GR5206: lecture 6

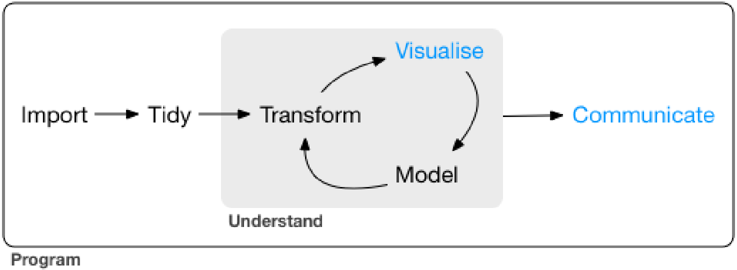
Computational Statistics
And Introduction to Data Science

Thibault Vatter

Department of Statistics, Columbia University

10/11/2019

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. Labels, axes, annotations and legends
7. Colors, zooming and themes

Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

4 Coordinate systems

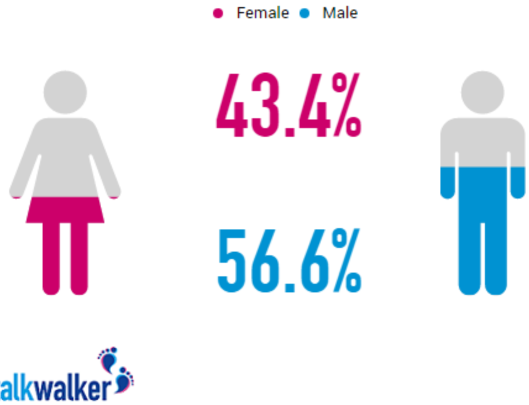
5 The layered grammar of graphics

6 Labels, axes, annotations and legends

7 Colors, zooming and themes

Data content

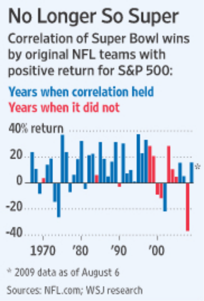
- Makes no sense to use graphs for very small amounts of data.
- The human brain is capable of grasping a few values.



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.



No Longer So Super

Correlation of Super Bowl wins by original NFL teams with positive return for S&P 500:

* 2009 data as of August 6

Sources: NFL.com; WSJ research

- 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”!

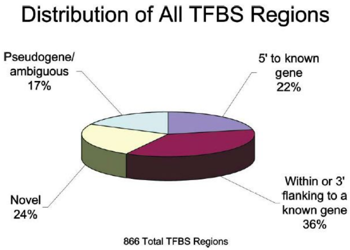


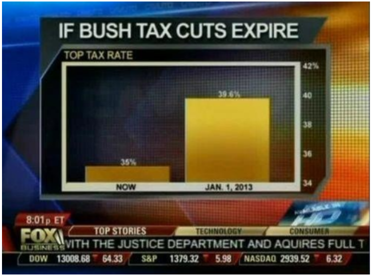
Figure 1. Classification of TFBS Regions

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.

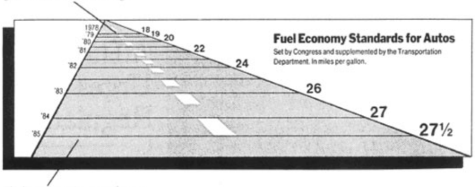


source: statisticshowto.com/misleading-graphs/

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 $= \dfrac{\text{size of effect shown in graphic}}{\text{size of effect shown in data}}$
    - Don't take this too seriously.
    - Defined by Ed Tufte of Yale.
  - If lie factor is > 1, the graph is exaggerating the effect.

This line, representing 18 miles per gallon in 1978, is 0.6 inches long.



This line, representing 27.5 miles per gallon in 1985, is 5.3 inches long.

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

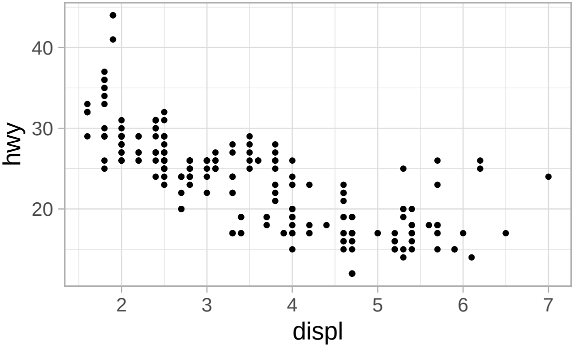
- Data from the US EPA on 38 models of car:

```
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

```r
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

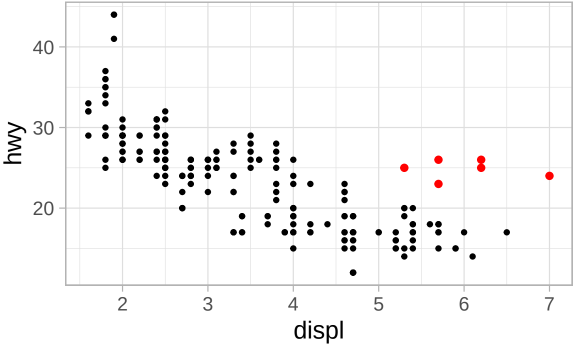
5 The layered grammar of graphics

6 Labels, axes, annotations and legends

7 Colors, zooming and themes

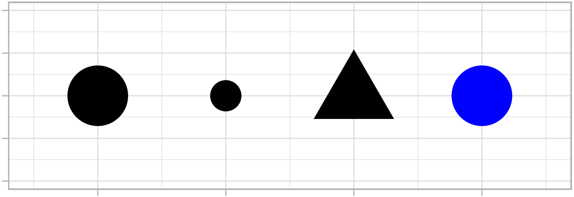
Aesthetic mappings

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



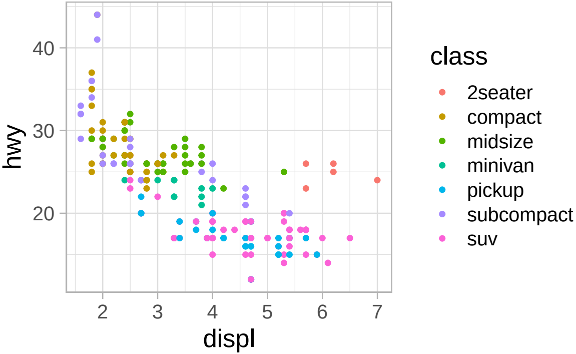
Aesthetic

- 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))
```

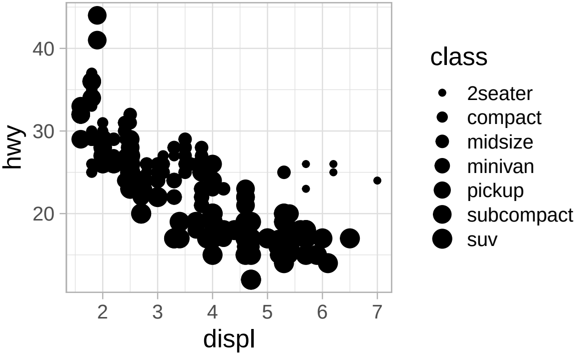


- If you prefer British English, use `colour` instead of `color`.

The size aesthetic

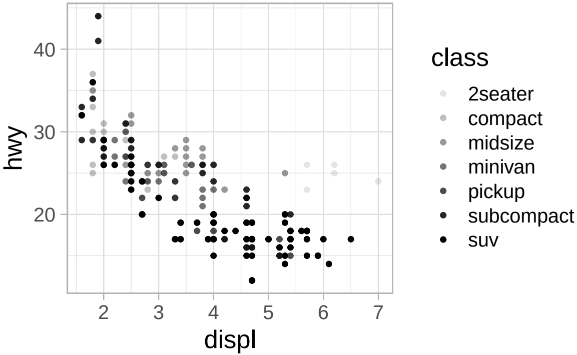
Columbia University
In the City of New York

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



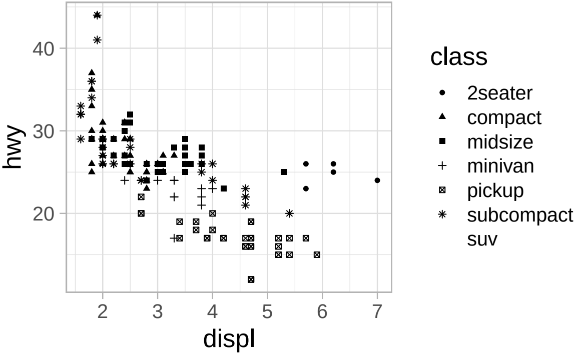
The alpha aesthetic

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



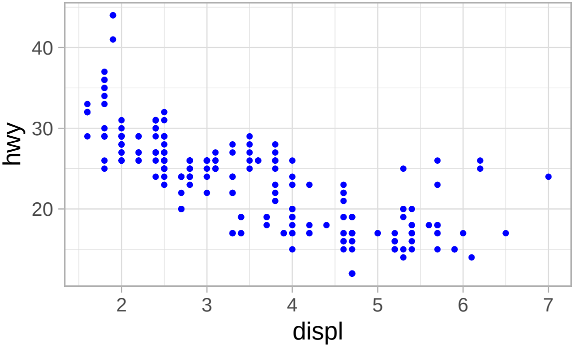
The shape aesthetic

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



Set the aesthetics manually

```
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.

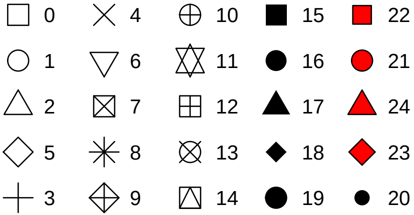
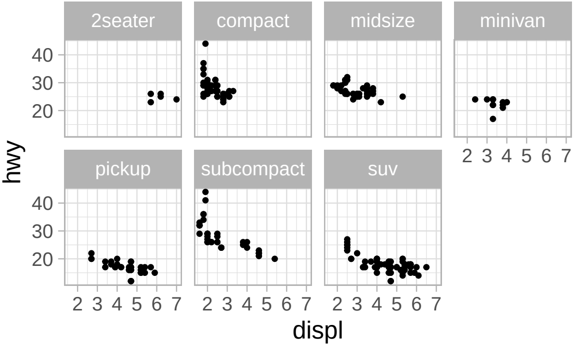


Figure 1: The hollow shapes (0–14) have a border determined by ‘color‘; the solid shapes (15–18) are filled with ‘color‘; the filled shapes (21–24) have a border of ‘color‘ and are filled with ‘fill‘.

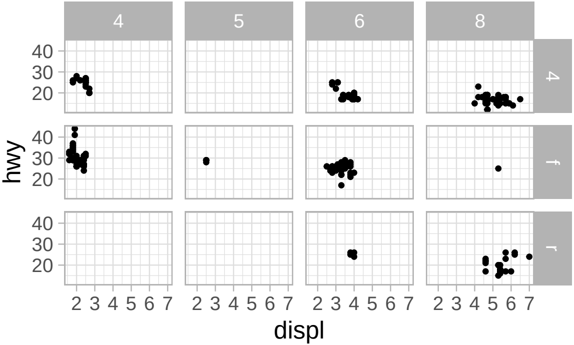
Facets wrap

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



Facets grid

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

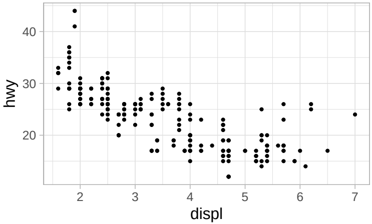


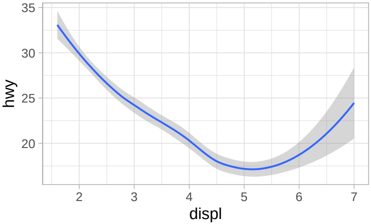
Outline

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. Labels, axes, annotations and legends
7. Colors, zooming and themes

How are these two plots similar?

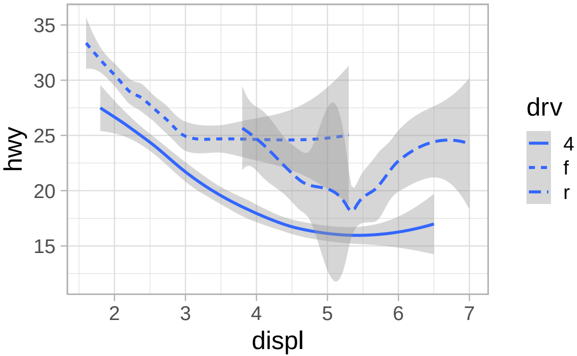
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))

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))
```

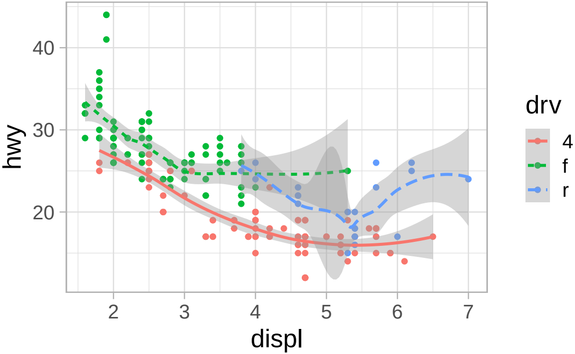


Geometric objects

- 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

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

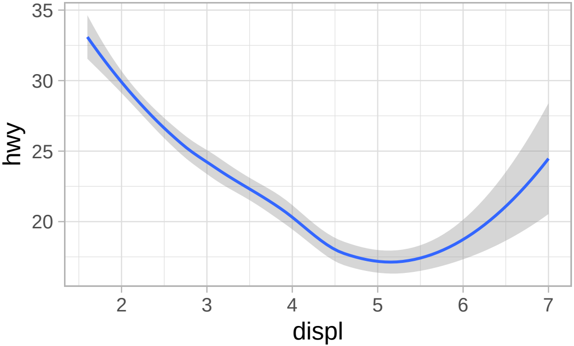


More on geoms

- 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.

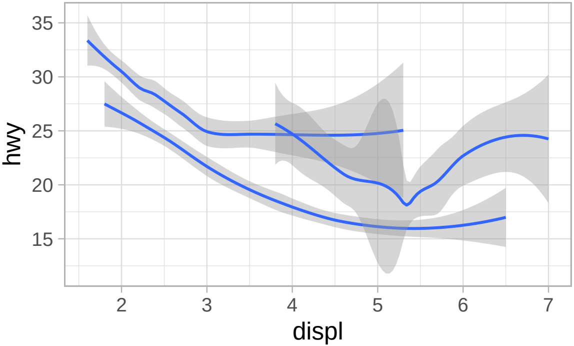
Geoms and legends

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



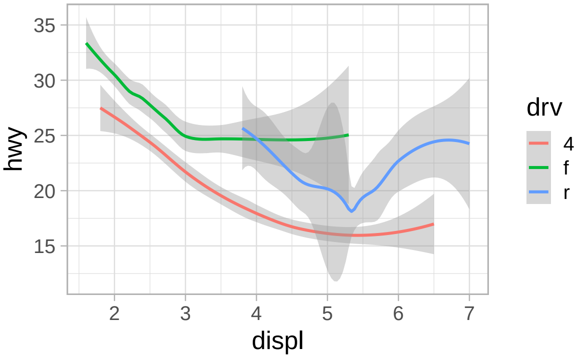
Geoms and legends

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



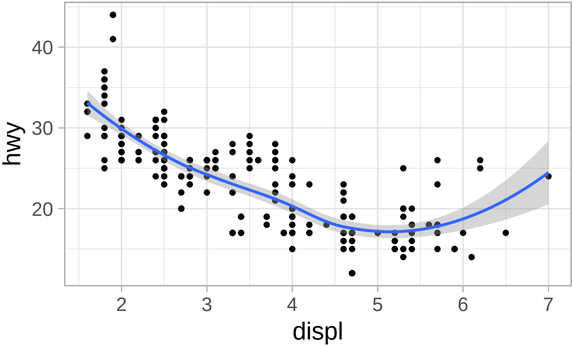
Geoms and legends

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



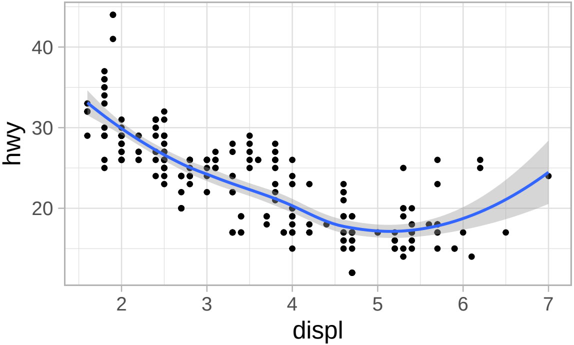
Multiple geoms in the same plot

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



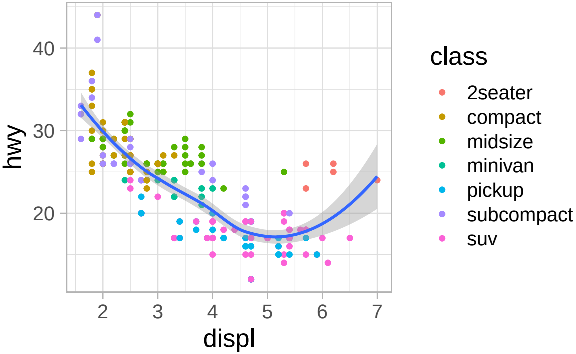
A better way

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



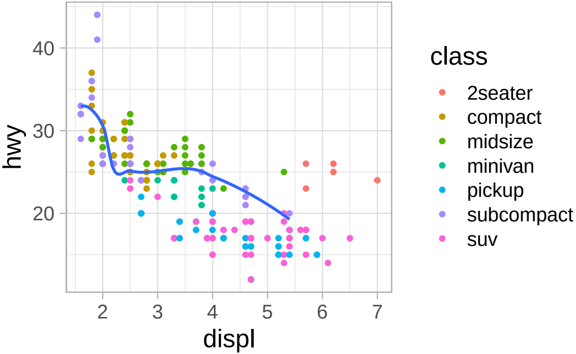
Local vs global mappings

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



Layer dependent data

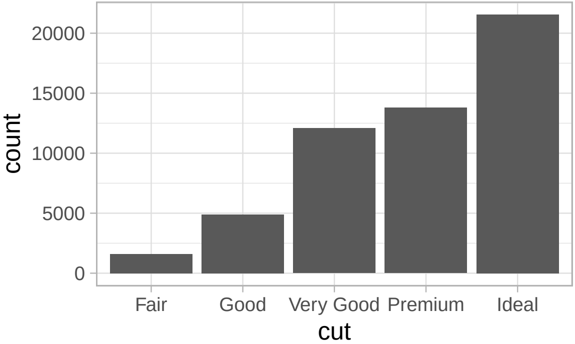
```r
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(data = filter(mpg, class == "subcompact"), se = FALSE)
```



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))
```

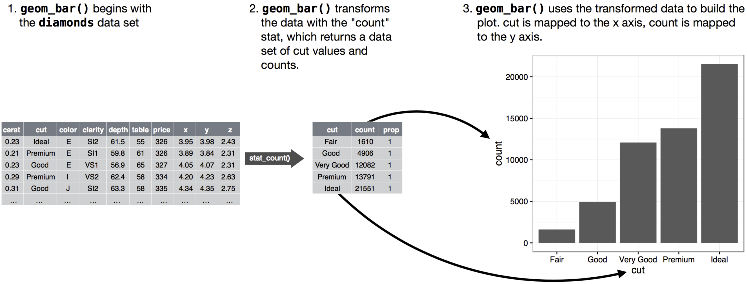


Beyond scatterplots

- 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

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



1. **`geom_bar()`** begins with the **`diamonds`** data set

2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

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

- `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

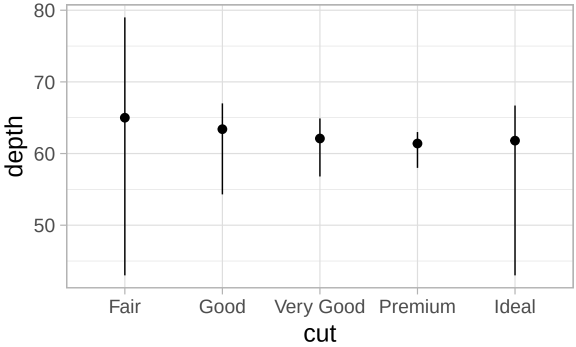
- Every geom has a default stat and conversely.
  - ?geom_bar shows that the default value for stat is "count".
  - Means that geom_bar() uses stat_count().
  - ?stat_count has a section called "Computed variables" with two new variables: count and prop.
- You can generally use geoms and stats interchangeably!

```r
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.

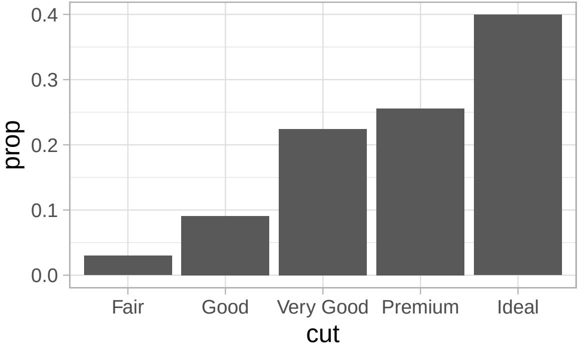
Use a stat explicitely I

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



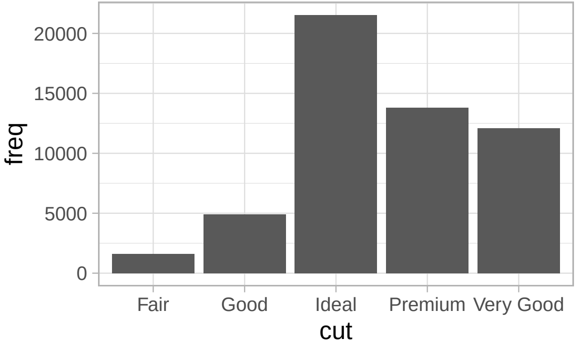
Use a stat explicitely II

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



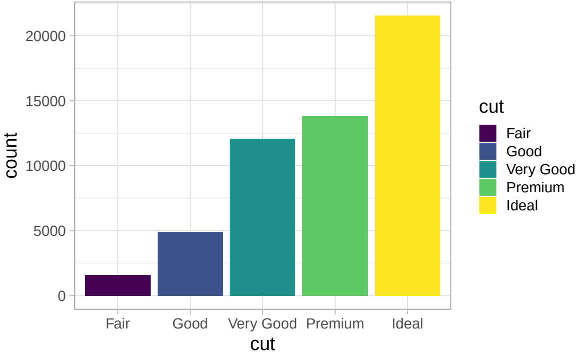
Use a stat explicitely III

```
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")
```



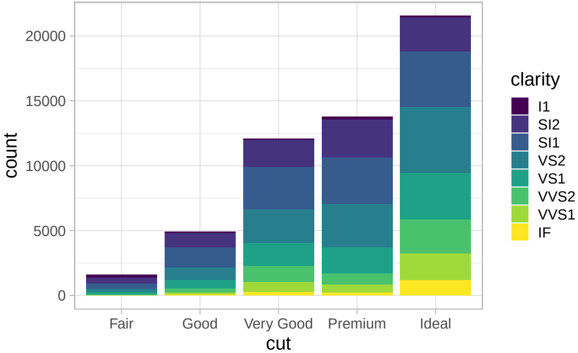
The fill aesthetic

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



Fill and position ajustements

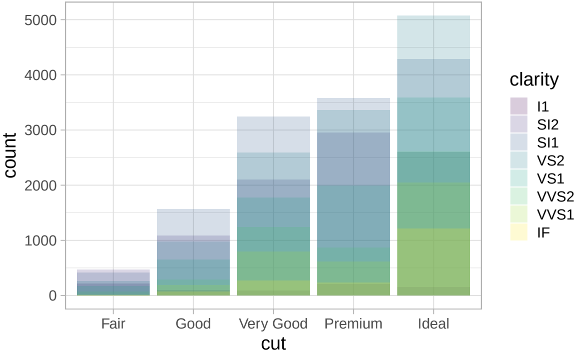
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



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

Fill with `position = "identity"`

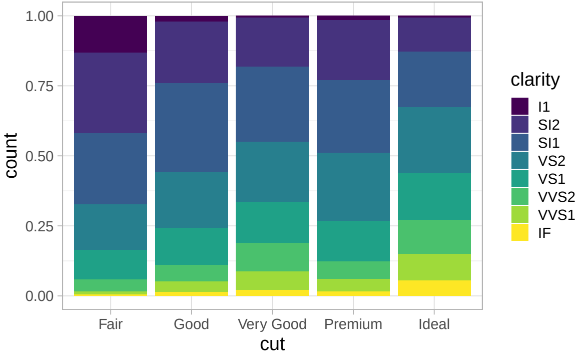
```r
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
  geom_bar(alpha = 1/5, position = "identity")
```



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

Fill with `position = "fill"`

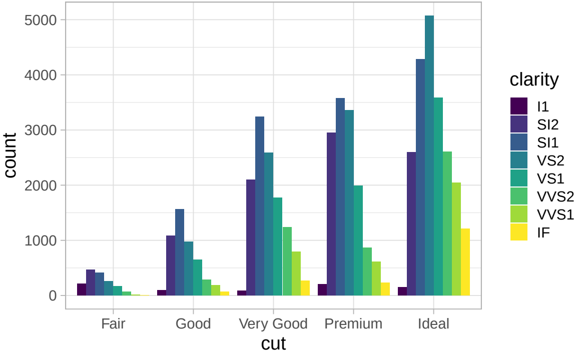
```r
ggplot(data = diamonds) +
    geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



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

Fill with `position = "dodge"`

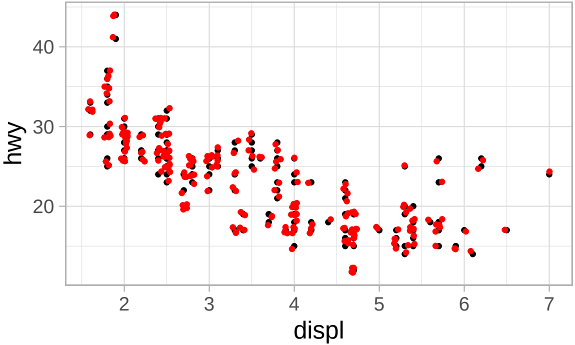
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```



- 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 Labels, axes, annotations and legends

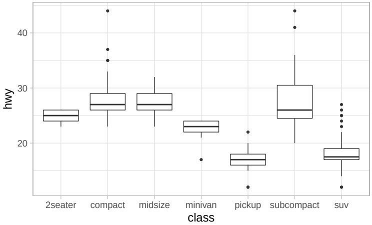
7 Colors, zooming and themes

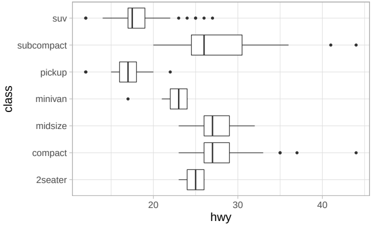
Coordinate systems

- 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()

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

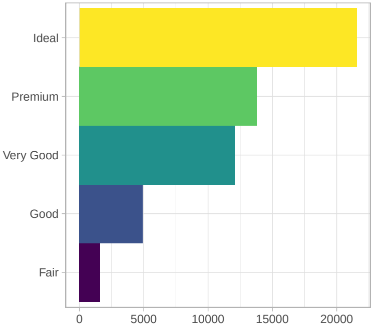


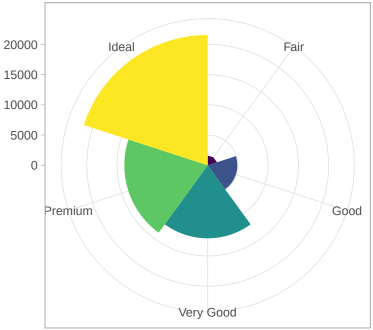


- Useful for:
  - horizontal boxplots,
  - and long labels.

coord_polar()

```r
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 Labels, axes, annotations and legends

7 Colors, zooming and themes

The layered grammar of graphics

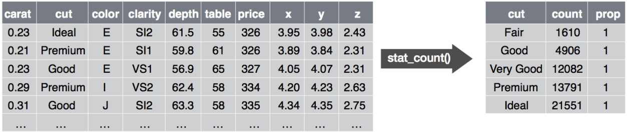
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

- 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()`**.



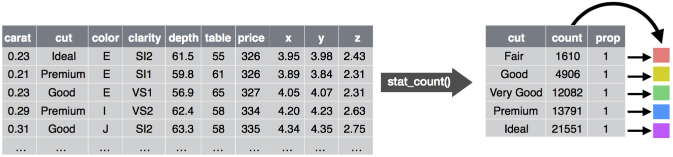
carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...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

stat_count()

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

Example

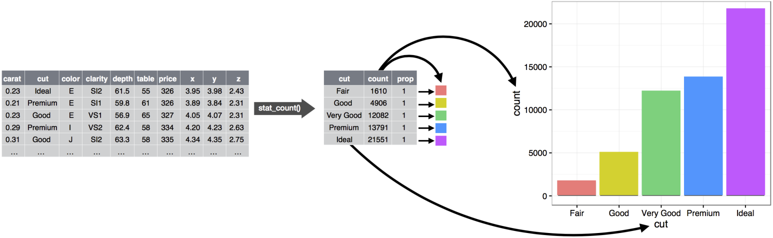
3. Represent each observation with a bar.
4. Map the **`fill`** of each bar to the **`..count..`** variable.



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
...	...	...	...	...	...	...	...	...	...

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

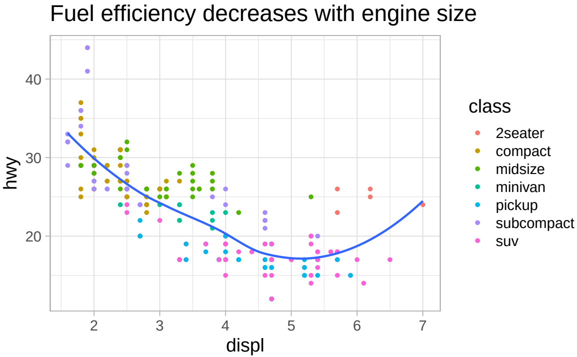
5 The layered grammar of graphics

6 **Labels, axes, annotations and legends**

7 Colors, zooming and themes

Label

```r
ggplot(mpg, aes(displ, hwy)) + geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(title = "Fuel efficiency decreases with engine size")
```

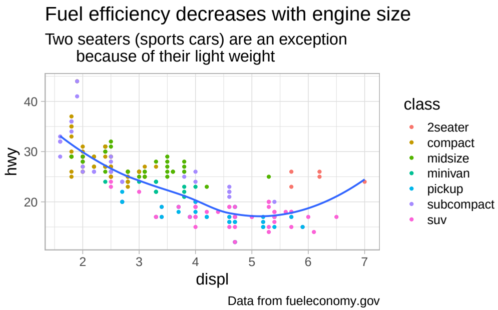


- Avoid titles that just describe what the plot is!

More text

- subtitle: additional details beneath the title.
- caption: text at the bottom right of the plot.

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) + geom_smooth(se = FALSE) +
  labs(title = "Fuel efficiency decreases with engine size",
       subtitle = "Two seaters (sports cars) are an exception
       because of their light weight",
       caption = "Data from fueleconomy.gov")
```



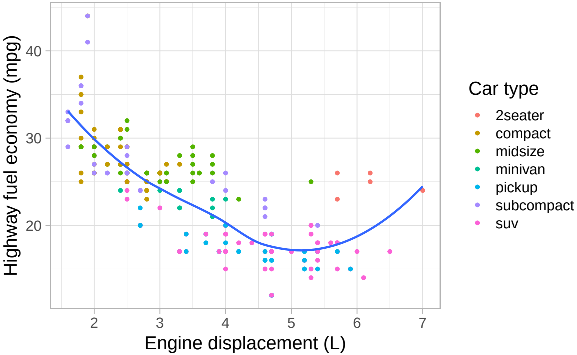
Fuel efficiency decreases with engine size

Two seaters (sports cars) are an exception
because of their light weight

Data from fueleconomy.gov

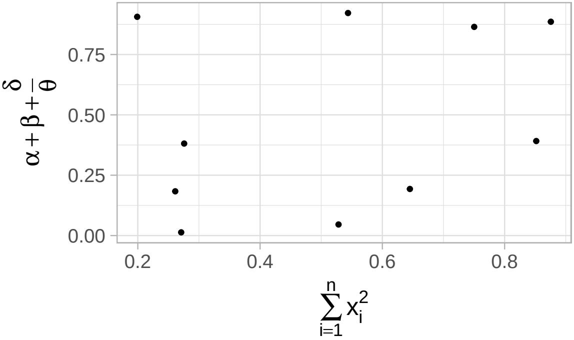
Axes

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



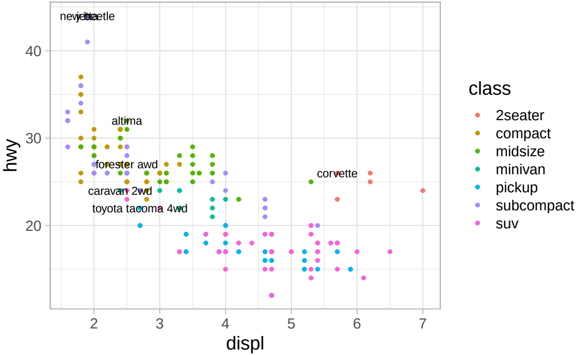
Mathematical expressions

```r
df <- tibble(x = runif(10),
             y = runif(10))
ggplot(df, aes(x, y)) + geom_point() +
  labs(x = quote(sum(x[i] ^ 2, i == 1, n)),
       y = quote(alpha + beta + frac(delta, theta)))
```



Annotations

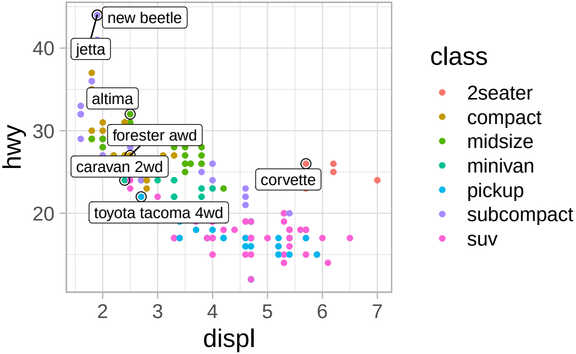
```r
best_in_class <- mpg %>%
  group_by(class) %>%
  filter(row_number(desc(hwy)) == 1)

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



Or better

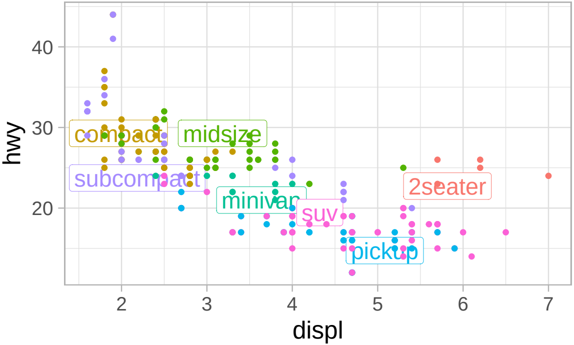
- Use the **ggrepel** package!

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



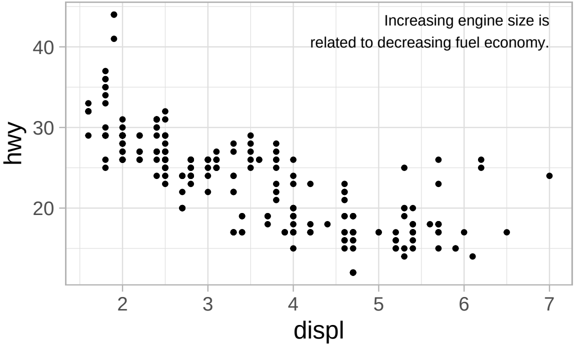
Replace legend by labels on the plot

```
class_avg <- mpg %>% group_by(class) %>%
  summarise(displ = median(displ), hwy = median(hwy))

ggplot(mpg, aes(displ, hwy, color = class)) +
  ggrepel::geom_label_repel(aes(label = class), data = class_avg,
                            size = 6, label.size = 0,
                            segment.color = NA) +
  geom_point() + theme(legend.position = "none")
```



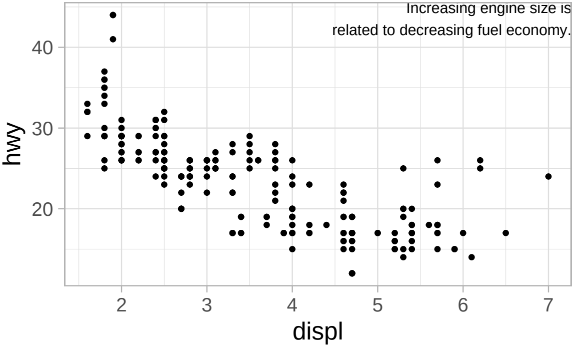
To add a single label to the plot

```
label <- mpg %>%
  summarise(displ = max(displ), hwy = max(hwy),
            label = "Increasing engine size is
            related to decreasing fuel economy.")

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



An alternative

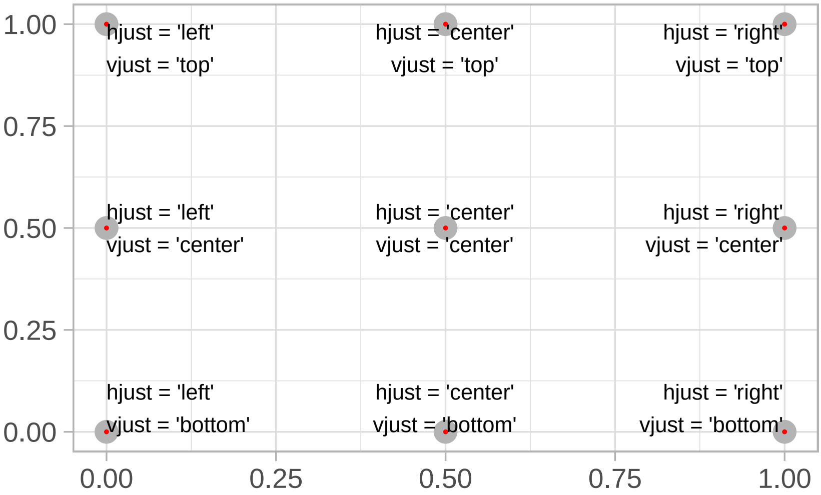
```
label <- tibble(displ = Inf, hwy = Inf,
                label = "Increasing engine size is
                related to decreasing fuel economy.")

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



To automatically add line breaks

```r
"Increasing engine size is related to decreasing fuel economy." %>%
  stringr::str_wrap(width = 40) %>%
  writeLines()
#> Increasing engine size is related to
#> decreasing fuel economy.
```

To control the alignment of the label

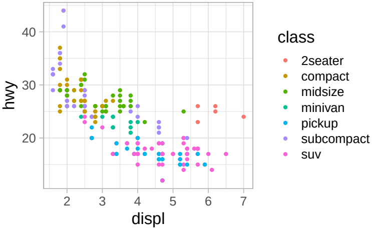


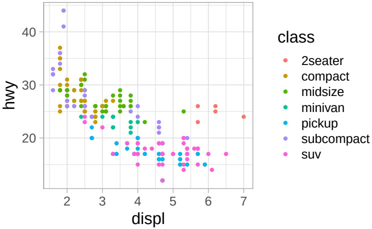
Geoms to help annotate your plot

- `geom_hline()` and `geom_vline()`:
  - Add reference lines.
  - Using e.g. `size = 2` is often a good idea.
- `geom_rect()`:
  - Draw a rectangle around points of interest.
  - Boundaries defined by `xmin`, `xmax`, `ymin`, `ymax`.
- `geom_segment()` with the `arrow` argument:
  - Draw attention to a point with an arrow.
  - `x`/`xend` and `y`/`yend` define the start/end locations.
- The only limit is your imagination (and patience)!

Scales

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

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  scale_x_continuous() +
  scale_y_continuous() +
  scale_color_discrete()
```

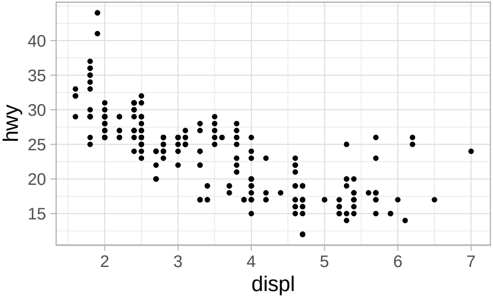




Axis ticks and legend keys

- To control the ticks on the axes and the keys on the legend:
  - `breaks`: controls the position of the ticks, or the values associated with the keys.
  - `labels`: controls the text label associated with each tick/key.

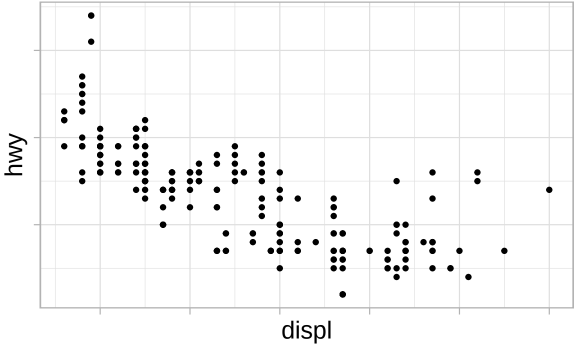
```r
ggplot(mpg, aes(displ, hwy)) + geom_point() +
  scale_y_continuous(breaks = seq(15, 40, by = 5))
```



Axis ticks and legend keys II

- A useful trick for maps, or for publishing plots where you can't share the absolute numbers:

```
ggplot(mpg, aes(displ, hwy)) + geom_point() +
  scale_x_continuous(labels = NULL) +
  scale_y_continuous(labels = NULL)
```



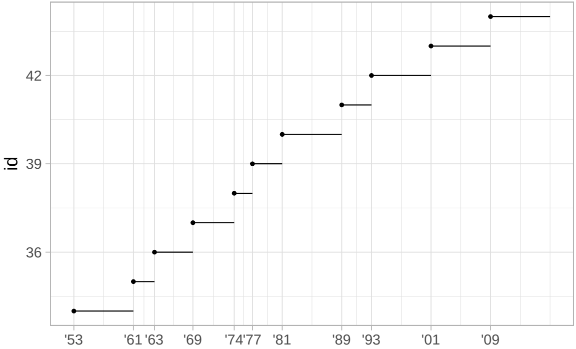
Two remarks

- Collectively axes and legends are called **guides**:
  - Axes are used for x and y aesthetics.
  - Legends are used for everything else.
  - You can also use `breaks` and `labels` to control the appearance of legends.
- Breaks and labels for date and datetime scales work differently:
  - `date_labels`: takes a format specification, see `?readr::parse_datetime()`.
  - `date_breaks`: takes a string like `"2 days"` or `"1 month"`.

breaks

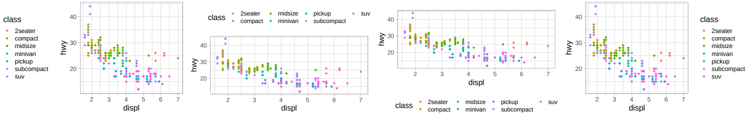
- Allow e.g. to highlight exactly where the observations occur:

```r
presidential %>%
  mutate(id = 33 + row_number()) %>%
  ggplot(aes(start, id)) + geom_point() +
    geom_segment(aes(xend = end, yend = id)) +
    scale_x_date(NULL, breaks = presidential$start, date_labels = "'%y")
```



Legend layout

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

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

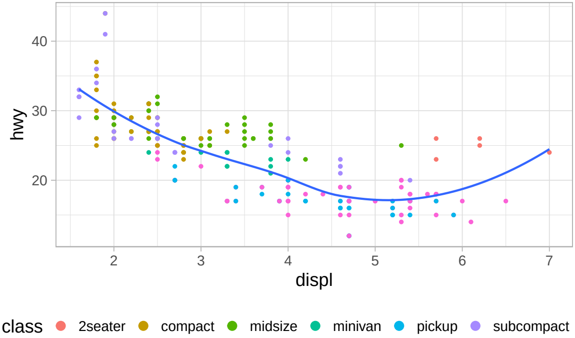


- `legend.position = "none"` suppresses the display of the legend!

To control individual legends

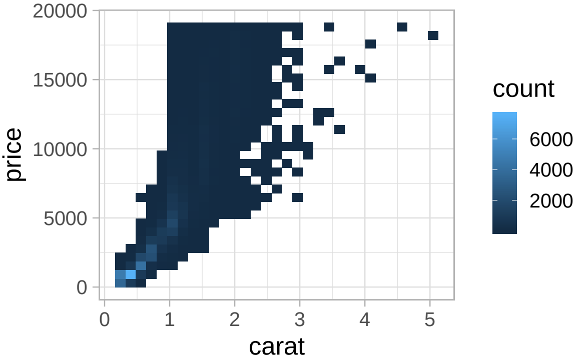
- Use guides(), guide_legend() or guide_colorbar():

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



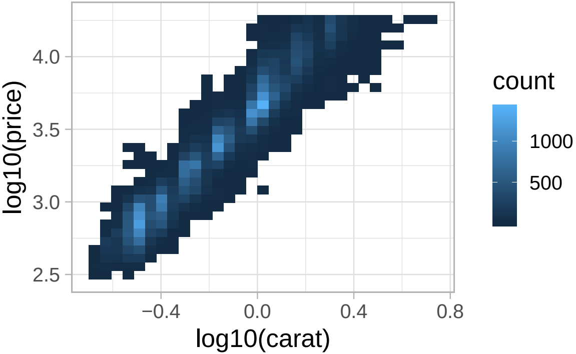
How could we improve the scale?

```r
ggplot(diamonds, aes(carat, price)) +
  geom_bin2d()
```



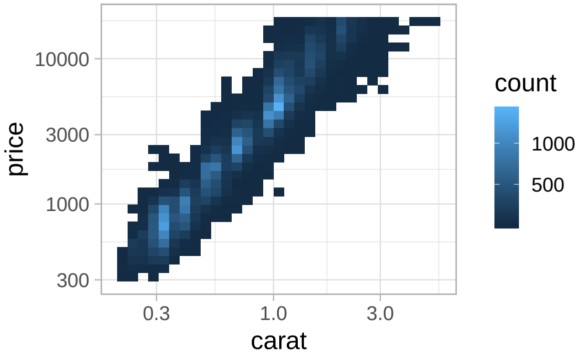
Log-transform the variables

```r
ggplot(diamonds, aes(log10(carat), log10(price))) +
  geom_bin2d()
```



. . . or simply replace the scale

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



Outline

1 From bad graphs to the grammar of graphics

2 Aesthetics and facetting

3 Geometric objects and statistical transformations

4 Coordinate systems

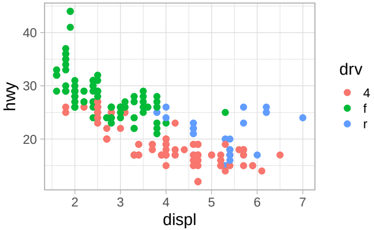
5 The layered grammar of graphics

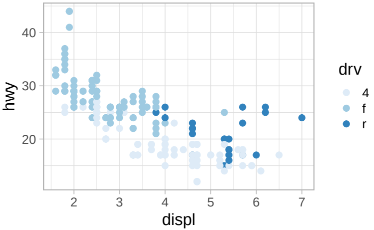
6 Labels, axes, annotations and legends

7 Colors, zooming and themes

Replacing color scales

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv), size = 3)

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv), size = 3) +
  scale_color_brewer(palette = "Blues")
```

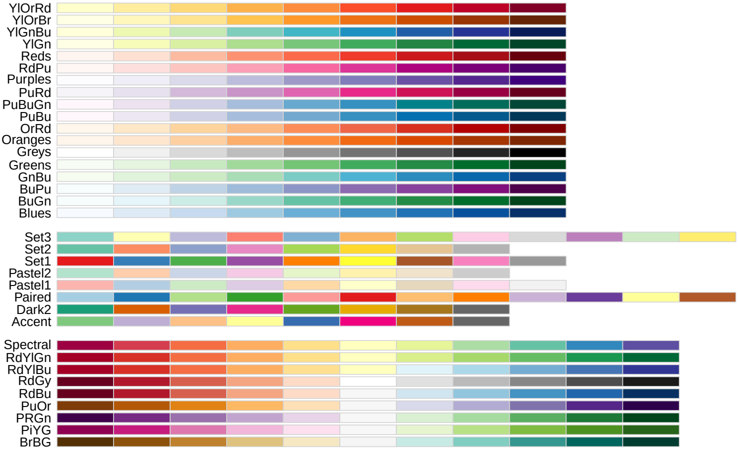




- Color scales come in two variety:
  - `scale_color_x()` for the `color` aesthetics (available in UK/US spellings).
  - `scale_fill_x()` for the `fill` aesthetics.

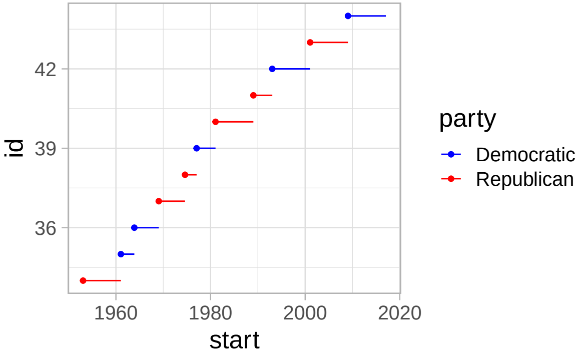
The ColorBrewer scales

- Documented online at http://colorbrewer2.org/
- Available via the **RColorBrewer** package.



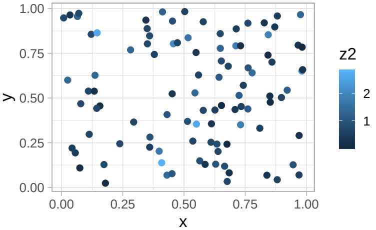
Using manually defined mappings

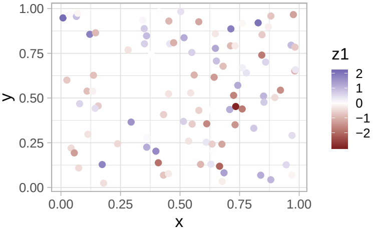
```r
presidential %>%
  mutate(id = 33 + row_number()) %>%
  ggplot(aes(start, id, color = party)) +
    geom_point() +
    geom_segment(aes(xend = end, yend = id)) +
    scale_color_manual(values = c(Republican = "red",
                                  Democratic = "blue"))
```



Continuous vs diverging color scales

```
df <- data.frame(x = runif(100), y = runif(100),
                 z1 = rnorm(100), z2 = abs(rnorm(100)))

ggplot(df, aes(x, y)) +
  geom_point(aes(color = z2), size = 3)

ggplot(df, aes(x, y)) +
  geom_point(aes(color = z1), size = 3) +
  scale_color_gradient2()
```

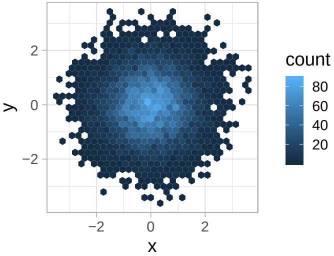


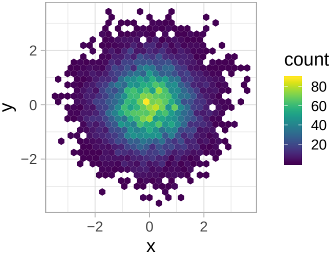


A continuous analog of ColorBrewer

- The viridis package!

```
df <- tibble(x = rnorm(10000), y = rnorm(10000))

ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed()

ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed() +
  viridis::scale_fill_viridis()
```

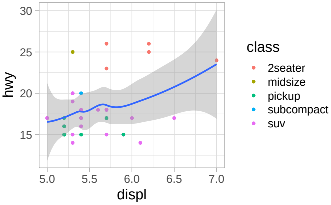


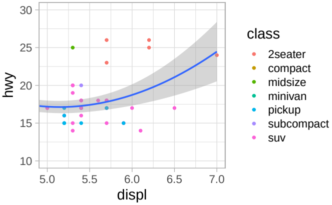


Zooming

- Three methods:
  - Adjust what data are plotted.
  - Set `xlim` and `ylim` in `coord_cartesian()`.
  - Set the limits in each scale.

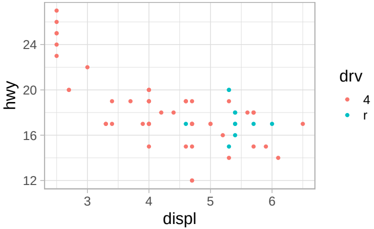
```
mpg %>%
  filter(displ >= 5, displ <= 7, hwy >= 10, hwy <= 30) %>%
  ggplot(aes(displ, hwy)) +
    geom_point(aes(color = class)) + geom_smooth()

ggplot(mpg, mapping = aes(displ, hwy)) +
  geom_point(aes(color = class)) + geom_smooth() +
  coord_cartesian(xlim = c(5, 7), ylim = c(10, 30))
```

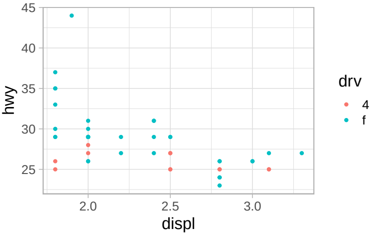




Zooming cont'd

```r
suv <- mpg %>%
  filter(class == "suv")
compact <- mpg %>%
  filter(class == "compact")

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

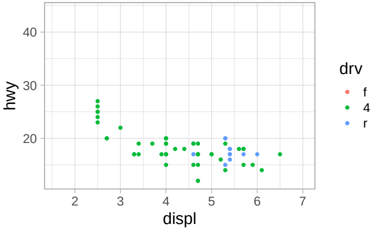


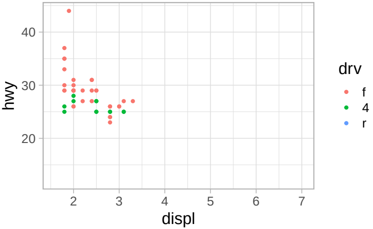


Share scales across multiple plots

- Training the scales with the `limits` of the full data:

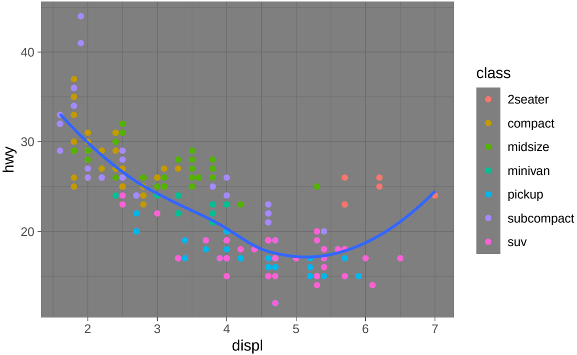
```
x_scale <- scale_x_continuous(limits = range(mpg$displ))
y_scale <- scale_y_continuous(limits = range(mpg$hwy))
col_scale <- scale_color_discrete(limits = unique(mpg$drv))

ggplot(suv, aes(displ, hwy, color = drv)) + geom_point() +
  x_scale + y_scale + col_scale
ggplot(compact, aes(displ, hwy, color = drv)) + geom_point() +
  x_scale + y_scale + col_scale
```



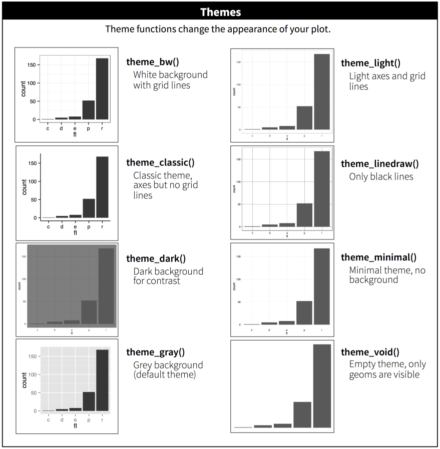


Themes

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme_dark()
```



ggplot2 default themes



Themes

Theme functions change the appearance of your plot.

theme_bw()
White background with grid lines

theme_light()
Light axes and grid lines

theme_classic()
Classic theme, axes but no grid lines

theme_linedraw()
Only black lines

theme_dark()
Dark background for contrast

theme_minimal()
Minimal theme, no background

theme_gray()
Grey background (default theme)

theme_void()
Empty theme, only geoms are visible

- More in add-on packages like ggthemes!