explicitely III

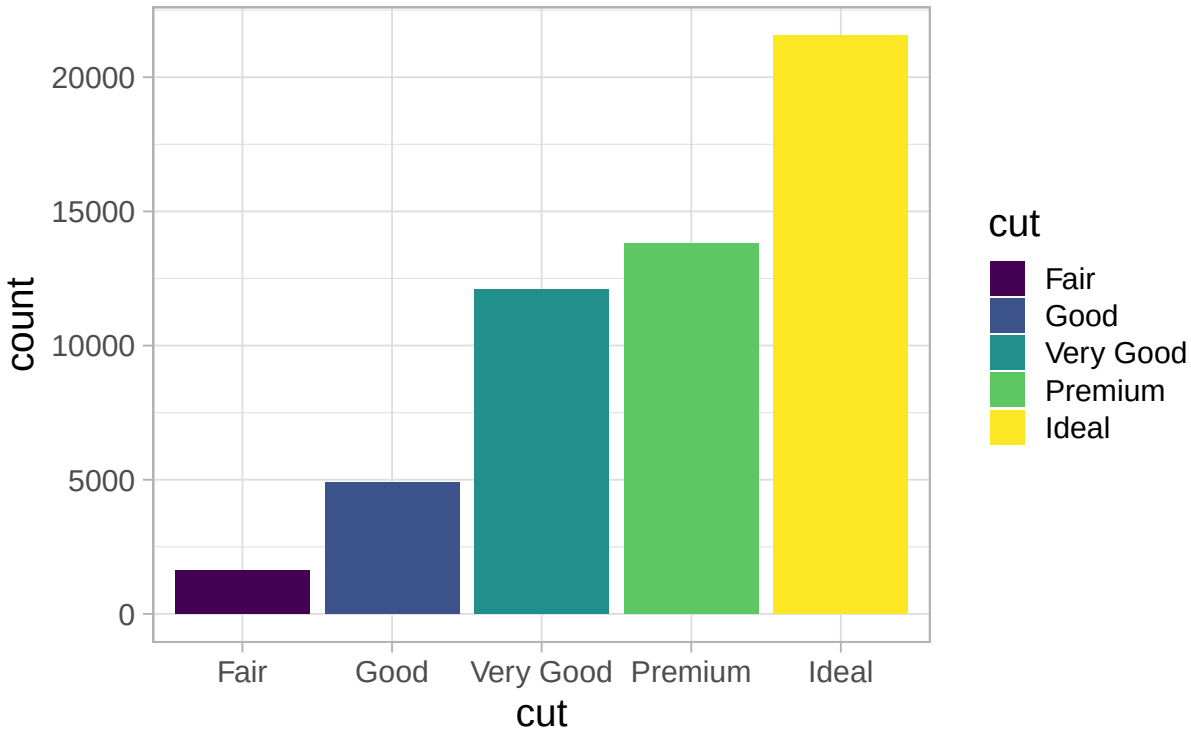
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
demo <- tribble(~cut,          ~freq,
                "Fair",        1610,
                "Good",        4906,
                "Very Good",   12082,
                "Premium",     13791,
                "Ideal",       21551)
 ggplot(data = demo) +
   geom_bar(mapping = aes(x = cut, y = freq), stat = "identity")
```



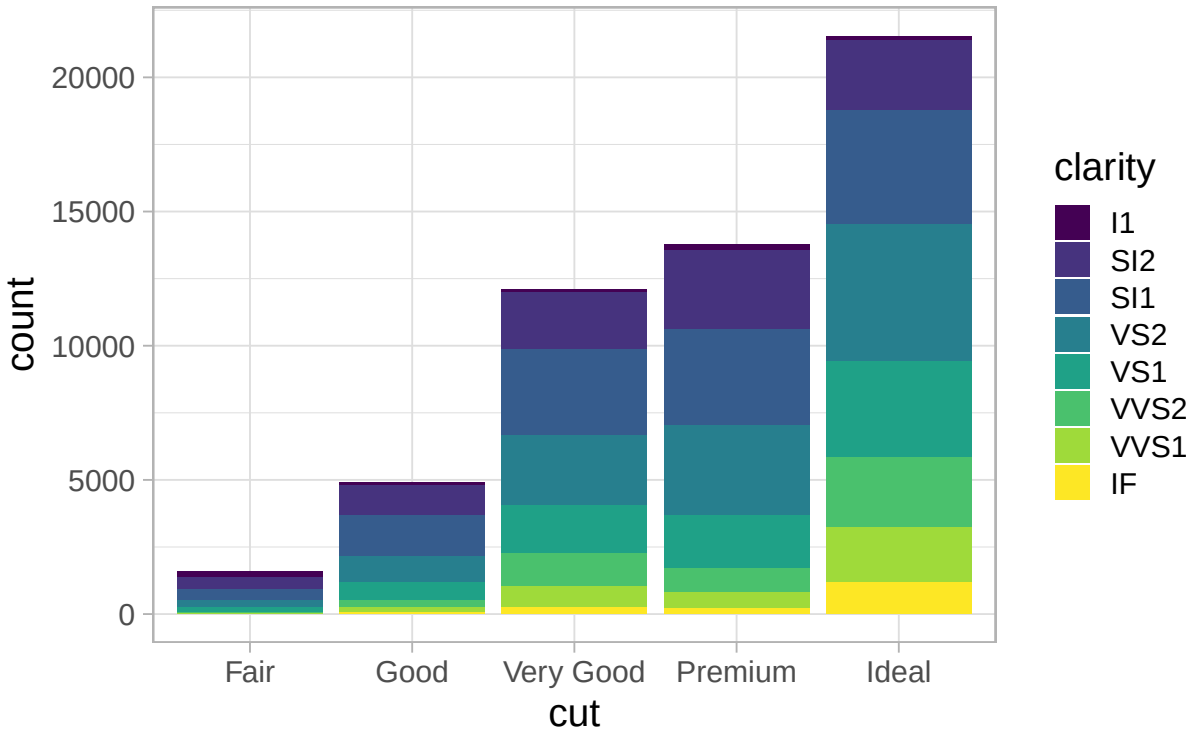
The fill aesthetic


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



Fill and position ajustements

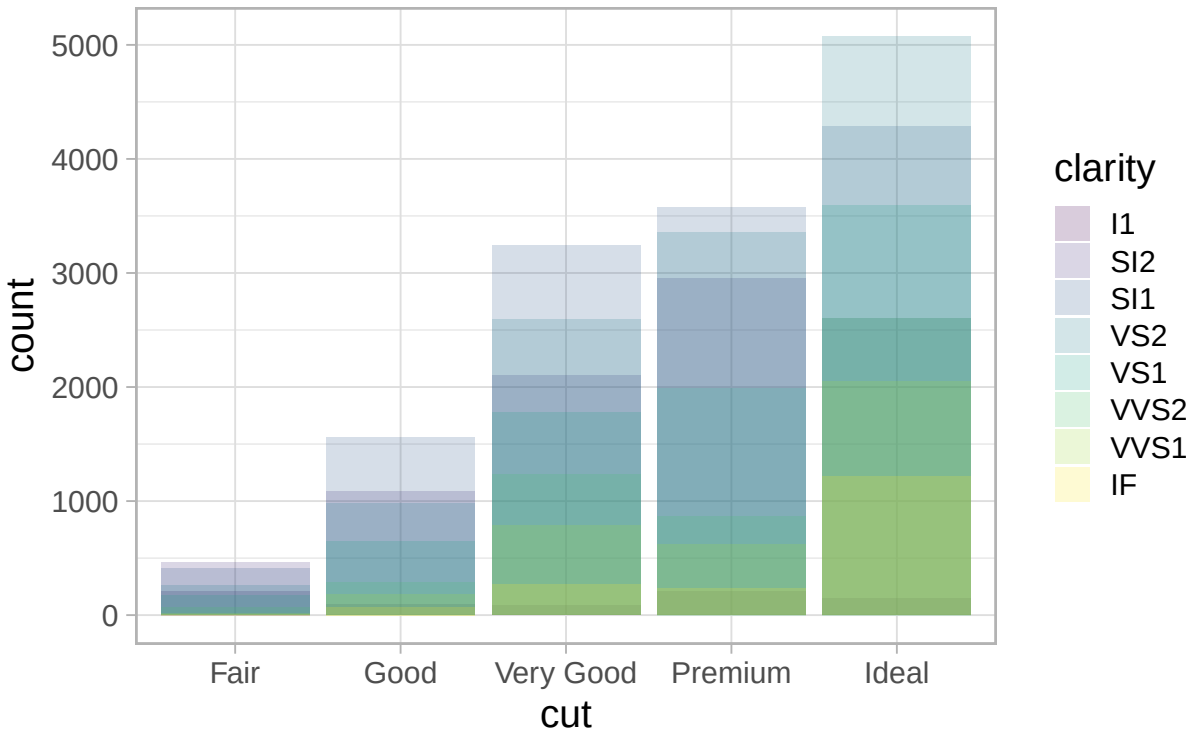

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



- Automatically stacked by the **position adjustement**.
- `?position_stack` to learn more.

Fill with position = "identity"

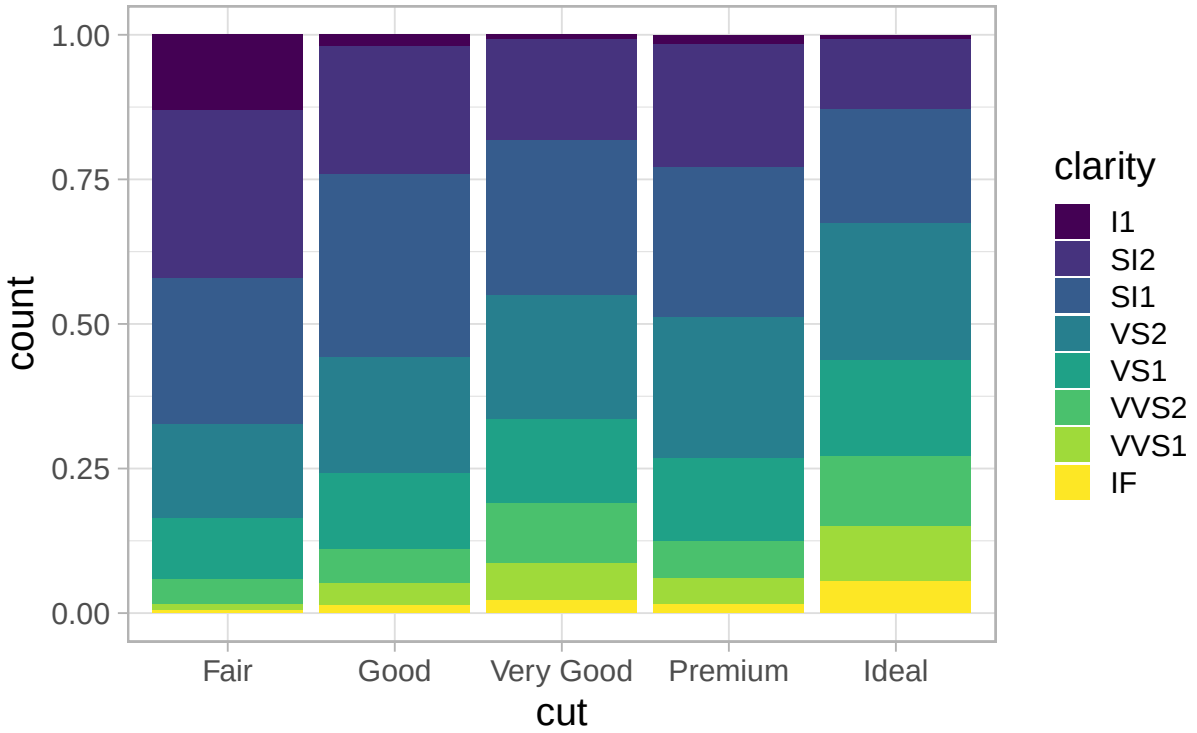

```
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
  geom_bar(alpha = 1/5, position = "identity")
```



- Not very useful for bars because of overlap.
- ?position_identity to learn more.

Fill with `position = "fill"`

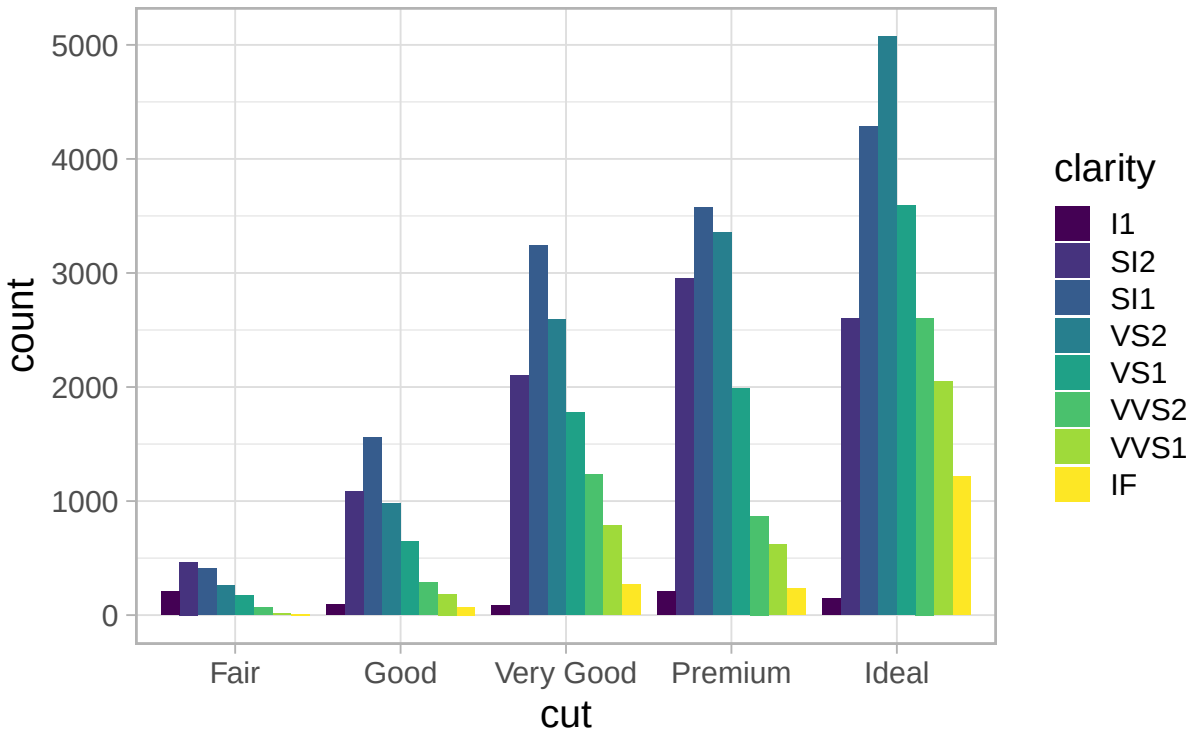

```
ggplot(data = diamonds) +
      geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



- Makes it easier to compare proportions across groups.
- `?position_fill` to learn more.

Fill with `position = "dodge"`

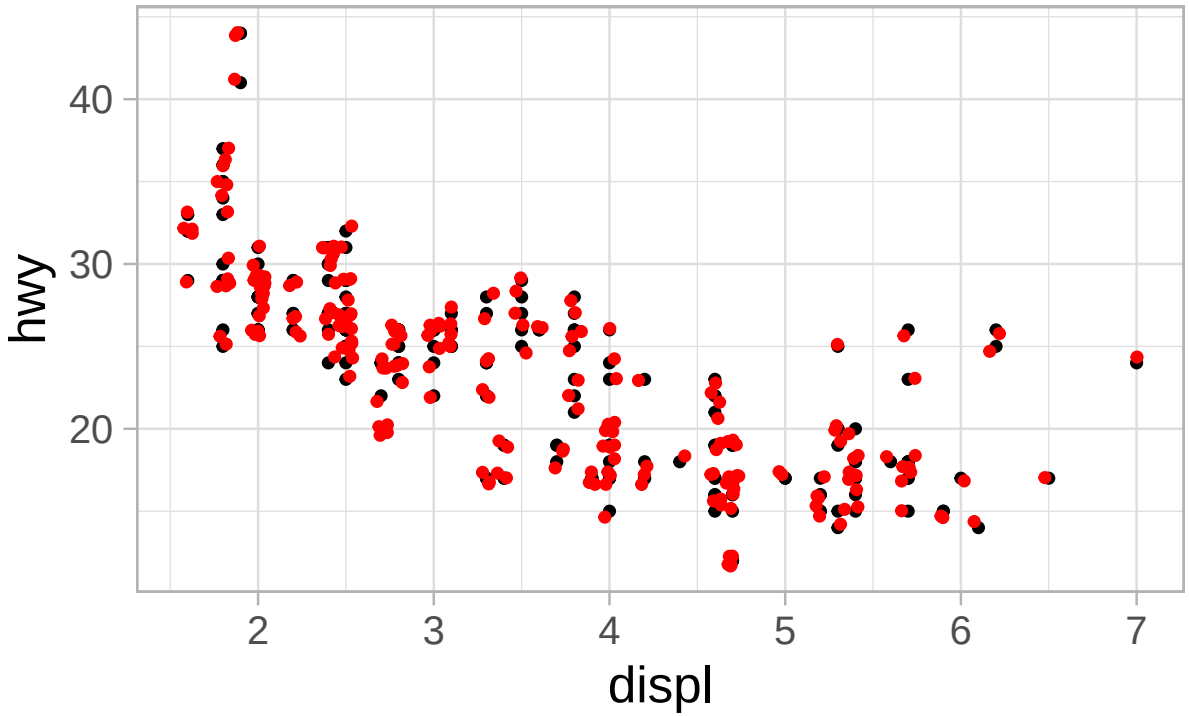

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```



- Makes it easier to compare individual values.
- `?position_dodge` to learn more.

position = "jitter"


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



- Graph less/**more** accurate/**revealing** at small/**large** scales.
- ?position_jitter to learn more.

Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

4 Coordinate systems

5 The layered grammar of graphics

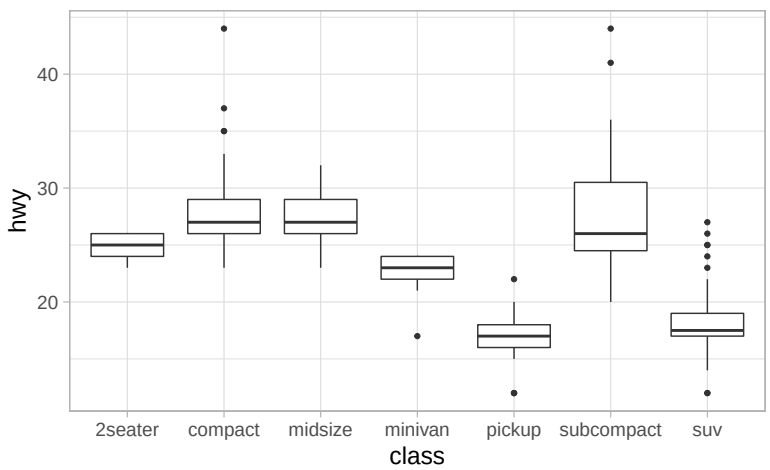
6 Tibbles

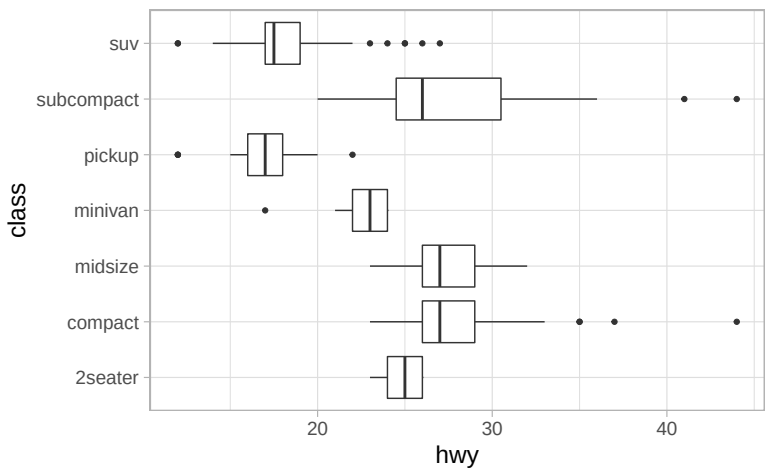
Coordinate systems

- The most complicated part of `ggplot2`.
- Default: the Cartesian coordinate system.
- Other systems occasionally helpful:
  - `coord_flip()` switches the x and y axes.
  - `coord_quickmap()` sets the aspect ratio correctly for maps.
  - `coord_polar()` uses polar coordinates.

coord_flip()


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

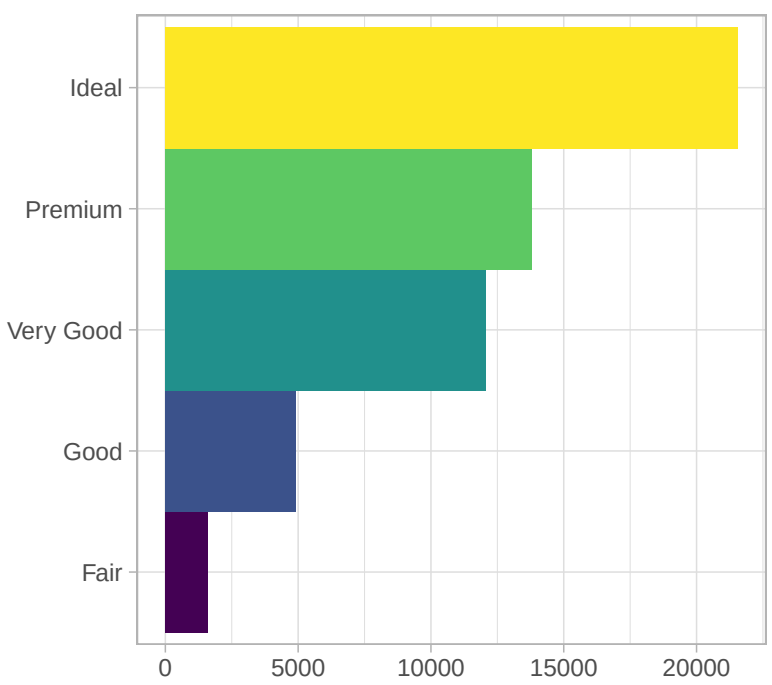


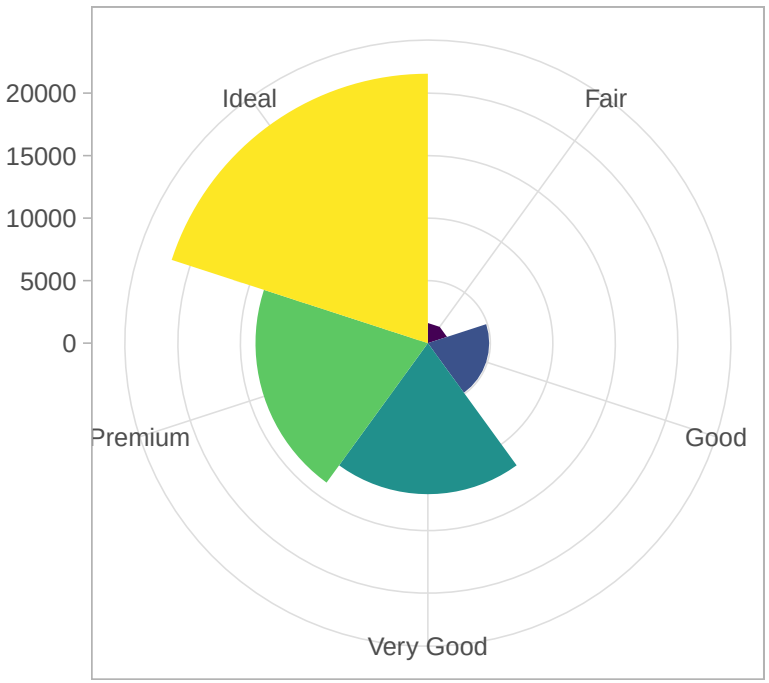


- Useful for:
  - horizontal boxplots,
  - and long labels.

coord_polar()


```
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()
```





Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

4 Coordinate systems

5 The layered grammar of graphics

6 Tibbles

The layered grammar of graphics

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

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

- A formal system for building plots,
- Uniquely describes *any* plot as a combination of
  - a dataset,
  - a geom,
  - a set of mappings,
  - a stat,
  - a position adjustment,
  - a coordinate system,
  - and a faceting scheme.

Example

1. Begin with the **`diamonds`** data set

2. Compute counts for each cut value with **`stat_count()`**.

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
…	…	…	…	…	…	…	…	…	…

stat_count()

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

Example

3. Represent each observation with a bar.
4. Map the **`fill`** of each bar to the **`..count..`** variable.

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
…	…	…	…	…	…	…	…	…	…

stat_count()

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

Example

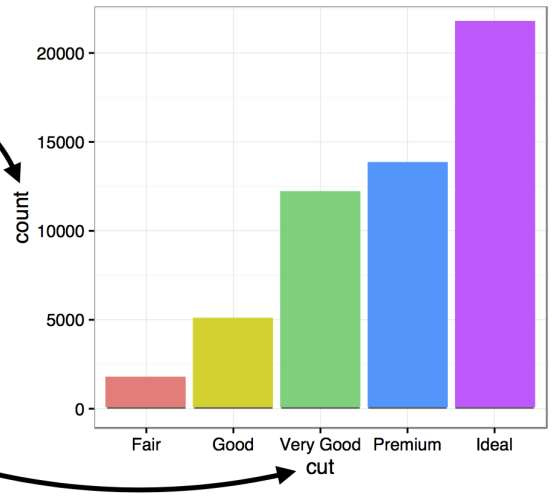
carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

stat_count()

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

5. Place geoms in a cartesian coordinate system.

6. Map the y values to **..count..** and the x values to **cut**.



Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

4 Coordinate systems

5 The layered grammar of graphics

6 Tibbles

What are tibbles?

- Alternative R's traditional `data.frame`.
- Tweak some older behaviours to make life easier.
- Part of the core tidyverse.
- Unifying feature of the tidyverse.
- Most functions from the tidyverse produce tibbles.
- To learn more, see `vignette("tibble")`.

Coerce a data frame to a tibble


```
as_tibble(iris)
#> # A tibble: 150 x 5
#>    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>           <dbl>       <dbl>        <dbl>       <dbl> <fct>
#>  1          5.1         3.5          1.4         0.2 setosa
#>  2          4.9         3            1.4         0.2 setosa
#>  3          4.7         3.2          1.3         0.2 setosa
#>  4          4.6         3.1          1.5         0.2 setosa
#>  5          5           3.6          1.4         0.2 setosa
#>  6          5.4         3.9          1.7         0.4 setosa
#>  7          4.6         3.4          1.4         0.3 setosa
#>  8          5           3.4          1.5         0.2 setosa
#>  9          4.4         2.9          1.4         0.2 setosa
#> 10          4.9         3.1          1.5         0.1 setosa
#> # ... with 140 more rows
```

Create from individual vectors

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
tibble(x = 1:5,
y = 1,
z = x ^ 2 + y)
#> # A tibble: 5 x 3
#>       x     y     z
#>   <int> <dbl> <dbl>
#> 1     1     1     2
#> 2     2     1     5
#> 3     3     1    10
#> 4     4     1    17
#> 5     5     1    26
```

Row-wise tibble creation


```
tribble(
~colA, ~colB,
"a",   1,
"b",   2,
"c",   3
)
#> # A tibble: 3 x 2
#>   colA   colB
#>   <chr> <dbl>
#> 1 a         1
#> 2 b         2
#> 3 c         3
```

Printing tibbles


```
(df <- tibble(a = lubridate::today() + runif(4e1) * 30,
b = 1:4e1,
c = runif(4e1),
d = sample(letters, 4e1, replace = TRUE)))
#> # A tibble: 40 x 4
#>    a                b       c d
#>    <date>       <int>   <dbl> <chr>
#>  1 2020-03-27       1 0.411   l
#>  2 2020-03-08       2 0.821   f
#>  3 2020-03-12       3 0.647   h
#>  4 2020-03-18       4 0.783   g
#>  5 2020-03-28       5 0.553   k
#>  6 2020-03-07       6 0.530   q
#>  7 2020-03-27       7 0.789   d
#>  8 2020-03-29       8 0.0233  m
#>  9 2020-03-20       9 0.477   h
#> 10 2020-03-19      10 0.732   y
#> # ... with 30 more rows
```

Modify default settings


```
print(df, n = 2, width = 30)
#> # A tibble: 40 x 4
#>   a                b     c d
#>   <date>       <int> <dbl> <chr>
#> 1 2020-03-27       1 0.411 l
#> 2 2020-03-08       2 0.821 f
#> # ... with 38 more rows
```

- `options(tibble.print_max = n, tibble.print_min = m)`: if more than `m` rows, print only `n` rows.
- `options(dplyr.print_min = Inf)` to always show all rows.
- `options(tibble.width = Inf)` to always print all columns.
- `package?tibble`

Subsetting tibbles


```
# Extract by name I
df$b
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
#> [22] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

# Extract by name II
df[["b"]]
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
#> [22] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

# Extract by position
df[[2]]
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
#> [22] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Compared to a `data.frame`:

- no partial matching
- warning if the column does not exist