# VISUALIZACIÓN

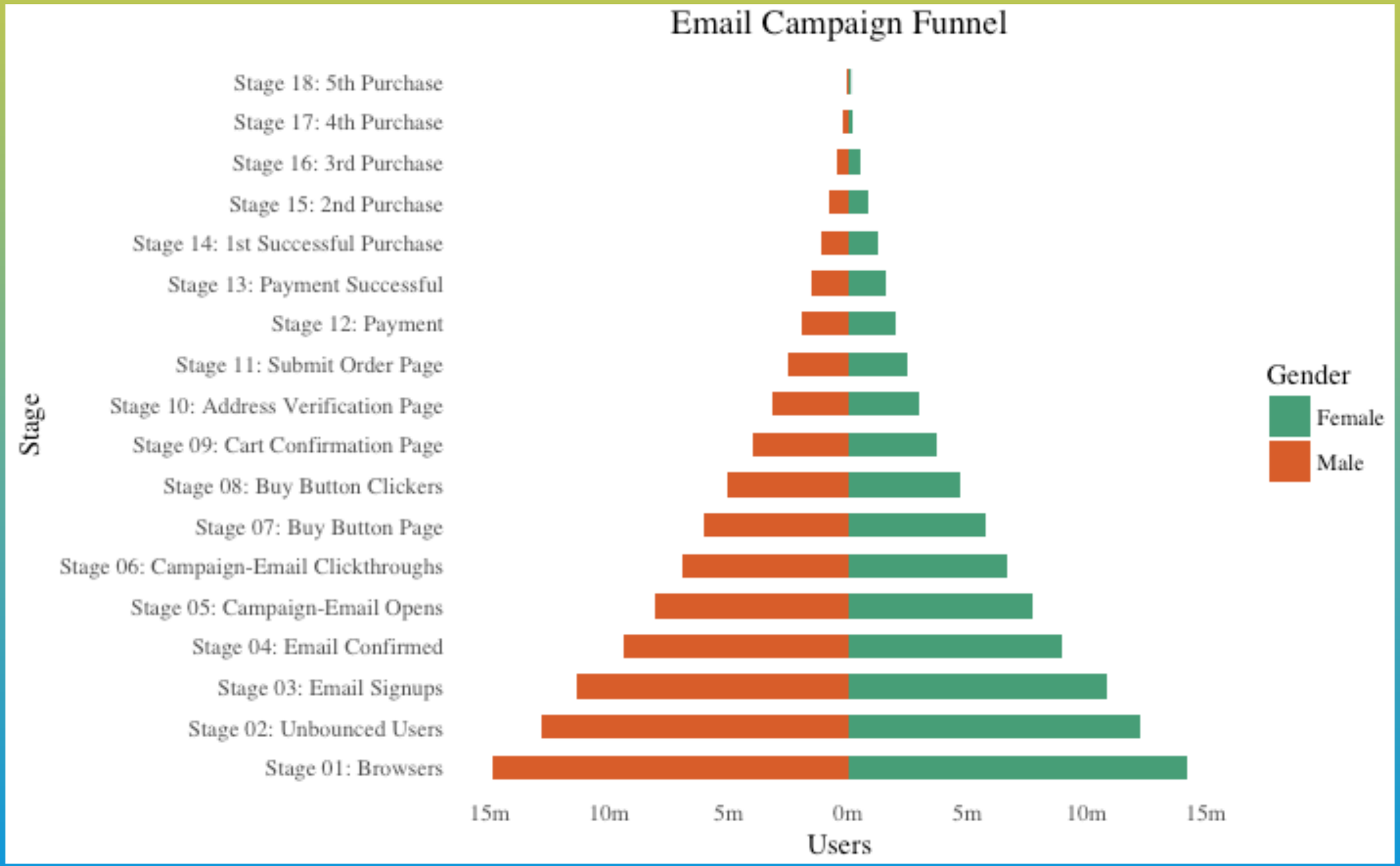
*Sebastián Niklitschek*

# INTRODUCCIÓN

"Una visualización simple y bien escogida, entrega más información al científico de datos que cualquier otra herramienta."

Sebastián Niklitschek

# ALGUNOS EJEMPLOS



# ALGUNOS EJEMPLOS

- https://shiny.rstudio.com/gallery/telephones-by-region.html
- Para más ejemplos, visitar:

http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html

# GGPLOT2

- Una de las librerías más elegantes y más versátiles;
- Sistema transferible a diversas aplicaciones;
- Implementa la llamada gramática gráfica. Leer:

http://vita.had.co.nz/papers/layered-grammar.pdf


# ALGUNOS REQUISITOS


- La “tidyverse” librería nos dará acceso a todas las herramientas gráficas de ggplot, así como también a los conjuntos de datos y funciones que usaremos en este módulo;

```
install.packages("tidyverse")
library(tidyverse)
```

# PRIMEROS PASOS

- A modo de ejemplo, intentemos responder a las siguientes preguntas:

¿Los automóviles con motores pequeños gastan menos combustible que los con motores grandes?

¿Cómo se ve la relación entre el tamaño del motor y su consumo? ¿Es lineal? ¿No lineal?

# MPG

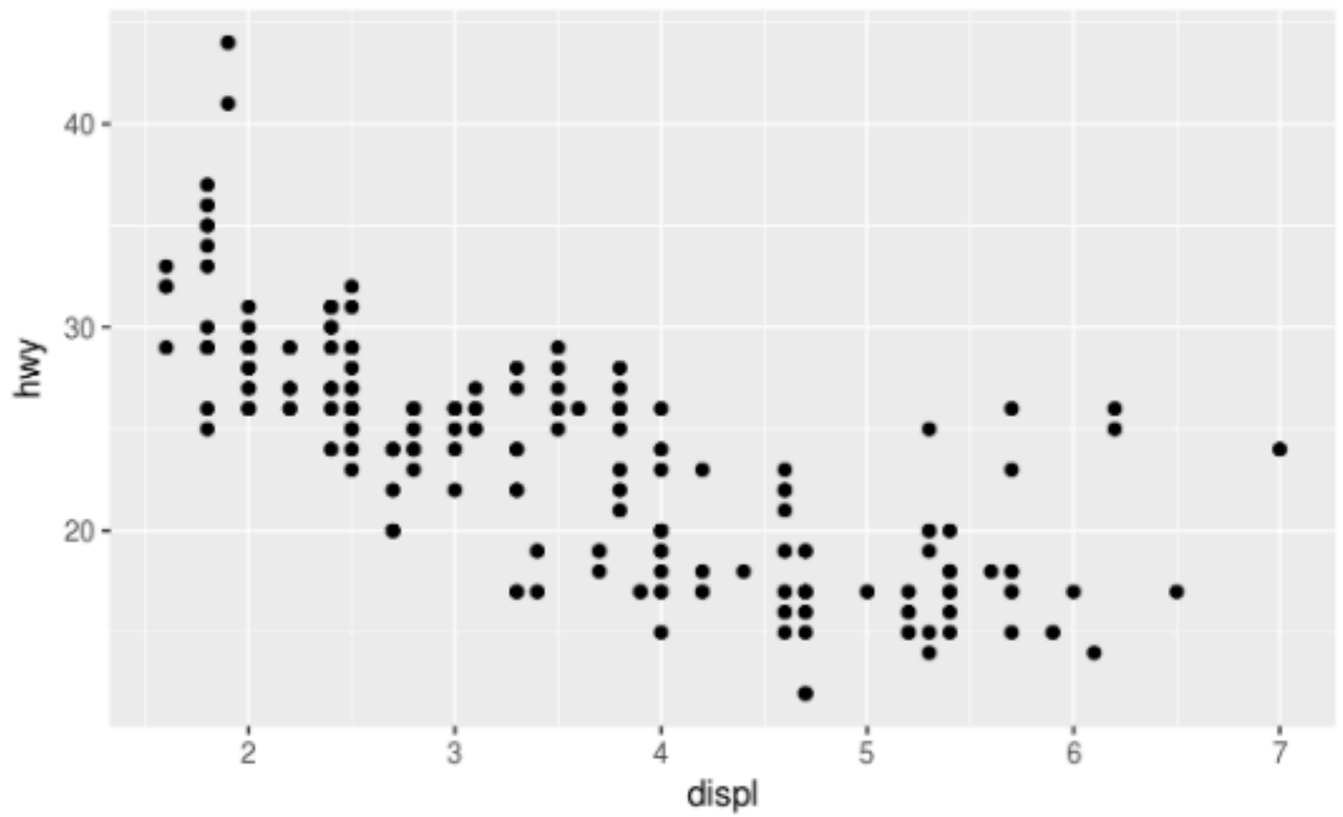
- Este conjunto de datos contiene información respecto a 38 modelos diferentes de automóviles recolectada en Estados Unidos.

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

# GRÁFICO

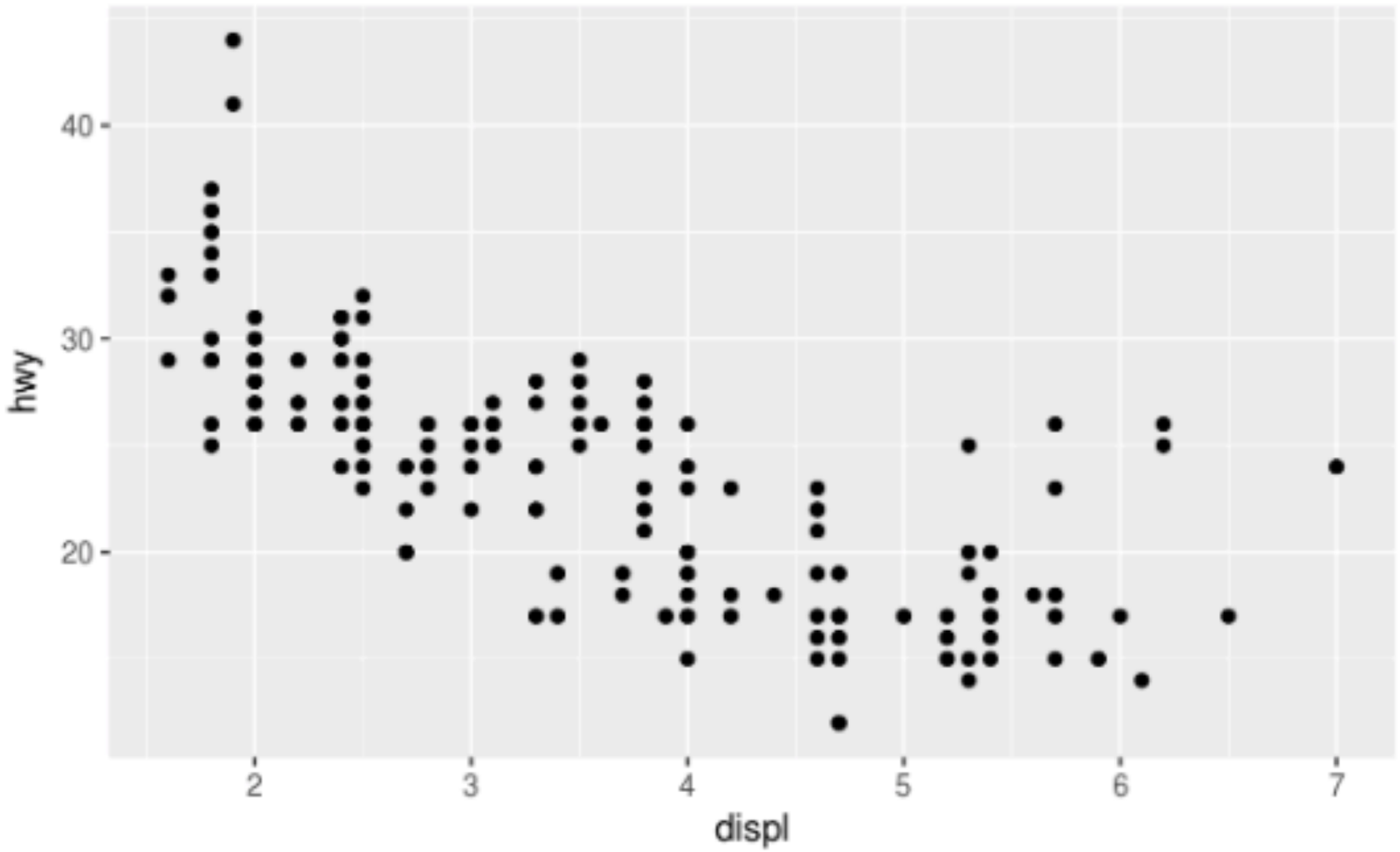
- Para crear el gráfico, pongamos la variable displ (tamaño del motor en litros) en el eje “x” y la variable hwy (eficiencia en carretera), en el eje “y”.

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



# ¿QUÉ PODEMOS OBSERVAR?

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



# ENTENDIENDO LO QUE HICIMOS

- Para crear un gráfico con ggplot2, lo que hicimos fue:
  - Usar la función ggplot(): Esta función crea un sistema de coordenadas al que luego podremos ir agregando más capas;
  - El primer argumento de la función ggplot() es el conjunto de datos a utilizar, así ggplot(data = mpg), crea un gráfico vacío;
- El gráfico se completa dibujando diferentes capas sobre la capa de base. La función geom_point() añade una capa de puntos sobre el gráfico, creando un gráfico de dispersión;
- ggplot2 viene con una serie de capas de tipo “geom”, que revisaremos.

# ENTENDIENDO LO QUE HICIMOS

- Cada función geom(), recibe sus argumentos mediante un mapeo. Este tipo de mapeo, define el cómo las variables en nuestro conjunto de datos se transforman en propiedades gráficas;
- El argumento de mapeo siempre está emparejado con “aes()”, y sus argumentos especifican qué variables mapear a cada eje. ggplot2 buscará los nombres asignados en aes(), dentro del conjunto de datos.

# UNA PLANILLA GENERAL

- Una planilla general quedaría entonces:

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

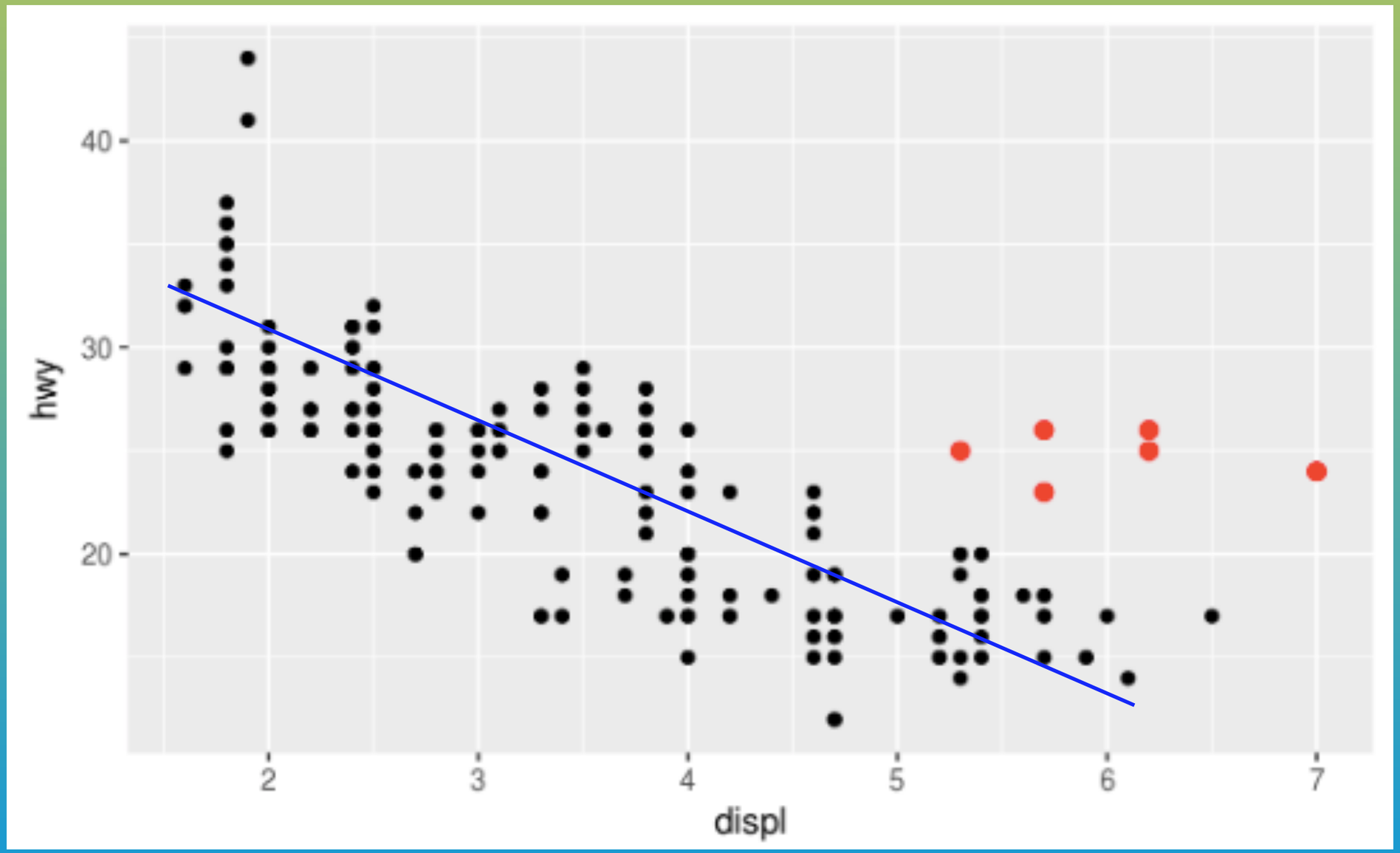
- Pasaremos algún tiempo descubriendo como completar y extender esta planilla para dibujar diferentes tipos de gráficos.

# MAPEOS DE ESTÉTICA

"El verdadero valor de una imagen aparece cuando nos obliga a notar algo que no esperábamos que estuviera ahí."

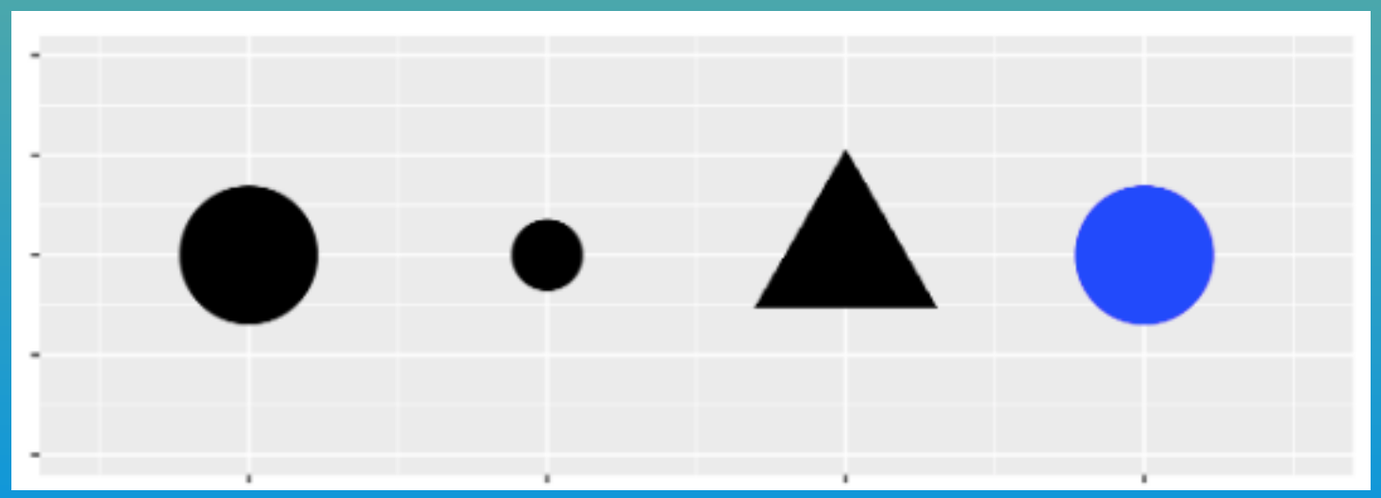
# VOLVIENDO AL EJEMPLO

- ¿Qué observa en la gráfica siguiente?



# AÑADIENDO VARIABLES

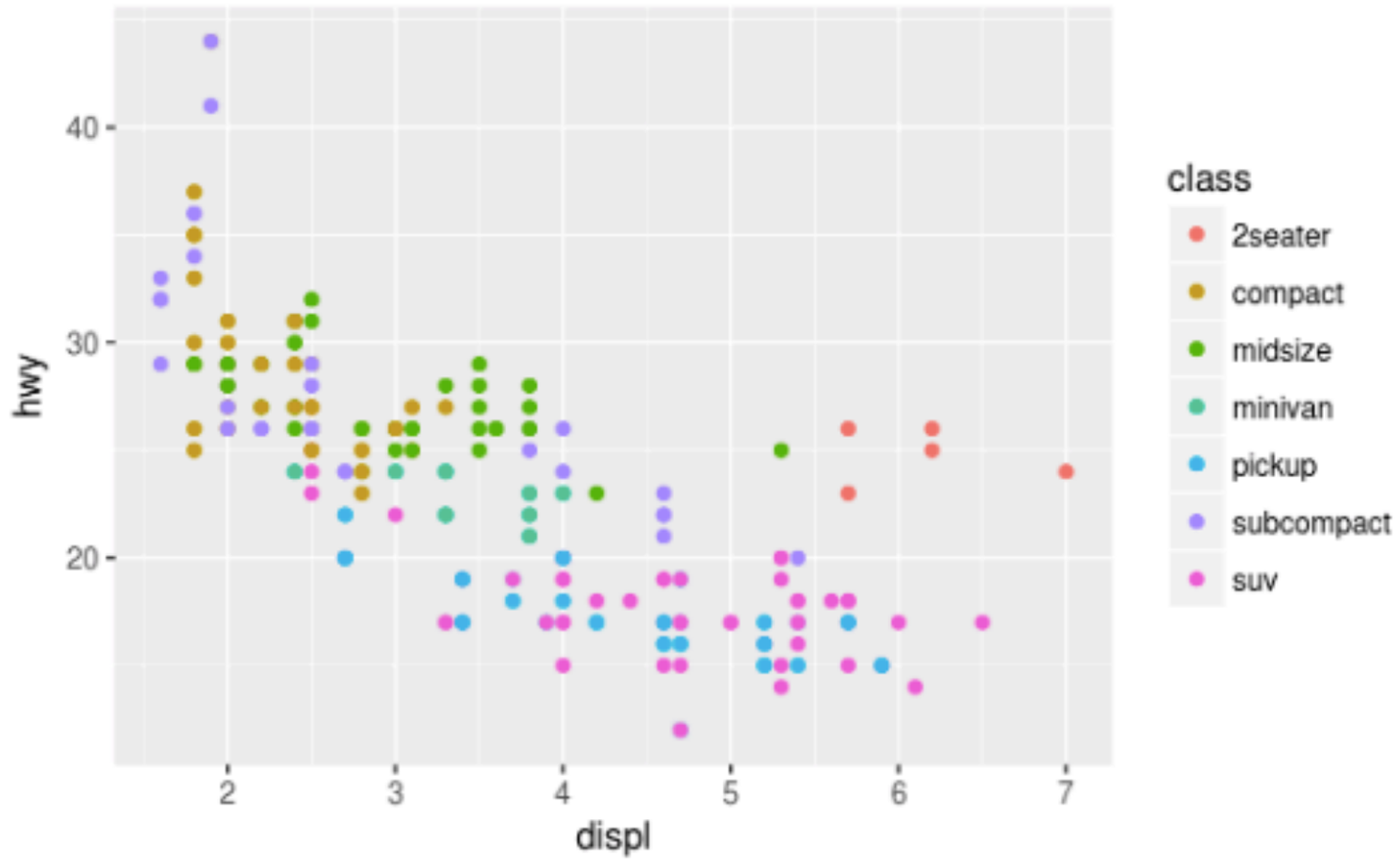
- Podemos agregar otra variable a un gráfico bi-dimensional, por ejemplo, la variable “class”, si la pasamos a la función como argumento estético;
- Los argumentos estéticos son propiedades de los objetos en el gráfico. Pueden incluir cosas como tamaño, forma o color;
- Llamaremos “niveles” a las propiedades estéticas.



# ENTENDIENDO EL COMPORTAMIENTO EXTRAÑO

- Agreguemos la variable “class” al mapeo “colors” para revelar la clase de cada uno de nuestros puntos;

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



# ENTENDIENDO LO OCURRIDO

- Para mapear un nivel estético a una variable, simplemente debemos asignar al nombre del nivel, la variable dentro de la función aes().

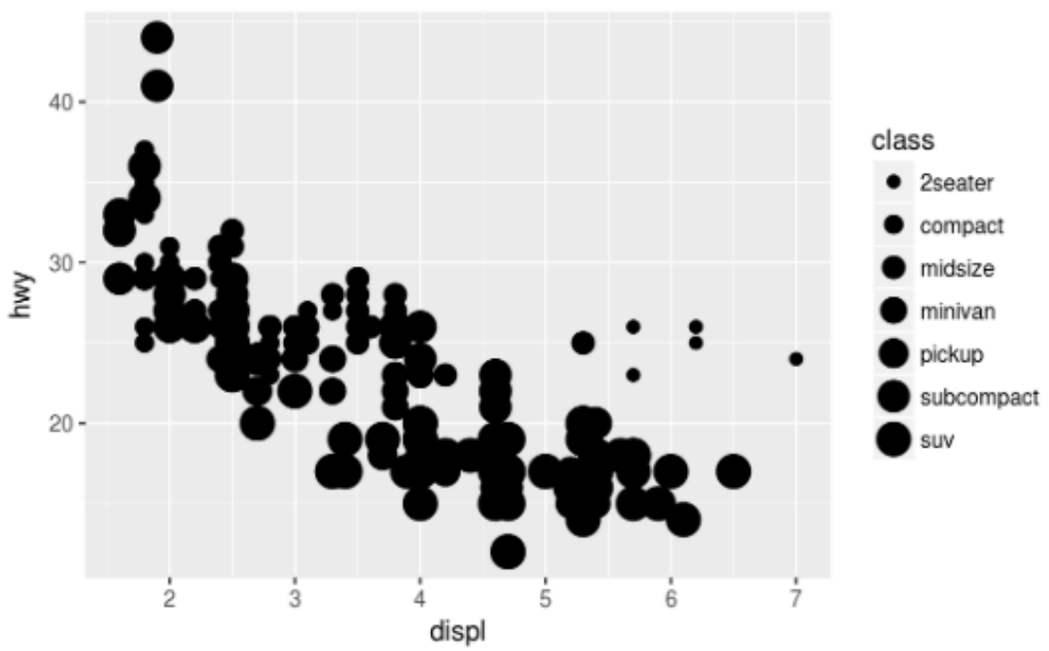
- ggplot2, asignará automáticamente un nivel único para la estética elegida para cada valor contenido en la variable asignada;

- ggplot2 agregará además una leyenda explicando qué nivel corresponde a qué valor.

# NOTA

- Por supuesto, podríamos hacer asociado la variable “class” a otro nivel estético. Por ejemplo, “size”. En este caso, el tamaño de cada punto representará la clase a la que pertenece el vehículo;
- ¿Qué problema podríamos encontrar?

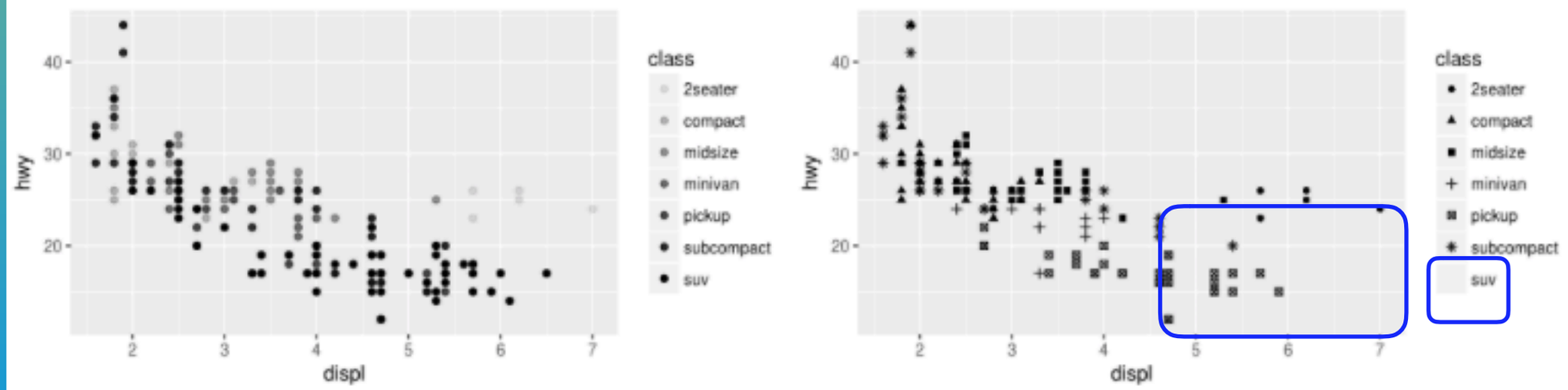
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```



# MÁS POSIBILIDADES

- Podríamos también haber utilizado otros niveles estéticos. Por ejemplo haber asignado la variable “class” a las niveles “alpha” o “shape”.

```r
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))
```



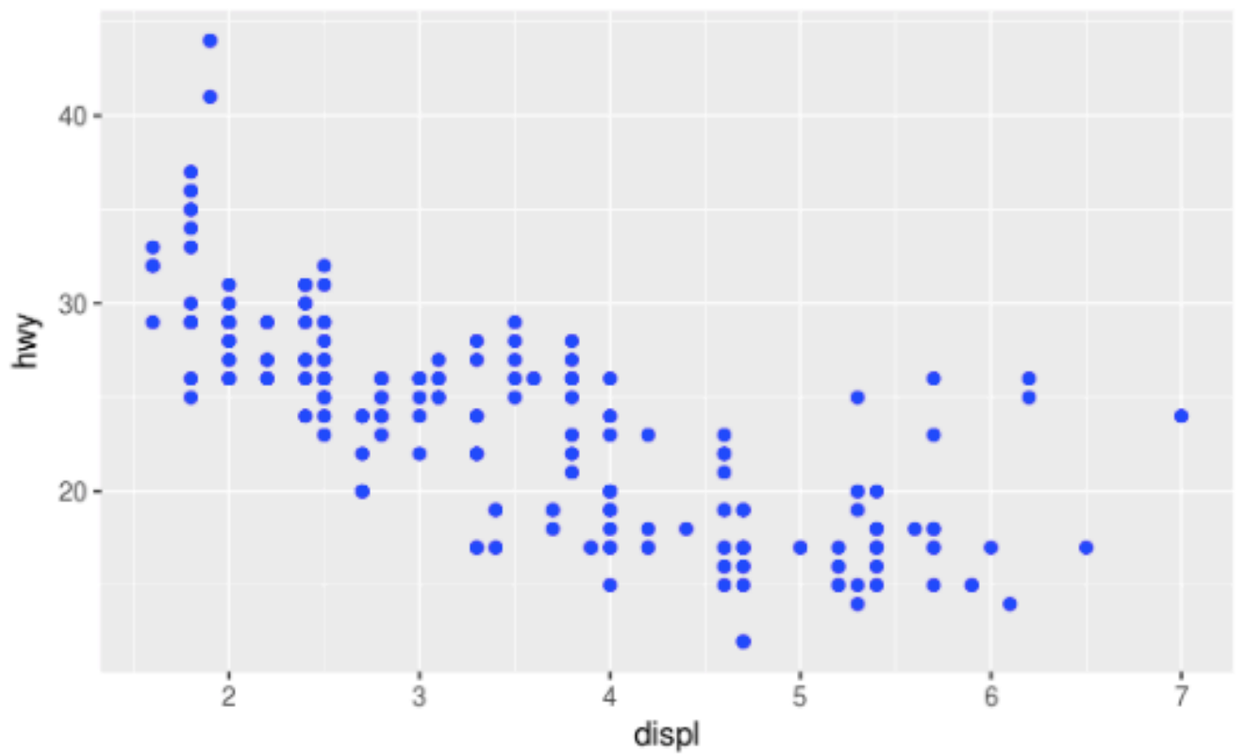
# OBSERVACIONES

- Para cada nivel estético, la función aes() nos permite asociar el nivel con una variable en el conjunto de datos;
- Esta función reúne todos los mapeos estéticos. Notemos que la posición de los puntos son niveles estéticos. En general, estos son propiedades visuales que pueden ser mapeadas a variables con la finalidad de desplegar información visual contenida en los datos;
- Una vez realizado el mapeo, la librería ggplot2 se hace cargo del resto, por ejemplo, elige la escala adecuada para utilizar en cada nivel, crea leyendas para explicar lo niveles de cada rasgo, crea etiquetas en los ejes para referenciar cada variable.

# FIJANDO PROPIEDADES MANUALMENTE

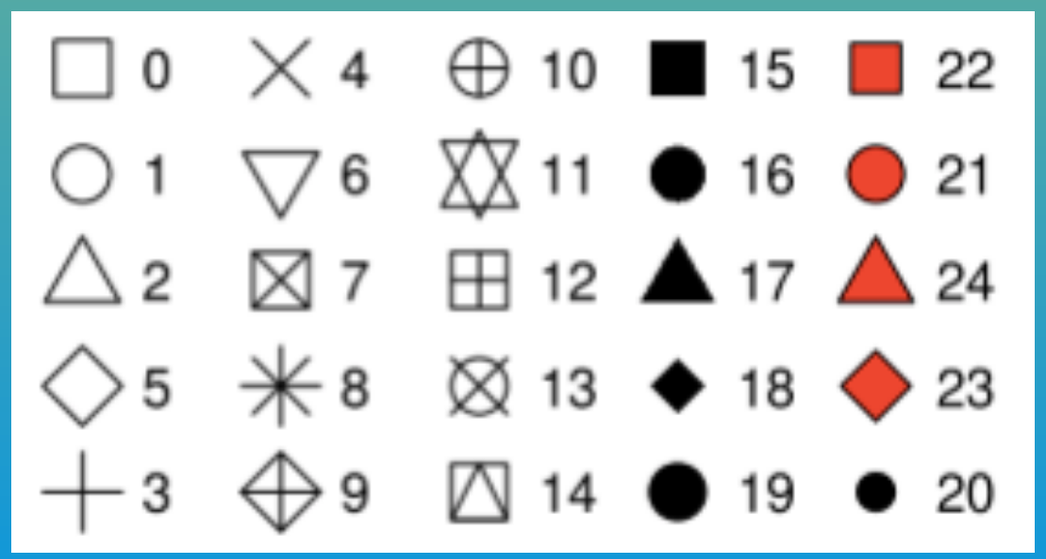
- Es posible fijar a algunas propiedades manualmente, evitando así que ggplot2 tome las decisiones;
- Por ejemplo, si quisiéramos que todos los puntos sean graficados en azul:

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



# OTRAS PROPIEDADES

- Para fijar niveles estéticos manualmente, éstos deben ser pasados directamente como argumentos a la función geom(), es decir, “fuera” de aes();
- Para ello, debemos escoger la codificación estándar de R, por ejemplo, los colores están codificados en cadenas de caracteres, el tamaño de los puntos en mm y la forma de los puntos mediante un número entero:



# NOTA

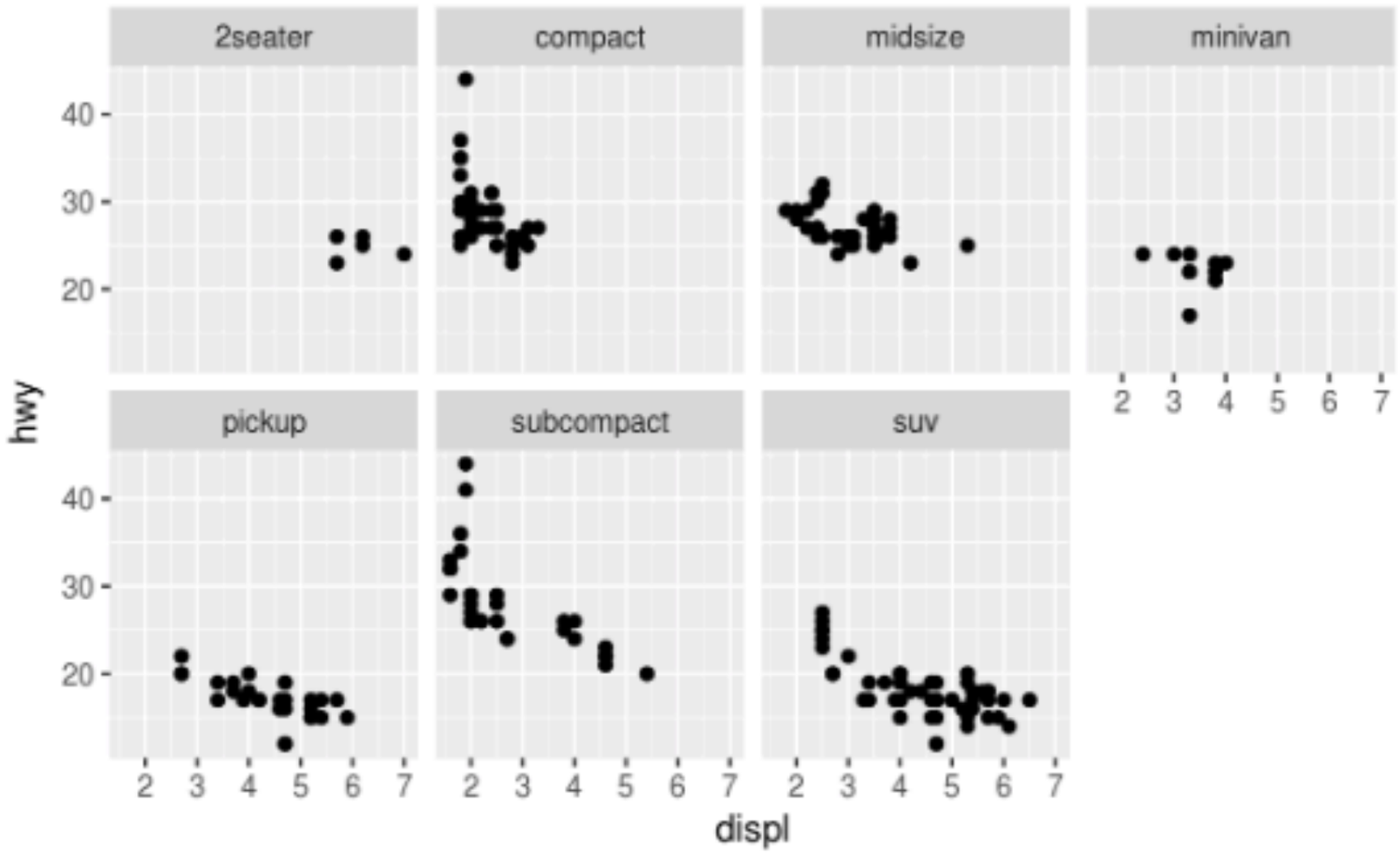
- Notemos que podríamos tener la impresión de que los tipos de puntos en la imagen anterior se repiten. Este es efectivamente el caso, las diferencias radican en el uso de los niveles color y fill.

# FACETAS

- Hemos visto como a través de niveles estéticos, es posible agregar nuevas variables en un gráfico bi-dimensional. Esto podría traer un problema, por ejemplo, si el número de puntos a graficar es muy grande, podría ser que los diferentes colores o tipos de puntos no se distingan bien;
- Otra posibilidad es dividir los datos utilizando subplots que desplegar subconjuntos de los datos;
- Para esto, usamos la función facet_wrap(). Su primer argumento corresponde a una fórmula, la que debe ser creada con "~" seguido por el nombre de una variable  discreta.

# FACETAS

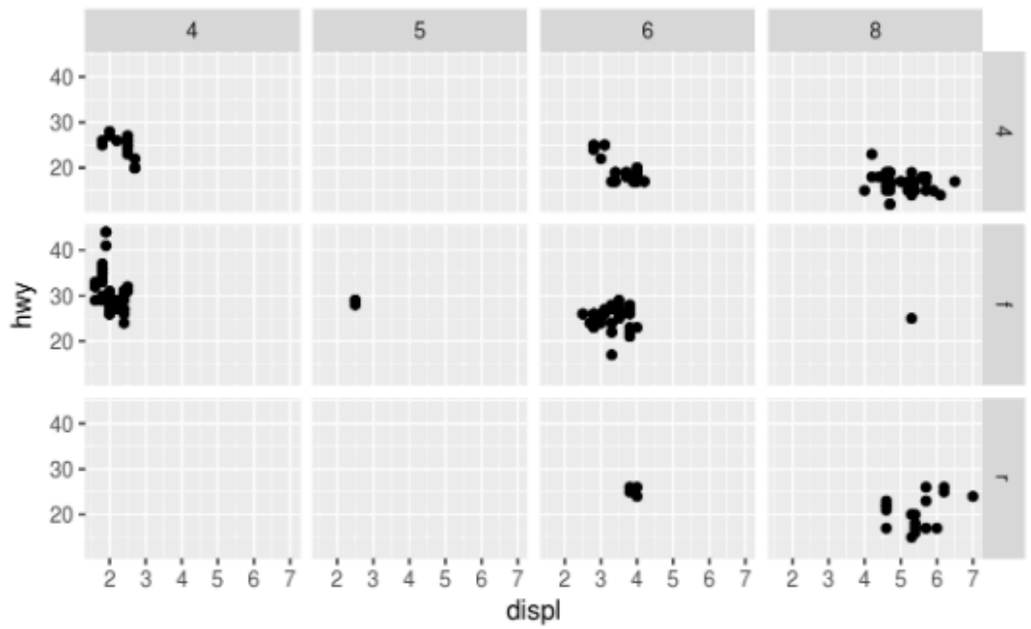
```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```



# FACETAS

- Podemos realizar la división con respecto a dos variables también, para ello, utilizamos la función facet_grid(). Su primer argumento es también una fórmula, esta vez, la fórmula debe contener dos variables categóricas separadas por el símbolo "~".

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

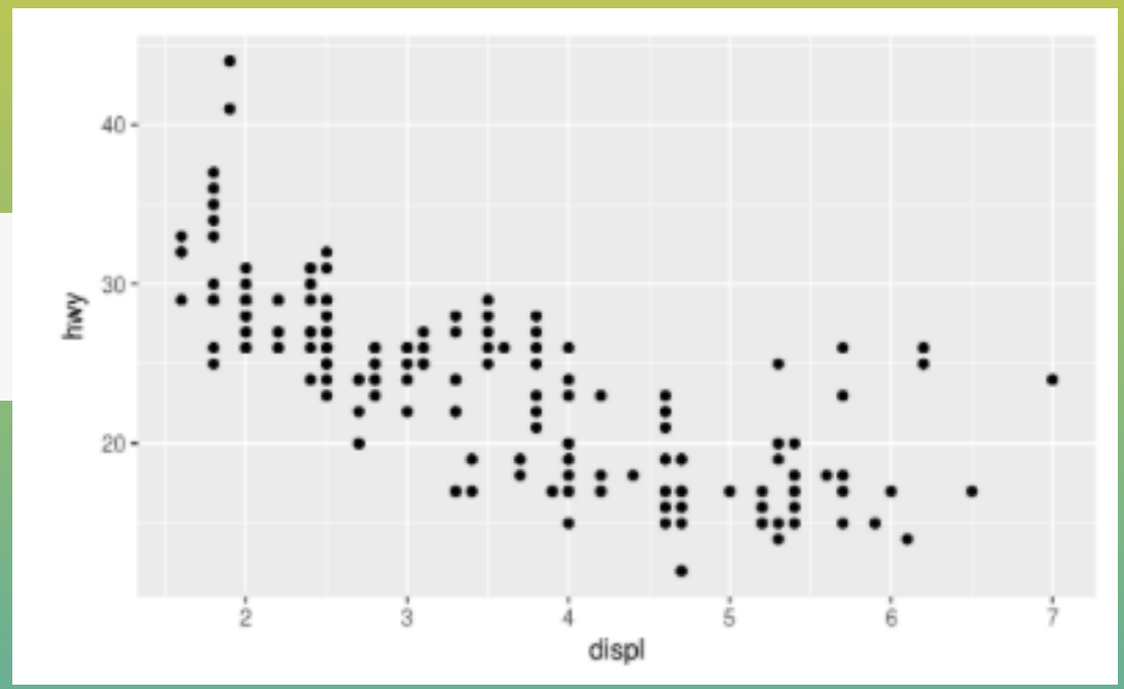


# OBJETOS GEOMÉTRICOS

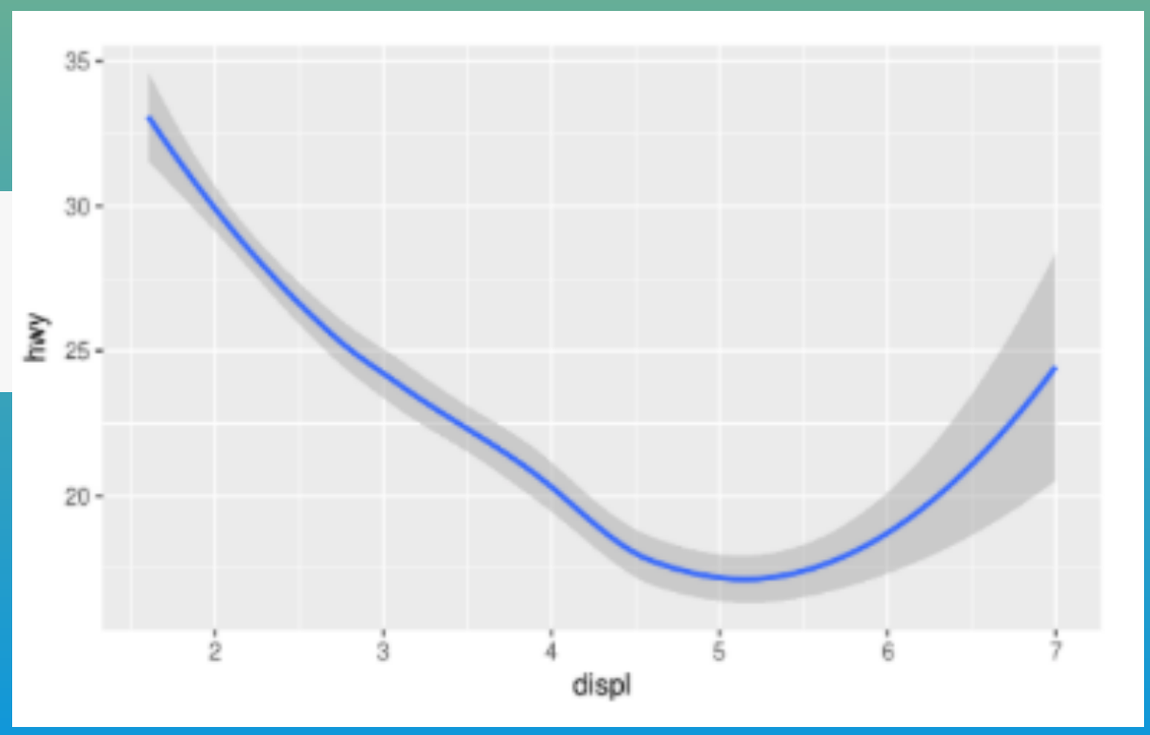
- geom hace referencia a un objeto geométrico que es utilizado en un gráfico para representar datos;
- Usualmente hacemos referencia a los gráficos mediante sus atributos geométricos, por ejemplo los gráficos de barras son geom bar, los gráficos de líneas usan geom line, los gráficos de caja o boxplots utilizan geom boxplot, etc.
- Es simple fijar el atributo geométrico del gráfico, basta simplemente cambiar la función geom que adicionamos a ggplot().

# EJEMPLOS

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




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



# NOTA IMPORTANTE

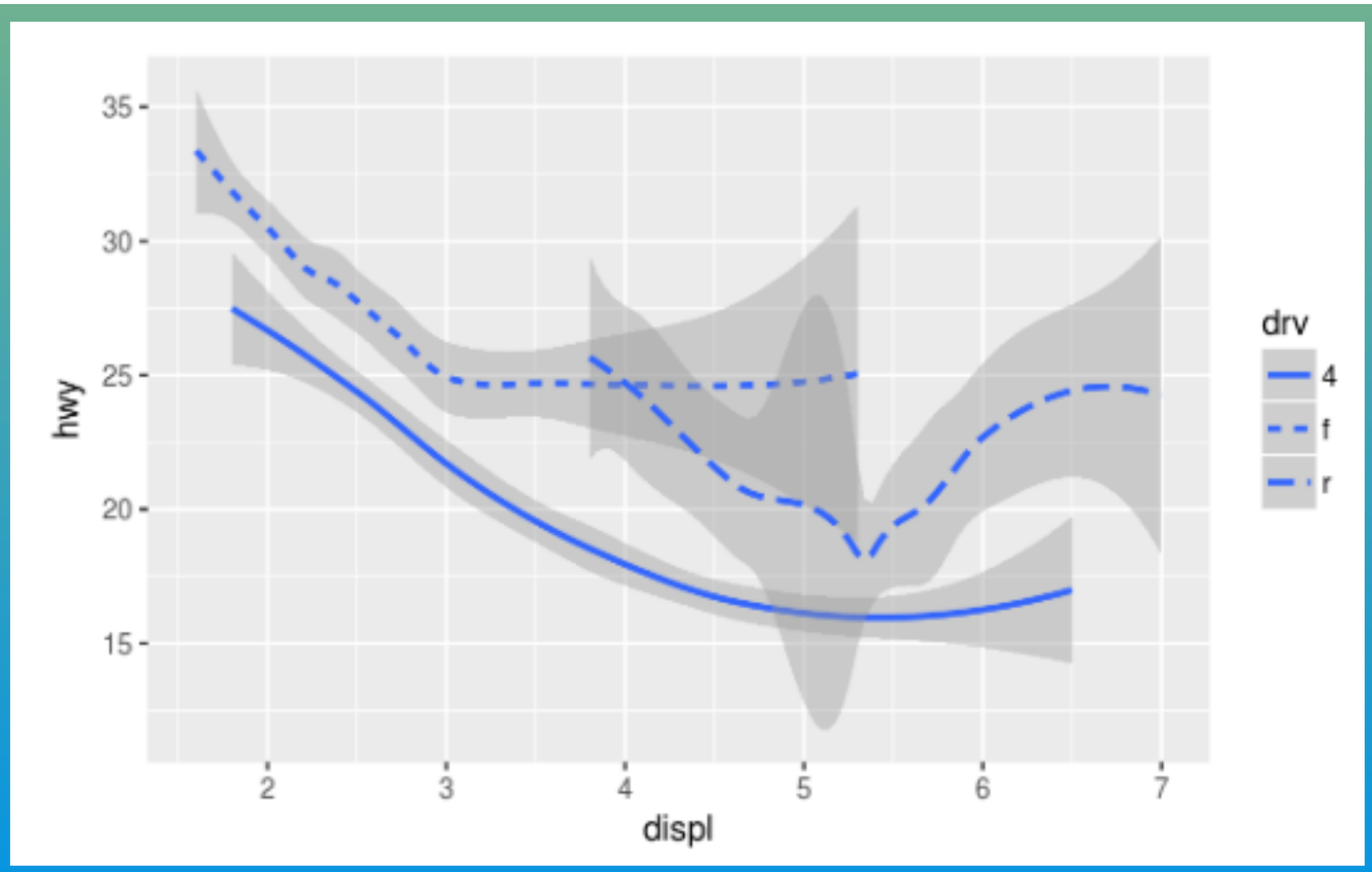
- Cada función de tipo geométrico en ggplot2 utiliza un mapeo como argumento, sin embargo, debemos tener cuidado puesto que no todo argumento funciona con cualquier “geom”;

- Por ejemplo, podemos fijar la forma “shape” de un point pero no de una línea.

# EJEMPLO

- geom_smooth() dibujará una línea diferente, utilizando un tipo diferente de línea, para cada valor único que sea pasado mediante el mapeo a la característica “linetype”.

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



# ¿CUÁNTOS HAY?

- ggplot2 incluye sobre 30 funciones de tipo geom;
- Muchas de las funciones geom, utilizan un objeto geométrico para desplegar muchas filas contenidas en los datos, en este caso podemos utilizar la característica estética “groups” para dibujar los datos separados utilizando una variable categórica;

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

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

# AGRUPAMIENTO AUTOMÁTICO

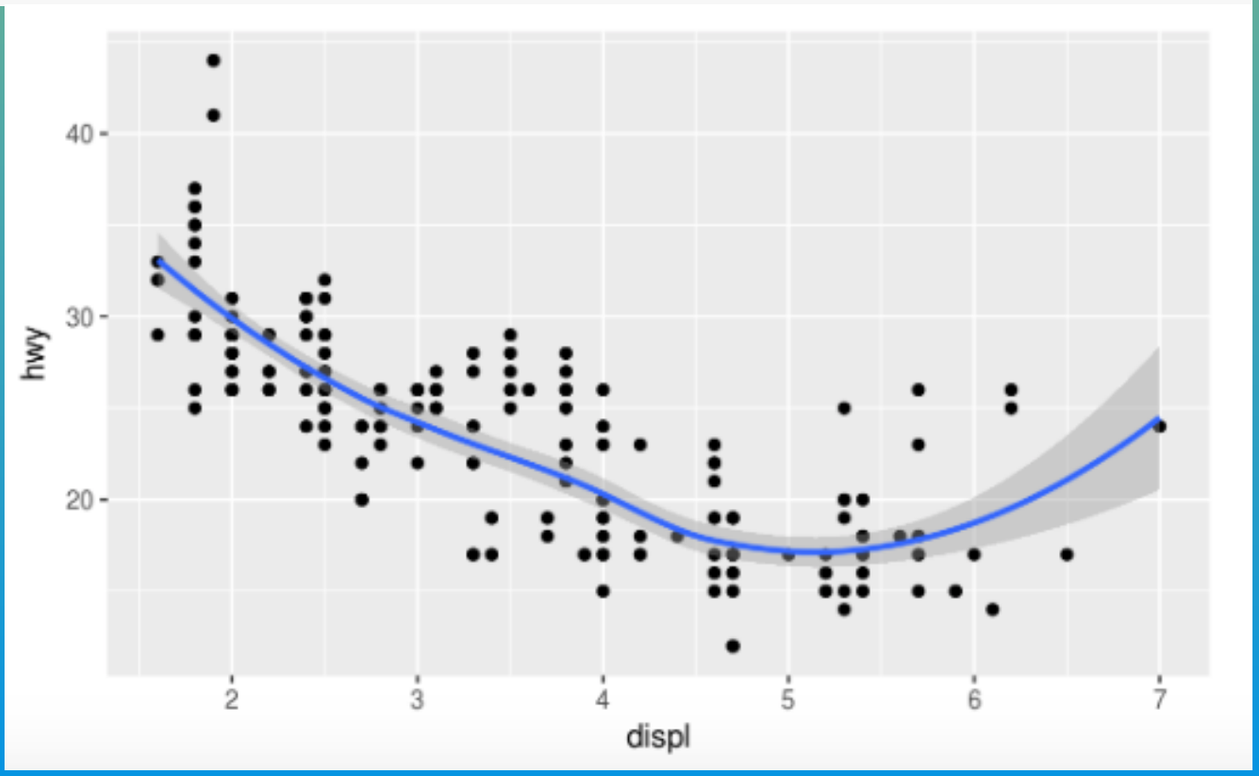
- ggplot2 hará este agrupamiento de manera automática si asociamos algún atributo estético a una variable categórica;
- Es mejor confiar en esta funcionalidad dado que agrega automáticamente una etiqueta a los datos.

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

# MÚLTIPLES ATRIBUTOS GEOMÉTRICOS

- Es posible dibujar varios atributos geométricos en la misma gráfica, para ello, sólo es necesario agregar múltiples funciones de tipo geom a ggplot()

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



# OBSERVACIÓN

- Un problema que surge es que la sintaxis introduce elementos duplicados en el código, por ejemplo, supongamos que quisiéramos cambiar el eje “y” por cty en lugar de hwy:

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

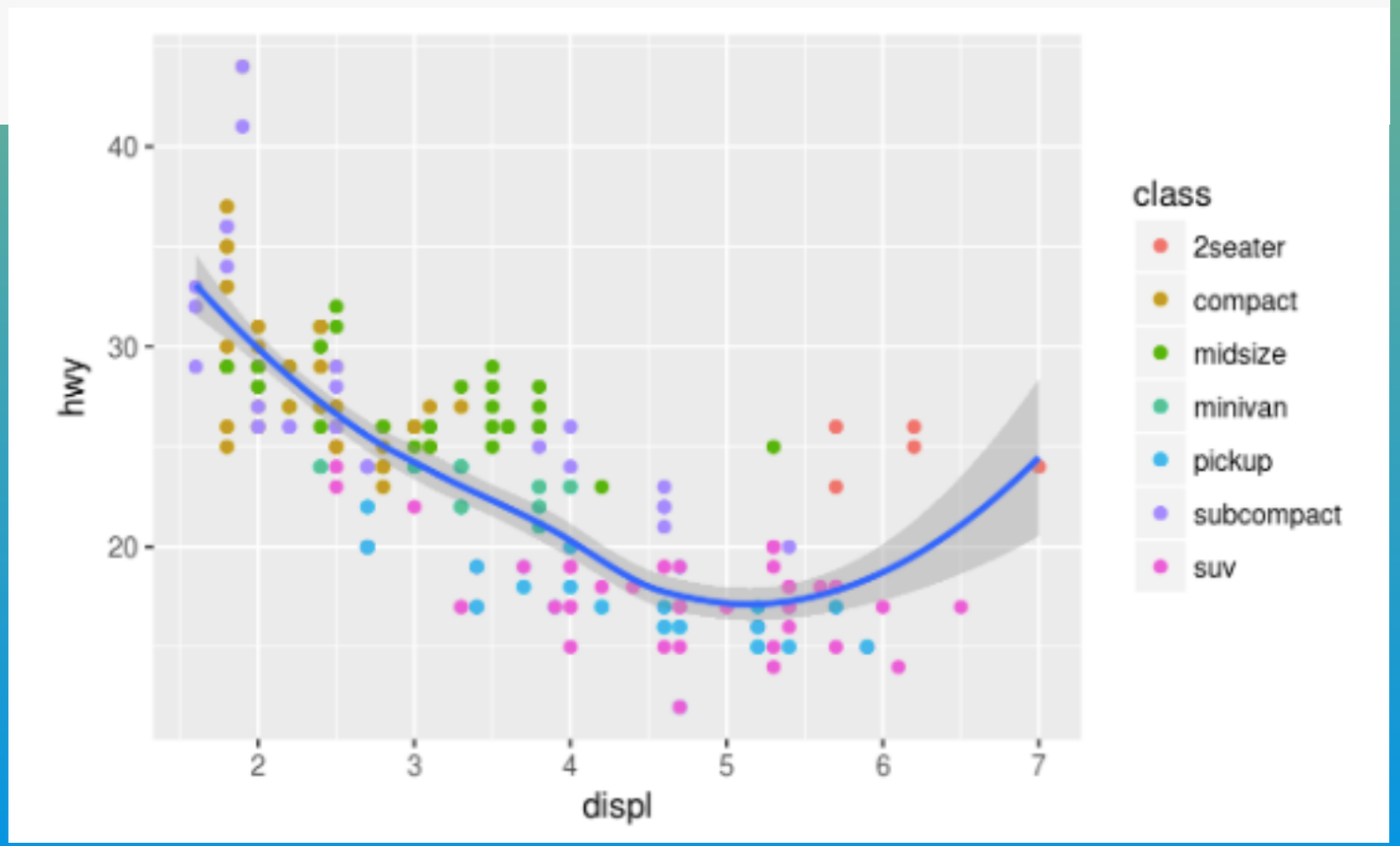
- Podemos evitar este tipo de duplicados simplemente agregando el mapeo a la función ggplot():

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

# ESTÉTICA LOCAL

- El mapeo agregado a la función ggplot() funcionará como un mareo global en la figura, sin embargo, si agregamos mapeos a un objeto geométrico, ggplot usará esos atributos sólo en esa capa:

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



# DATOS LOCALES

- Podemos utilizar la misma lógica para especificar diferentes datos para cada capa. Por ejemplo, el siguiente código despliega sólo un subconjunto de los datos contenidos en el arreglo mpg, aquellos que sólo corresponden a los autos de clase “subcompact”:

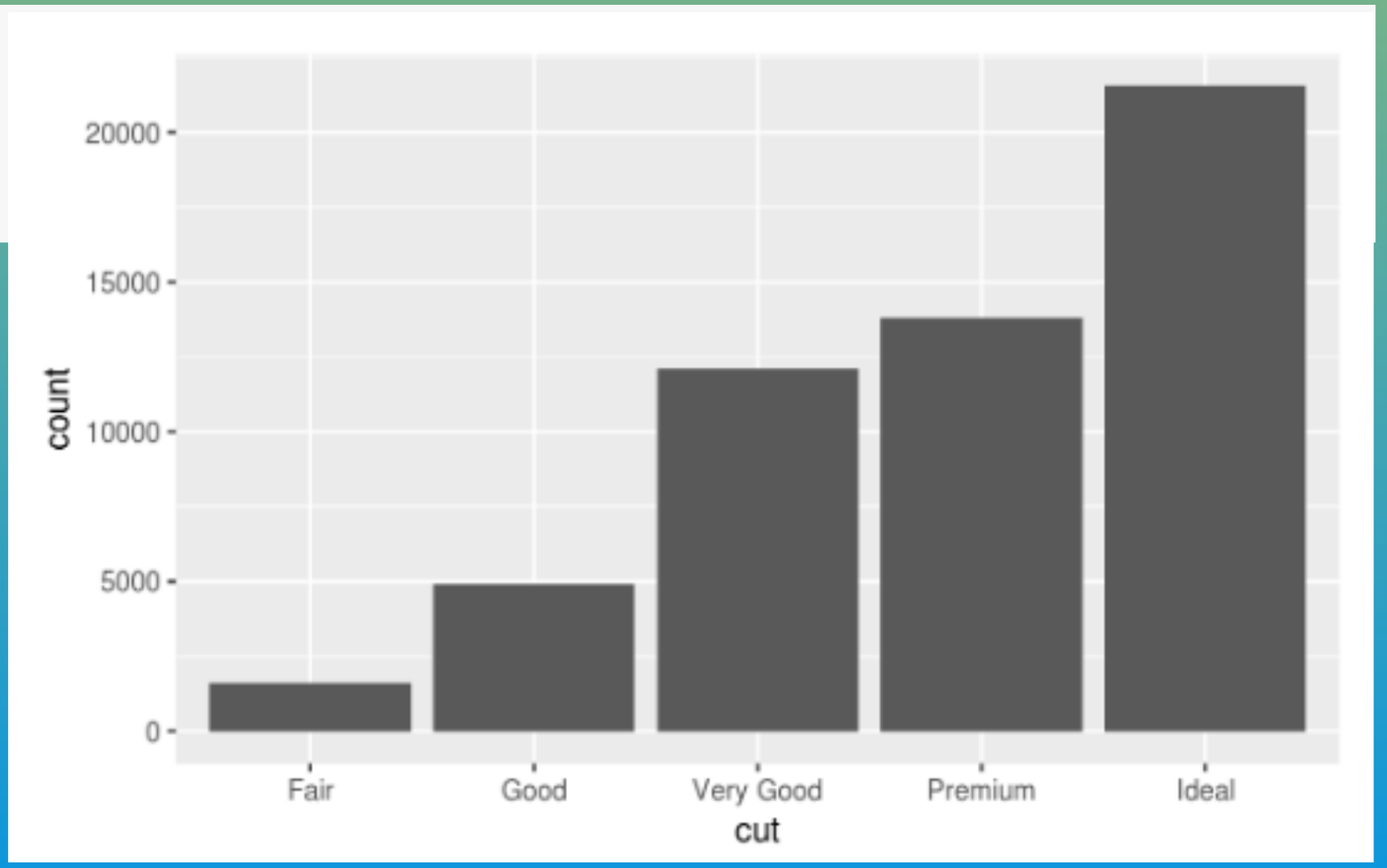
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(data = filter(mpg, class == "subcompact"), se = FALSE)
```

- Esta asignación sobre-escribe, sólo para esta capa, el mapeo argumento de ggplot().

# TRANSFORMACIONES ESTADÍSTICAS

- El conjunto de datos “diamonds” incluye información de alrededor de 54000 diamantes.
- Comencemos estudiando los gráficos de barra, la función geom_bar():

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



# OBSERVACIÓN IMPORTANTE

- En el gráfico anterior, desplegamos la variable cut en el eje “x”. En el eje “y”, desplegamos el conteo de diamantes que caen dentro de cada categoría en “cut”. ¡Esta variable no está en los datos originales!
- Varios tipos de gráfico, como los gráficos de dispersión, grafican los valores presentes en el conjunto de datos, en cambio otros, como los gráficos de barra, calculan nuevos valores para incluirlos en el gráfico.

# OBSERVACIÓN IMPORTANTE

- Los gráficos de barra, histogramas y polígonos de frecuencia, agrupan los datos y grafican el número de registros que caen dentro de cada categoría;
- Los suavizamientos (smooth), ajustan un modelo a los datos y luego grafican predicciones a partir del modelo;
- Los gráficos de caja o boxplot, calculan un resumen robusto de la distribución empírica de los datos y despliegan una caja que resume esta información.

# STATS

- El algoritmo utilizado para calcular nuevos valores para generar un gráfico se denomina “stat”, en el caso de los boxplot, funciona así:

*geom_bar() comienza con el conjunto de datos completo*

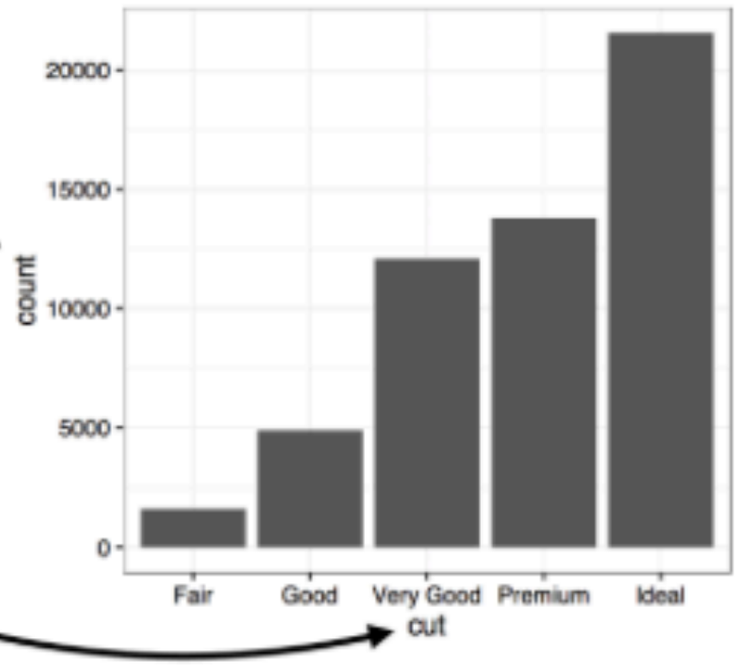
| carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|
| 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 0.21 | Premium | E | SI1 | 59.8 | 61 | 326 | 3.89 | 3.84 | 2.31 |
| 0.23 | Good | E | VS1 | 56.9 | 65 | 327 | 4.05 | 4.07 | 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()

*geom_bar() transforma los datos mediante el estadístico “conteo”*

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

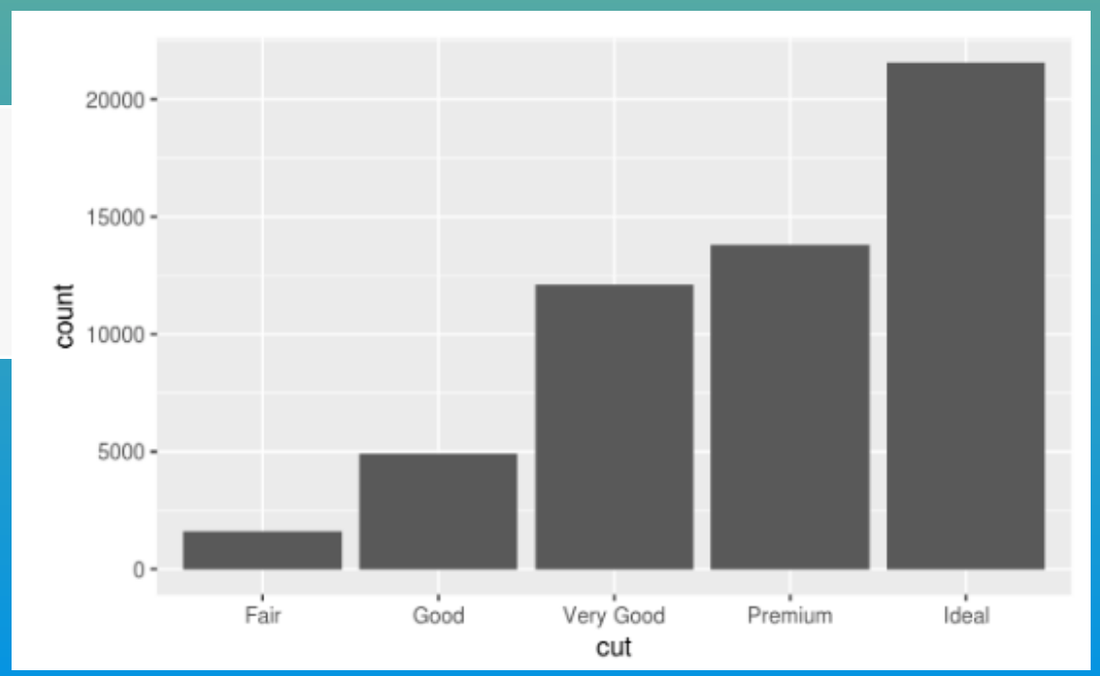
*geom_bar() utiliza la nueva tabla generada para construir el gráfico.*



# STATS

- Es posible conocer qué estadístico utiliza una función geom, mirando cuál es el valor por defecto para el argumento stat;
- Ejemplo: ?geom_bar muestra que el valor por defecto de stat es “count”, lo que quiere decir que geom_bar() utiliza stat_count;
- Es posible utilizar las funciones de tipo geom o las estadísticas, stat, de forma intercambiable;

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



# NOTA

- Lo anterior funciona porque cada función de tipo geom, tiene un stat por defecto y cada stat, tiene por defecto una función de tipo geom;
- En general podríamos vernos tentados a cambiar los valores por defecto en una de las tres situaciones siguientes:
  - Si queremos cambiar el valor por defecto de stat, por ejemplo, si el conteo ya se encontrara contenido en los datos, debemos cambiar el stat en geom_bar() a “identity”;
  - Podríamos querer cambiar el mapeo por defecto de las variables transformadas a argumentos estéticos, por ejemplo, podríamos querer dibujar un gráfico de barras con las proporciones en lugar del conteo;

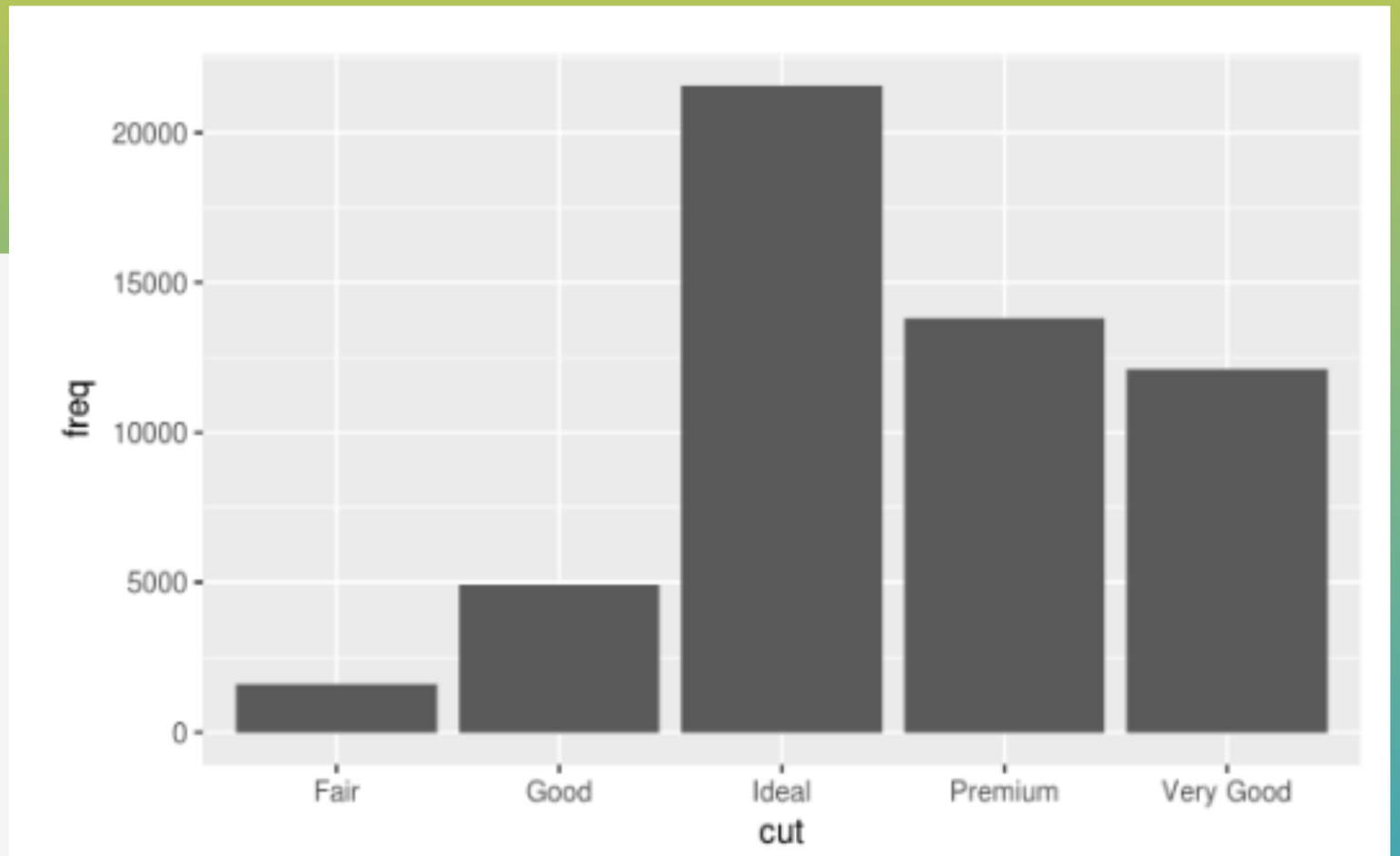
# CONTINUACIÓN NOTA

- Podríamos querer precisar con más detalle la transformación que será graficada para cada valor de “x”.
- ggplot2 tiene más de 20 estadísticos que podemos utilizar. Cada stat es una función, así que podemos acceder al detalle de su uso de la manera habitual, esto es, utilizando “?”.

# EJEMPLOS DE LOS TRES CASOS

➤ Caso 1:

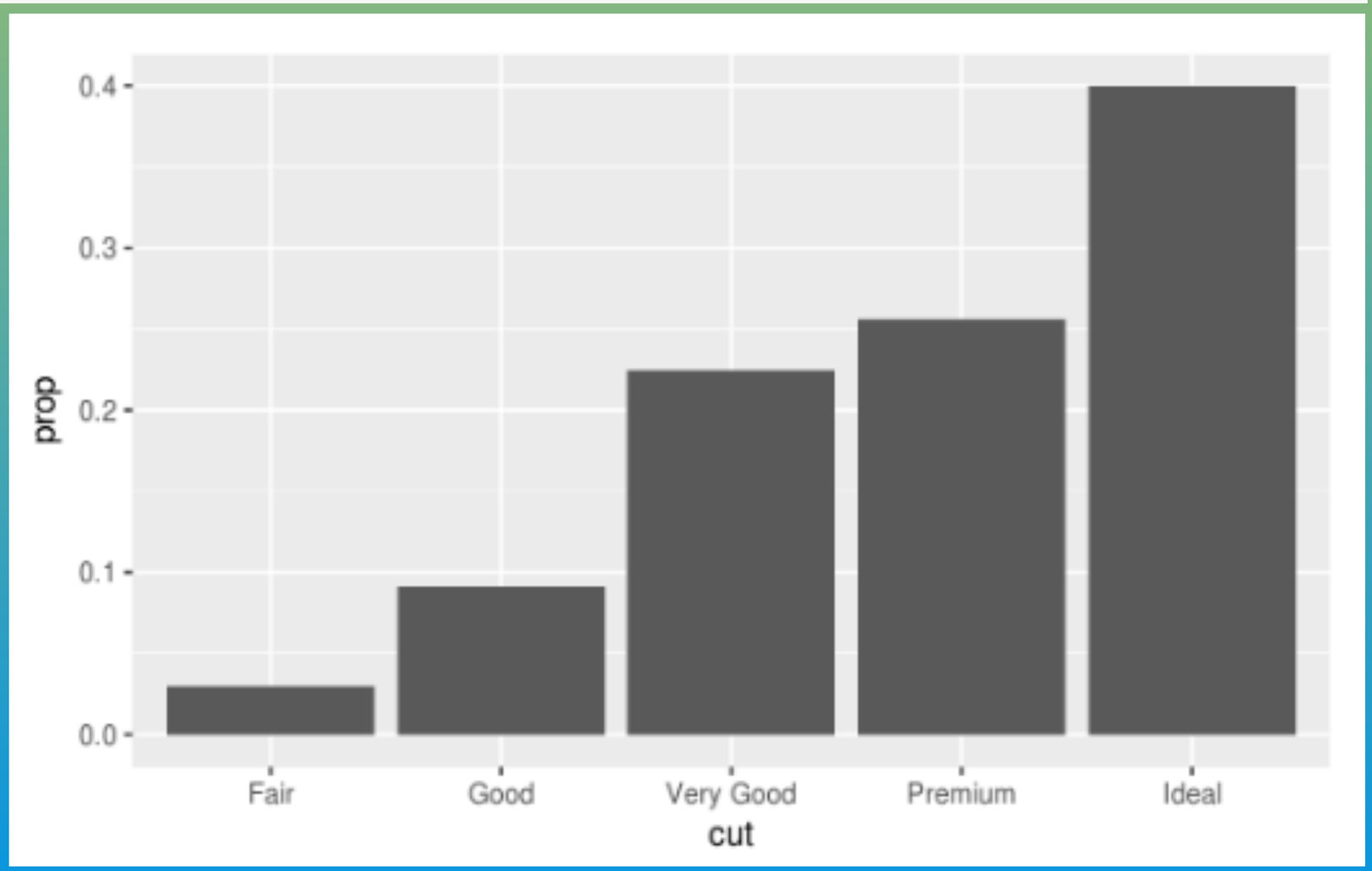
```
demo <- tribble(
  ~cut,         ~freq,
  "Fair",       1610,
  "Good",       4906,
  "Very Good",  12082,
  "Premium",    13791,
  "Ideal",      21551
)

ggplot(data = demo) +
  geom_bar(mapping = aes(x = cut, y = freq), stat = "identity")
```



# EJEMPLOS DE LOS TRES CASOS

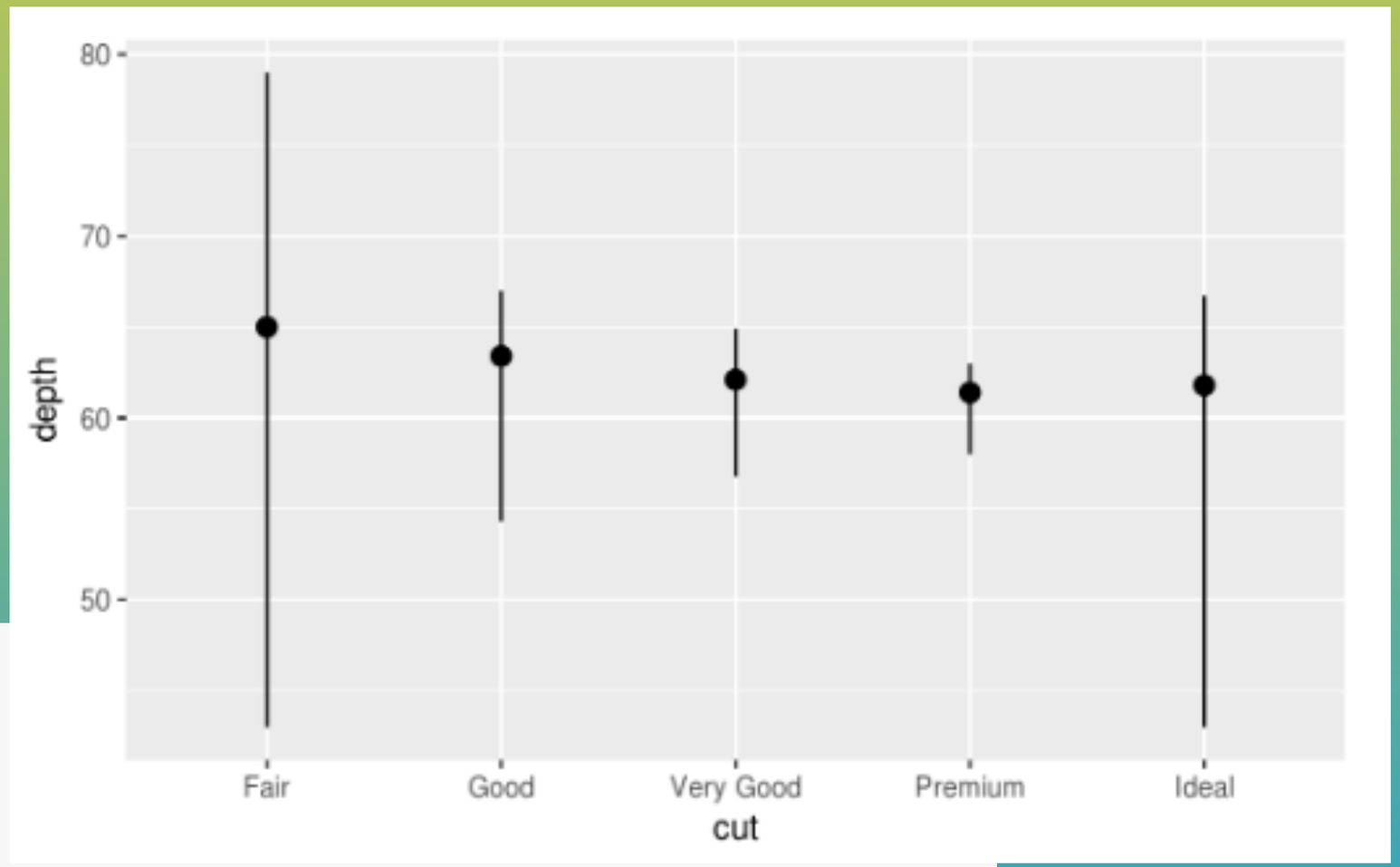
- Caso 2:

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



# EJEMPLOS DE LOS TRES CASOS

- Caso 3:

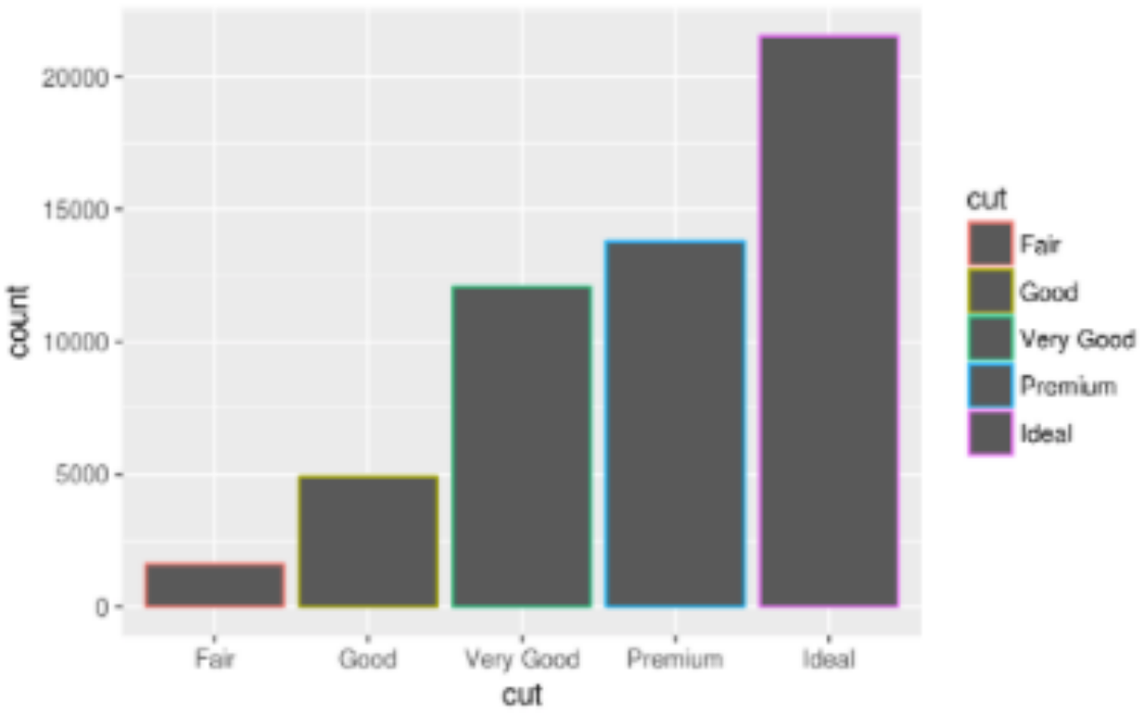


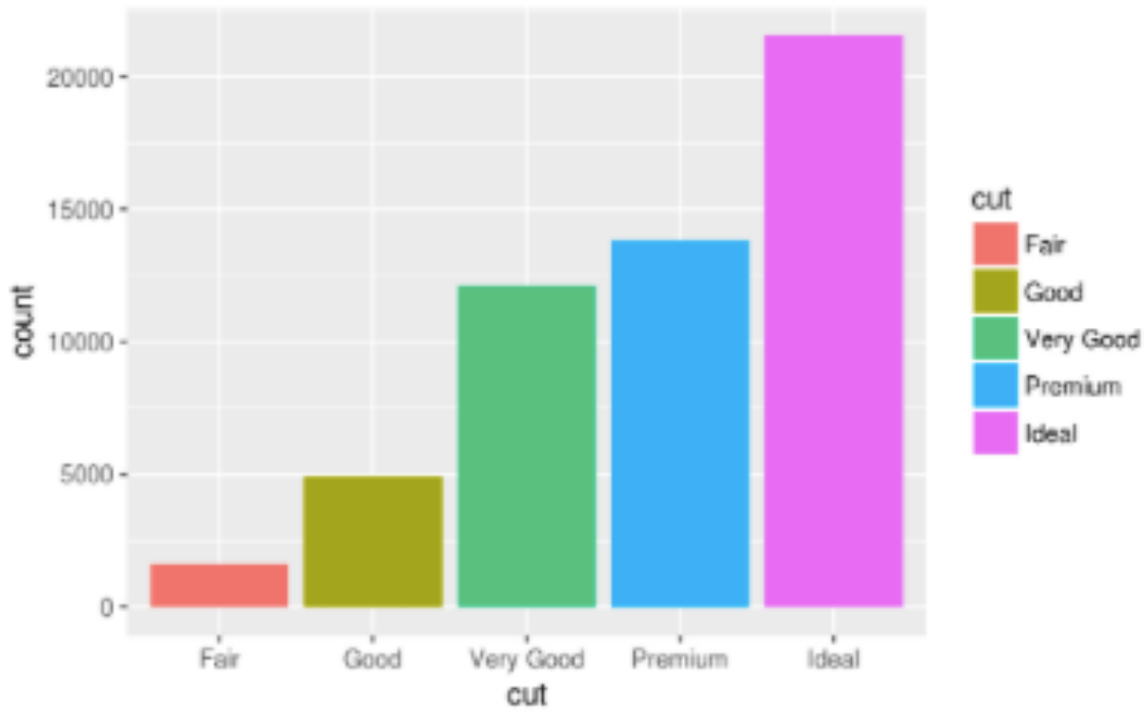

```r
ggplot(data = diamonds) +
  stat_summary(
    mapping = aes(x = cut, y = depth),
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
  )
```

# ARGUMENTOS DE POSICIÓN

- Existen algunos argumentos mágicos asociados con los gráficos de barras. Por ejemplo, podemos colorear las barras, usando la estética “colour”, o aún mejor, “fill”;

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

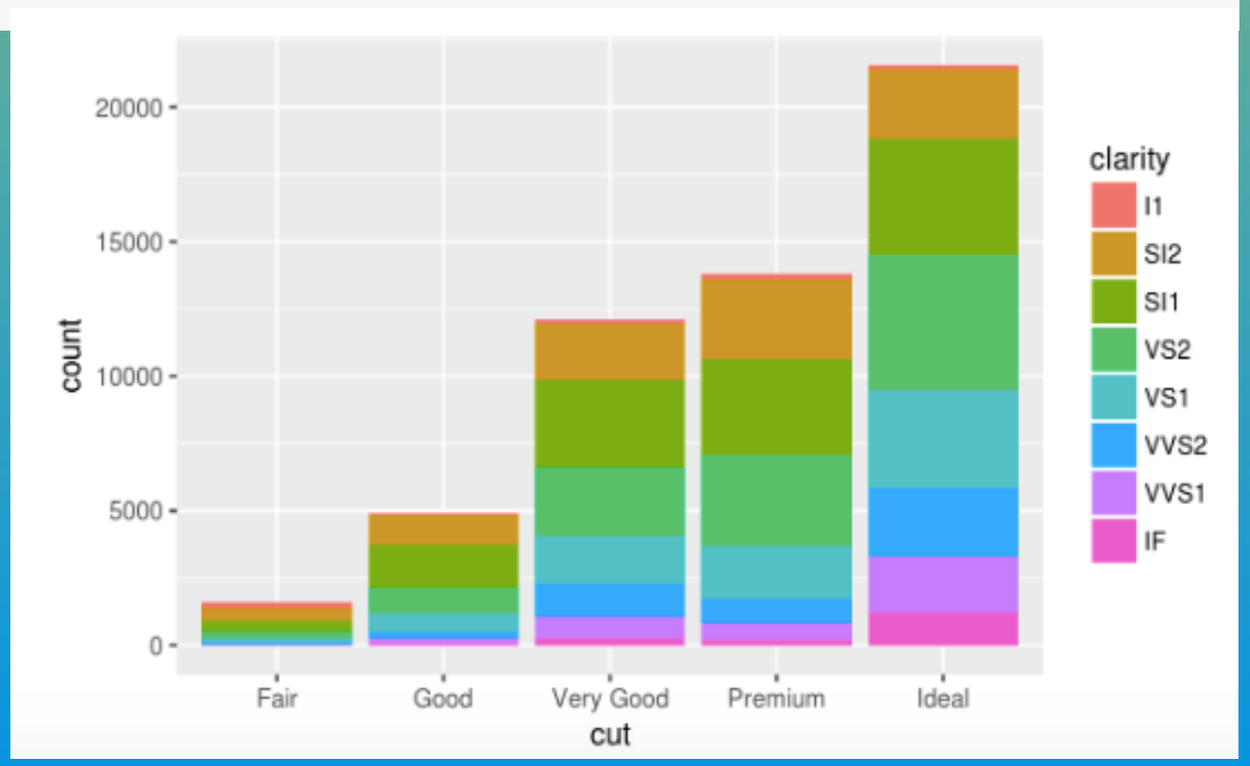


# ARGUMENTOS DE POSICIÓN

- ¿Qué pasará si mapeamos fill a una variable diferente de la mapeada al eje x?
- Por ejemplo, si mapeamos a la estética “fill”, la variable “clarity”, las barras son automáticamente divididas;

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

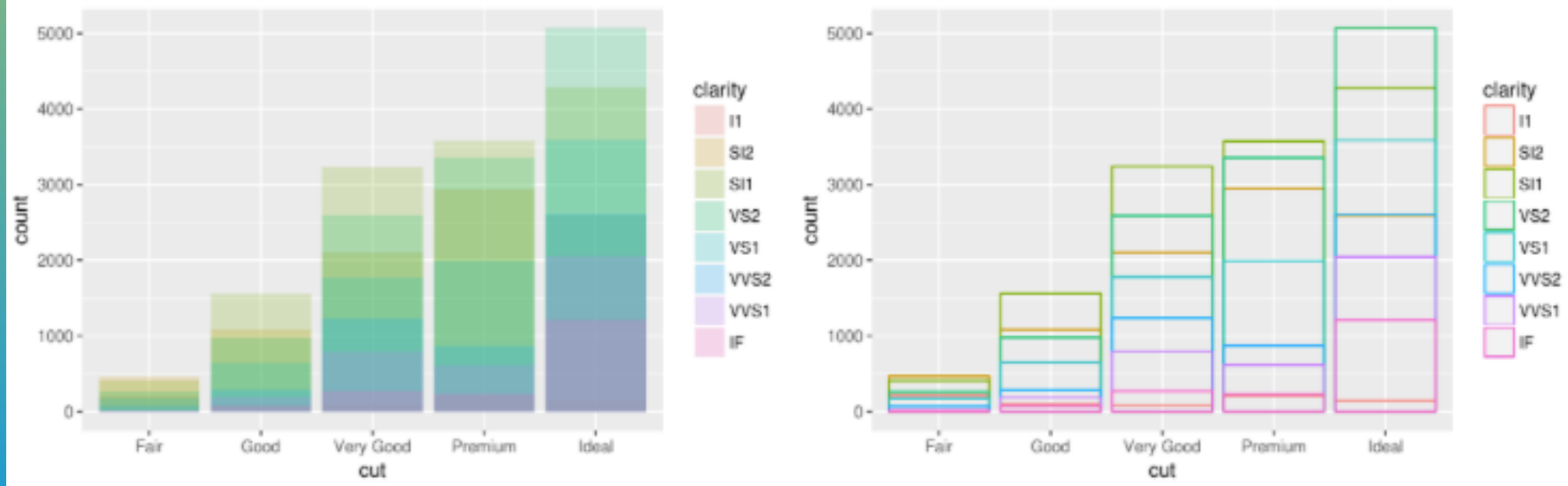


# MÁS OPCIONES

- Una opción que no es tan adecuada en general es fijar la estética de posición, “position”, como “identity”;
- Esto usualmente solapa las barras por lo que se hace recomendable fijar la transparencia en un valor bajo o fill = NA;
- Esto último es más útil en gráficos de dispersión que en este contexto.

# MÁS OPCIONES

```r
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")
```

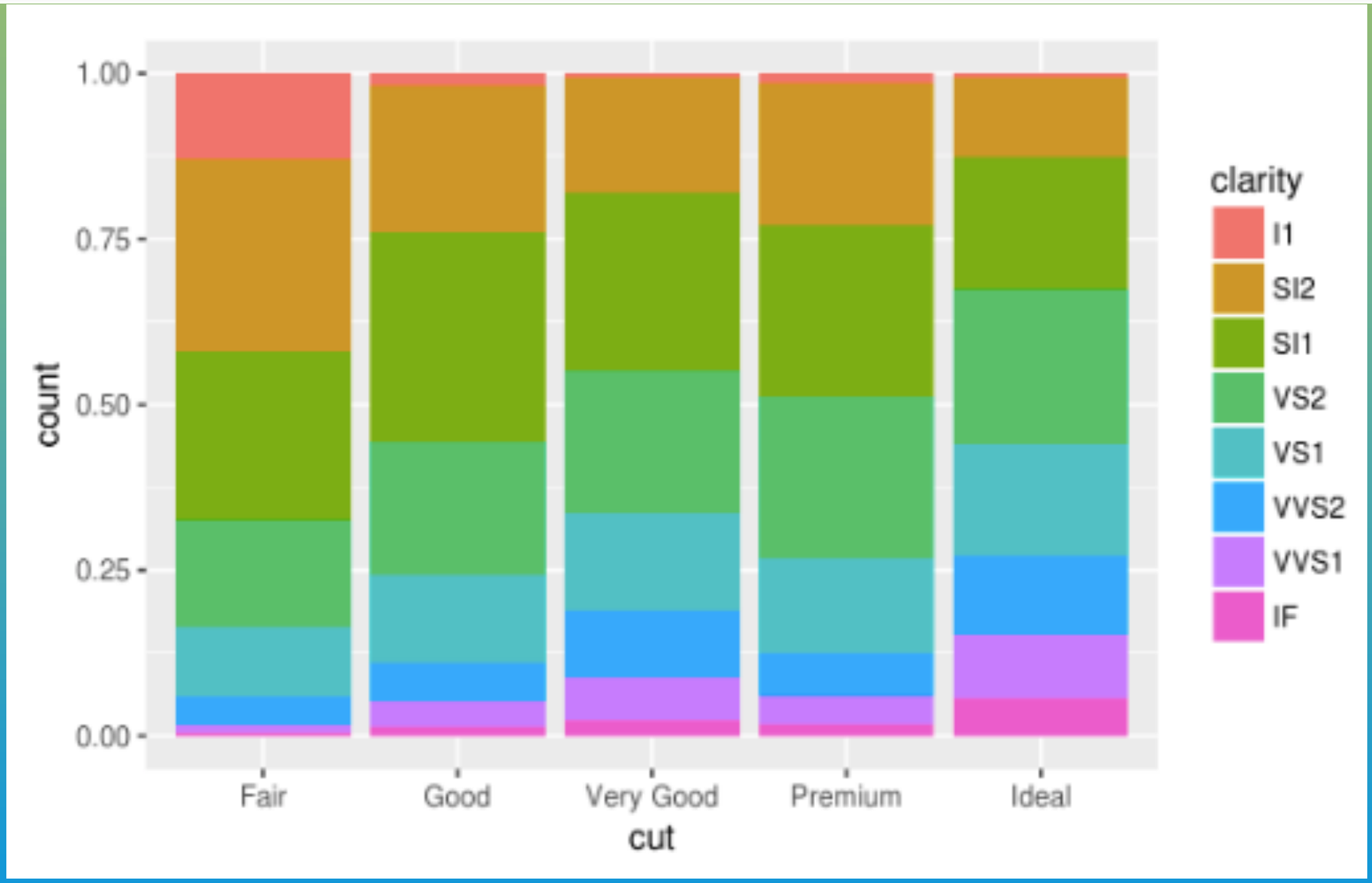


# MÁS OPCIONES

- Si fijamos la estética de posición como “fill”, obtendremos también barras divididas, pero esta vez de igual altura para cada elemento en el eje “x”;
- Esto puede servir para comparar de manera más precisa, proporciones a través de los diferentes grupos.

# MÁS OPCIONES

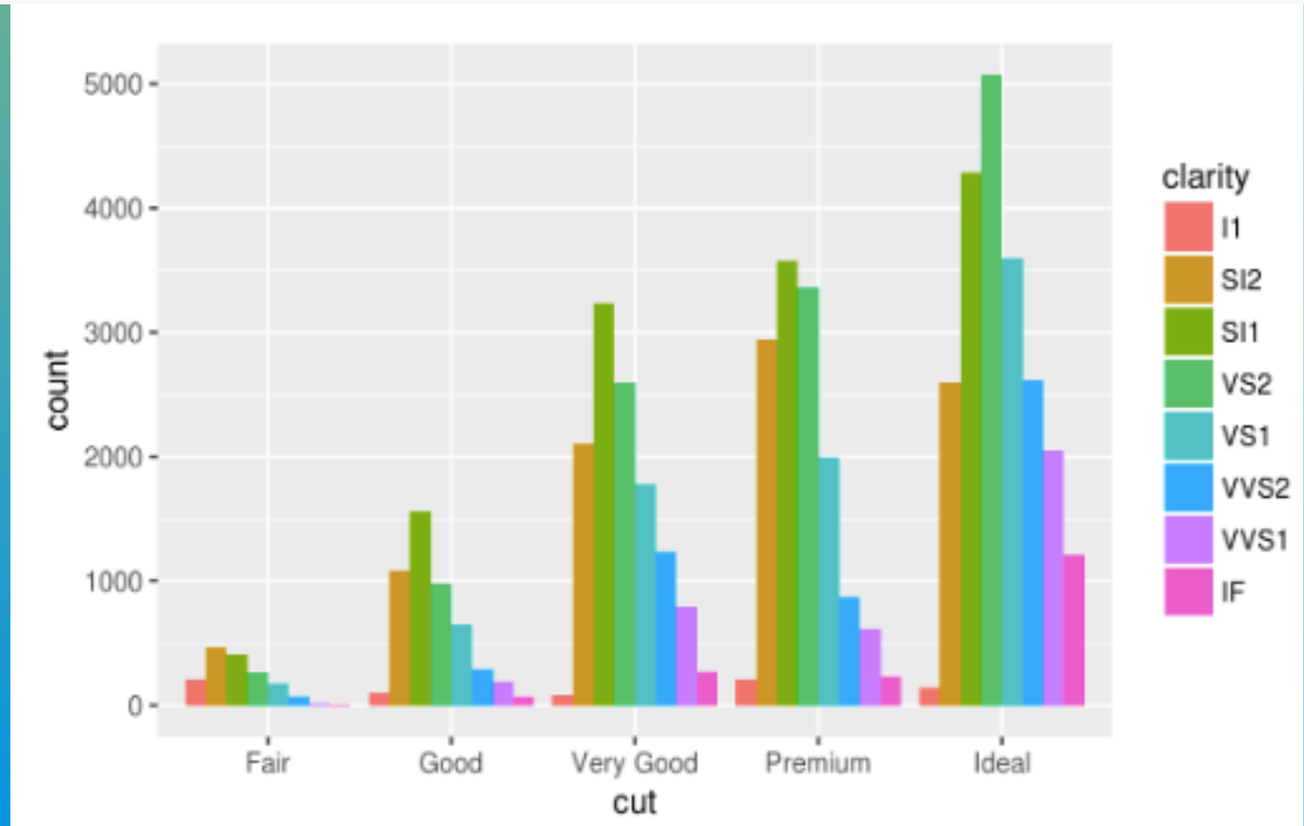
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



# MÁS OPCIONES

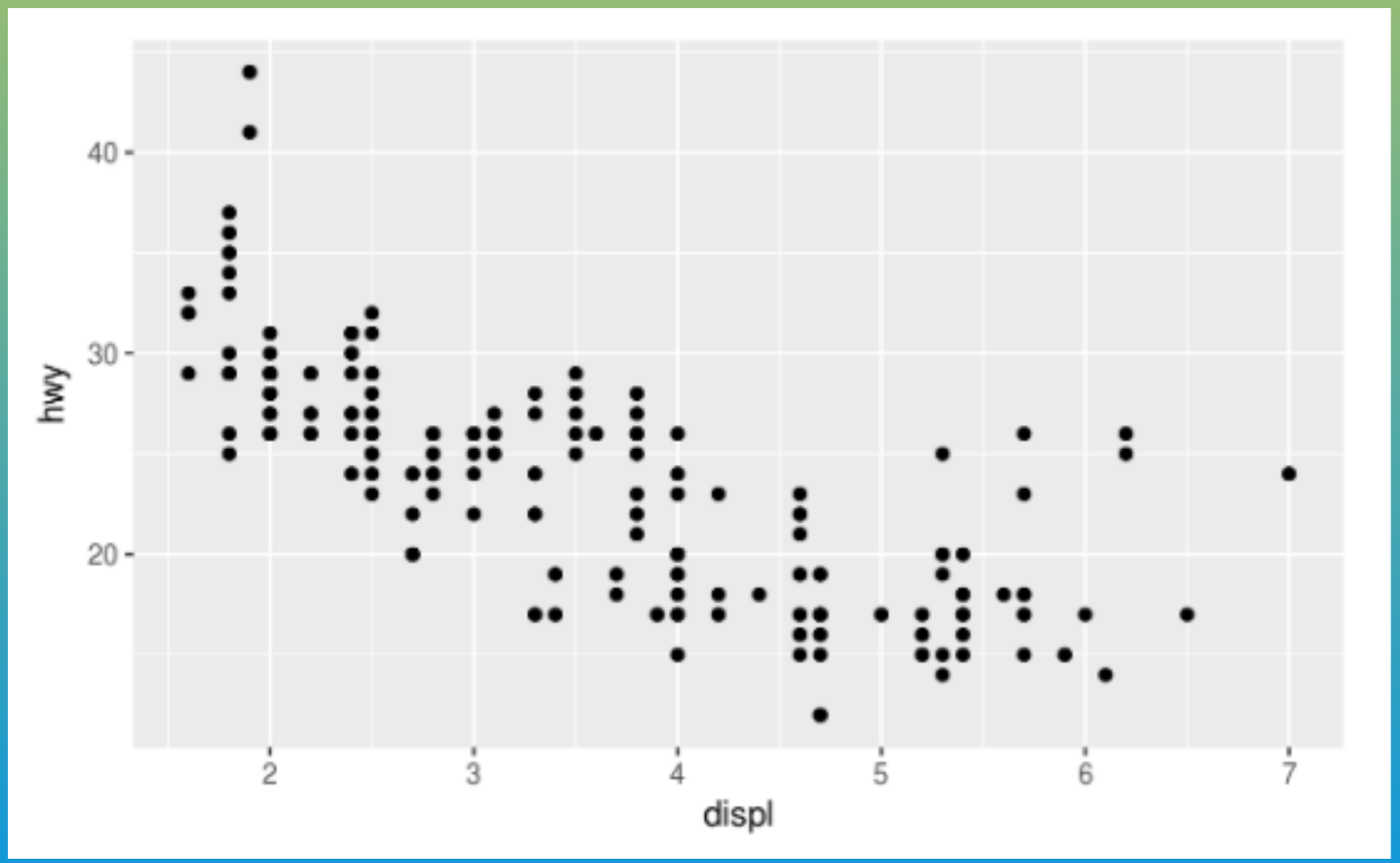
- Si fijamos la estética de posición como “dodge”, los objetos que ocupan la misma categoría se dibujan uno al lado de otro, esto hace más sencillo el comparar valores individuales, esto es, conteo para una categoría en el eje “x”.

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



# QUIZÁS NO LO HAYAN NOTADO...

- ¿Cuántas filas tenía el conjunto de datos mpg?
- ¿Cuántos puntos aparecen dibujados en el gráfico siguiente?



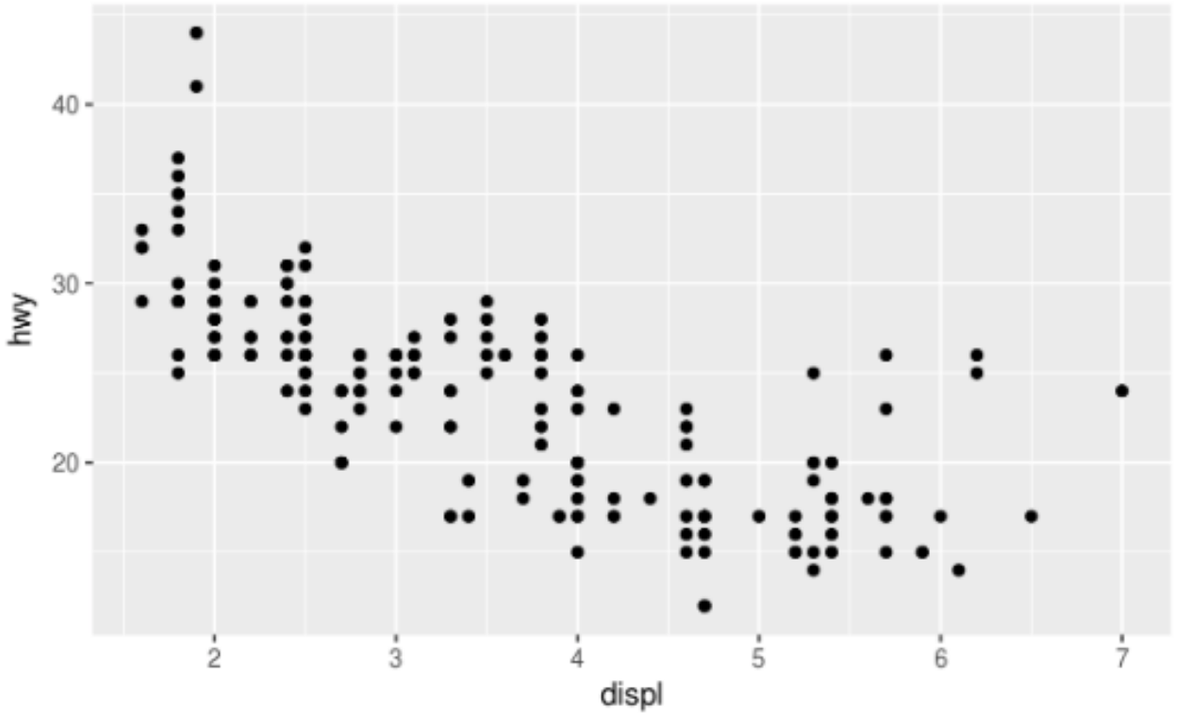
# QUIZÁS NO LO HAYAN NOTADO

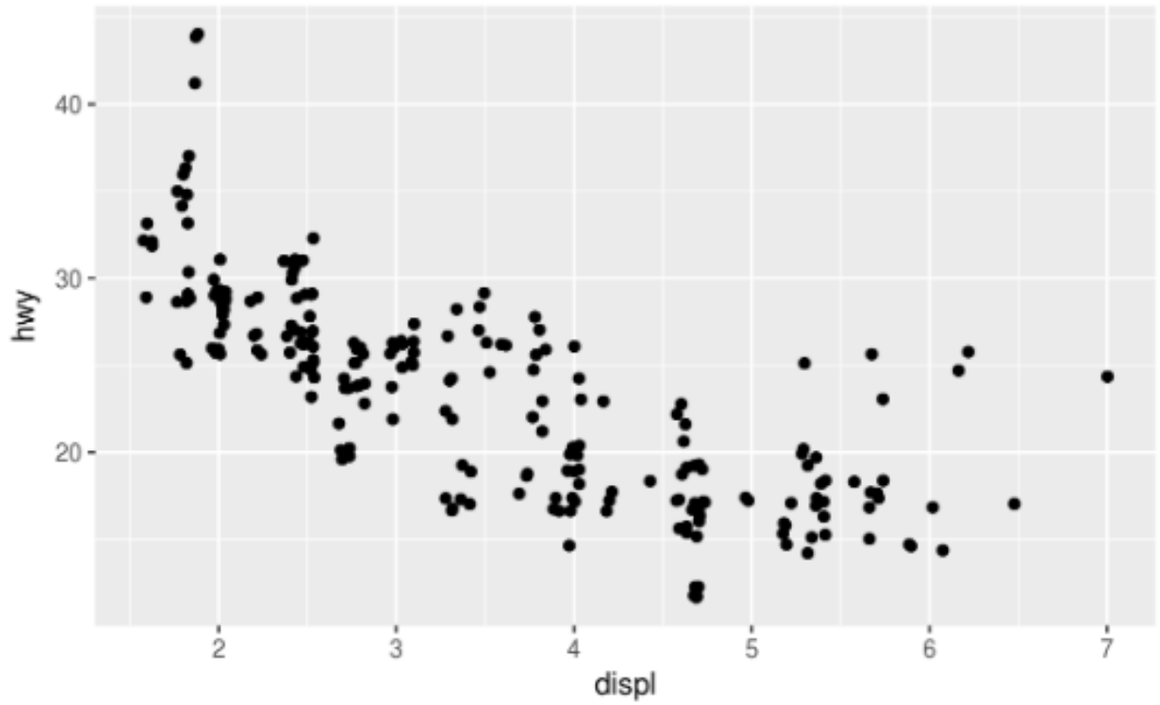
- Notemos que los valores de las variables hwy y displ son enteros por lo que en la gráfica varios puntos se superponen;
- Este problema se llama sobre-graficado;
- Notemos que no podemos decir, en la gráfica, si todos los puntos sobre-graficados se encuentran en la misma ubicación o dispersos.

# QUIZÁS NO LO HAYAN NOTADO

- Es posible evitar el sobre-graficado, añadiendo ruido a cada observación;
- Lo anterior dispersa los puntos porque ninguno recibirá la misma cantidad de ruido;
- Esto se logra fijando la estética de posición en “jitter”.

# QUIZÁS NO LO HAYAN NOTADO



# NO TODO ES TAN MARAVILLOSO

- El añadir ruido a las observaciones hace menos preciso el gráfico a escalas pequeñas, sin embargo puede ser de gran utilidad a gran escala;
- Dado que esta operación es tan útil, ggplot2 tiene una función que nos facilitará este proceso:

geom_point(position = “jitter”) = geom_jitter()

# SISTEMAS DE COORDENADAS

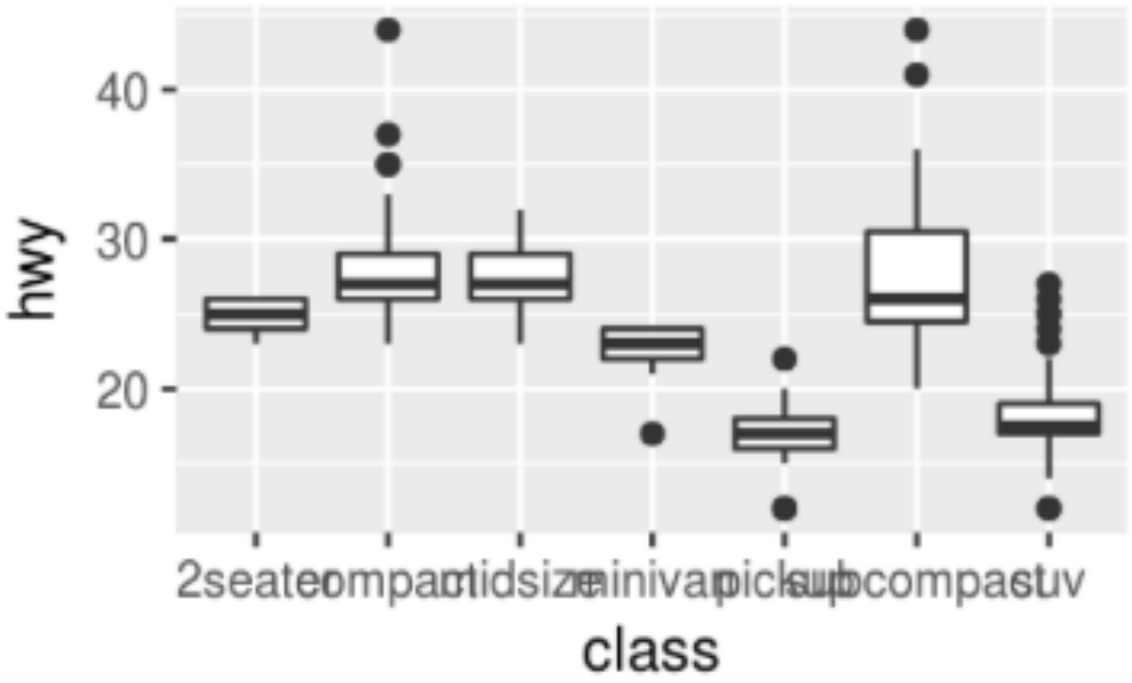
- Los sistemas de coordenadas son, en opinión de muchos, uno de los aspectos más complejos en el uso de la librería ggplot2;
- Los ejes coordenados por defecto corresponden a ejes cartesianos, donde las posiciones, “x” e “y”, actúan de manera independiente para determinar la posición de cada punto;
- Existen muchos otros sistemas de coordenadas que revisaremos a continuación.

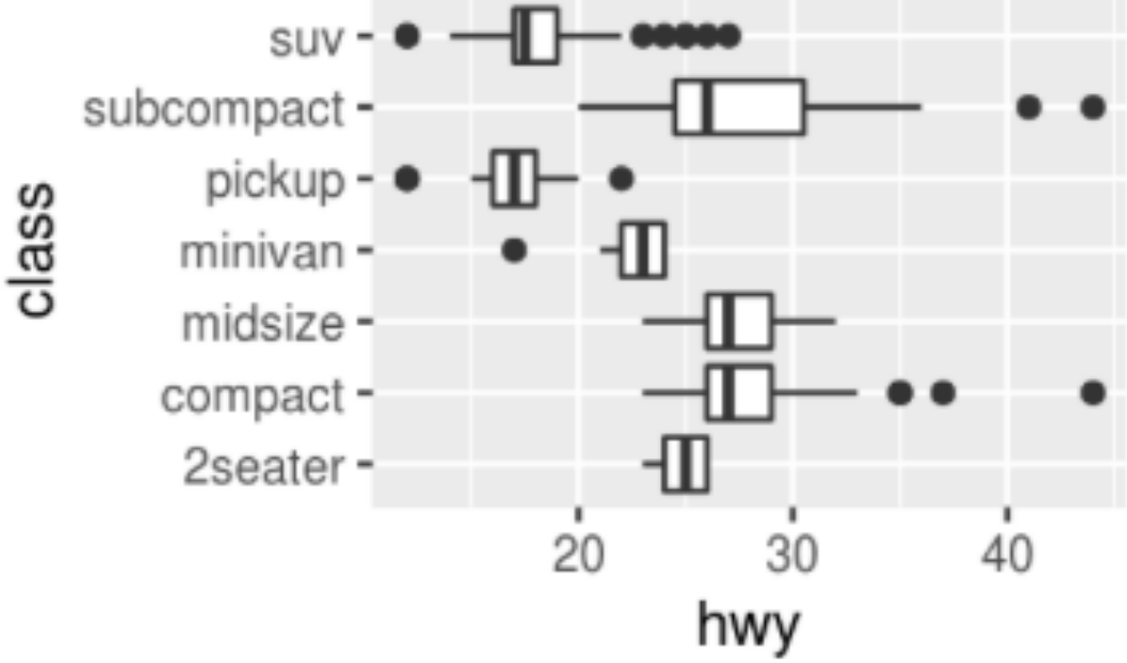
# COORD_FLIP

- Como su nombre podría indicar, coord_flip, intercambia los ejes “x” e “y”;
- Esto podría ser de utilidad si quisiésemos, por ejemplo, dibujar gráficos de caja horizontales o si las etiquetas desplegadas en el eje “x” son demasiado largas.

# 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()
```



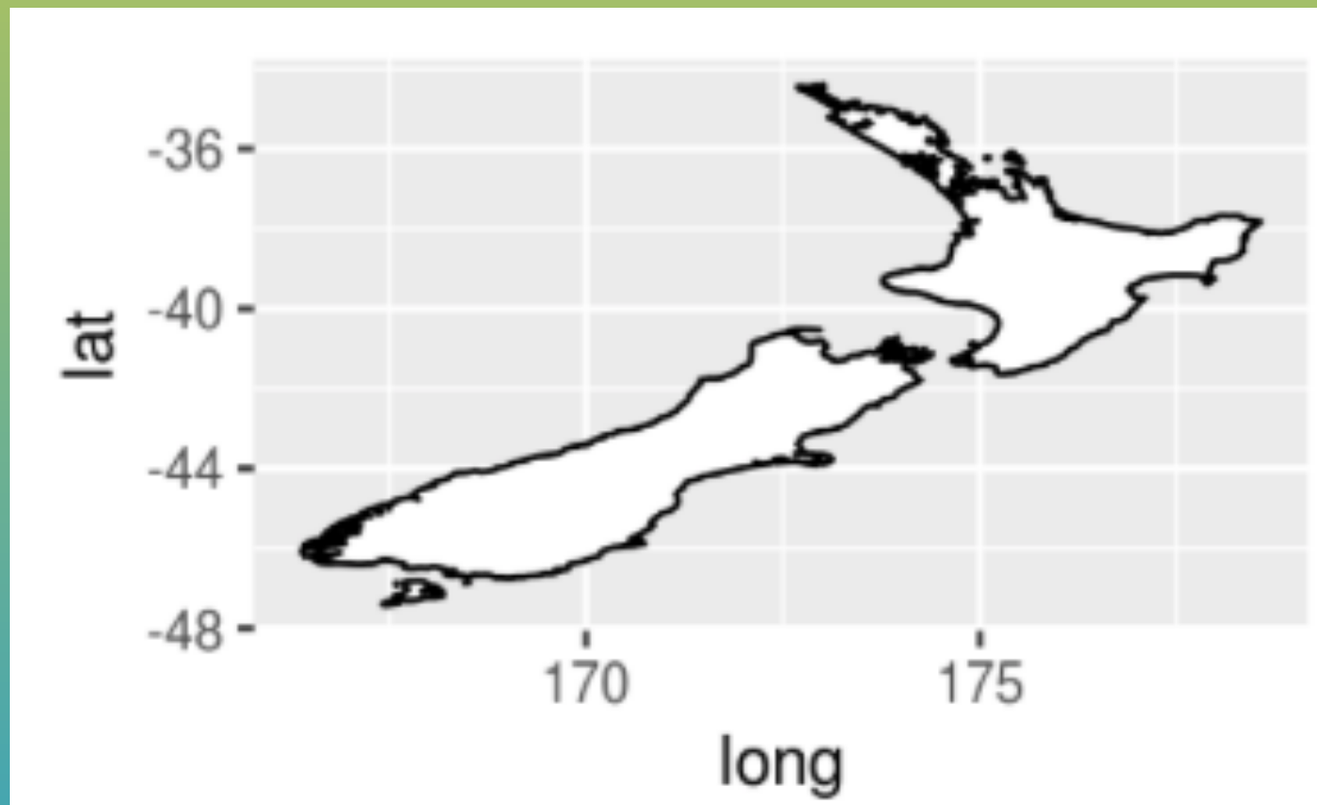



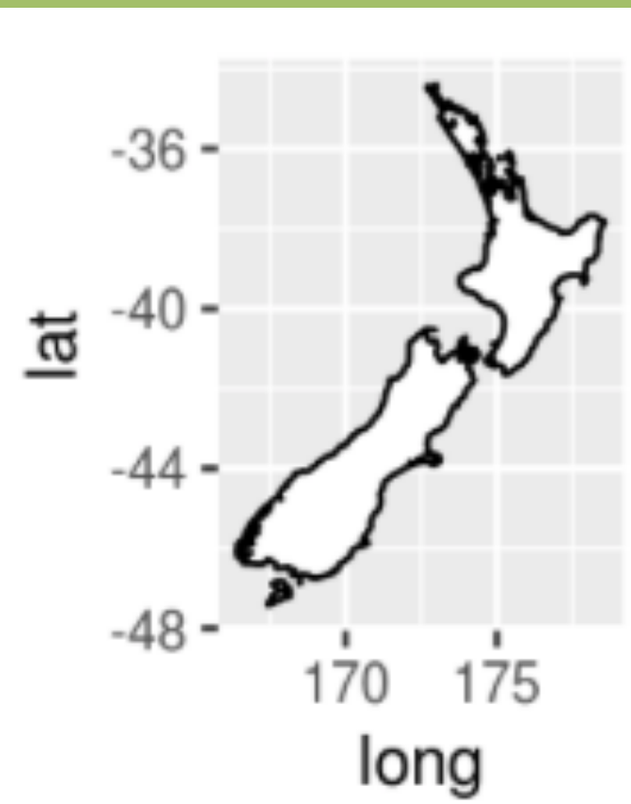
# COORD_QUICKMAP

- Esta función fija automáticamente la relación entre los ejes para desplegar mapas;
- Esto es de vital importancia cuando traficamos datos geo-referenciados con ggplot2.

```
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()
```

# COORD_QUICKMAP



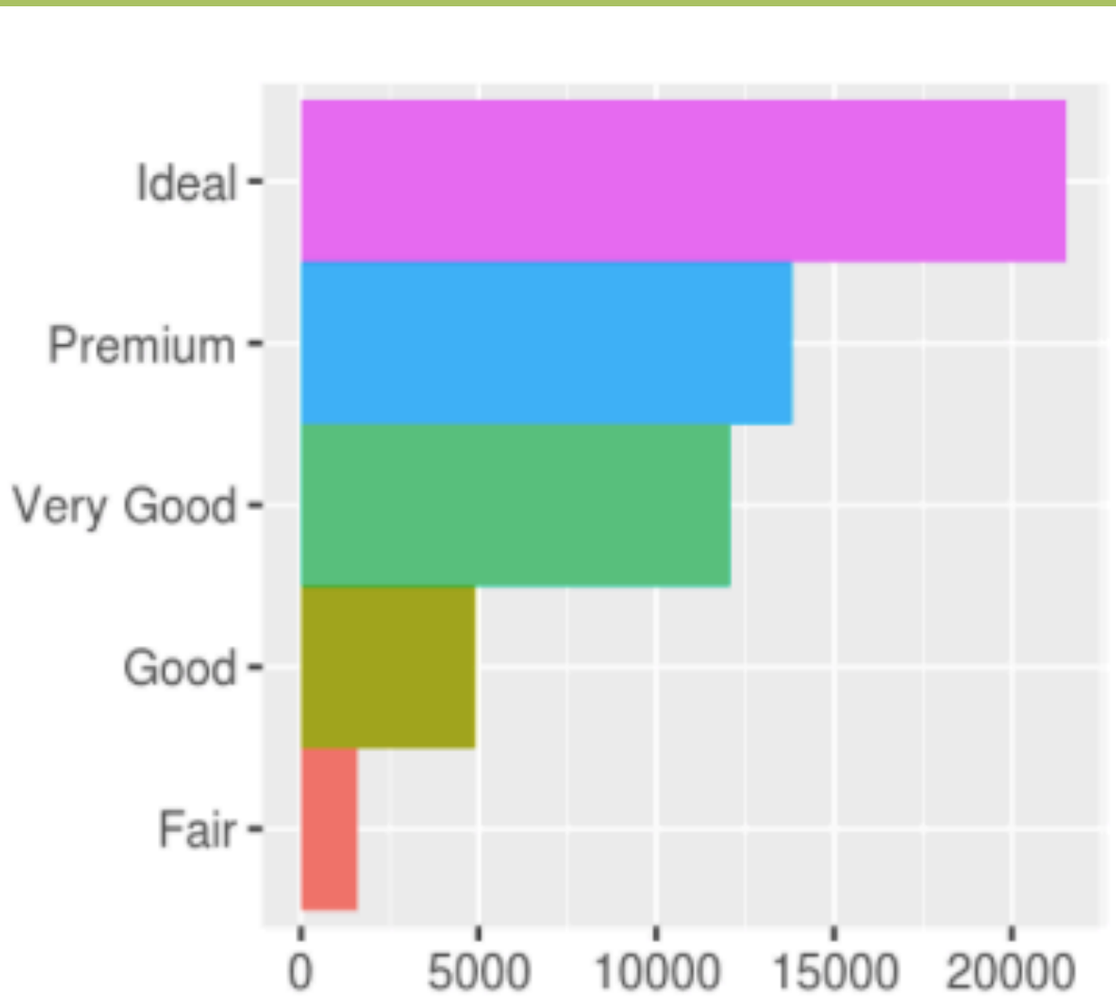



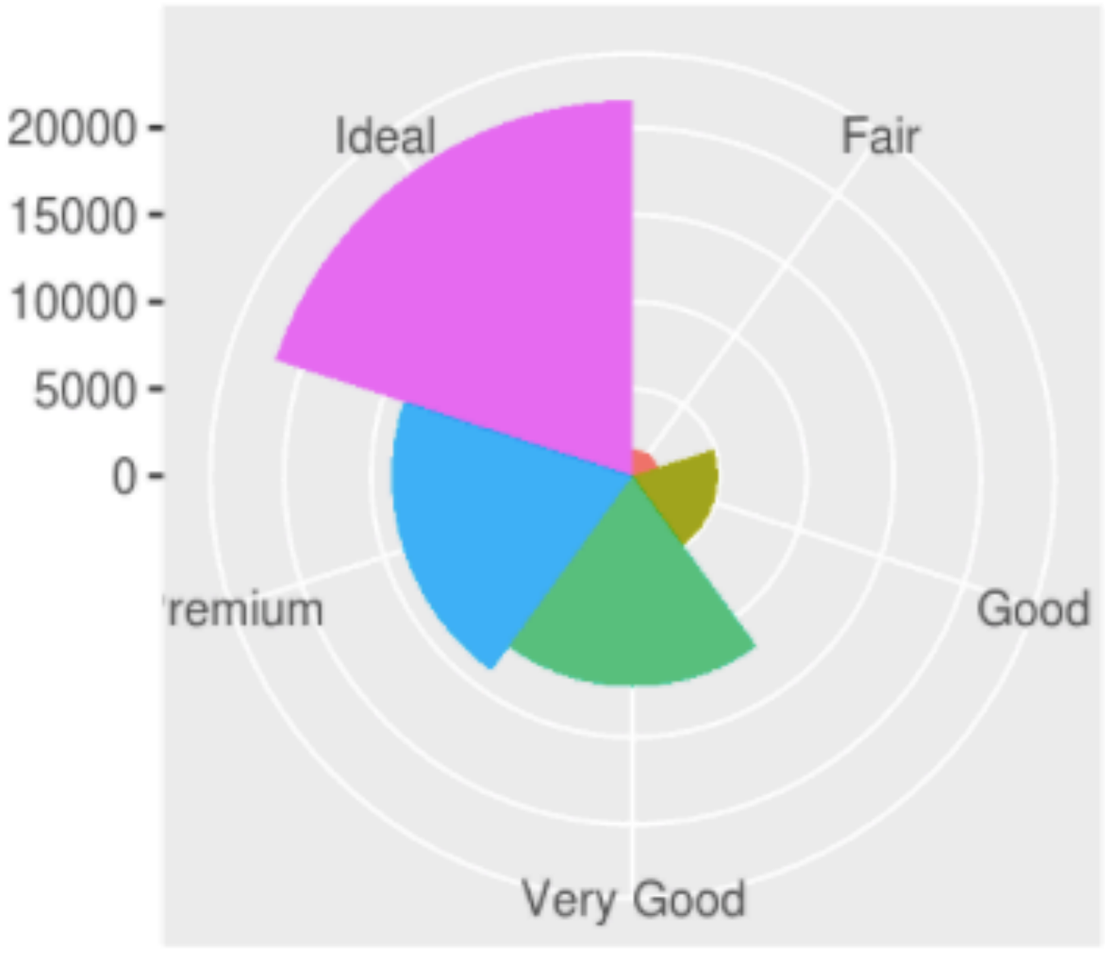
# COORD_POLAR()

- Esta función permite el uso de coordenadas polares;
- Este tipo de coordenadas permiten visualizar de manera interesante la información proporcionada por los gráficos de barra.

```
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()
```

# COORD_POLAR()


Ideal
Premium
Very Good
Good
Fair
0
5000
10000
15000
20000


20000
15000
10000
5000
0
Ideal
Fair
remium
Good
Very Good