Taller sobre el lenguaje R

Clase 2: Graficando con ggplot2

Lic. Lucio José Pantazis

March 24, 2021

Características generales

ggplot2

- ggplot2 es un paquete de R que permite hacer gráficos muy precisos y específicos.

ggplot2

- ggplot2 es un paquete de R que permite hacer gráficos muy precisos y específicos.
- Esto se debe a que permite graficar distintos objetos sobre gráficos previos, dando la chance de ajustar detalles de forma iterativa.

ggplot2

- ggplot2 es un paquete de R que permite hacer gráficos muy precisos y específicos.
- Esto se debe a que permite graficar distintos objetos sobre gráficos previos, dando la chance de ajustar detalles de forma iterativa.
- Además, con unos pocos conceptos principales, permite cambiar prácticamente cualquier detalle del gráfico.

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.
- Aesthetics: Variables cuyo valor van a definir cualidades estéticas de los elementos del gráfico (ubicación, color, forma, tamaño).

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.
- Aesthetics: Variables cuyo valor van a definir cualidades estéticas de los elementos del gráfico (ubicación, color, forma, tamaño).
- Geoms: formas geométricas que se proyectan sobre el gráfico (puntos, líneas, polígonos, etc.)

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.
- Aesthetics: Variables cuyo valor van a definir cualidades estéticas de los elementos del gráfico (ubicación, color, forma, tamaño).
- Geoms: formas geométricas que se proyectan sobre el gráfico (puntos, líneas, polígonos, etc.)
- Stats: Medidas estadísticas que se aplican a las variables antes de graficarlas (cálculo de frecuencias, medias, etc.)

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.
- Aesthetics: Variables cuyo valor van a definir cualidades estéticas de los elementos del gráfico (ubicación, color, forma, tamaño).
- Geoms: formas geométricas que se proyectan sobre el gráfico (puntos, líneas, polígonos, etc.)
- Stats: Medidas estadísticas que se aplican a las variables antes de graficarlas (cálculo de frecuencias, medias, etc.)
- Scales: Determinación de nombres y valores que toman las variables que determinan las cuestiones estéticas.

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.
- Aesthetics: Variables cuyo valor van a definir cualidades estéticas de los elementos del gráfico (ubicación, color, forma, tamaño).
- Geoms: formas geométricas que se proyectan sobre el gráfico (puntos, líneas, polígonos, etc.)
- Stats: Medidas estadísticas que se aplican a las variables antes de graficarlas (cálculo de frecuencias, medias, etc.)
- Scales: Determinación de nombres y valores que toman las variables que determinan las cuestiones estéticas.
- Coord: Un sistema de coordenadas para el plano.

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.
- Aesthetics: Variables cuyo valor van a definir cualidades estéticas de los elementos del gráfico (ubicación, color, forma, tamaño).
- Geoms: formas geométricas que se proyectan sobre el gráfico (puntos, líneas, polígonos, etc.)
- Stats: Medidas estadísticas que se aplican a las variables antes de graficarlas (cálculo de frecuencias, medias, etc.)
- Scales: Determinación de nombres y valores que toman las variables que determinan las cuestiones estéticas.
- Coord: Un sistema de coordenadas para el plano.
- Facet: División de los datos según grupos.

gg = Grammar of graphics

ggplot2 se basa en una publicación de Wilkinson (2005) que hace referencia a "la gramática de los gráficos", en el que sintetiza los componentes de cualquier gráfico estadístico:

- Data: La base que contiene los datos que serán parte del gráfico.
- Aesthetics: Variables cuyo valor van a definir cualidades estéticas de los elementos del gráfico (ubicación, color, forma, tamaño).
- Geoms: formas geométricas que se proyectan sobre el gráfico (puntos, líneas, polígonos, etc.)
- Stats: Medidas estadísticas que se aplican a las variables antes de graficarlas (cálculo de frecuencias, medias, etc.)
- Scales: Determinación de nombres y valores que toman las variables que determinan las cuestiones estéticas.
- Coord: Un sistema de coordenadas para el plano.
- Facet: División de los datos según grupos.
- Theme: Detalles concernientes a formato del texto, de paneles, ejes, etc.

Comando ggplot

Datos: mpg

Primero recordemos cómo era el data.frame mpg, perteneciente al paquete ggplot2:

```
require(ggplot2)
head(mpg)
```

```
##   manufacturer model displ year cyl      trans drv cty hwy fl   class
## 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
```

Datos: mpg

Primero recordemos cómo era el data.frame mpg, perteneciente al paquete ggplot2:

```
require(ggplot2)
str(mpg)
```

```
## 'data.frame':    234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ..
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr  "p" "p" "p" "p" ...
##  $ class       : chr  "compact" "compact" "compact" "compact" ...
```

Descripción de los datos

Usaremos las siguientes variables:

- manufacturer: El nombre de la empresa fabricante (cualitativa, tipo character).

Descripción de los datos

Usaremos las siguientes variables:

- manufacturer: El nombre de la empresa fabricante (cualitativa, tipo character).
- displ: desplazamiento del motor en litros (cuantitativa continua, tipo numeric).

Descripción de los datos

Usaremos las siguientes variables:

- manufacturer: El nombre de la empresa fabricante (cualitativa, tipo character).
- displ: desplazamiento del motor en litros (cuantitativa continua, tipo numeric).
- year: año de manufactura (cuantitativa discreta, tipo integer).

Descripción de los datos

Usaremos las siguientes variables:

- manufacturer: El nombre de la empresa fabricante (cualitativa, tipo character).
- displ: desplazamiento del motor en litros (cuantitativa continua, tipo numeric).
- year: año de manufactura (cuantitativa discreta, tipo integer).
- cyl: número de cilindros (cuantitativa discreta, tipo integer).

Descripción de los datos

Usaremos las siguientes variables:

- manufacturer: El nombre de la empresa fabricante (cualitativa, tipo character).
- displ: desplazamiento del motor en litros (cuantitativa continua, tipo numeric).
- year: año de manufactura (cuantitativa discreta, tipo integer).
- cyl: número de cilindros (cuantitativa discreta, tipo integer).
- drv: ejes de maniobra (cualitativa, tipo character).

Descripción de los datos

Usaremos las siguientes variables:

- manufacturer: El nombre de la empresa fabricante (cualitativa, tipo character).
- displ: desplazamiento del motor en litros (cuantitativa continua, tipo numeric).
- year: año de manufactura (cuantitativa discreta, tipo integer).
- cyl: número de cilindros (cuantitativa discreta, tipo integer).
- drv: ejes de maniobra (cualitativa, tipo character).
- hwy: rendimiento en autopista, medido en millas por galón de nafta (cuantitativa continua, tipo integer).

Descripción de los datos

Usaremos las siguientes variables:

- manufacturer: El nombre de la empresa fabricante (cualitativa, tipo character).
- displ: desplazamiento del motor en litros (cuantitativa continua, tipo numeric).
- year: año de manufactura (cuantitativa discreta, tipo integer).
- cyl: número de cilindros (cuantitativa discreta, tipo integer).
- drv: ejes de maniobra (cualitativa, tipo character).
- hwy: rendimiento en autopista, medido en millas por galón de nafta (cuantitativa continua, tipo integer).
- fl: tipo de nafta (cualitativa, tipo character).

Aesthetics: Ubicación

Recordando que las variables estéticas son las que definen las principales características del gráfico, se incluyen mediante el comando aes. Por ejemplo aquí definimos las coordenadas x e y según las variables cuantitativas displ y hwy:

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

Aesthetics: Ubicación

Recordando que las variables estéticas son las que definen las principales características del gráfico, se incluyen mediante el comando aes. Por ejemplo aquí definimos las coordenadas x e y según las variables cuantitativas displ y hwy:

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

Este comando inicia las condiciones básicas del gráfico que se quiere realizar.

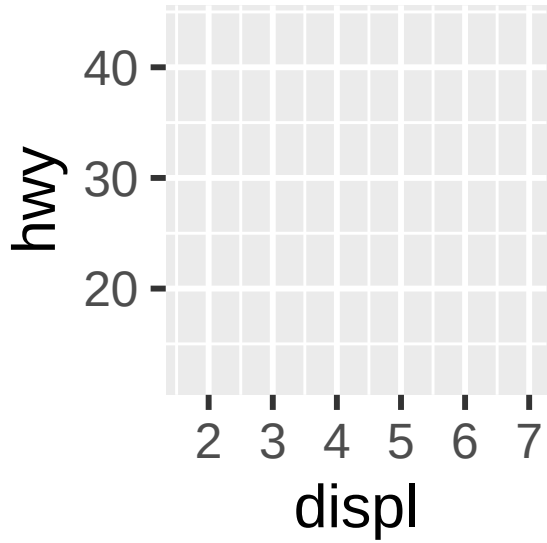
Aesthetics: Ubicación

Si corremos sólo este comando, nos aparecerá un plano simple con los ejes nombrados según las variables como están en la base mpg:

Aesthetics: Ubicación

Si corremos sólo este comando, nos aparecerá un plano simple con los ejes nombrados según las variables como están en la base mpg:

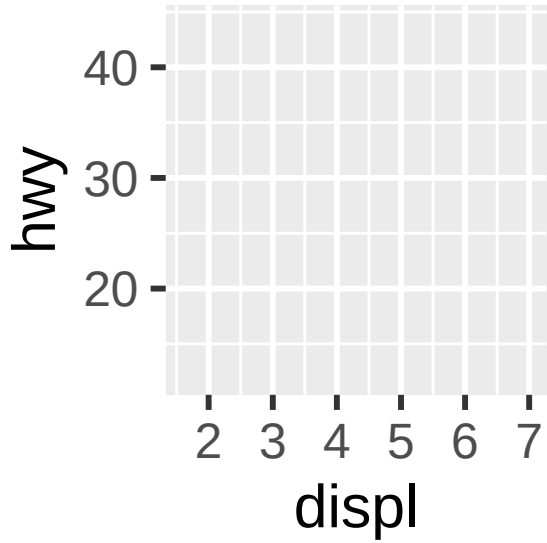
```
ggplot(data = mpg,
       mapping =aes(x=displ,y=hwy))
```



Aesthetics: Ubicación

Si corremos sólo este comando, nos aparecerá un plano simple con los ejes nombrados según las variables como están en la base mpg:

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



Notar que los ejes se mueven entre los valores máximos y mínimos de las variables:

```
c(min(mpg$displ),max(mpg$displ),min(mpg$hwy),max(mpg$hwy))
```

```
## [1]  1.6  7.0 12.0 44.0
```

Capas (layers)

Gráfico de puntos

Sobre este plano, se agregan el resto de los objetos del gráfico en forma de capas (layers) usando el signo +, y luego especificando una forma geométrica para los objetos a agregar. Los comandos tienen la estructura "geom_*". Por ejemplo, para hacer un scatter plot, agregamos a lo anterior el comando geom_point, que usa la información de x e y definidas anteriormente:

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

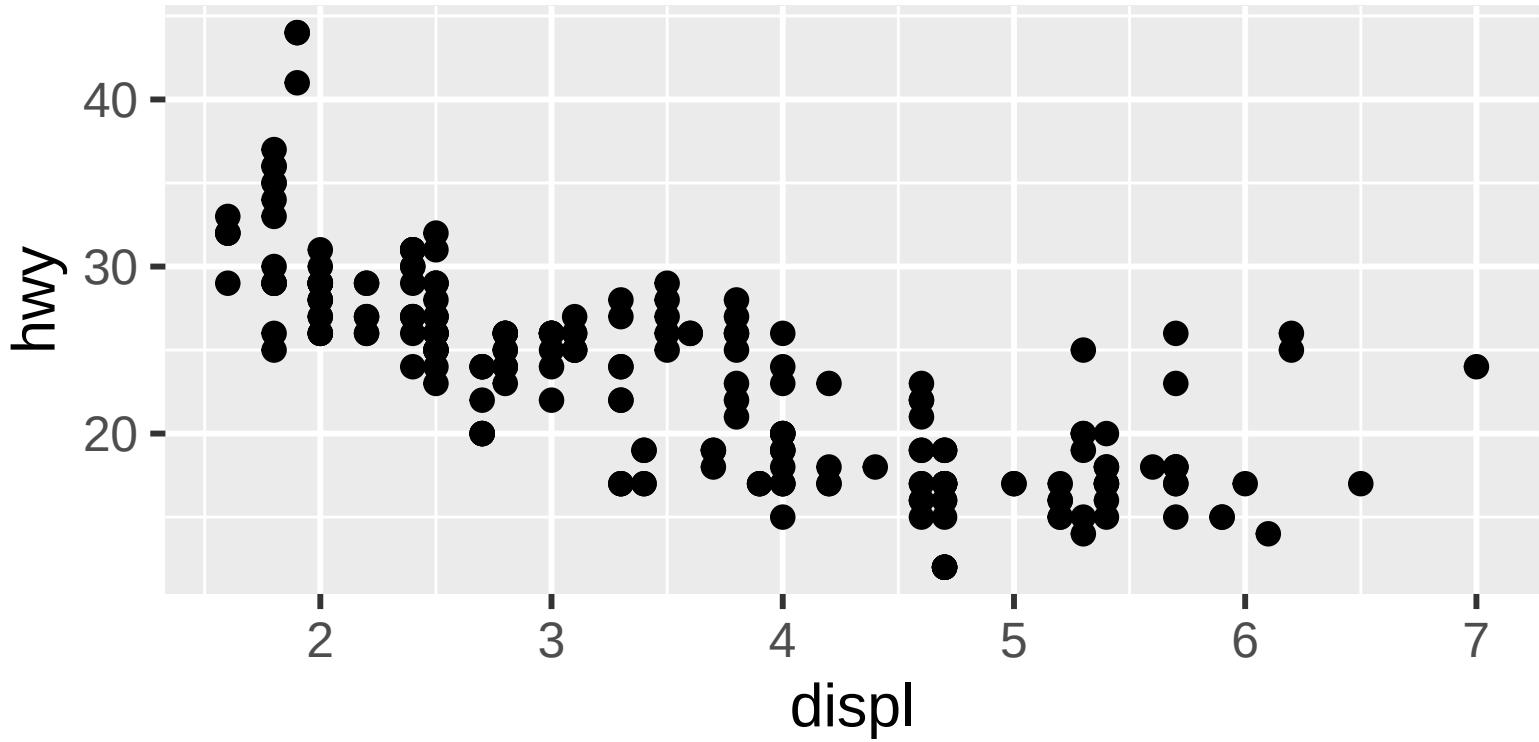


Gráfico de puntos

Por otro lado, para enfatizar de que las formas geométricas se agregan en forma de "capas" le podemos agregar una línea horizontal azul (usando el comando geom_hline) a la altura de la media de la variable hwy:

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

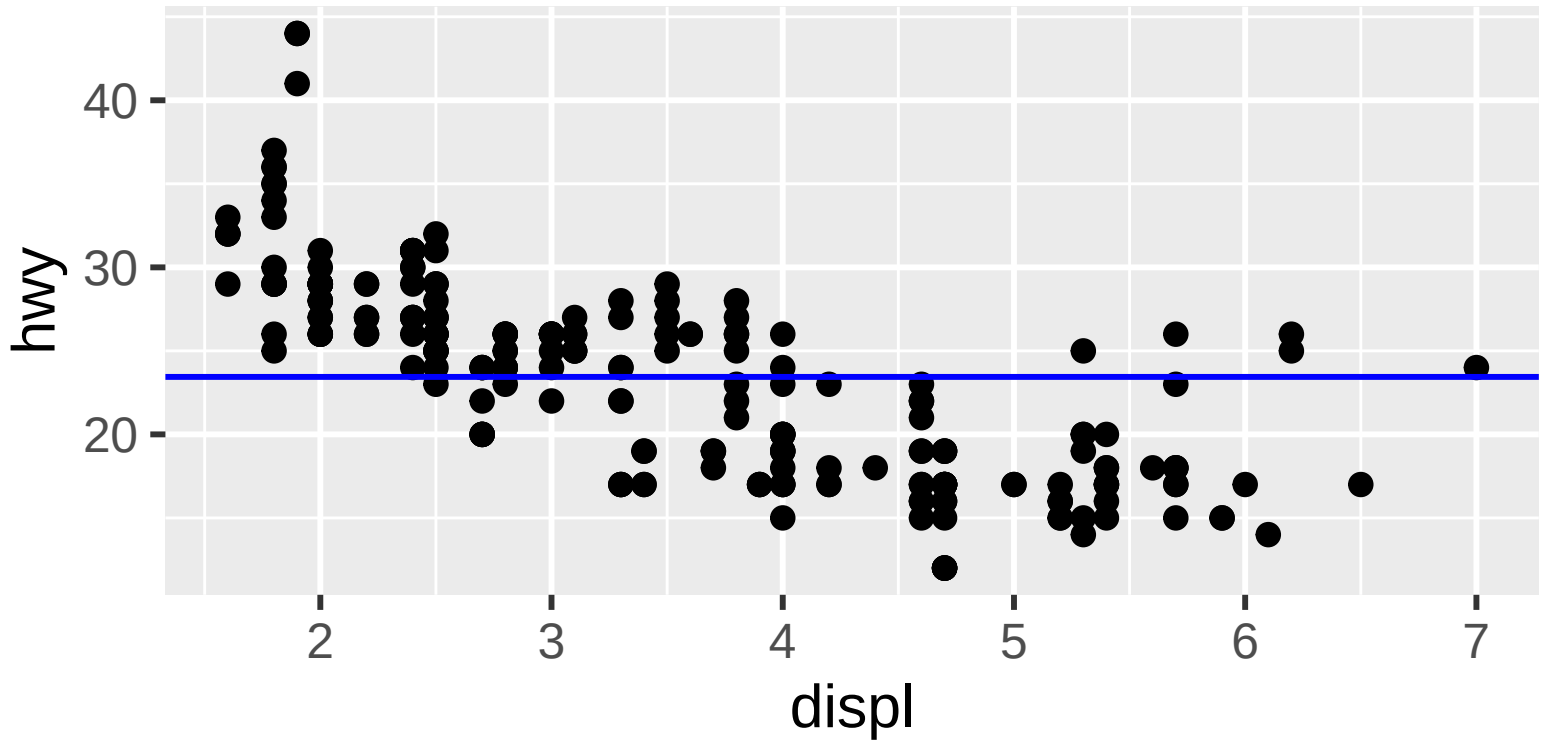


Gráfico de puntos

Por otro lado, le podemos agregar una línea vertical roja (usando el comando geom_vline) a la altura de la media de la variable displ:

```
ggplot(data = mpg)+
  geom_point(mapping =aes(x=displ,y=hwy))+
  geom_hline(mapping = aes(yintercept=mean(hwy)),color="blue")+
  geom_vline(mapping = aes(xintercept=mean(displ)),color="red")
```

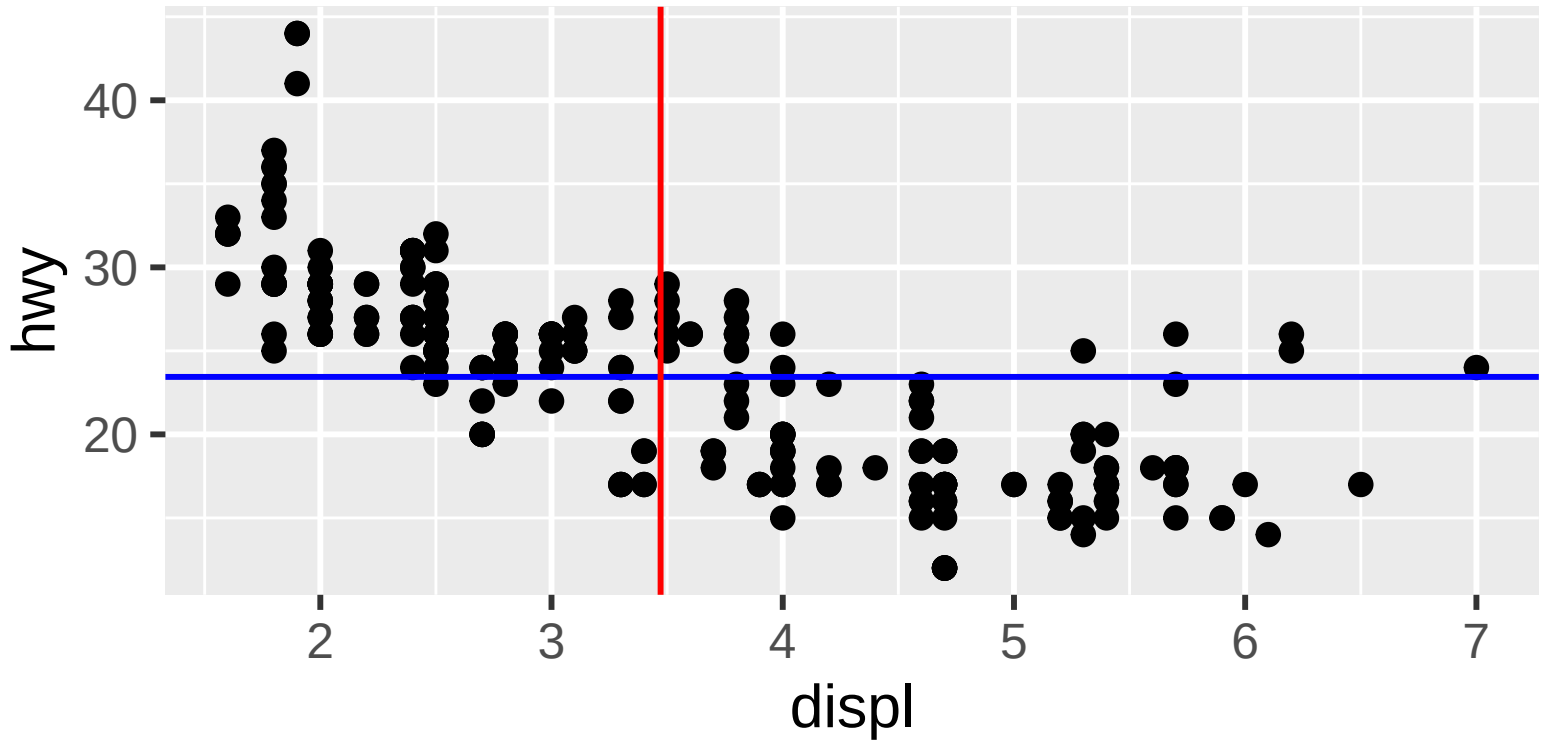


Gráfico de puntos

Además, como va agregando por capas los objetos, el orden de las capas altera el resultado. Por ejemplo, en este caso, los puntos se sobreponen a las rectas:

```
ggplot(data = mpg)+
  geom_vline(mapping = aes(xintercept=mean(displ)),color="red")+
  geom_hline(mapping = aes(yintercept=mean(hwy)),color="blue")+
  geom_point(mapping =  aes(x=displ,y=hwy))
```

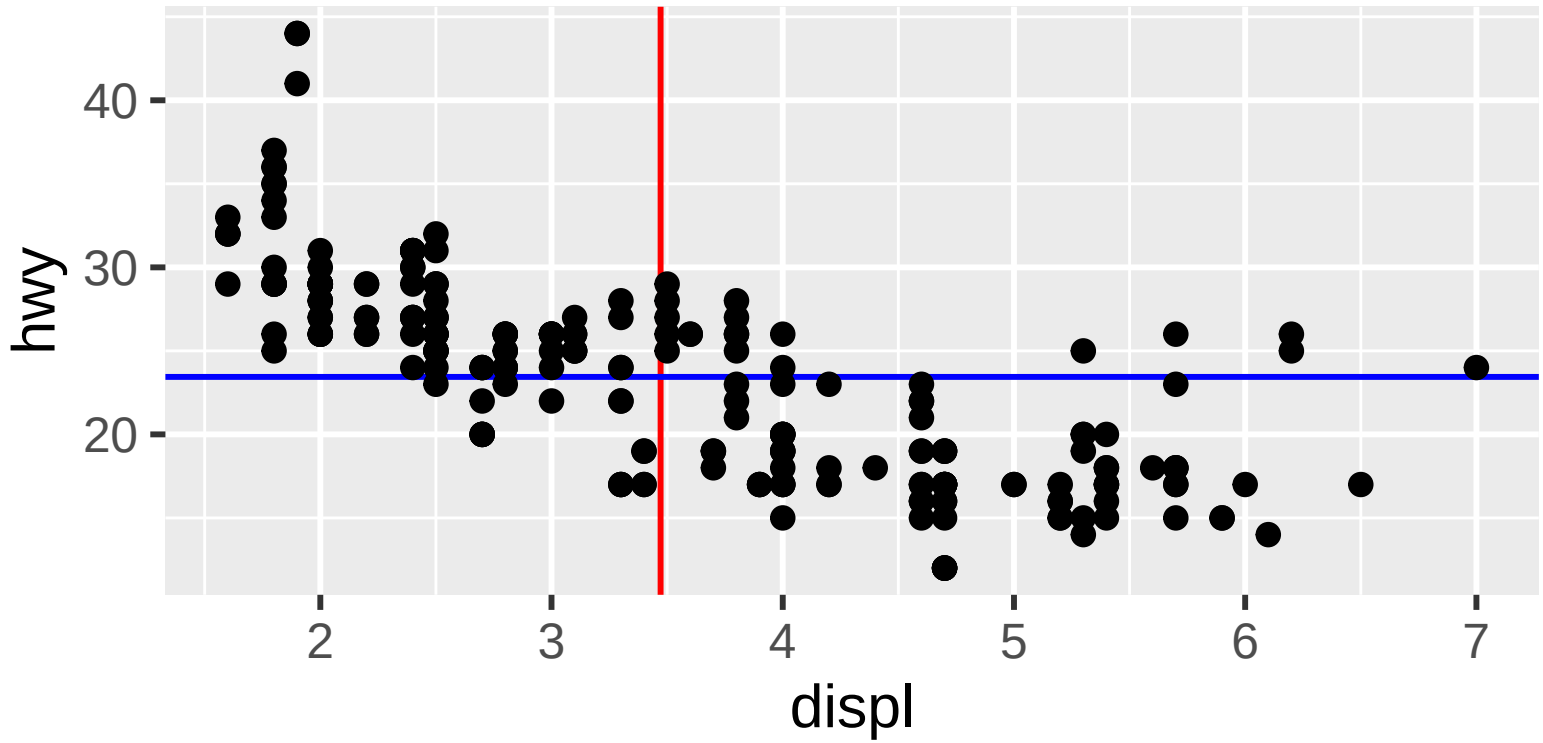


Gráfico de puntos

Además, las variables estéticas pueden ser específicas a cada forma geométrica utilizada. Por ejemplo, si no se definen al principio, se pueden definir en el mismo comando geom_point, obteniendo el mismo gráfico:

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

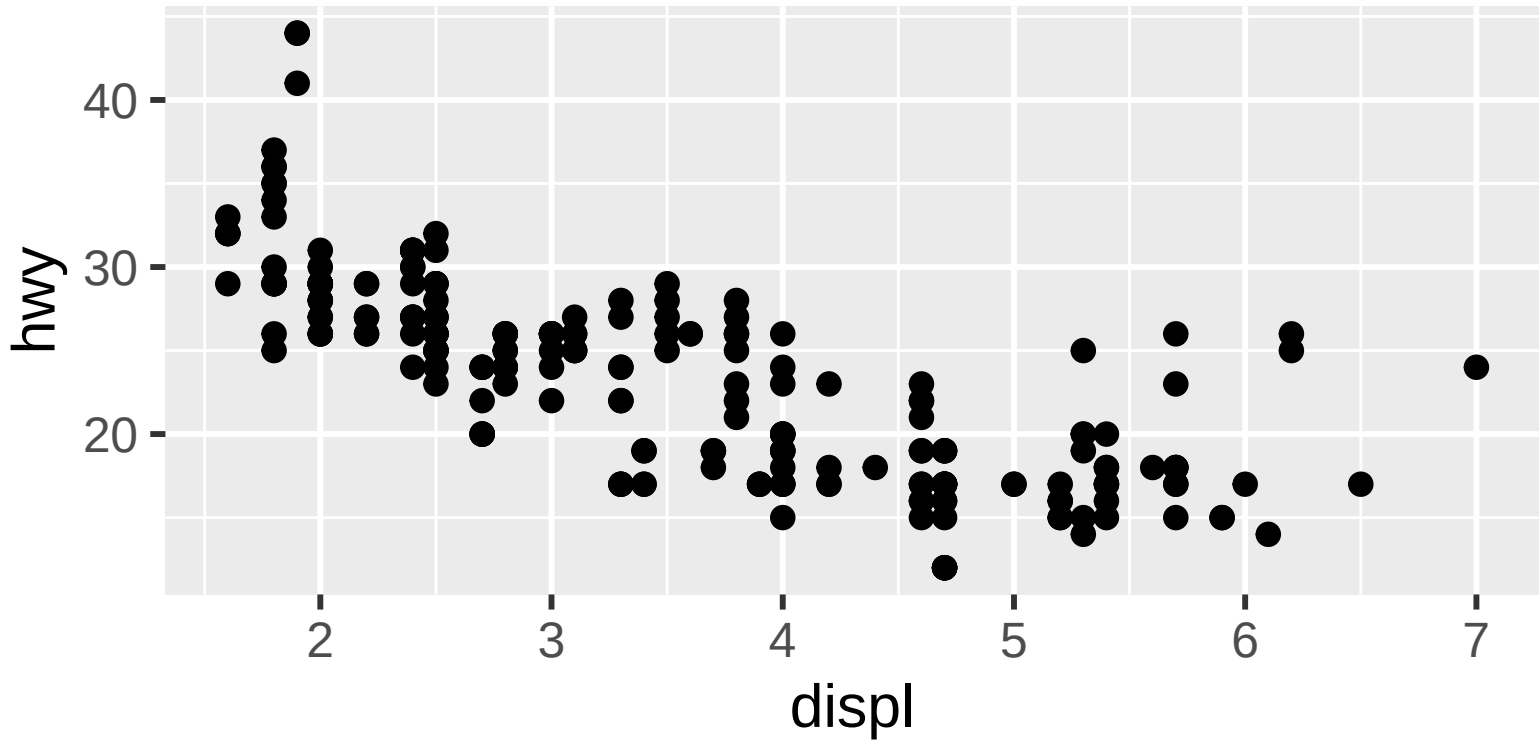


Gráfico de puntos

Para algunas formas geométricas, hay una cantidad requerida de variables estéticas. Por ejemplo, para un punto, hacen falta coordenadas x y coordenadas y:

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

```
## Error: geom_point requires the following missing aesthetics: y
```

Gráfico de puntos

Por default, cada forma geométrica hereda las definiciones de las variables estéticas del principio. Por ejemplo, puedo definir *y* al principio y luego *x*:

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

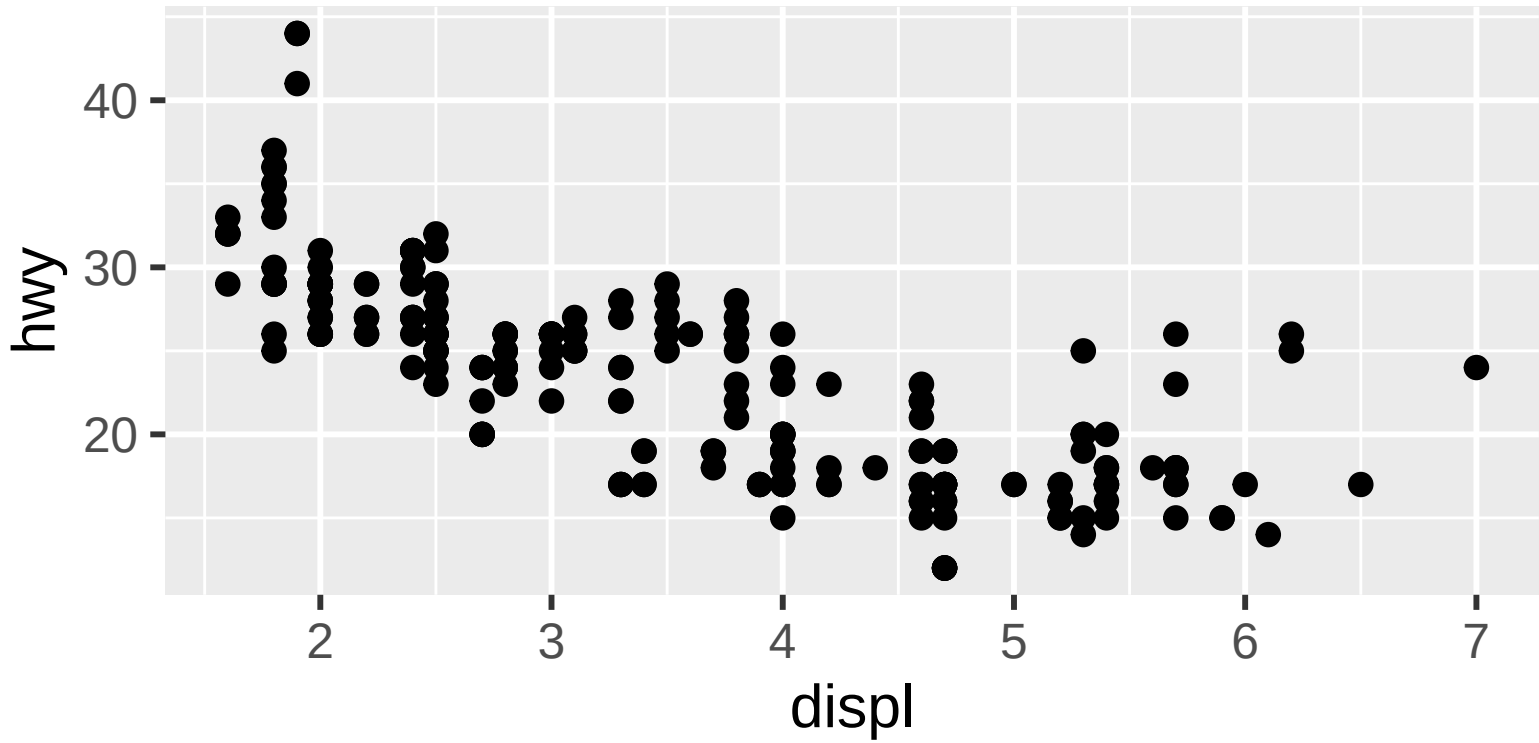
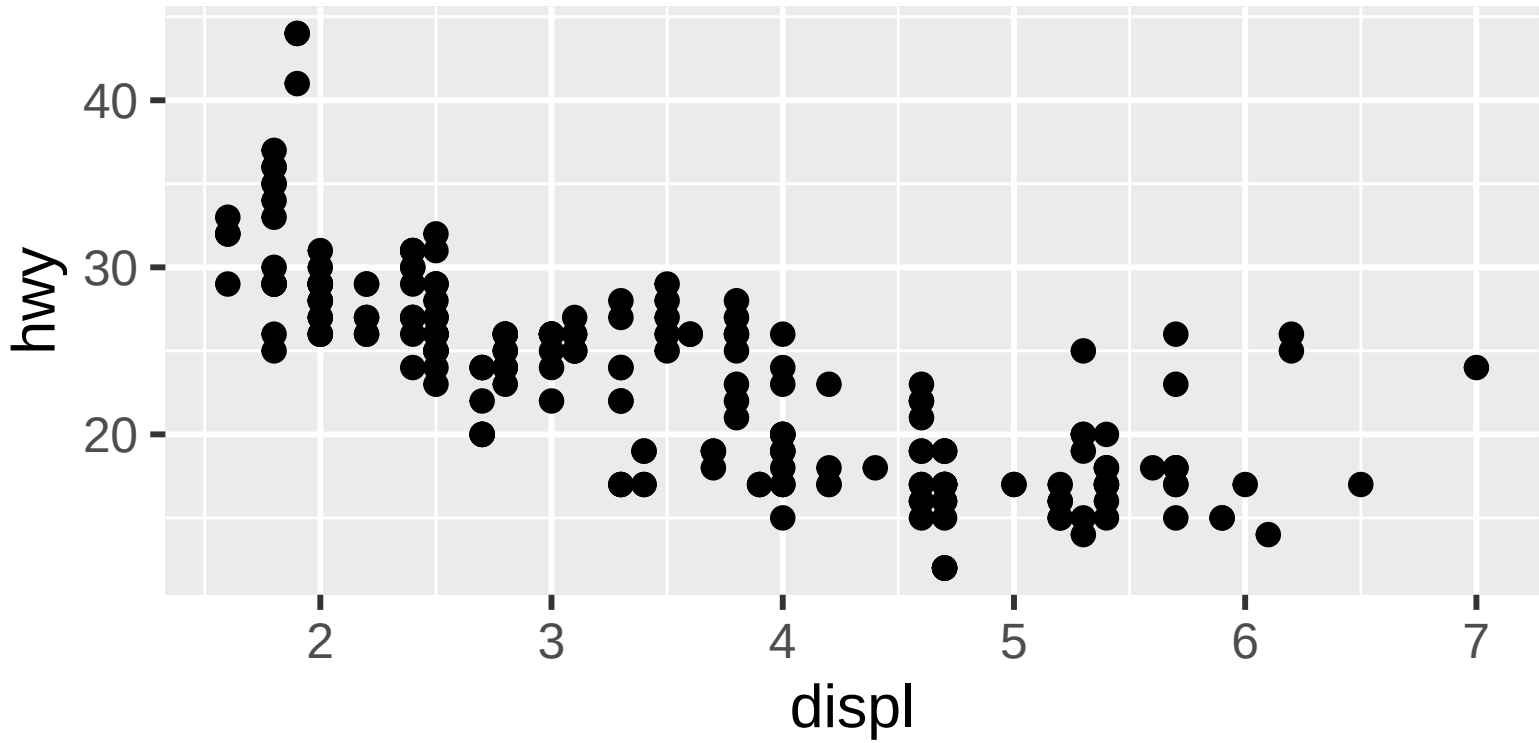


Gráfico de puntos

Por default, cada forma geométrica hereda las definiciones de las variables estéticas del principio. Por ejemplo, puedo definir *y* al principio y luego *x*:

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



Notar que no siempre hace falta poner data y mapping, son los dos primeros argumentos y agiliza el código.

Gráfico de puntos

Del mismo modo, por default, cada forma geométrica hereda la base de datos del principio. Sin embargo, cada forma geométrica puede tener su propia base de datos. Vamos a calcular la media de las variables hwy y displ por cada nivel de la variable drv:

```
# Divide la base según la variable drv
SplitMPG=split(mpg,mpg$drv)
# Calcula la media de hwy para cada valor de drv
mHWY.drv=sapply(SplitMPG, function(x){mean(x$hwy)})
mHWY.drv
```

```
##        4        f        r
## 19.17476 28.16038 21.00000
```

```
# Calcula la media de displ para cada valor de drv
mDISPL.drv=sapply(SplitMPG, function(x){mean(x$displ)})
mDISPL.drv
```

```
##        4        f        r
## 3.998058 2.558491 5.176000
```

Gráfico de puntos

Puedo juntar los datos de cada media en un data.frame:

```
Levs.drv=names(mHWY.drv);Levs.drv
```

```
## [1] "4" "f" "r"
```

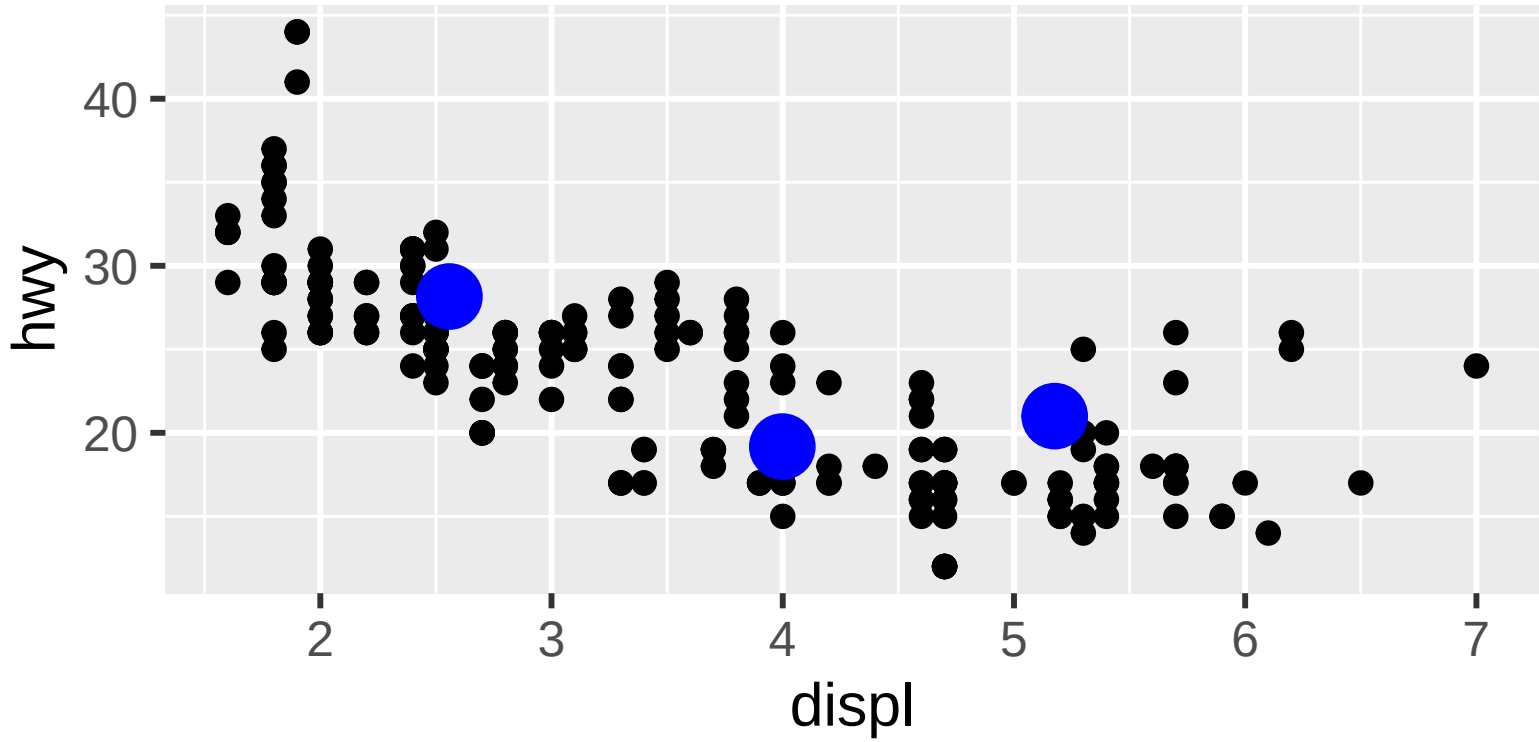
```
names(mHWY.drv)=NULL
names(mDISPL.drv)=NULL
MPG.drv=data.frame(DRV=Levs.drv,mHWY=mHWY.drv,mDISPL=mDISPL.drv)
MPG.drv
```

```
##   DRV     mHWY   mDISPL
## 1   4 19.17476 3.998058
## 2   f 28.16038 2.558491
## 3   r 21.00000 5.176000
```

Gráfico de puntos

Teniendo las medias por cada grupo, podemos agregar puntos donde se ubican estas medias:

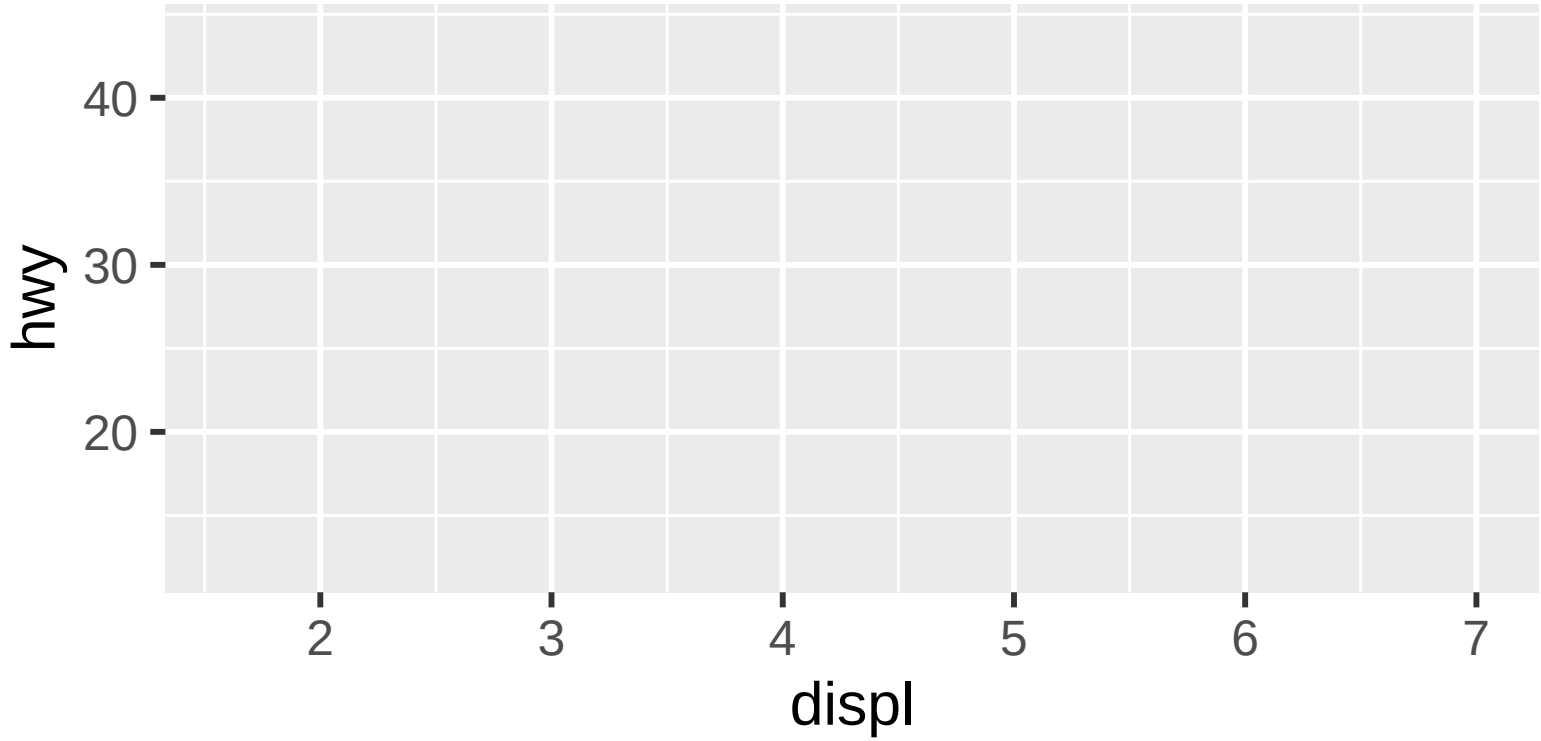
```
ggplot(data = mpg,aes(x=displ,y=hwy))+
  geom_point()+
  # Aquí sí es importante definir data=MPG.drv
  # porque difiere del original mpg
  geom_point(data = MPG.drv,
             aes(x=mDISPL,y=mHWY),
             color="blue",size=5)
```



Gráficos secuenciales

Por otro lado, esta forma secuencial de armar gráficos permite guardar un gráfico básico en una variable en el que se le van agregando capas:

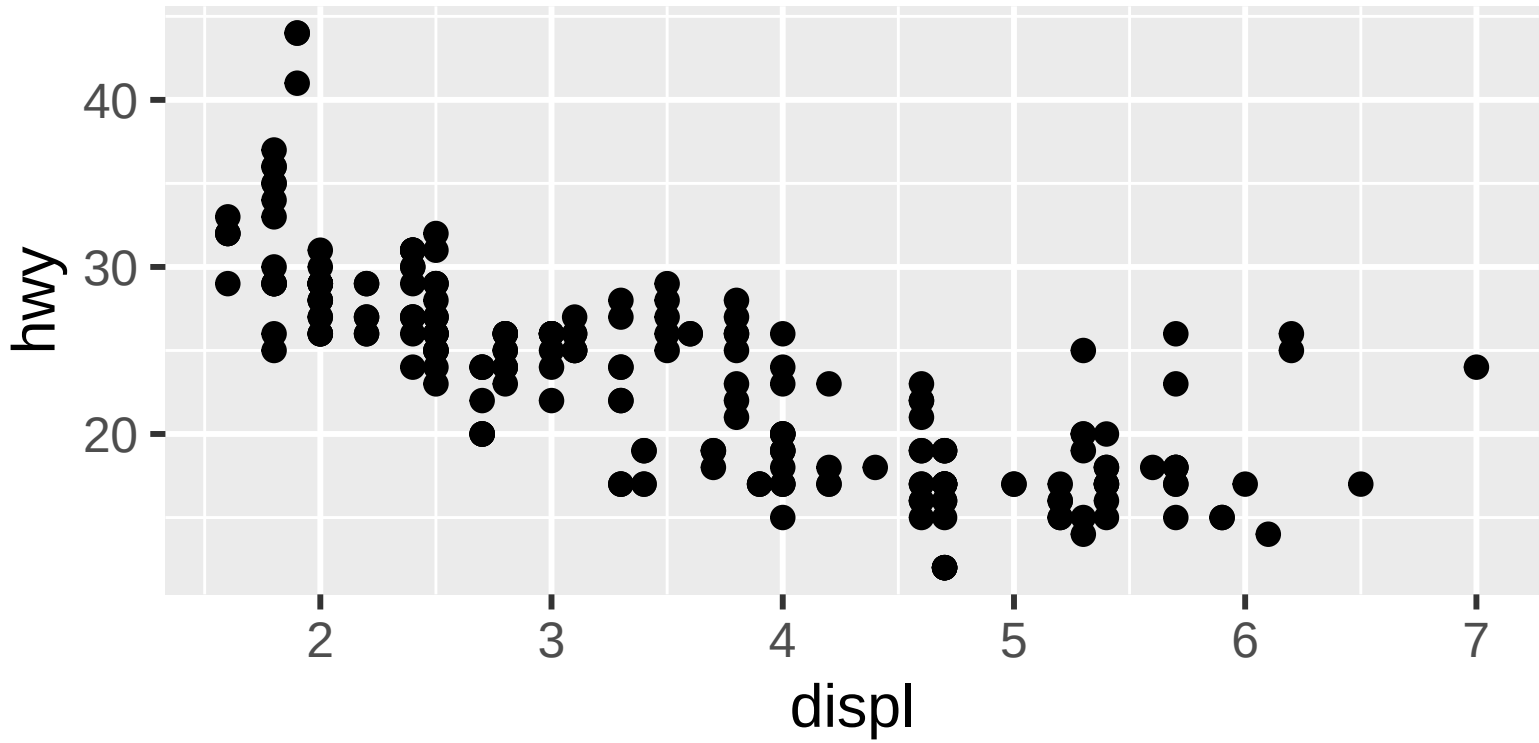
```
GG.Plano=ggplot(data = mpg,mapping = aes(x=displ,y=hwy))
GG.Plano
```



Gráficos secuenciales

Por otro lado, esta forma secuencial de armar gráficos permite guardar un gráfico básico en una variable en el que se le van agregando capas:

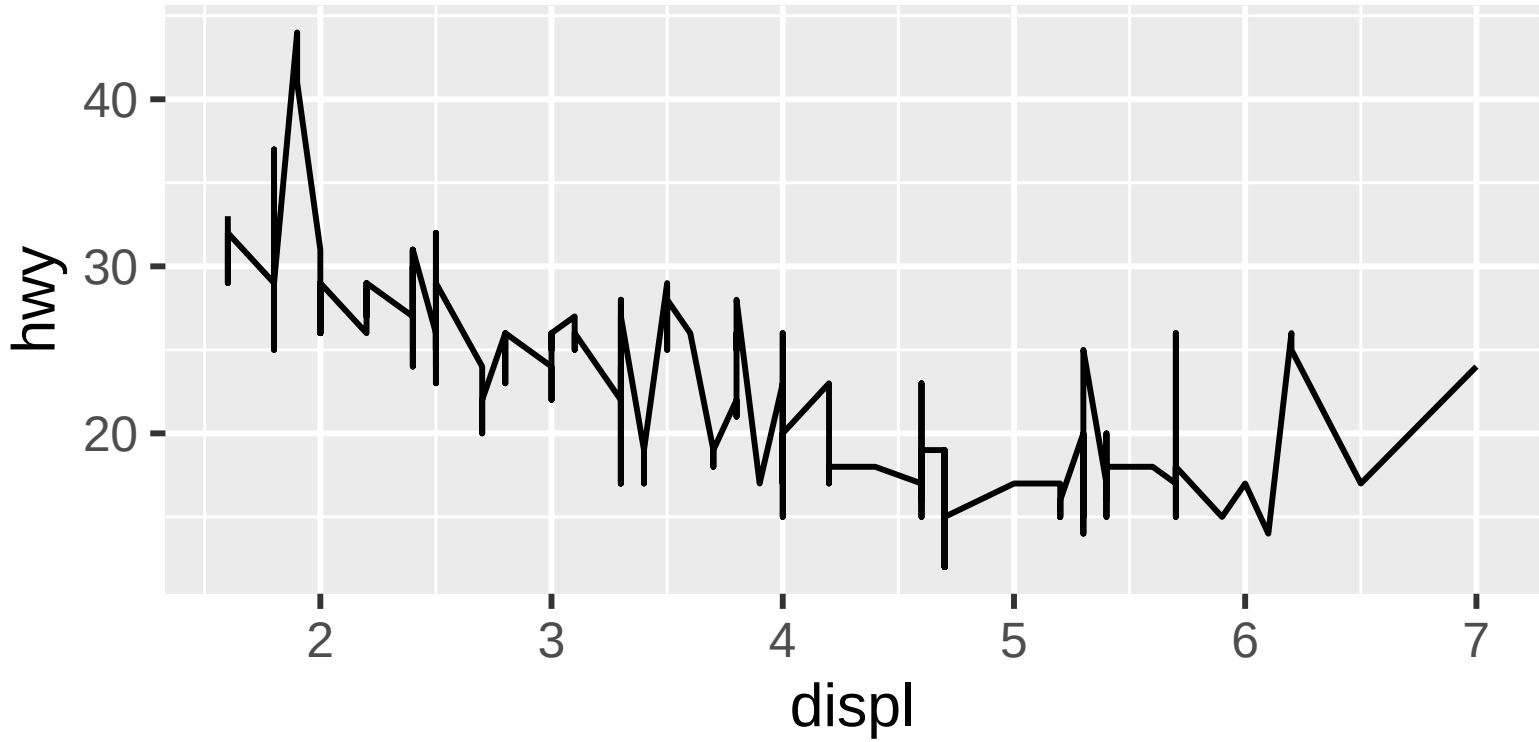
```
GG.Scatter=GG.Plano+
  geom_point()
GG.Scatter
```



Gráficos secuenciales

Por otro lado, esta forma secuencial de armar gráficos permite guardar un gráfico básico en una variable en el que se le van agregando capas:

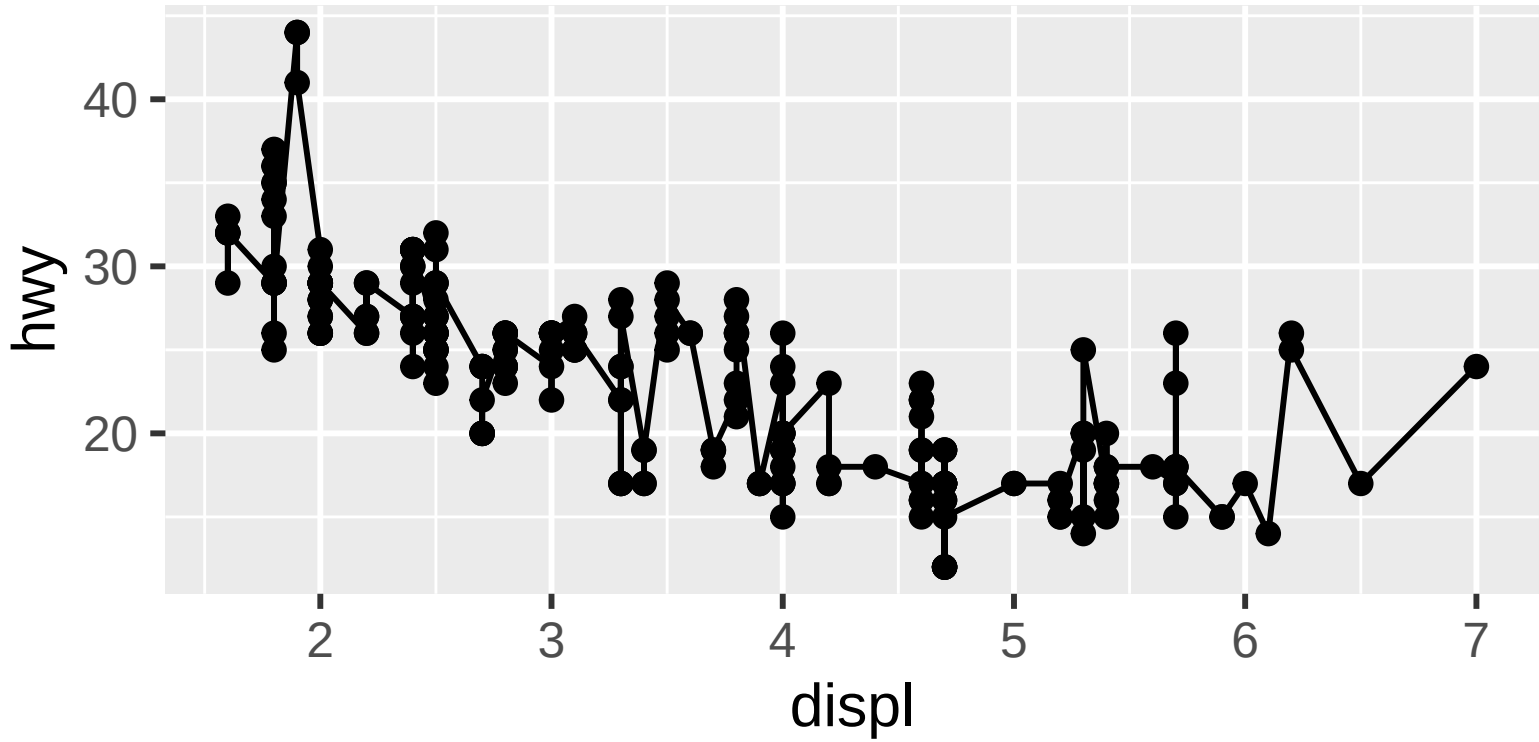
```
GG.Plano+
  geom_line()
```



Gráficos secuenciales

Por otro lado, esta forma secuencial de armar gráficos permite guardar un gráfico básico en una variable en el que se le van agregando capas:

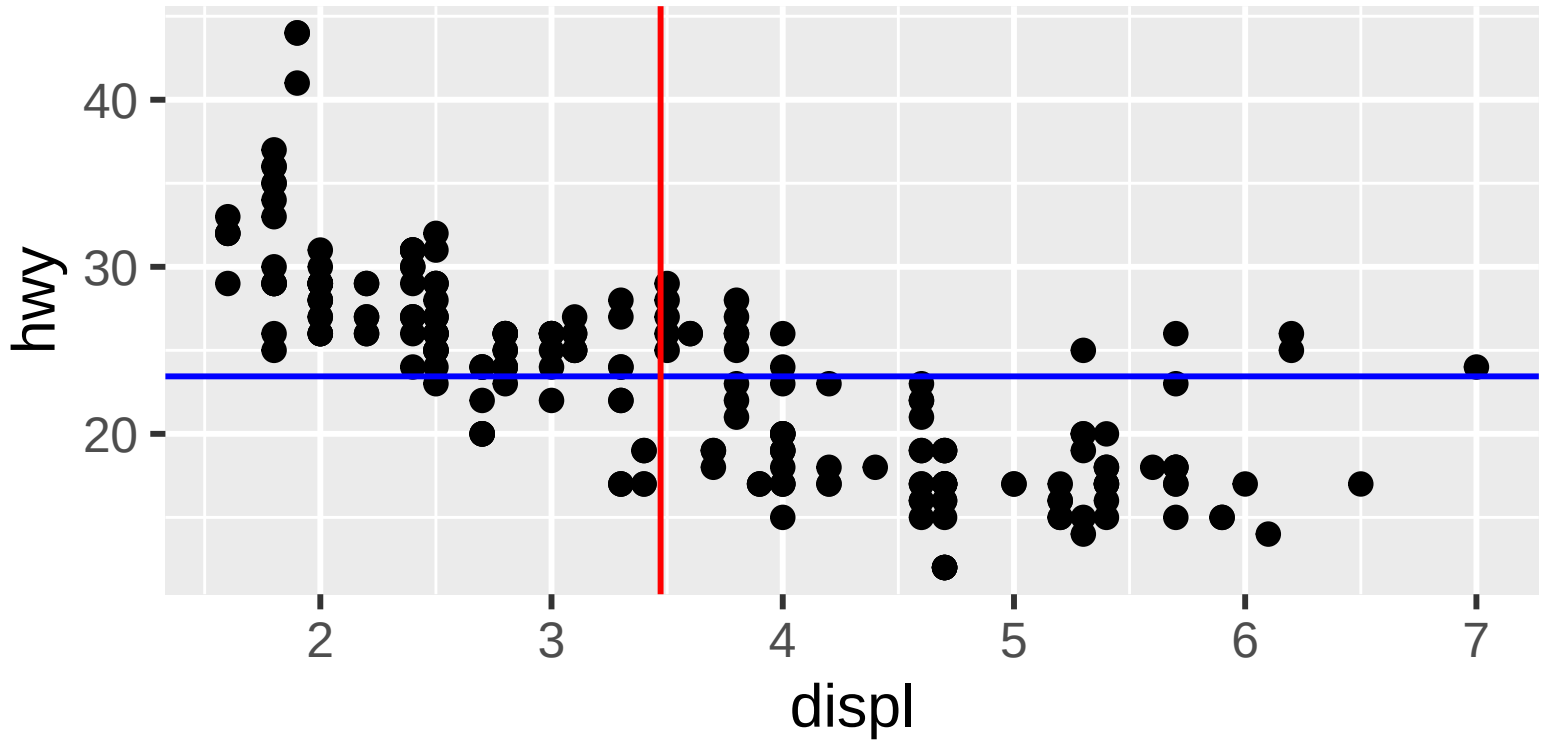
```
GG.Scatter+
  geom_line()
```



Gráficos secuenciales

Por otro lado, esta forma secuencial de armar gráficos permite guardar un gráfico básico en una variable en el que se le van agregando capas:

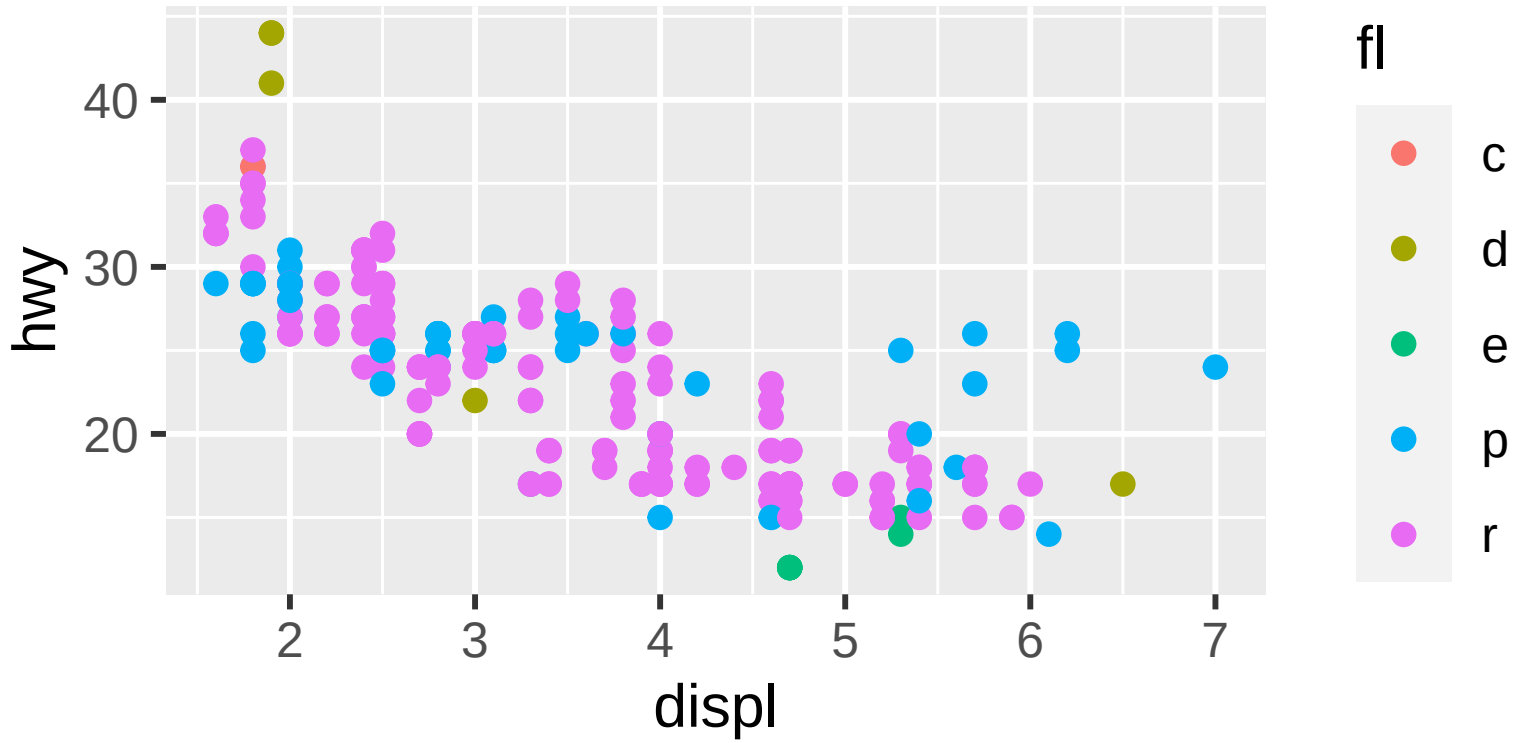
```
GG.Scatter+
  geom_hline(aes(yintercept=mean(hwy)),color="blue")+
  geom_vline(aes(xintercept=mean(displ)),color="red")
```



Otras variables estéticas: Color

Por ejemplo, también podemos hacer que distintas variables determinen las características de los gráficos:

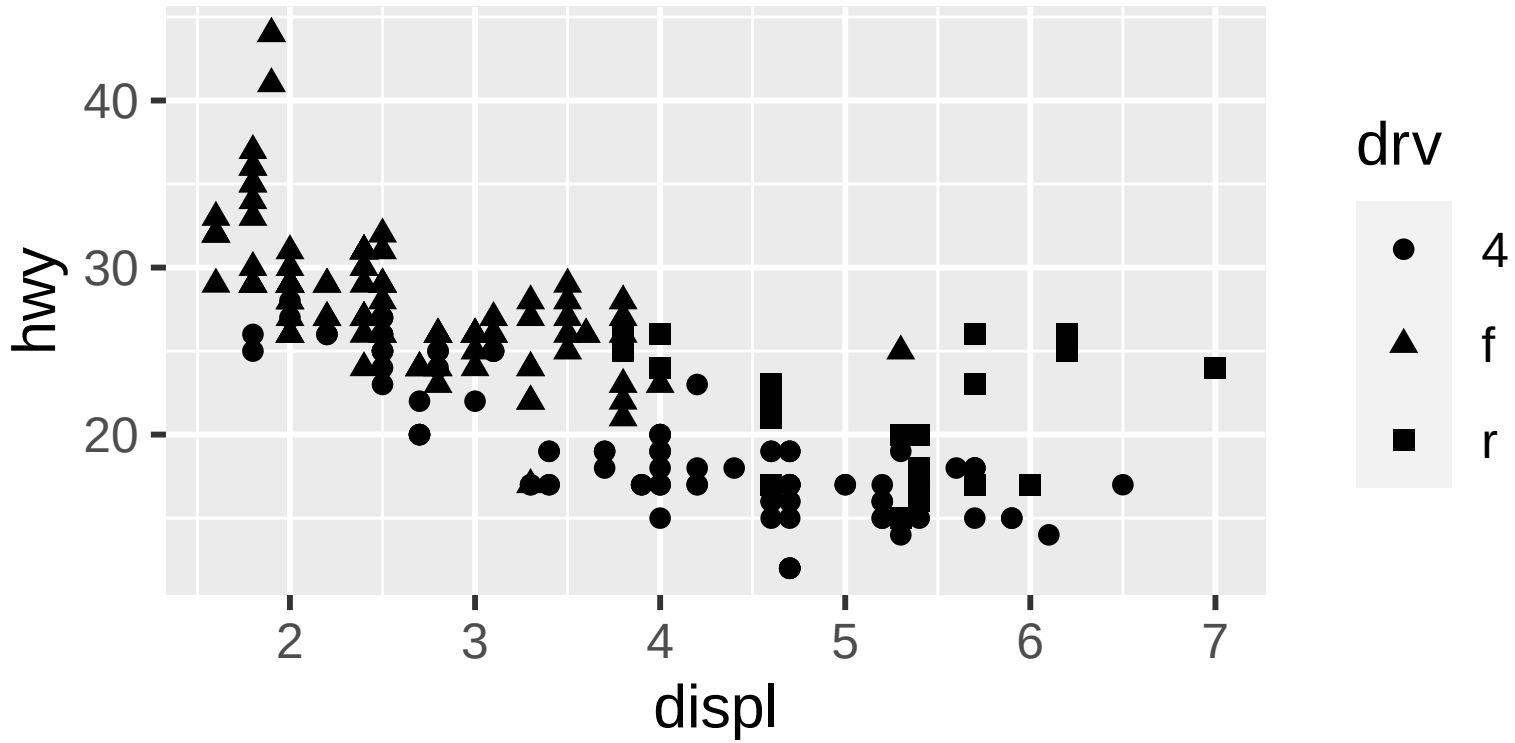
```
ggplot(data = mpg,
       mapping = aes(x=displ,y=hwy,color=fl))+
  geom_point()
```



Otras variables estéticas: Forma

Por ejemplo, también podemos hacer que distintas variables determinen las características de los gráficos:

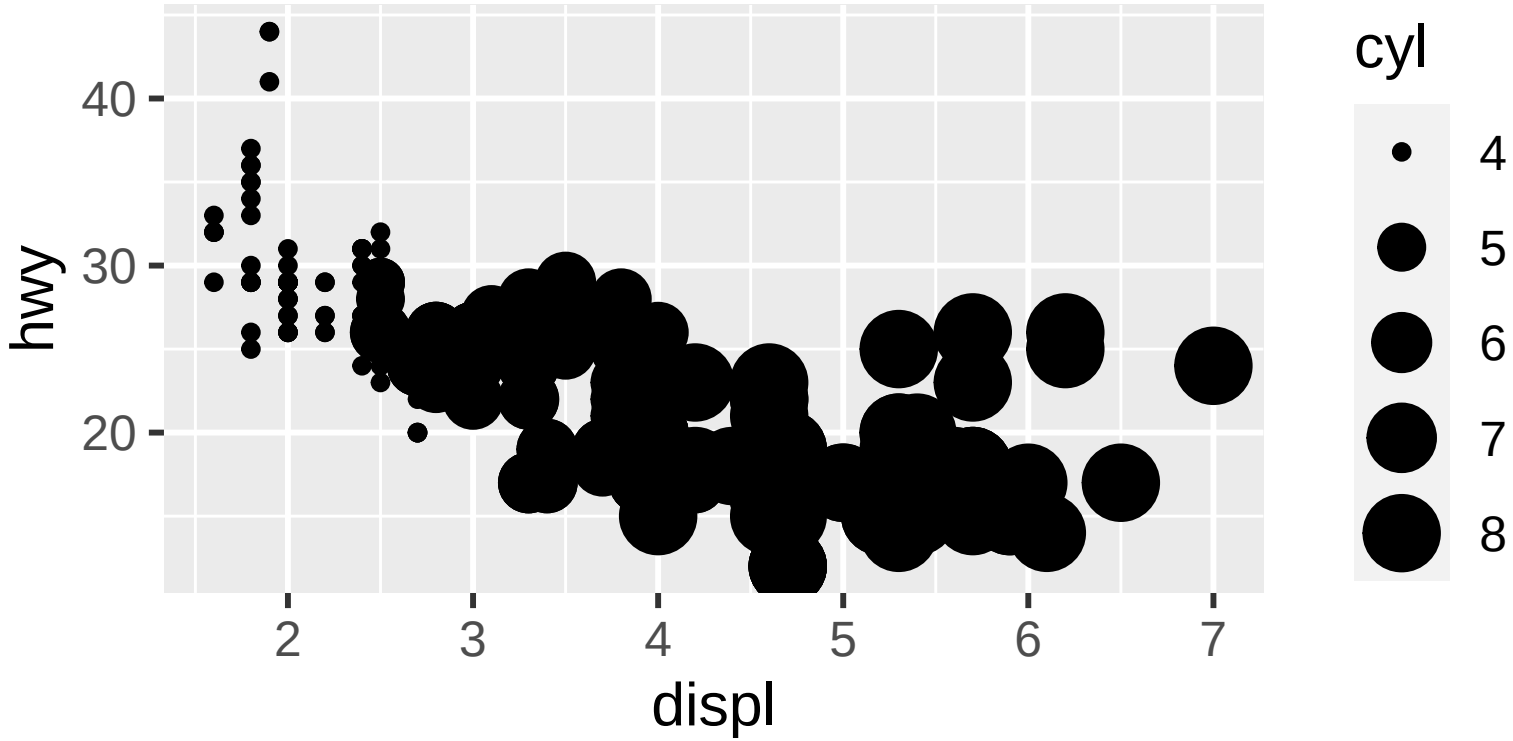
```
ggplot(data = mpg,mapping = aes(x=displ,y=hwy,shape=drv))+
  geom_point()
```



Otras variables estéticas: Tamaño

Por ejemplo, también podemos hacer que distintas variables determinen las características de los gráficos:

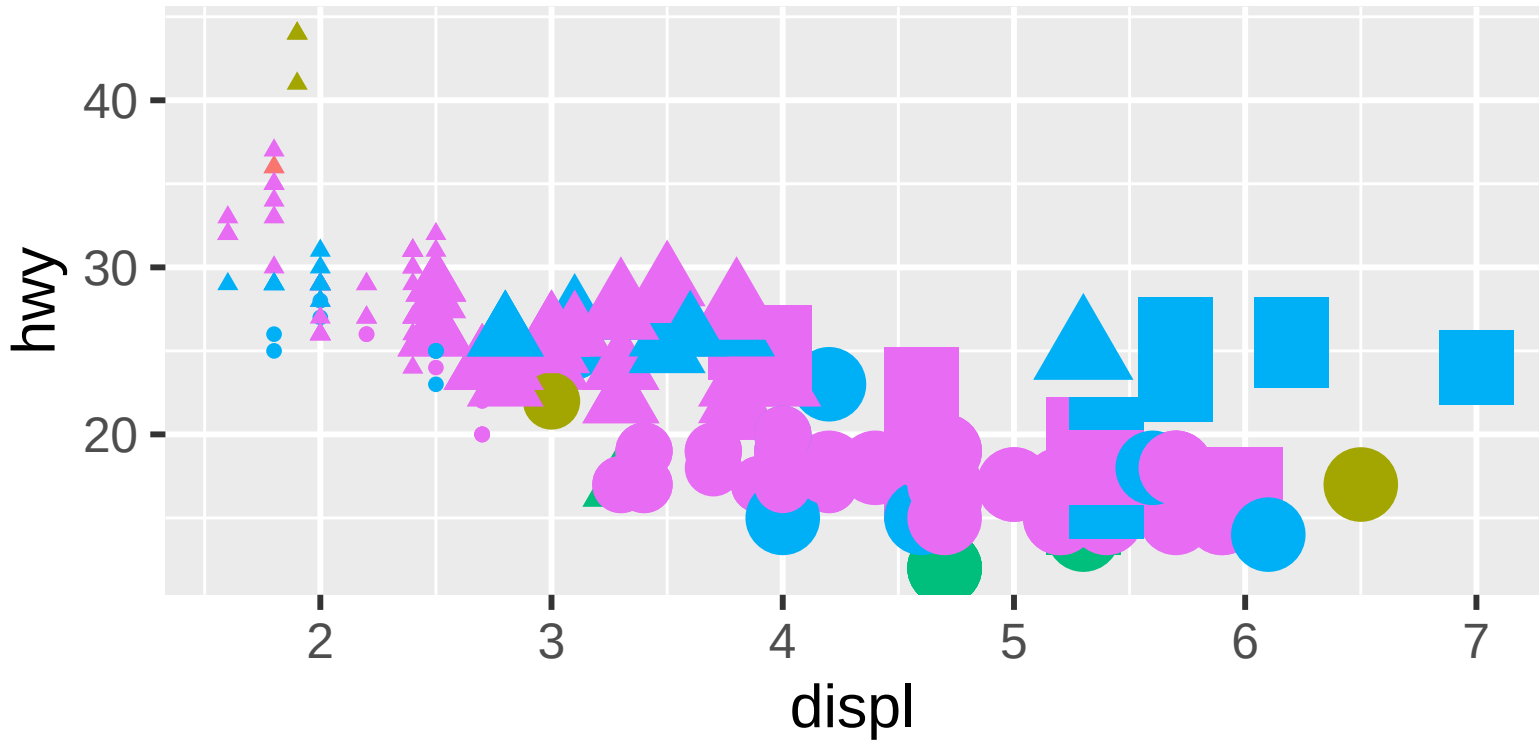
```
ggplot(data = mpg,mapping = aes(x=displ,y=hwy,size=cyl))+
  geom_point()
```



Otras variables estéticas: All together now

Por ejemplo, también podemos hacer que distintas variables determinen las características de los gráficos (no siempre de forma informativa):

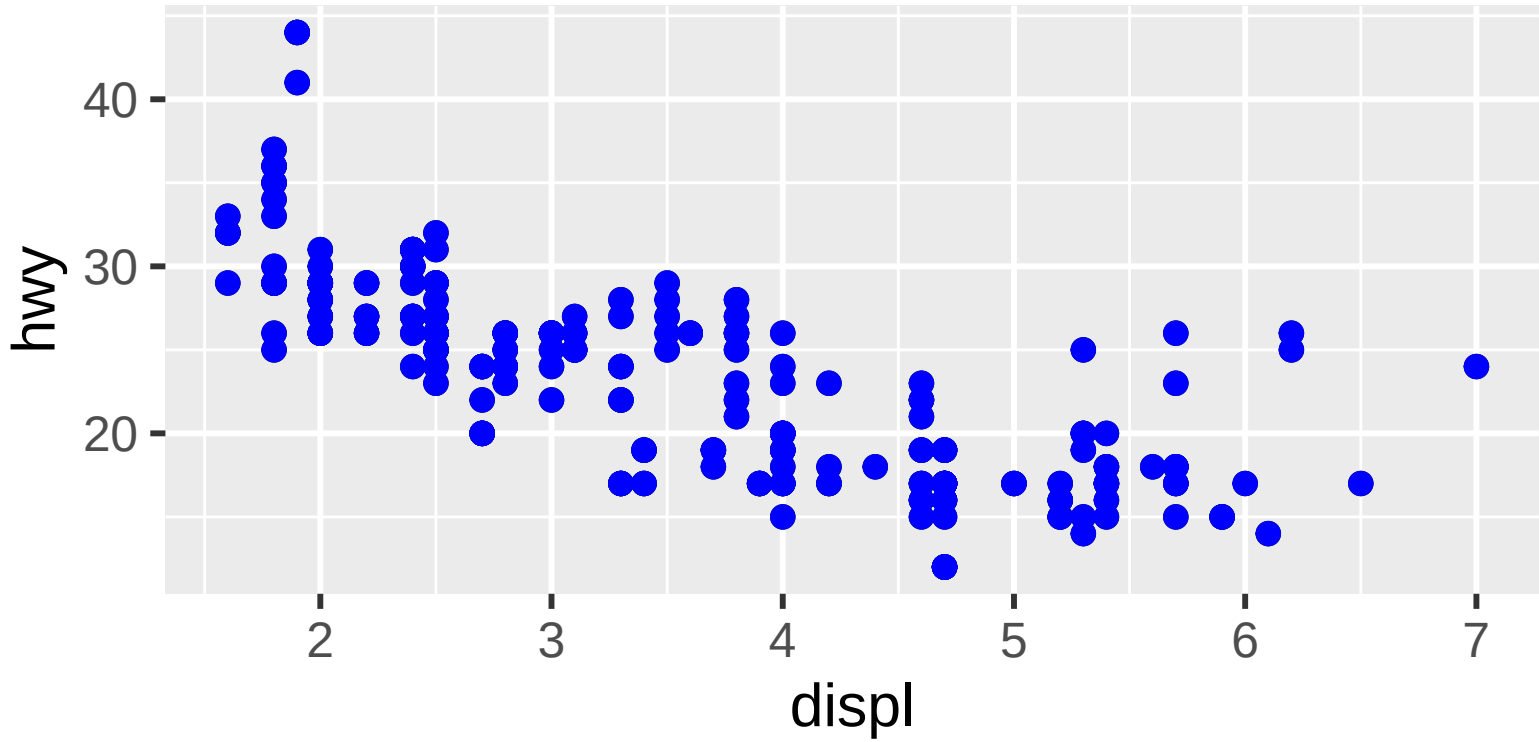
```
ggplot(data = mpg,
       aes(x=displ,y=hwy,color=fl,size=cyl,shape=drv))+
  geom_point()+
  theme(legend.position = "none" )
```



Estéticas constantes: Color

No siempre las variables estéticas deben ser variables, pueden ser constantes, en este caso, no hace falta poner el comando aes porque no se consideran variables estéticas:

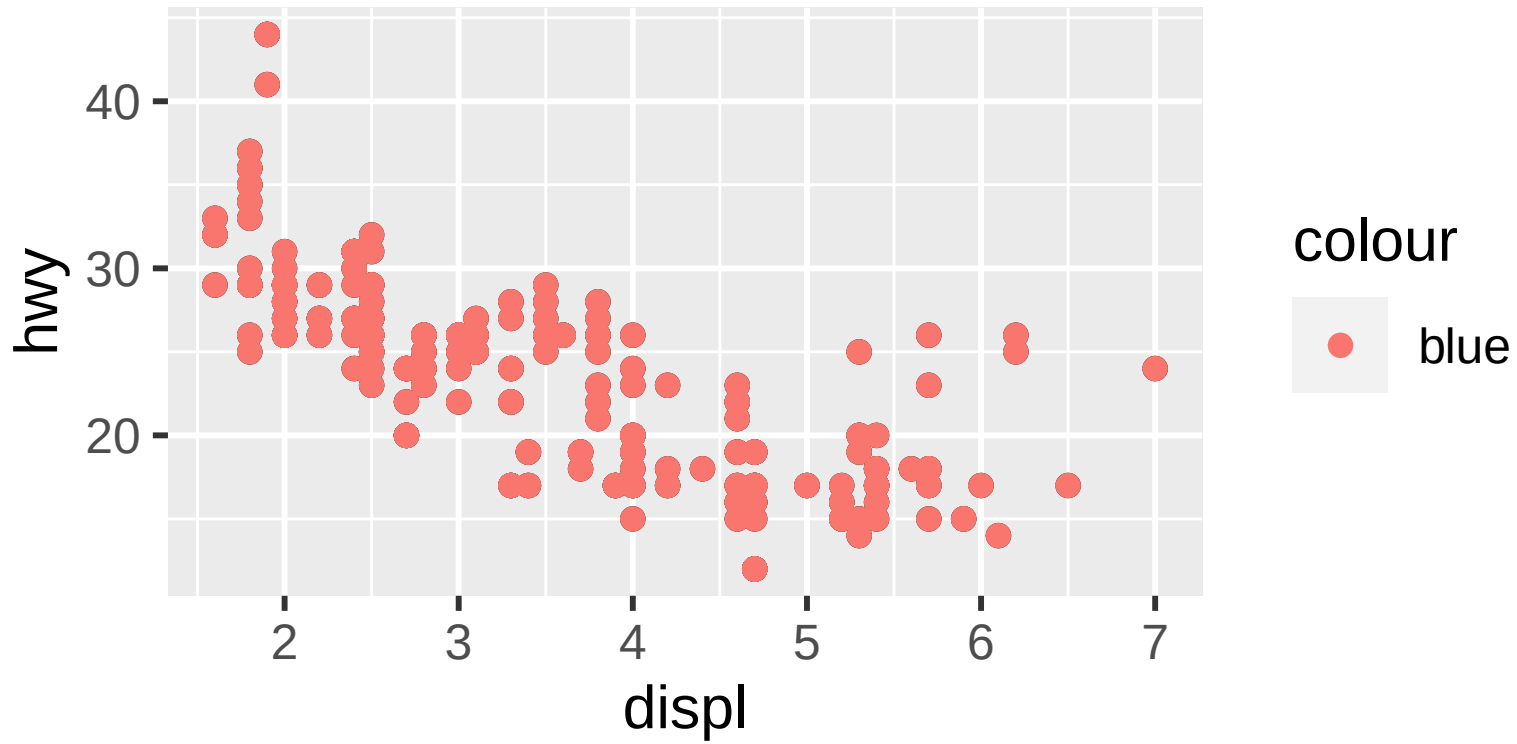
```
GG.Scatter+
  geom_point(color="blue")
```



Estéticas constantes: Color

Si se quiere poner como variable estética, la considera una variable de un solo valor y por lo tanto, devuelve una leyenda:

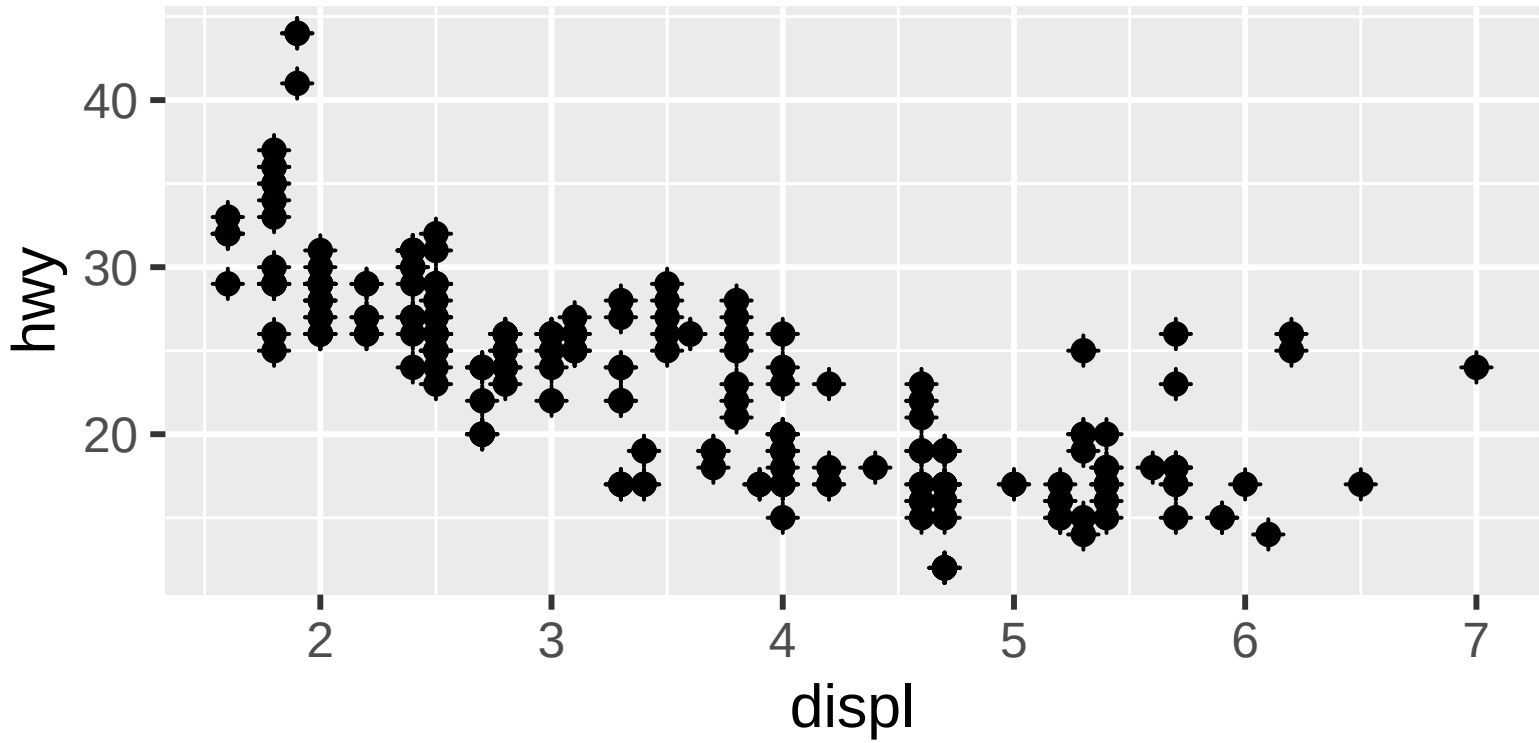
```
GG.Scatter+
  geom_point(aes(color="blue"))
```



Estéticas constantes: Forma

No siempre las cualidades estéticas deben ser variables, pueden ser constantes:

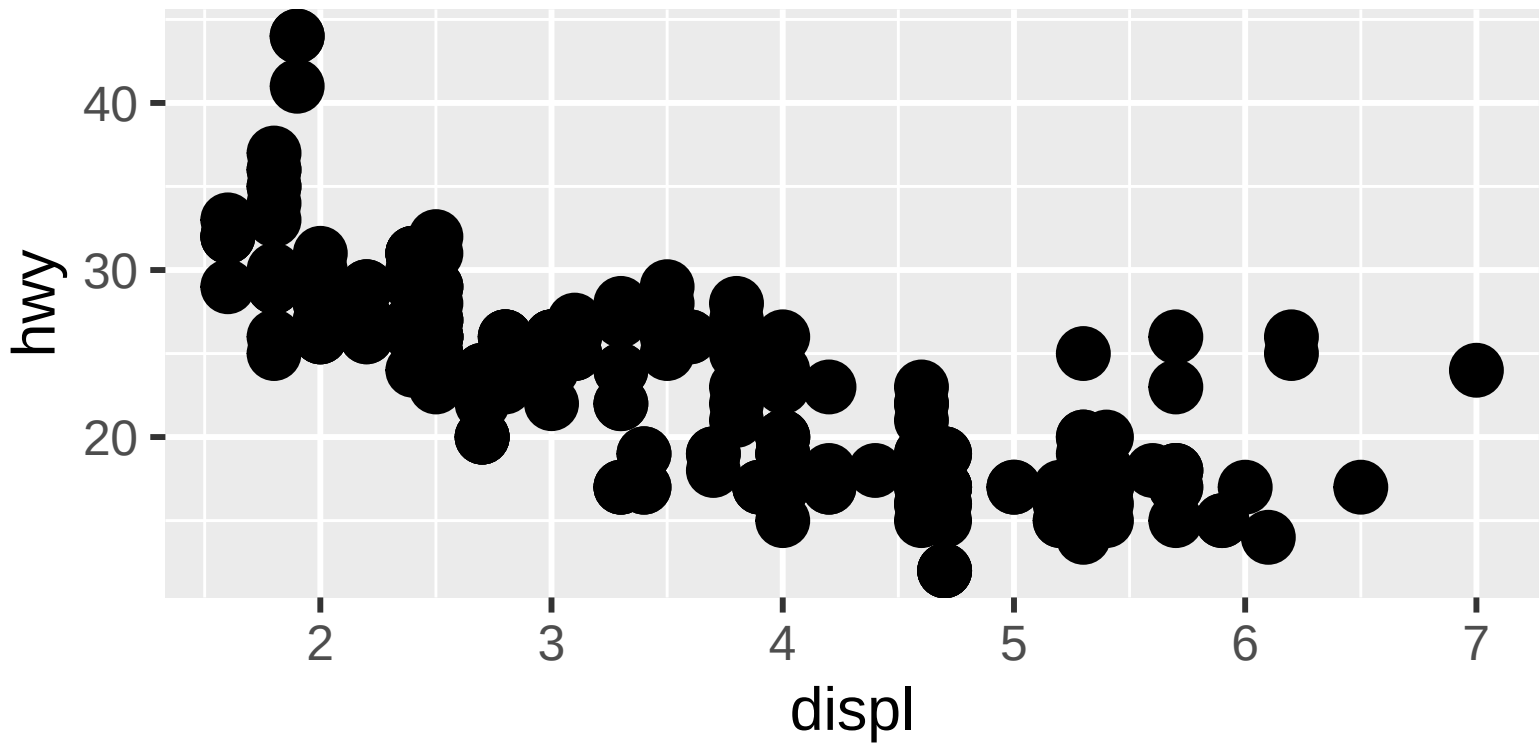
```
GG.Scatter+
  geom_point(shape=3)
```



Estéticas constantes: Tamaño

No siempre las variables estéticas deben ser variables, pueden ser constantes:

```
GG.Scatter+
  geom_point(size=4)
```

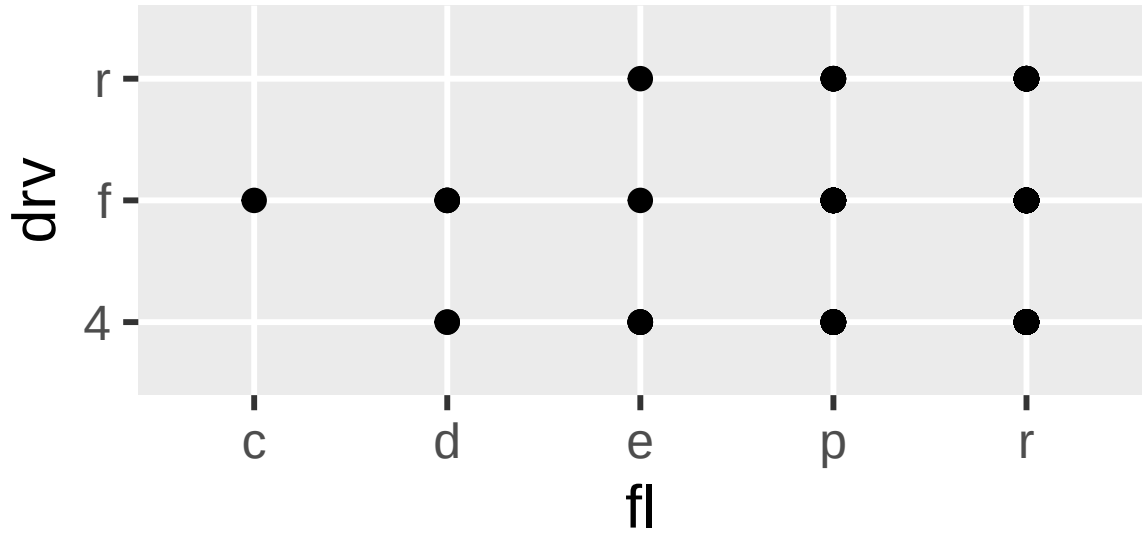


Overplotting

Variables cualitativas.

Nuevamente, como suele pasar en R, ggplot maneja con versatilidad en el caso de que las variables sean cualitativas:

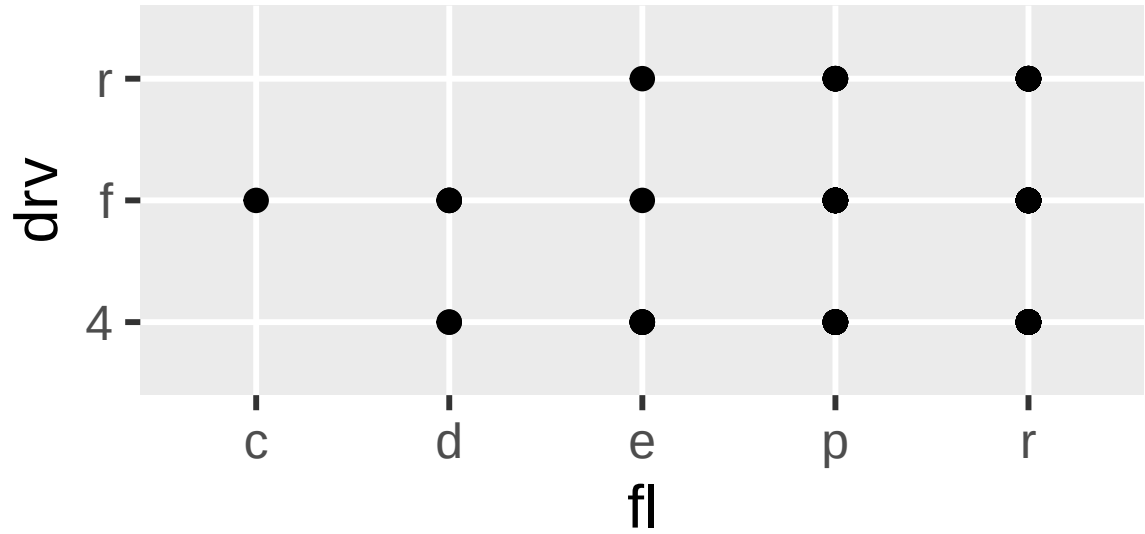
```
GG.Cat=ggplot(data = mpg,aes(x=fl,y=drv))
GG.Cat+
  geom_point()
```



Variables cualitativas.

Nuevamente, como suele pasar en R, ggplot maneja con versatilidad en el caso de que las variables sean cualitativas:

```
GG.Cat=ggplot(data = mpg,aes(x=fl,y=drv))
GG.Cat+
  geom_point()
```

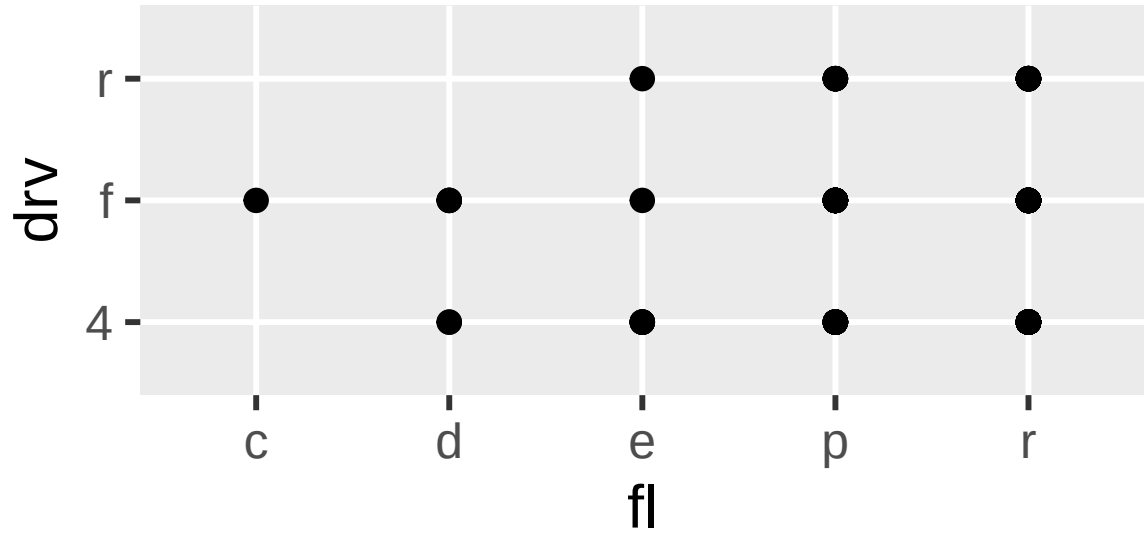


Aquí inventa una escala discreta separada de forma unitaria.

Variables cualitativas.

Nuevamente, como suele pasar en R, ggplot maneja con versatilidad en el caso de que las variables sean cualitativas:

```
GG.Cat=ggplot(data = mpg,aes(x=fl,y=drv))
GG.Cat+
  geom_point()
```



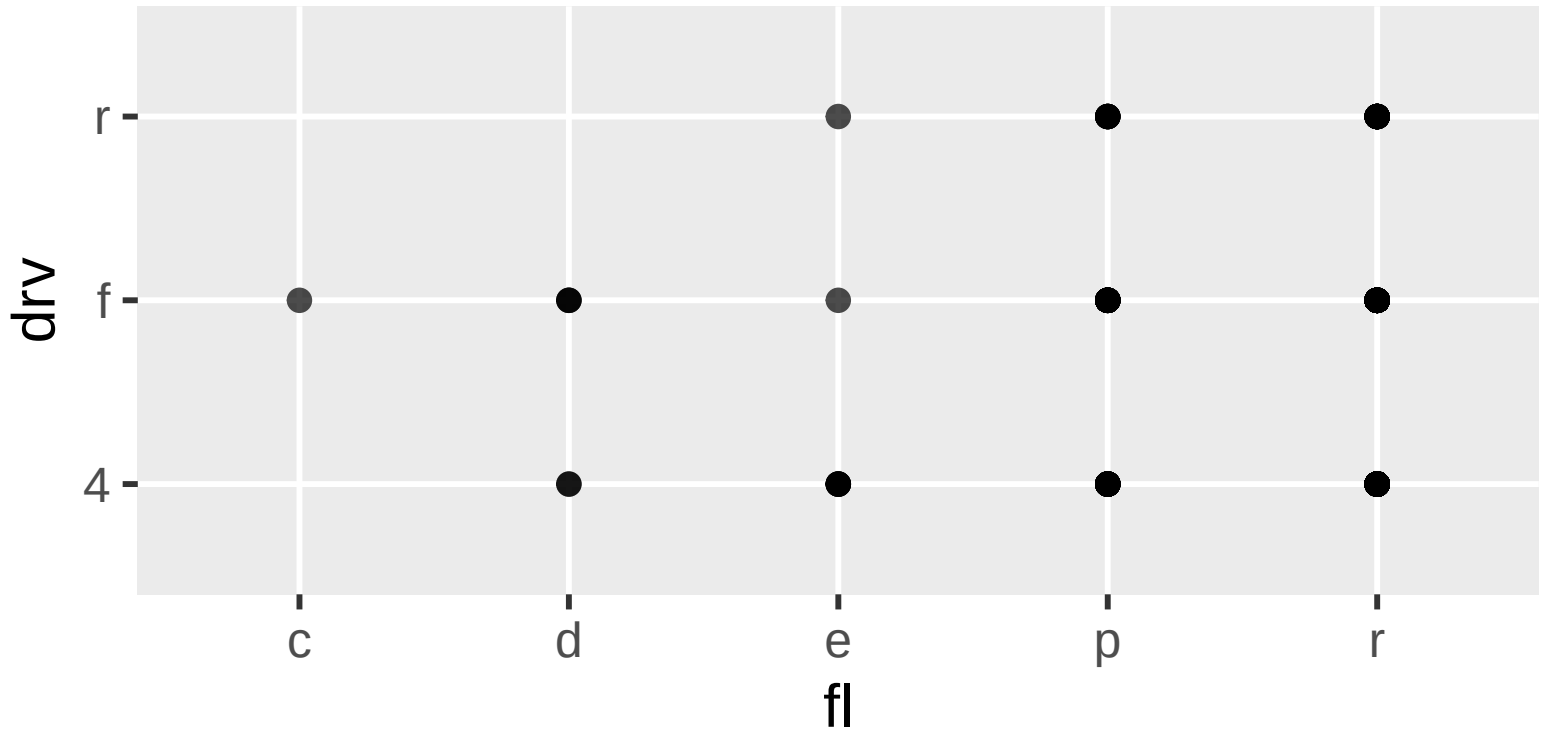
Aquí inventa una escala discreta separada de forma unitaria.

Sin embargo, notemos que todos los puntos se superponen entre sí, por lo que no está informando cómo se concentran los datos. Este fenómeno se llama overplotting.

Opacidad

Una forma de evitar esta concentración es agregando un parámetro de opacidad alpha (entre 0 y 1):

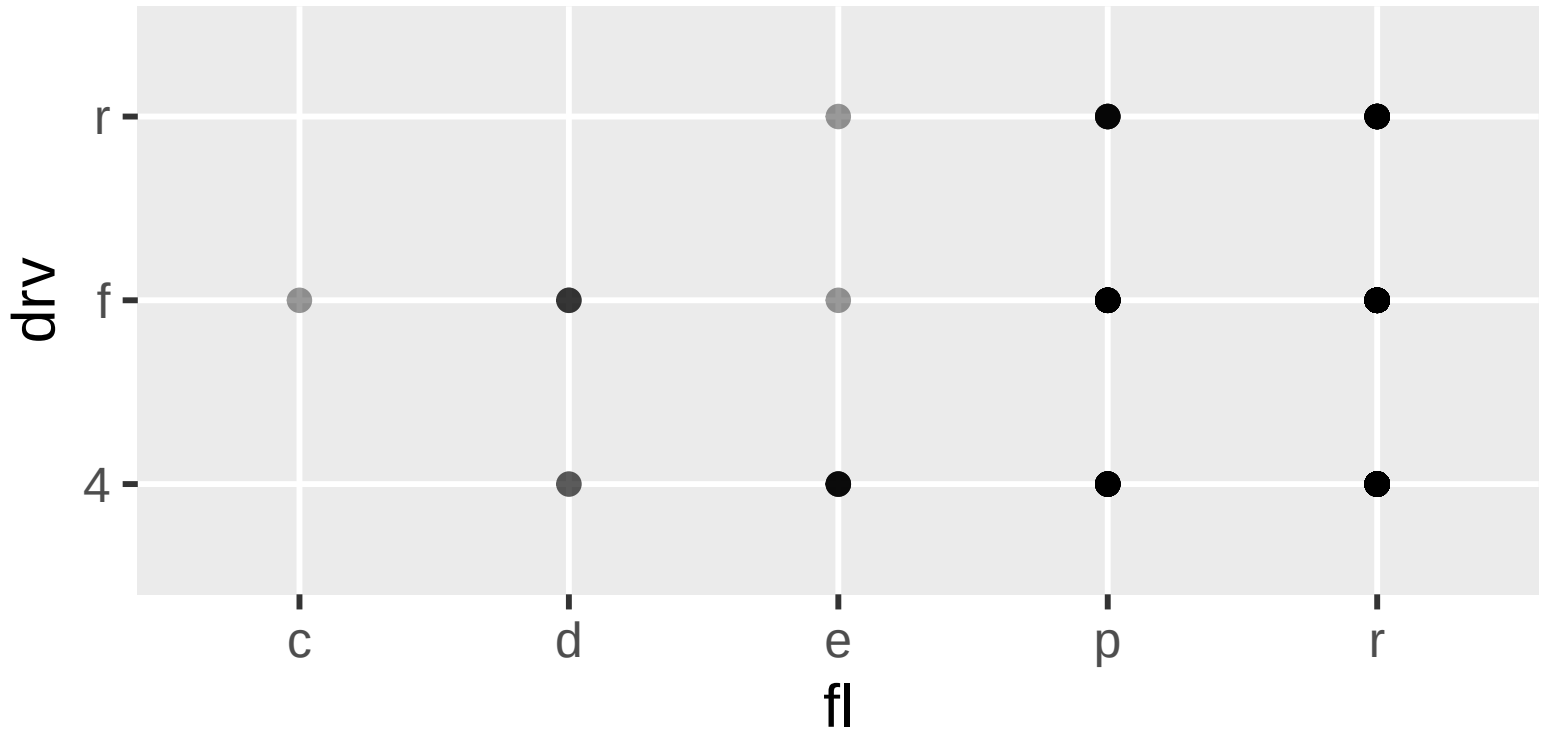
```
GG.Cat+
  geom_point(alpha=0.7)
```



Opacidad

Una forma de evitar esta concentración es agregando un parámetro de opacidad alpha (entre 0 y 1):

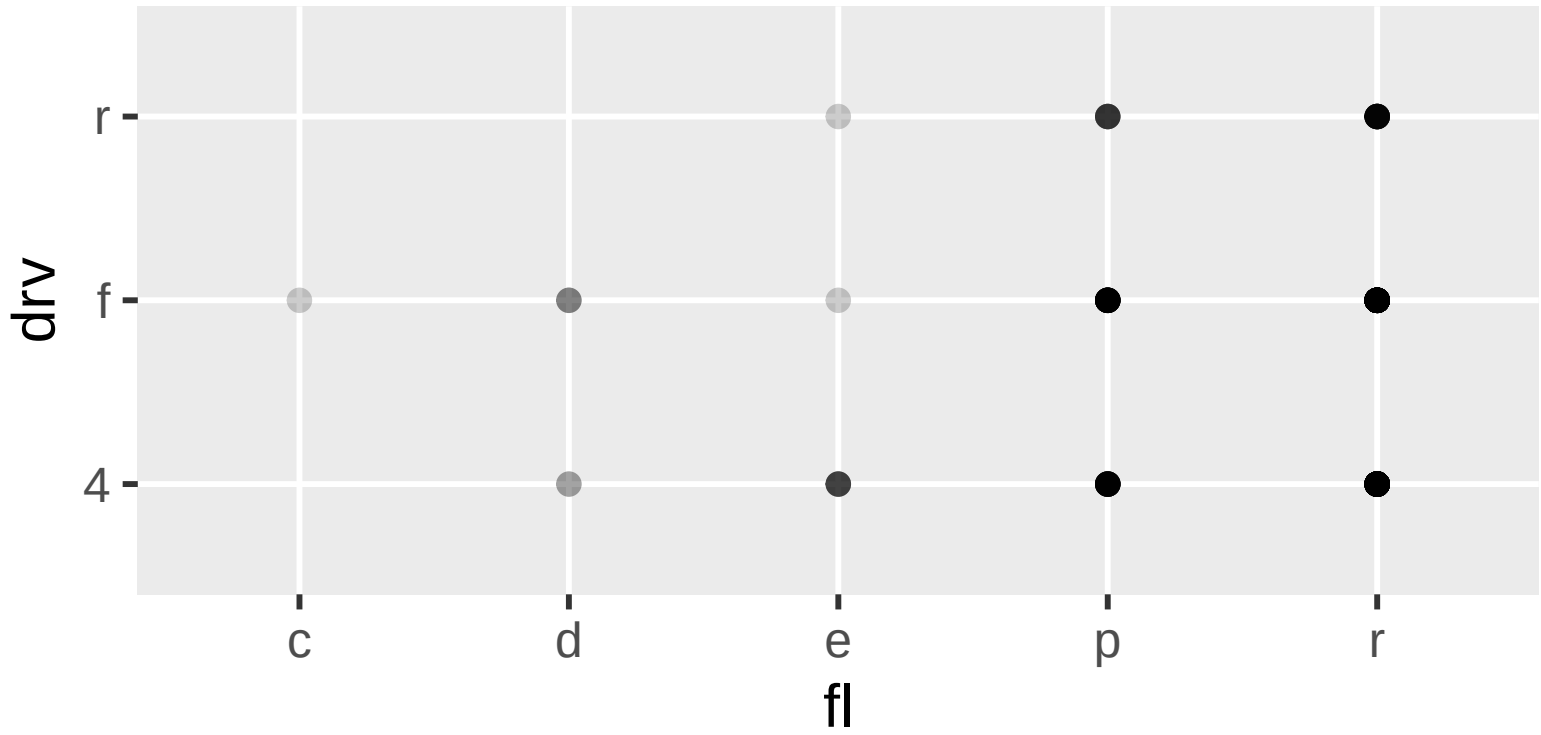
```
GG.Cat+
  geom_point(alpha=0.4)
```



Opacidad

Una forma de evitar esta concentración es agregando un parámetro de opacidad alpha (entre 0 y 1):

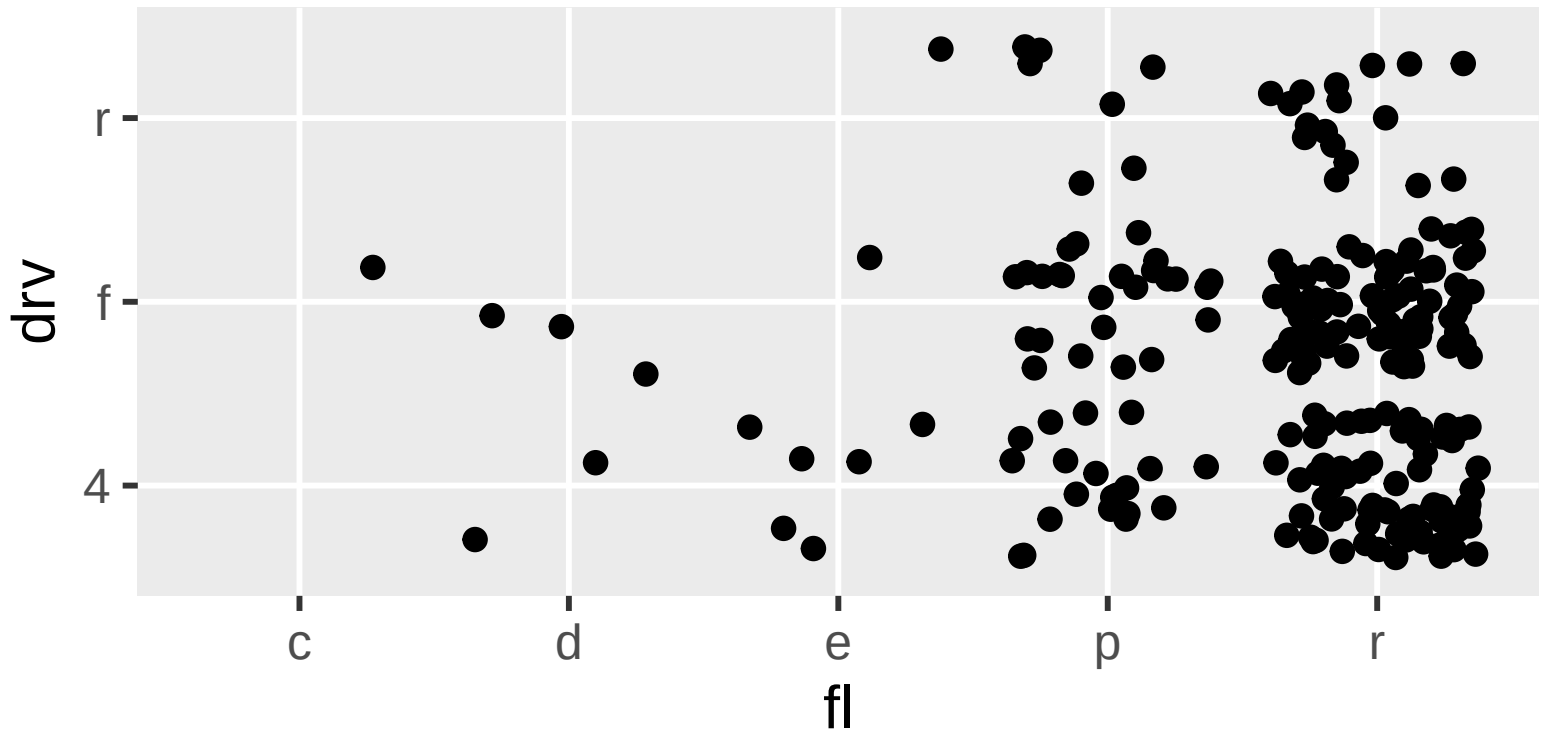
```
GG.Cat+
  geom_point(alpha=0.2)
```



Dispersión

Una forma de evitar esta concentración es agregando un parámetro que permite dispersar los datos jitter:

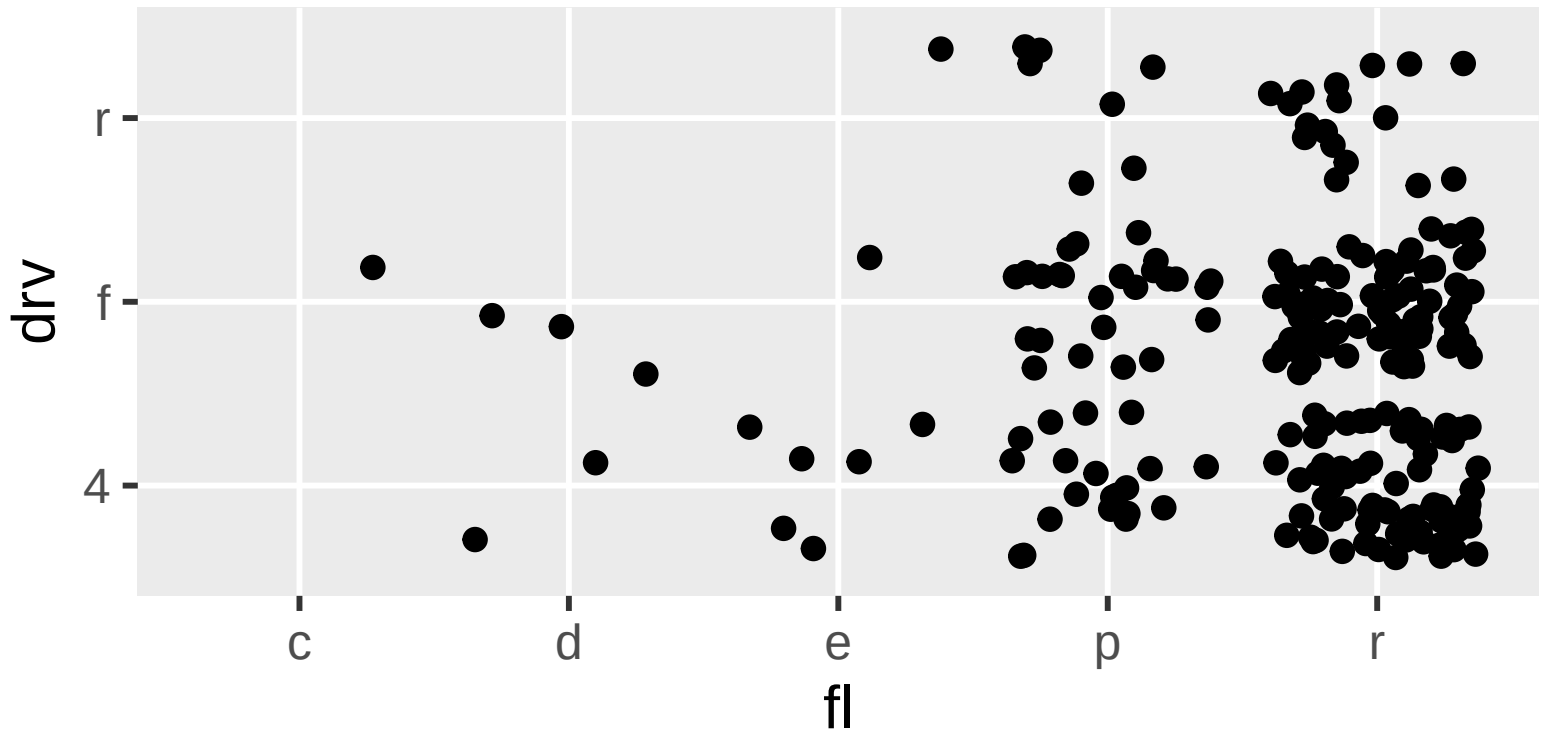
```
GG.Cat+
  geom_point(position = "jitter")
```



Dispersión

Una forma de evitar esta concentración es agregando un parámetro que permite dispersar los datos jitter:

```
GG.Cat+
  geom_point(position = "jitter")
```

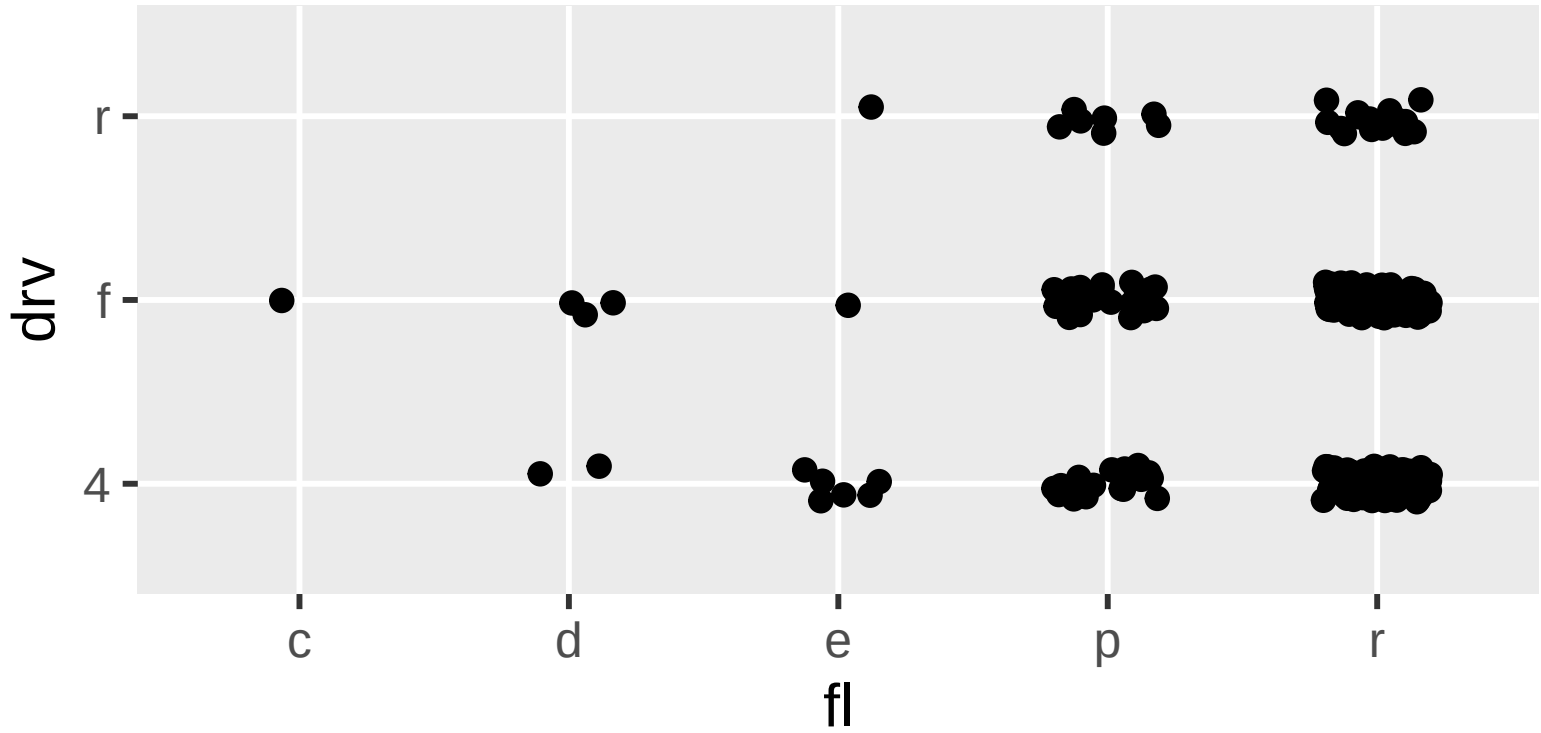


Se dispersan demasiado y no permite visualizar dónde se concentran más.

Dispersión

Se pueden ajustar los parámetros de dispersión:

```
GG.Cat+
  geom_point(position = position_jitter(width=0.2,height = 0.1))
```



Dispersión

El jitter tiene su propio comando geom:

```
GG.Cat+
  geom_jitter(width=0.2,height = 0.1)
```

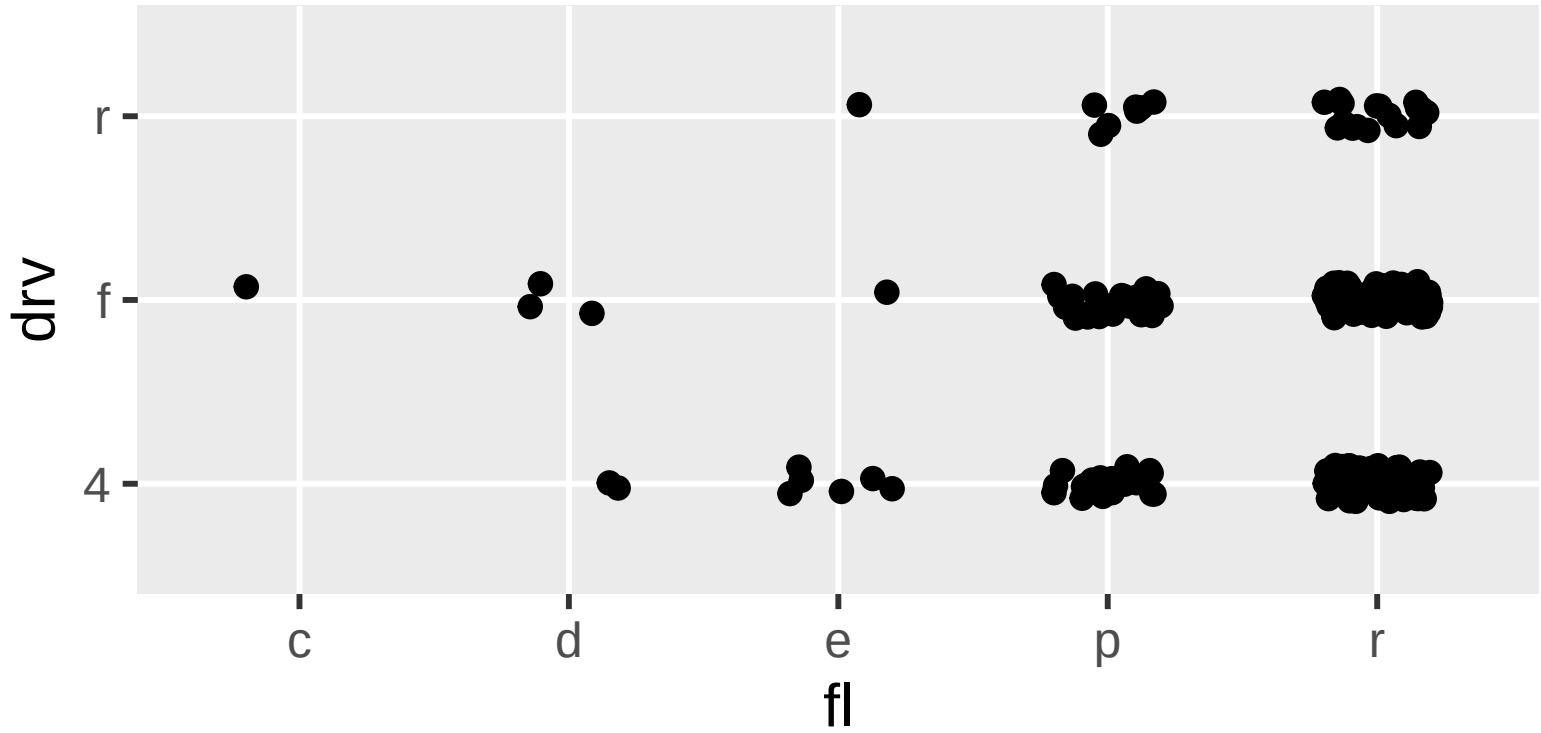


Gráfico de texto

Aquí tenemos otro ejemplo en el que en vez de sumarle una capa de puntos, podemos sumarle una capa con texto, cuyo output está especificado en la variable estética "label":

```
GG.Scat.dc=ggplot(data = mpg,mapping = aes(x=displ,y=hwy,color=drv))
GG.Scat.dc+
  geom_text(aes(label=cyl))
```

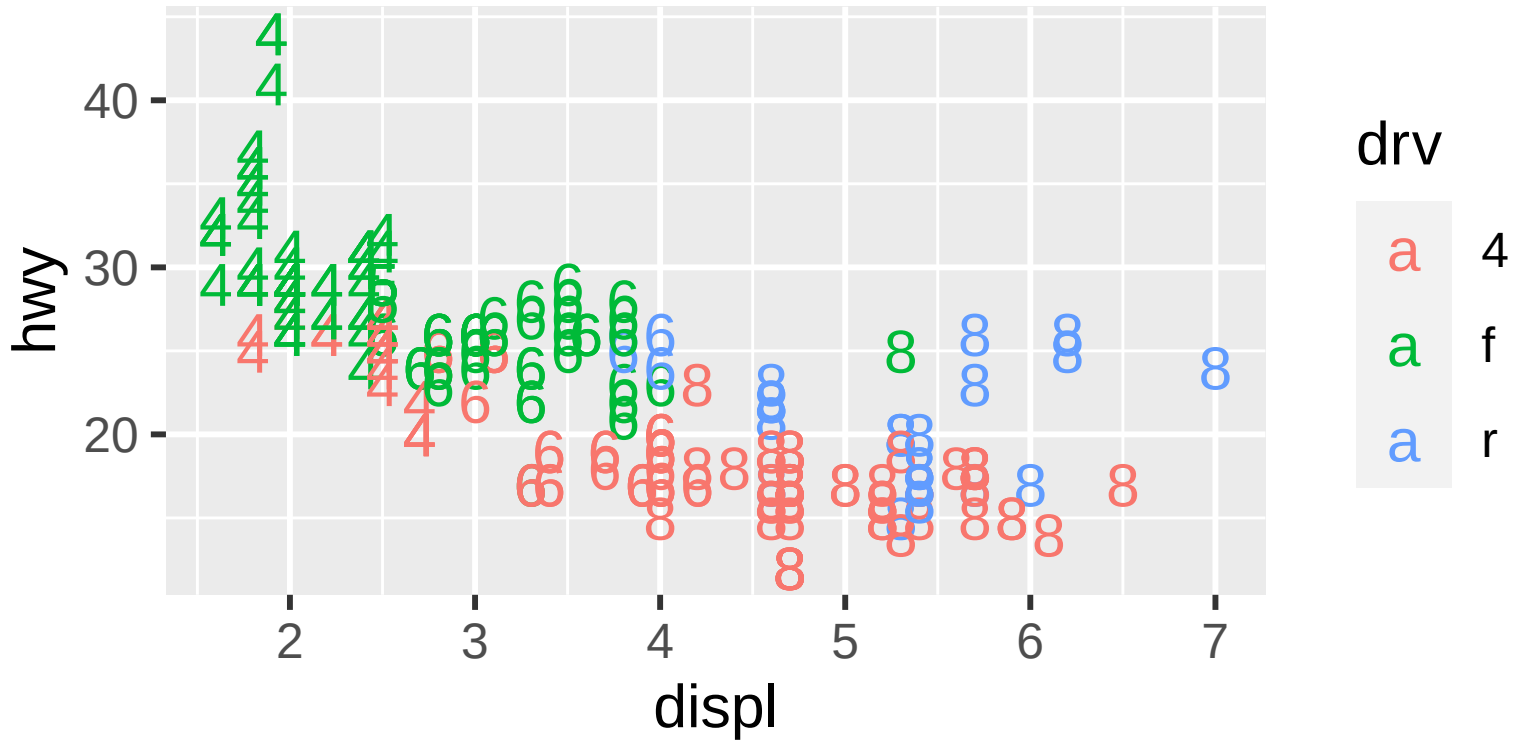


Gráfico de texto

También en este caso se pueden sacar los elementos que se superponen para poder visualizar mejor:

```
GG.Scat.dc+
  geom_text(aes(label=cyl),check_overlap = T)
```

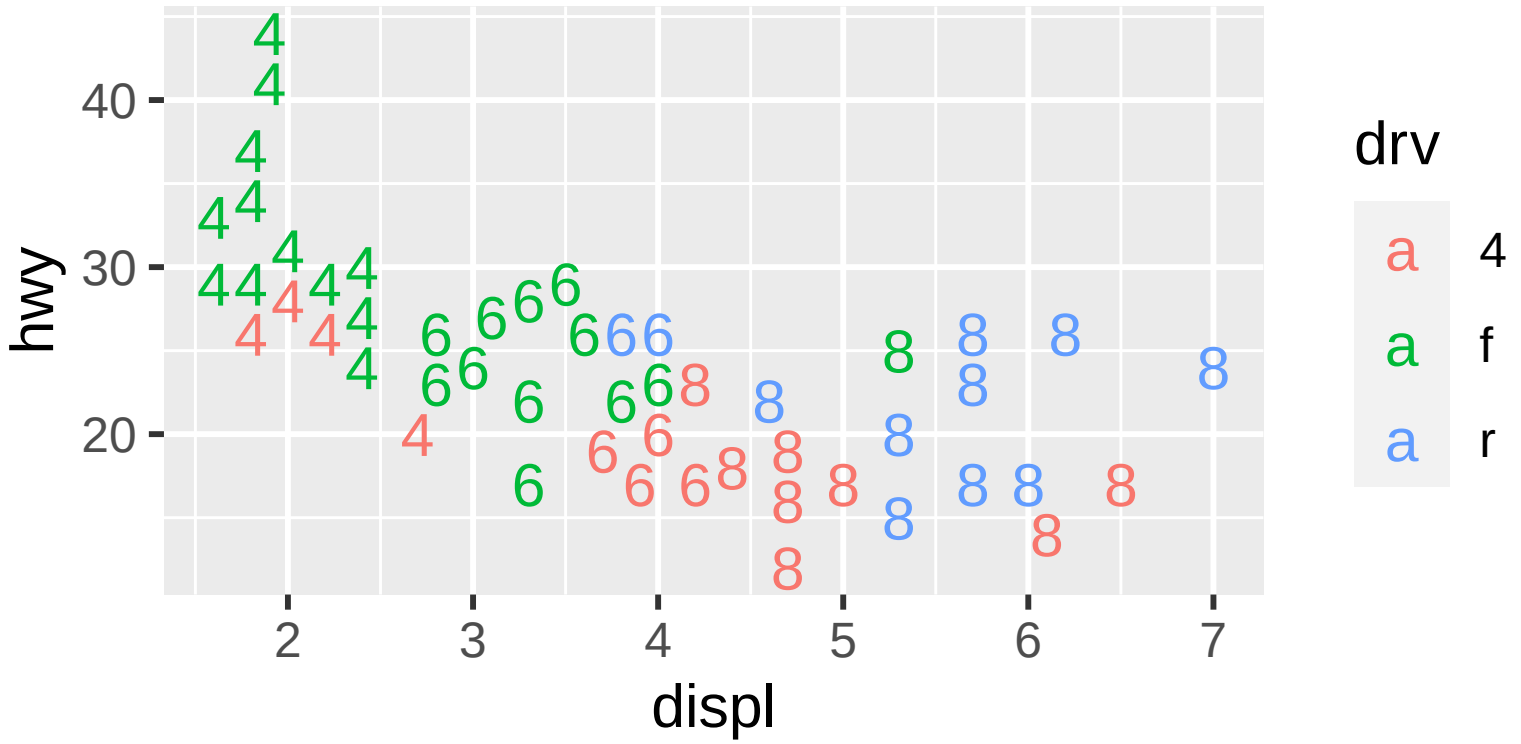
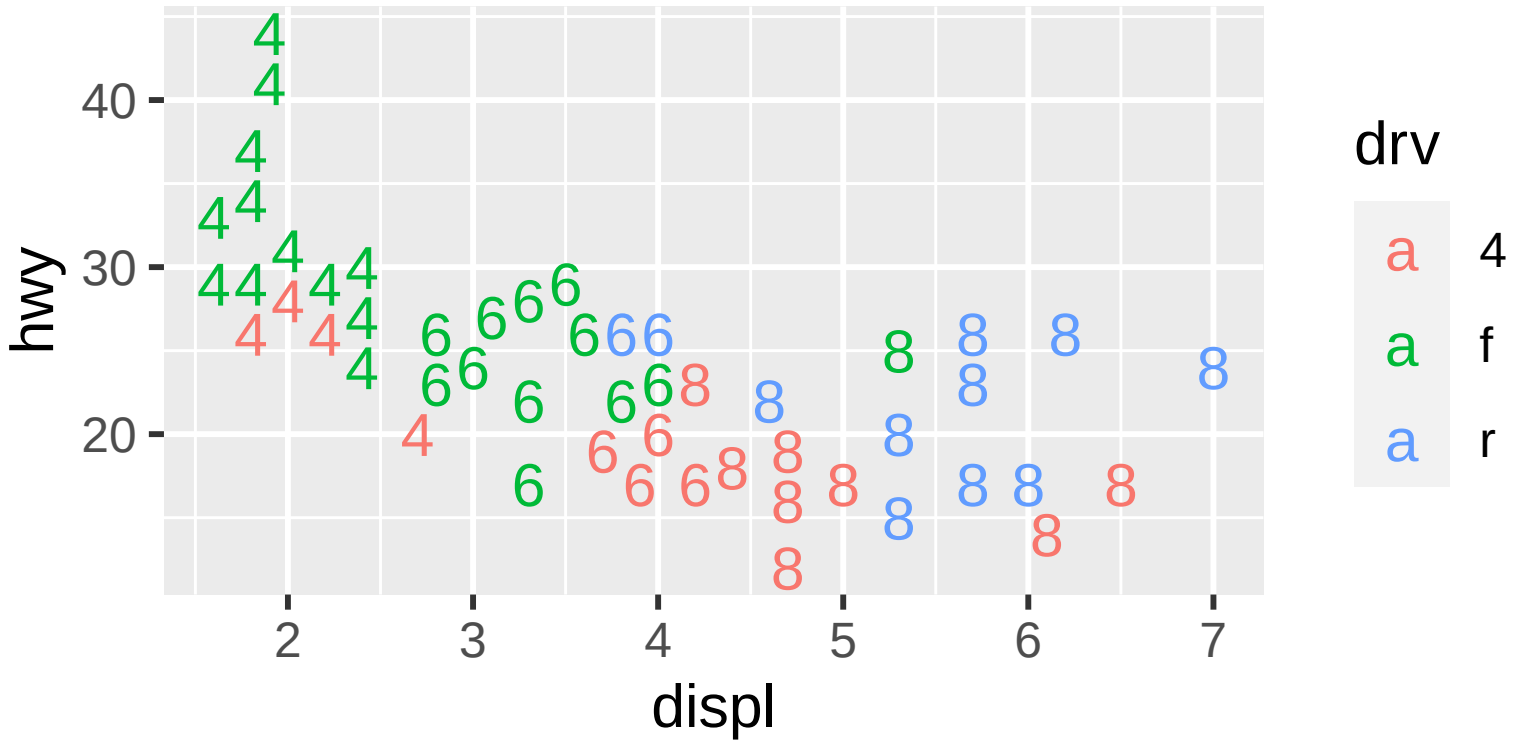


Gráfico de texto

También en este caso se pueden sacar los elementos que se superponen para poder visualizar mejor:

```
GG.Scat.dc+
  geom_text(aes(label=cyl),check_overlap = T)
```

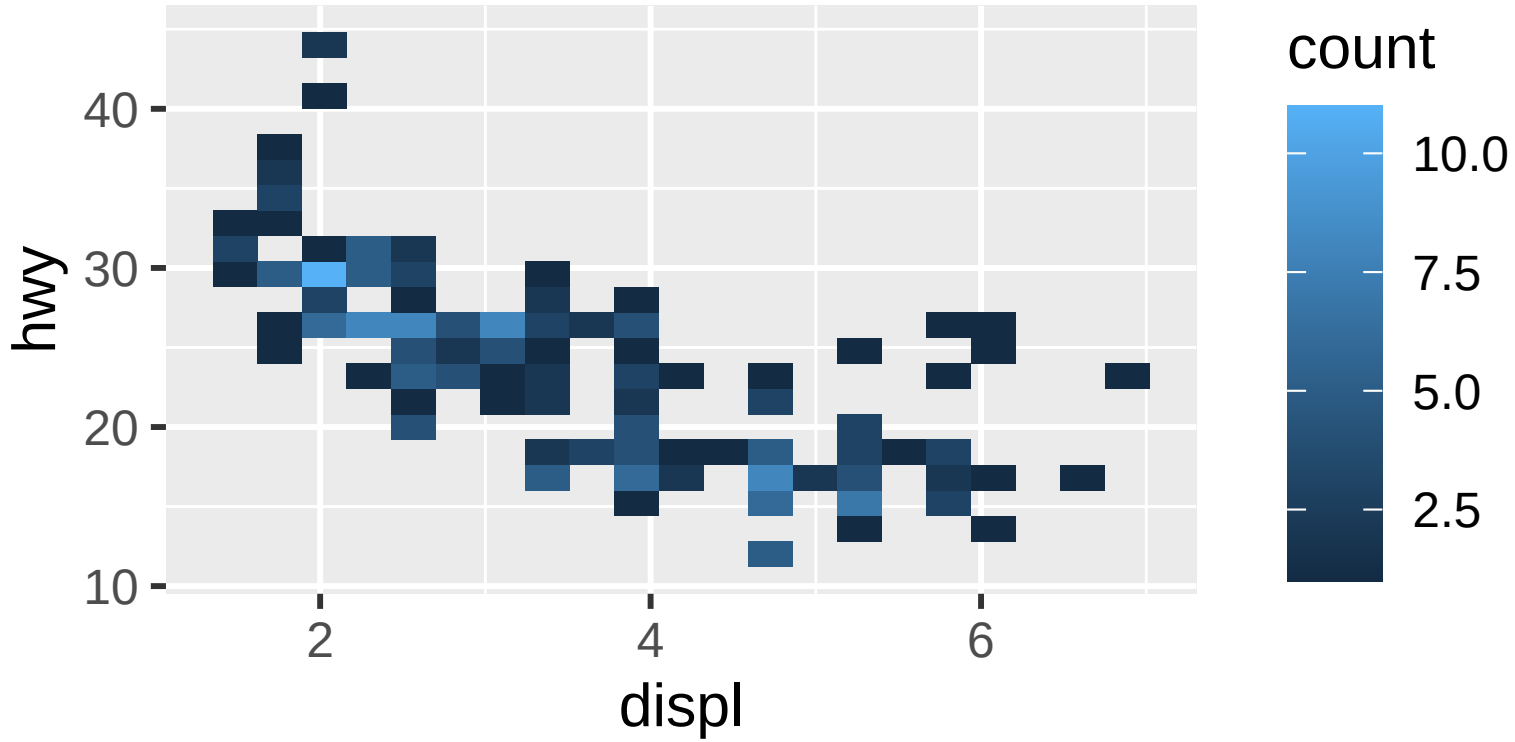


Vale aclarar que es sólo una herramienta de visualización y que se pierde información al suprimir los puntos.

Mapas de calor

Otra forma de lidiar con overplotting es usando mapas de calor, seccionando el plano en rectángulos y sumando la cantidad de datos en cada rectángulo:

```
GG.Plano+
  geom_bin2d(bins=20)
```



Mapas de calor

Otra forma de lidiar con overplotting es usando mapas de calor, seccionando el plano en hexágonos y sumando la cantidad de datos en cada hexágonos:

```
GG.Plano+
  geom_hex(bins=20)
```

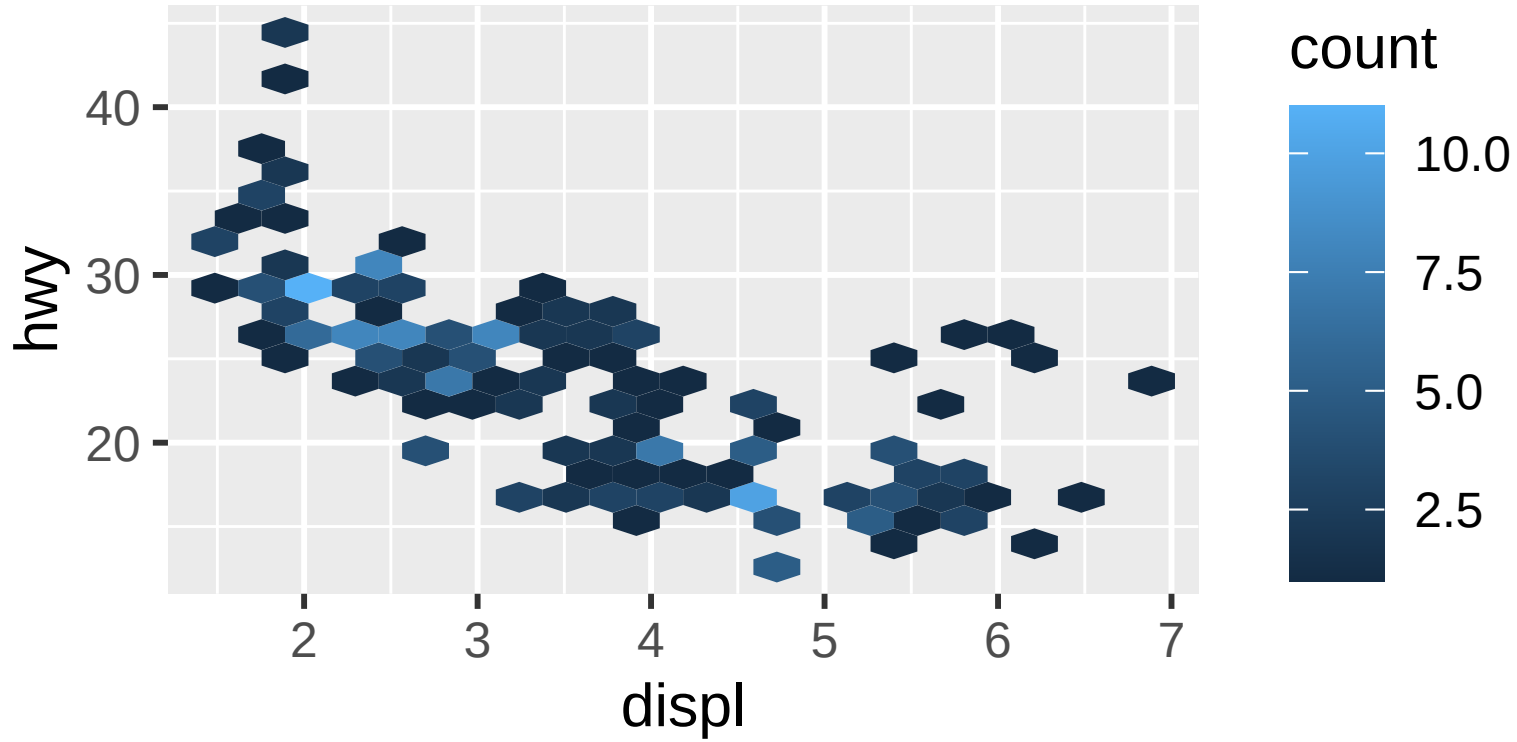


Gráfico de líneas

También podemos analizar mediante un gráfico de líneas el rendimiento en autopistas de los autos según el orden que tienen en la base de datos. 1:N recorre las filas de la base y hwy les da el valor de la ordenada:

```
N=nrow(mpg)
ggplot(data=mpg,aes(x=1:N,y=hwy))+
  geom_line()
```

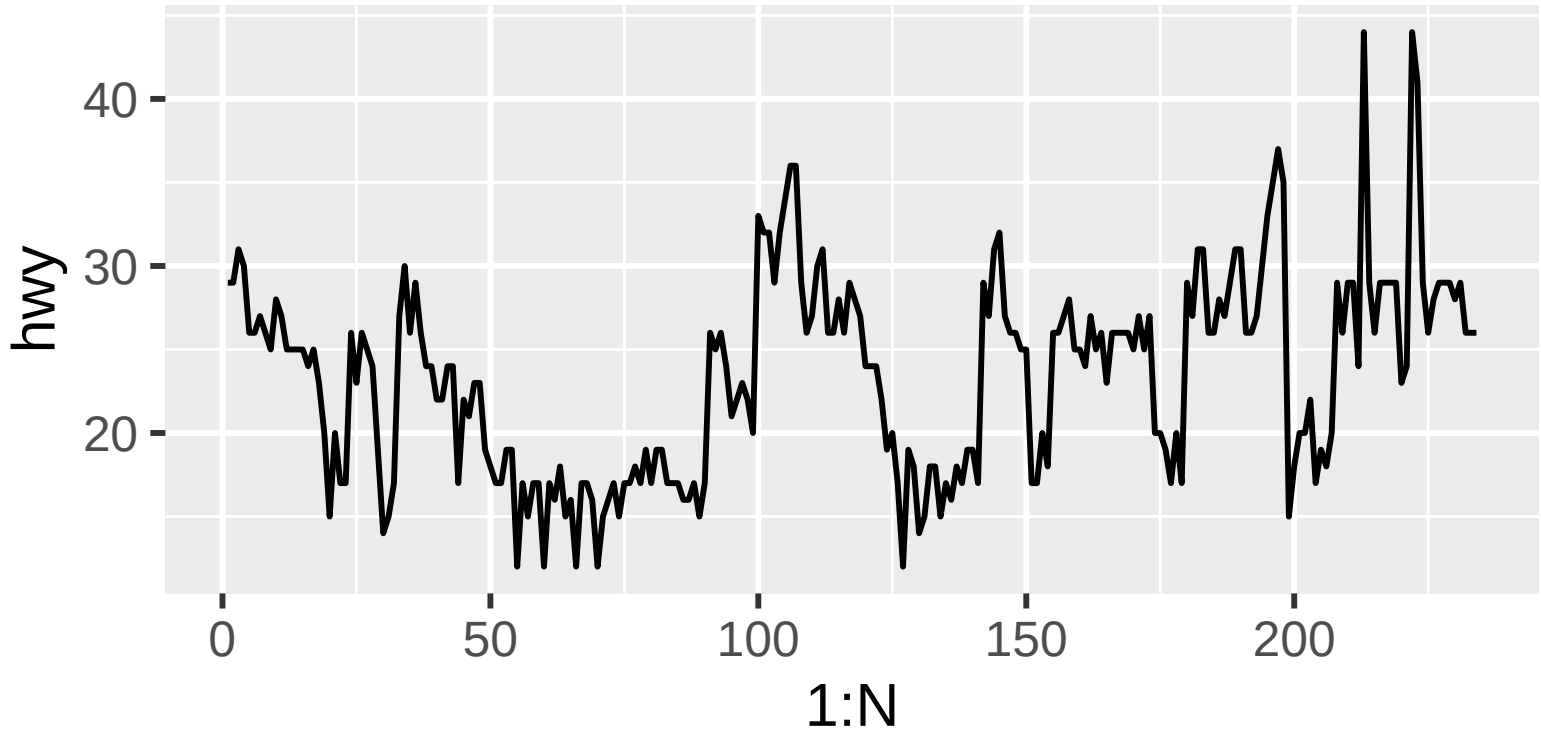
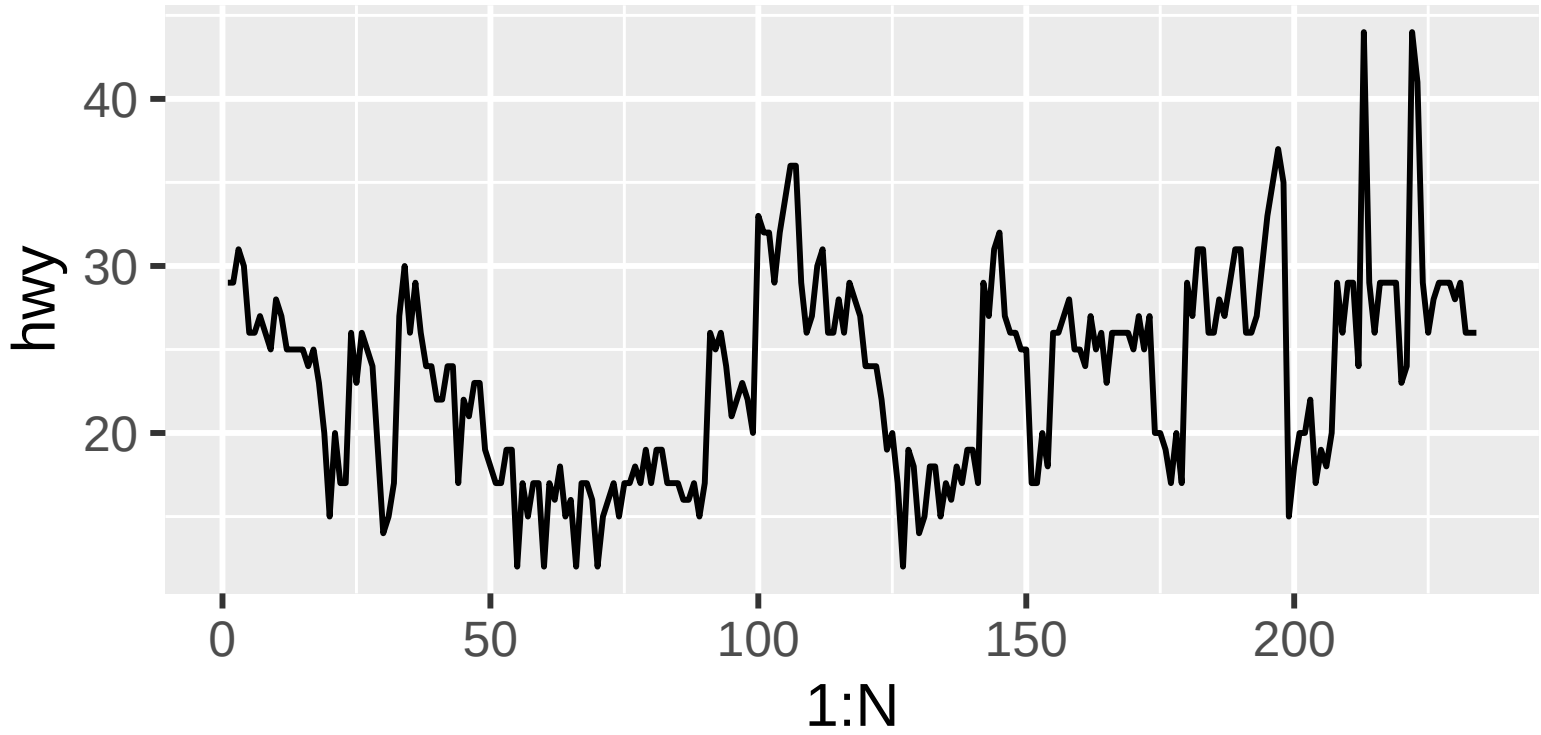


Gráfico de líneas

También podemos analizar mediante un gráfico de líneas el rendimiento en autopistas de los autos según el orden que tienen en la base de datos. 1:N recorre las filas de la base y hwy les da el valor de la ordenada:

```
N=nrow(mpg)
ggplot(data=mpg,aes(x=1:N,y=hwy))+
  geom_line()
```



Notamos que hay algunas tendencias de crecimiento y decrecimiento en los rendimientos.

Gráfico de líneas

Esa estructura que se ve según el orden de los datos, se debe a que están ordenados según la empresa fabricante.

```
GG.TS=ggplot(data=mpg,aes(x=1:N,y=hwy,color=manufacturer))+
  geom_line()
```

```
GG.TS
```

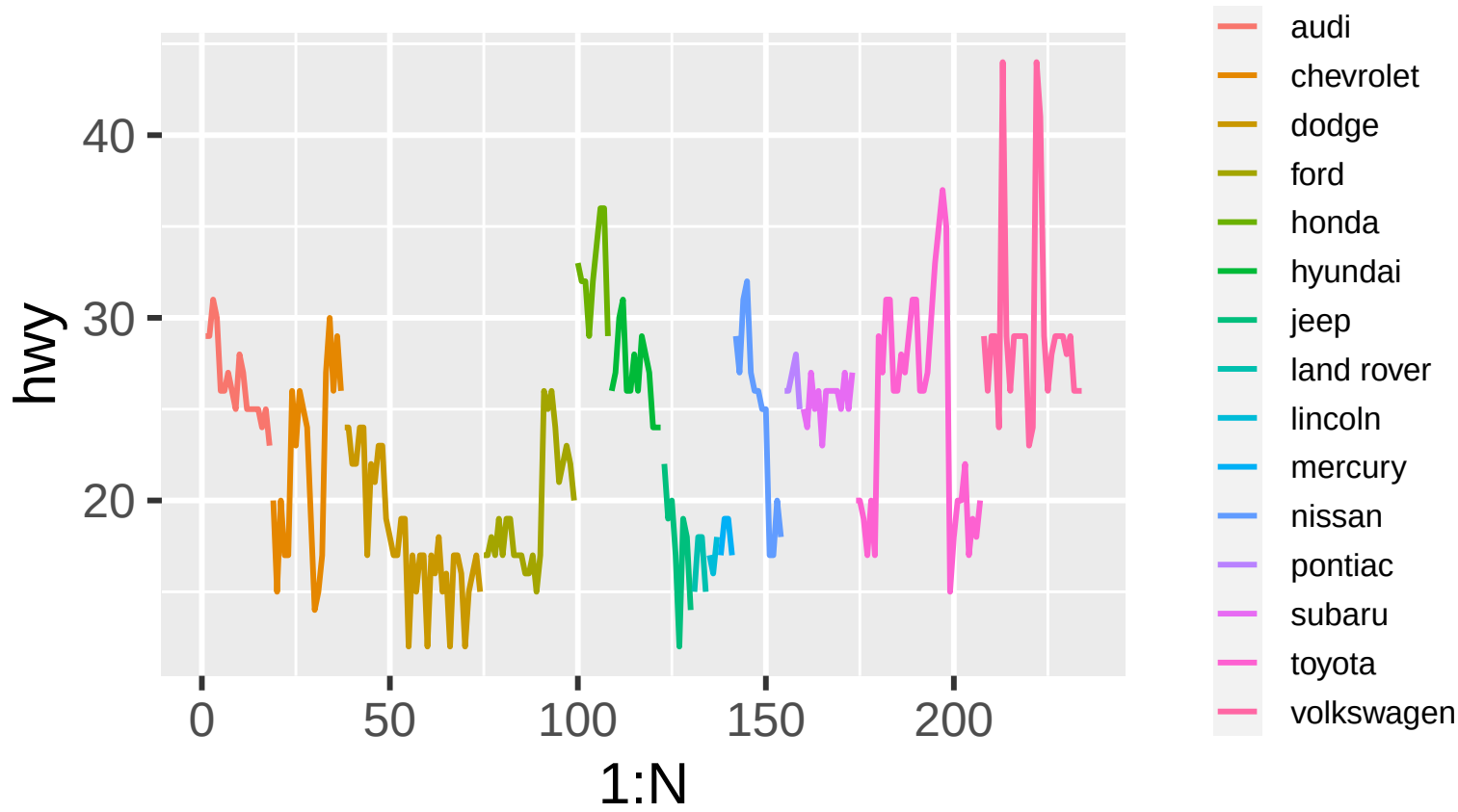
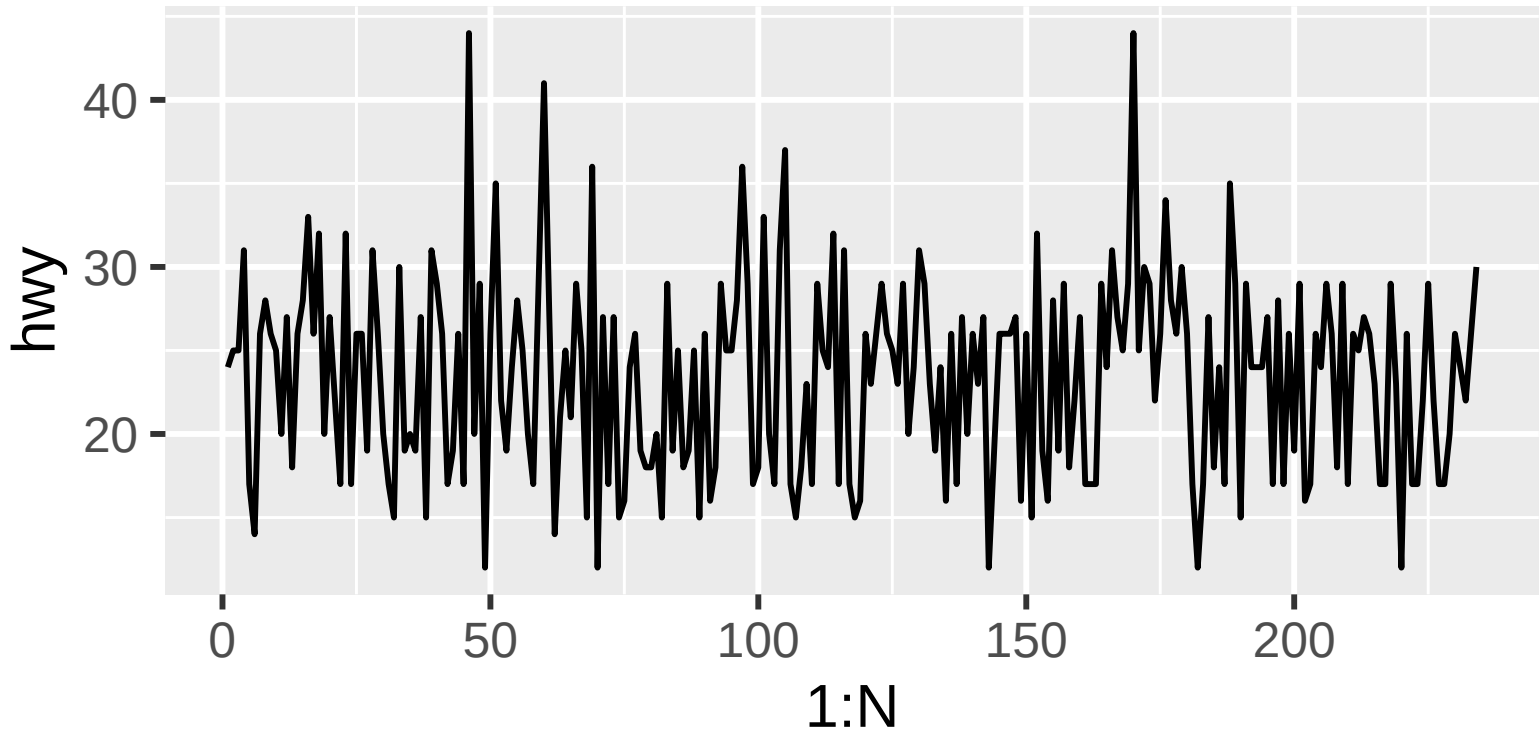


Gráfico de líneas

Sin embargo, si los datos se permutan no necesariamente se observa una estructura en los datos.

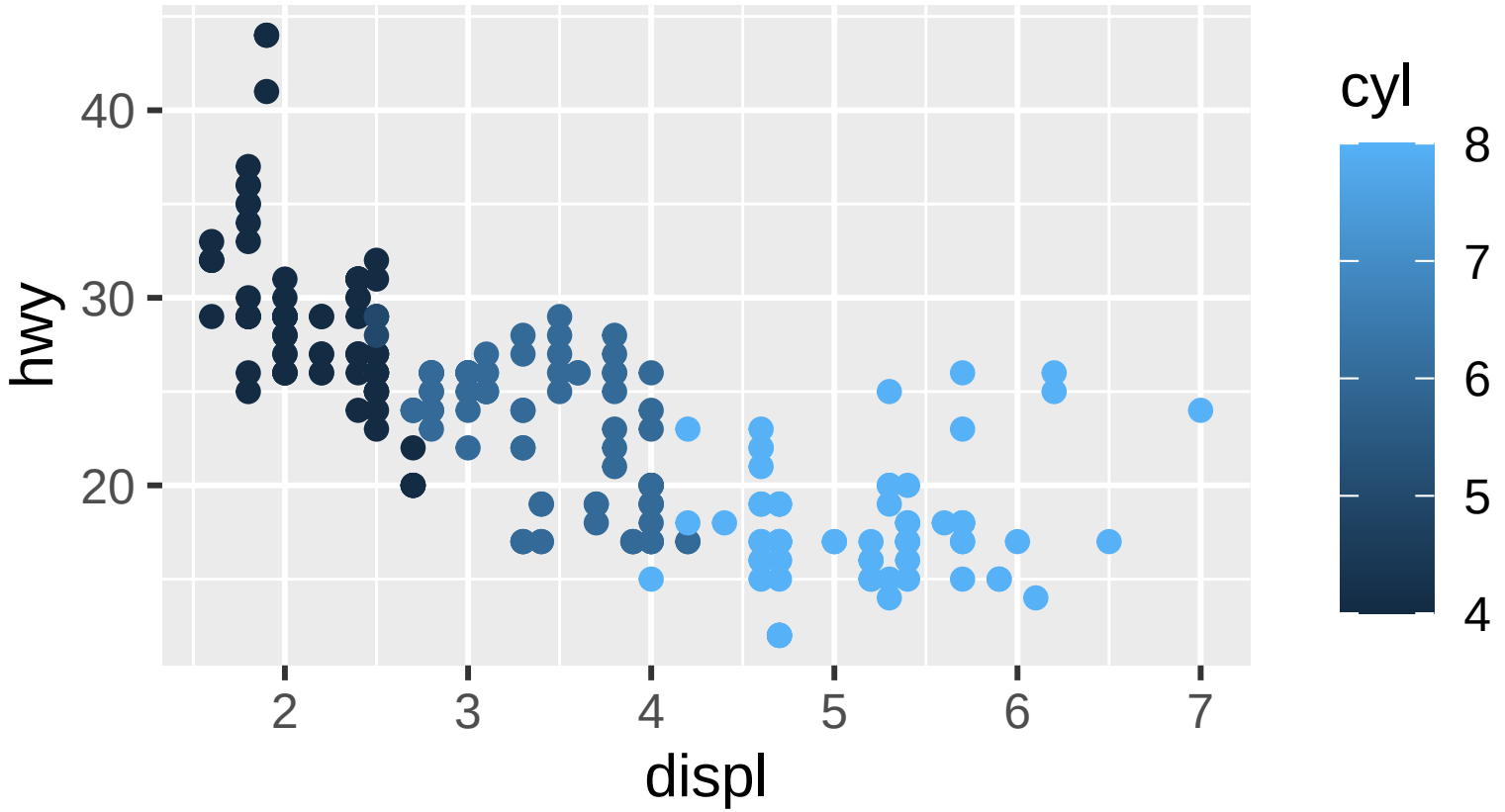
```
N=nrow(mpg)
Perm=sample(1:N,N,replace = F)
PermMPG=mpg[Perm,]
ggplot(data=PermMPG,aes(x=1:N,y=hwy))+
  geom_line()
```



Variables discretas y continuas

Para ggplot, las variables se consideran discretas (generalmente factores) o continuas (generalmente numéricas o enteras). Veamos qué sucede cuando queremos hacer el scatterplot del principio agregando como colores la cantidad de cilindros:

```
GG.Plano+
  geom_point(aes(color=cyl))
```



Esto se debe a que al ser entera, considera la variable cyl como continua.

Variables discretas y continuas

Sin embargo, sabemos que la variable cyl no puede tener valores fraccionarios, vamos a copiar sus datos en un factor llamado cylF:

```
mpg$cylF=as.factor(mpg$cyl);
head(mpg$cyl)
```

```
## [1] 4 4 4 4 6 6
```

```
head(mpg$cylF)
```

```
## [1] 4 4 4 4 6 6
## Levels: 4 5 6 8
```

```
class(mpg$cyl)
```

```
## [1] "integer"
```

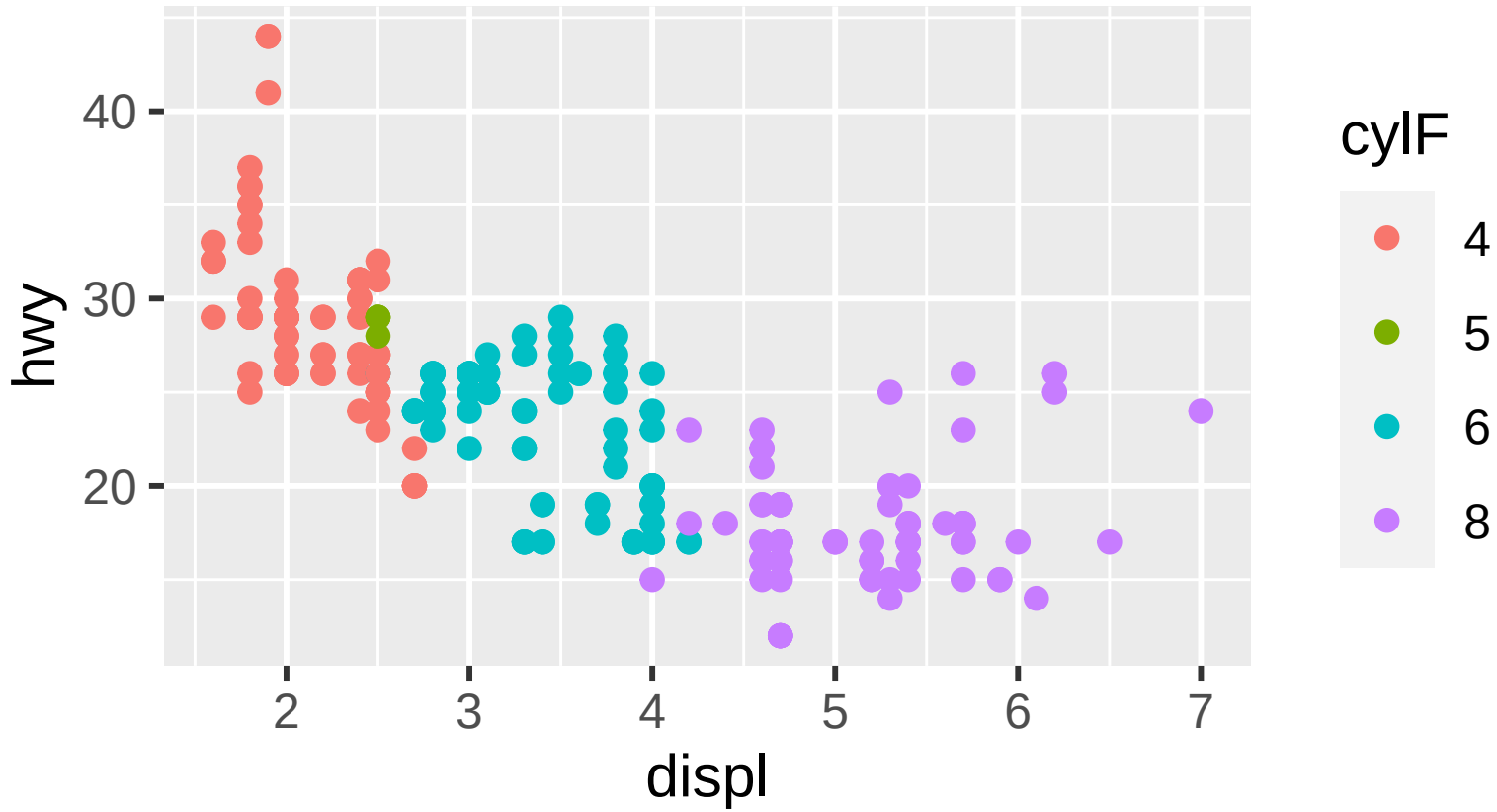
```
class(mpg$cylF)
```

```
## [1] "factor"
```

Variables discretas y continuas

De este modo, nos aseguramos de que ggplot vea a cyl como una variable discreta:

```
GG.Plano +
  geom_point(aes(color=cylF))
```



Análisis de variables cualitativas

Gráfico de barras

El comando geom_bar hace gráficos de barras, pensado para variables categóricas. Sólo requiere una variable estética x. Por default, calcula la frecuencia absoluta de cada categoría:

```
GG.Bar=ggplot(data = mpg,aes(x=drv))
GG.Bar+
  geom_bar()
```

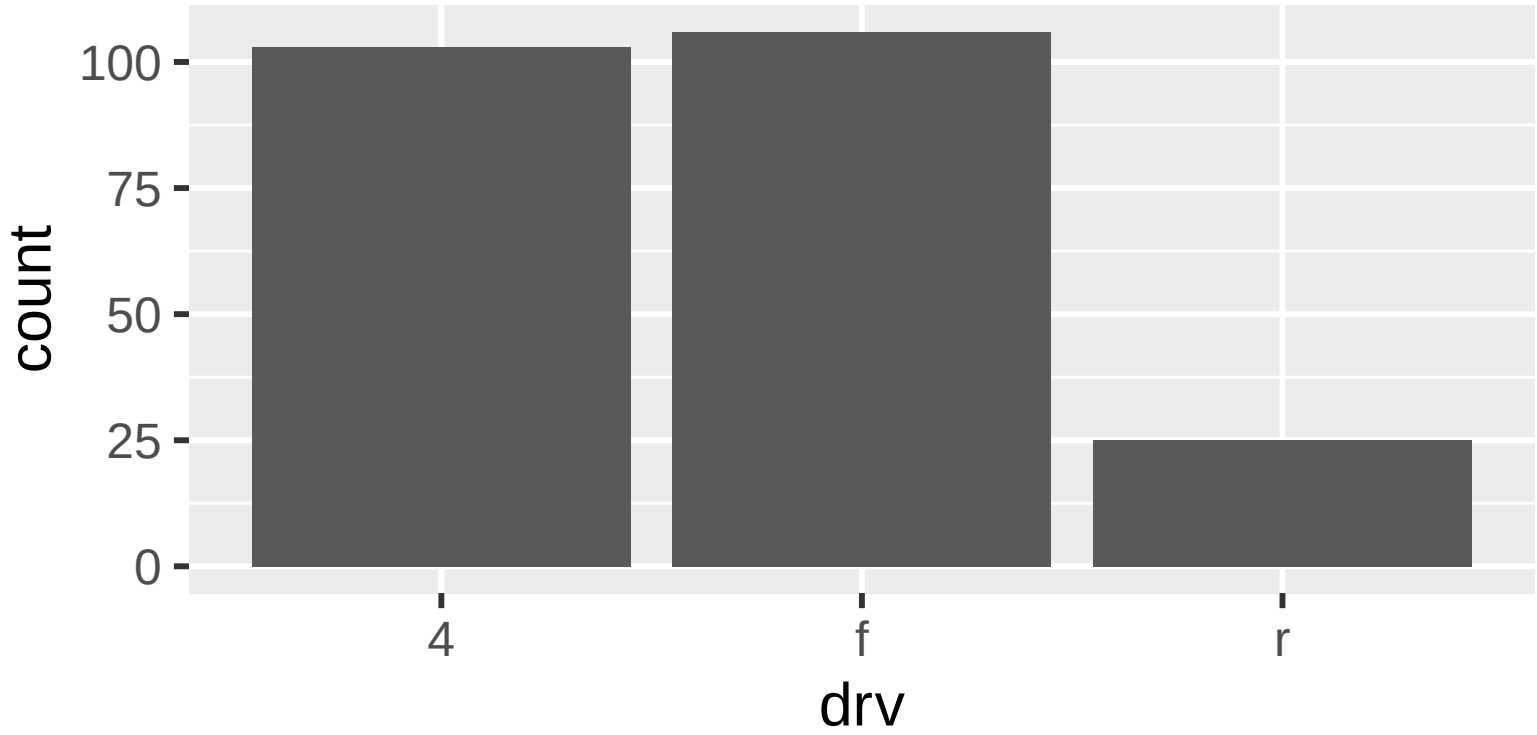


Gráfico de barras

Para sacarle el gris ese horrible, uno pensaría que poniendo color="blue" lo solucionaría:

```
GG.Bar+
  geom_bar(color="blue")
```

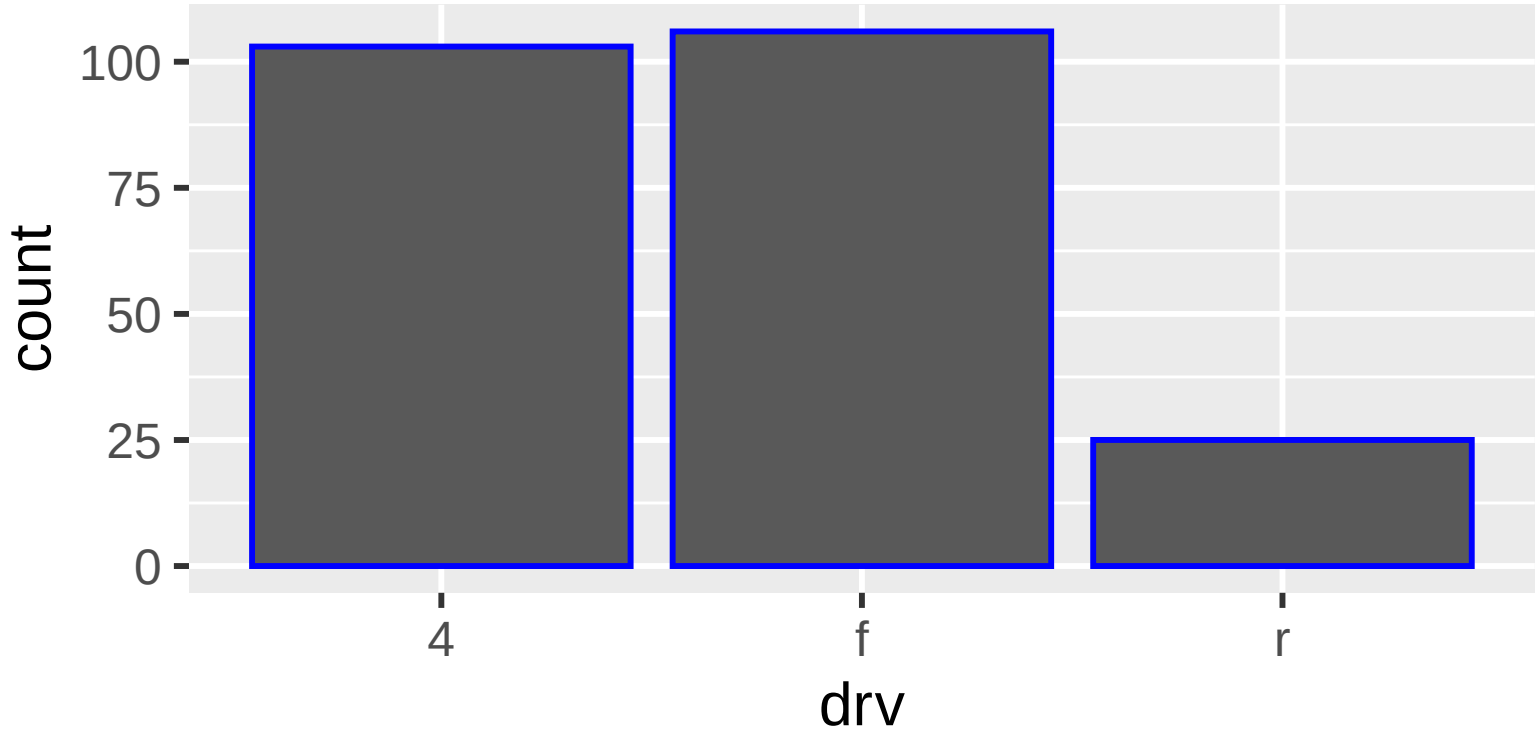
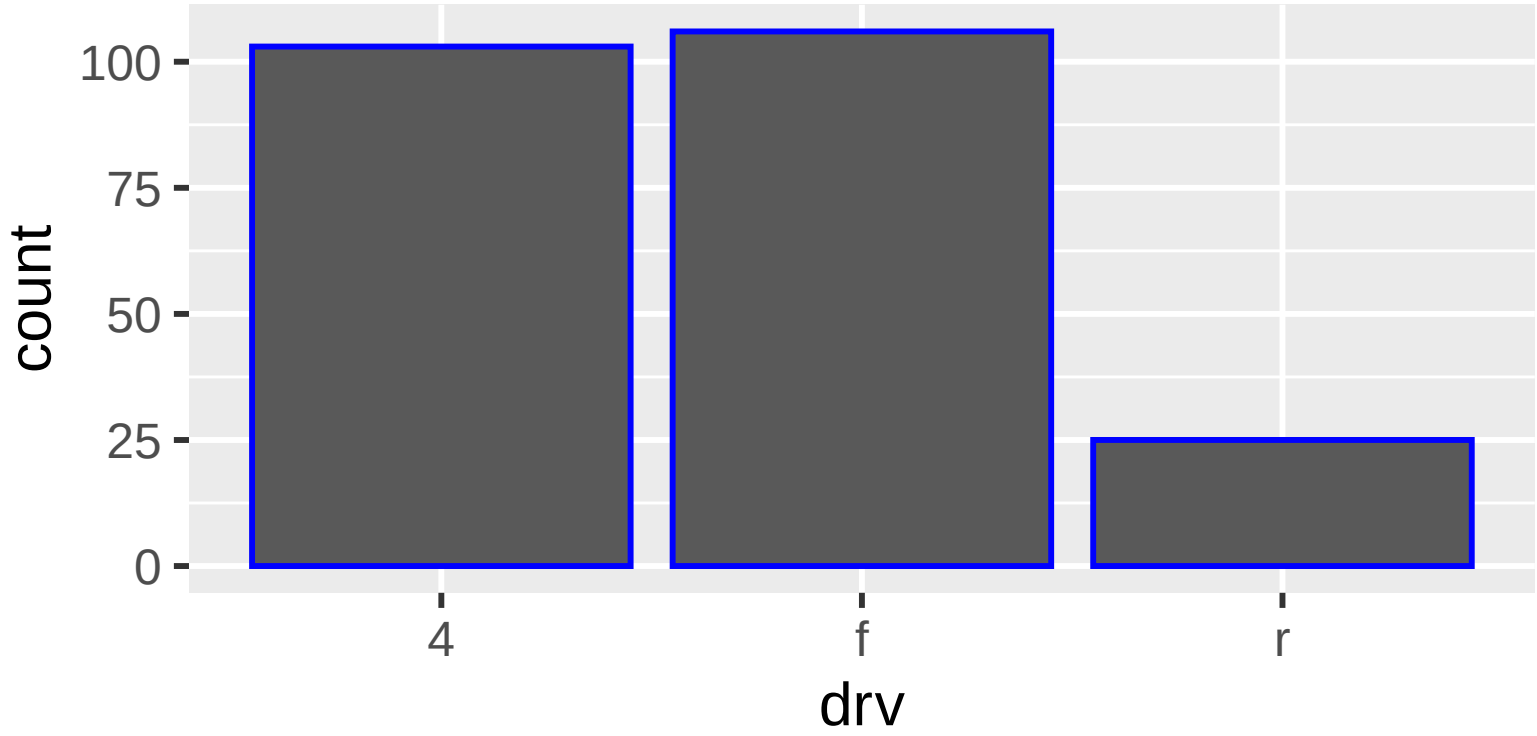


Gráfico de barras

Para sacarle el gris ese horrible, uno pensaría que poniendo color="blue" lo solucionaría:

```
GG.Bar+
  geom_bar(color="blue")
```



Pues no mi ciela.

Gráfico de barras

El comando geom_bar necesita usar el parámetro fill para llenar de un color:

```
GG.Bar+
  geom_bar(fill="blue")
```

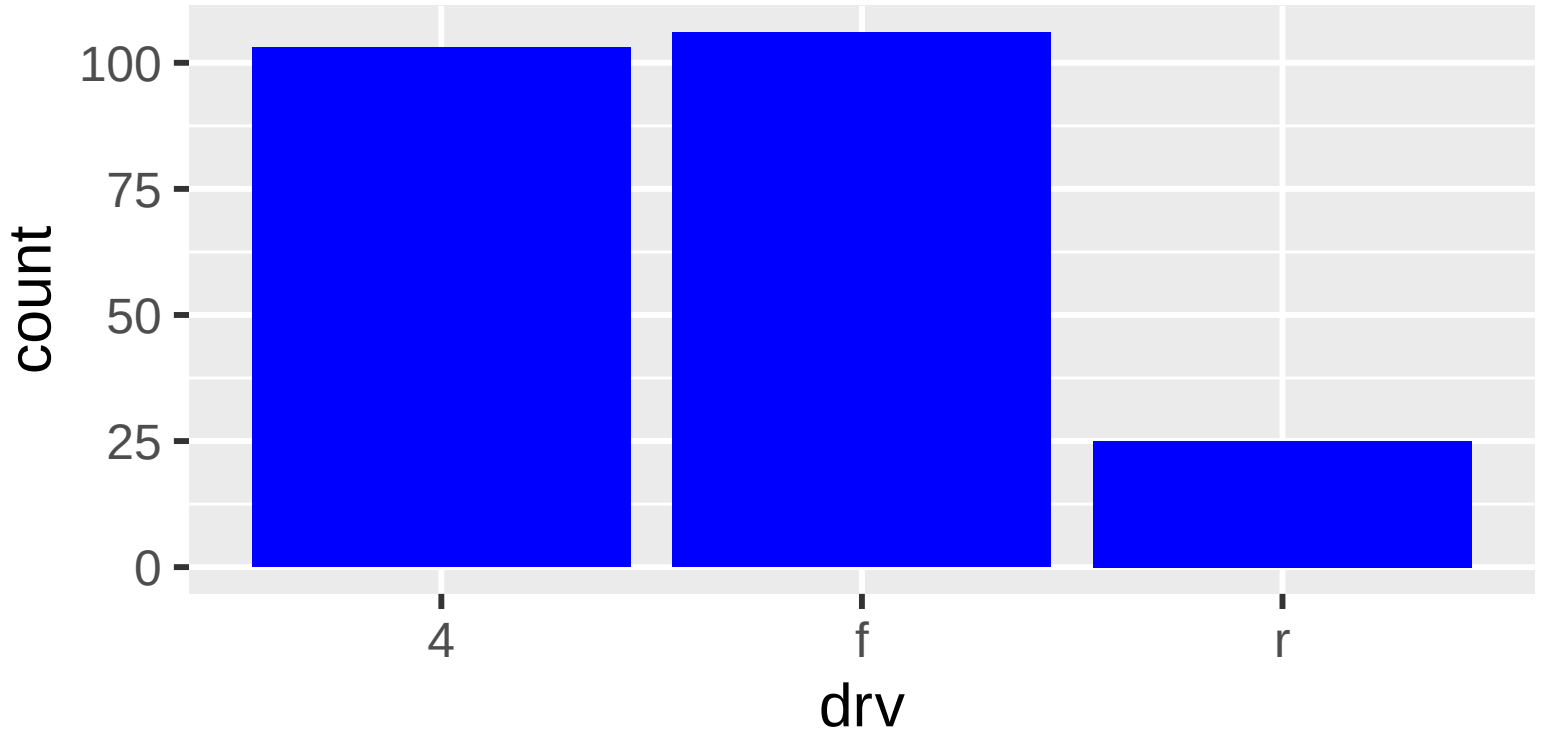


Gráfico de barras

El parámetro fill puede ser una variable categórica, dando noción de cómo es la interacción entre distintos factores:

```
GG.Bar+
  geom_bar(aes(fill=cylF))
```

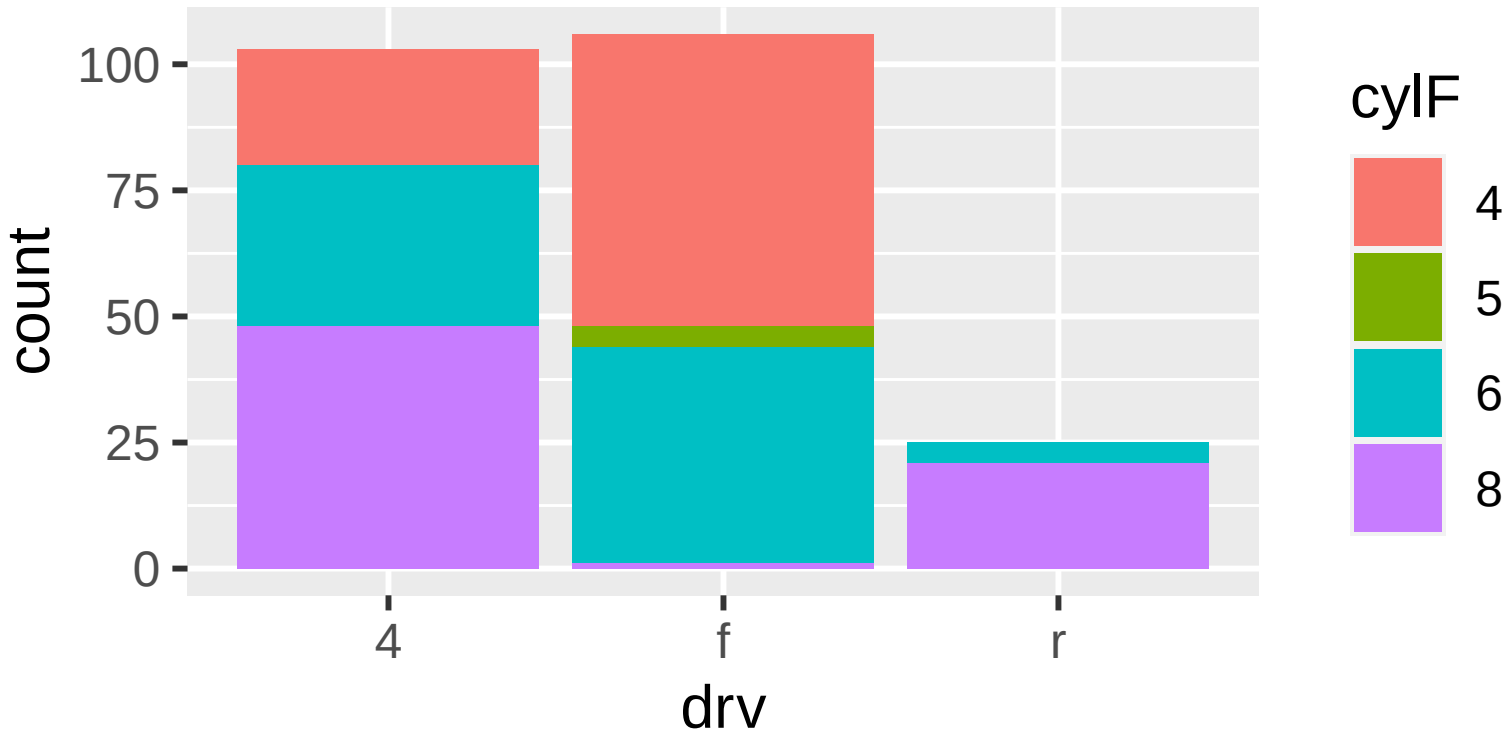


Gráfico de barras

Por otro lado, se pueden subdividir en barras que mantengan el color deseado, sin necesidad de que estén apilados:

```
GG.Bar+
  geom_bar(aes(fill=cylF),position = "dodge")
```

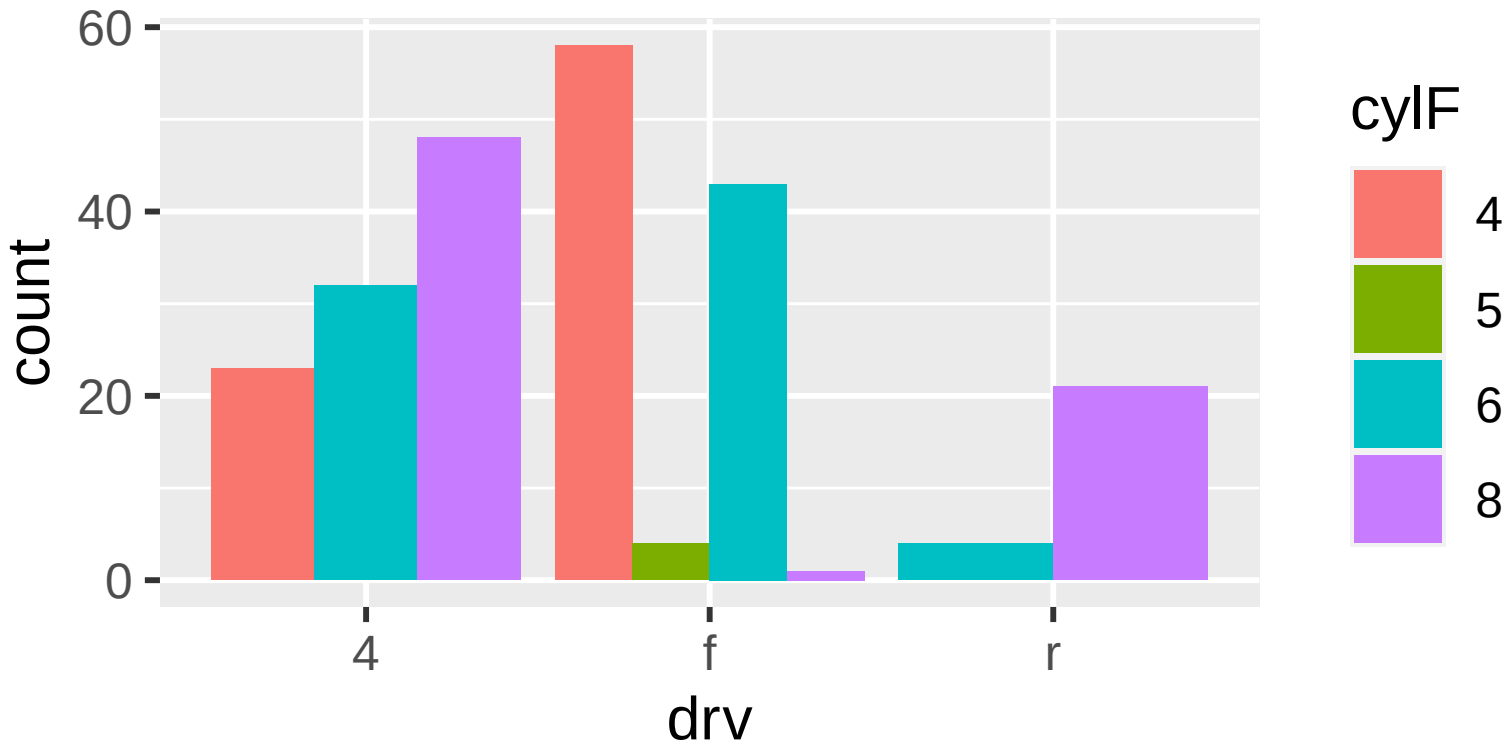


Gráfico de barras

Las barras se pueden separar más o menos usando position_dodge:

```
GG.Bar+
  geom_bar(aes(fill=cylF),
           width=0.25,
           position = position_dodge(width = 0.5))
```

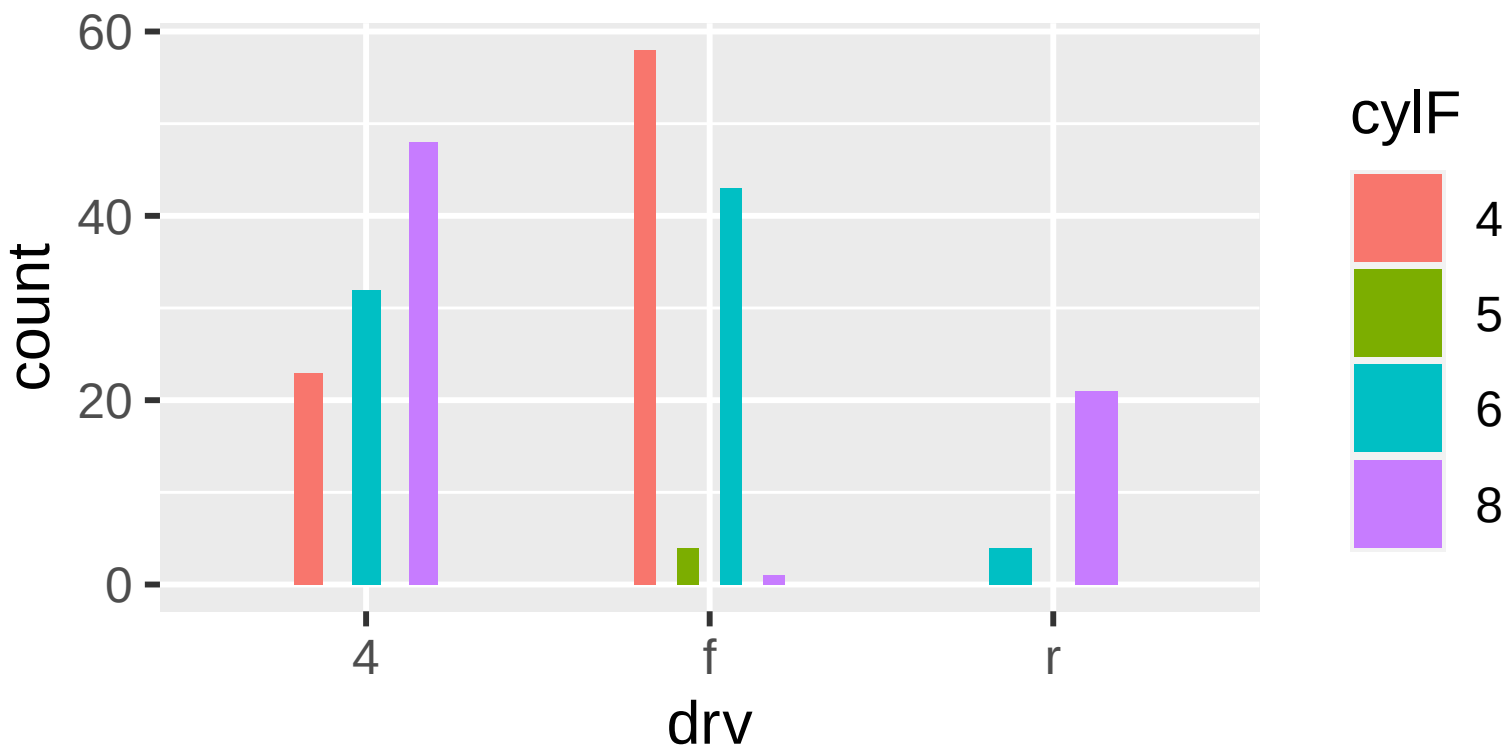


Gráfico de barras

Por último, position="fill" permite ver la proporción de cada nivel de un factor dentro del otro:

```
GG.Bar+
  geom_bar(aes(fill=cylF),
           position = "fill")
```

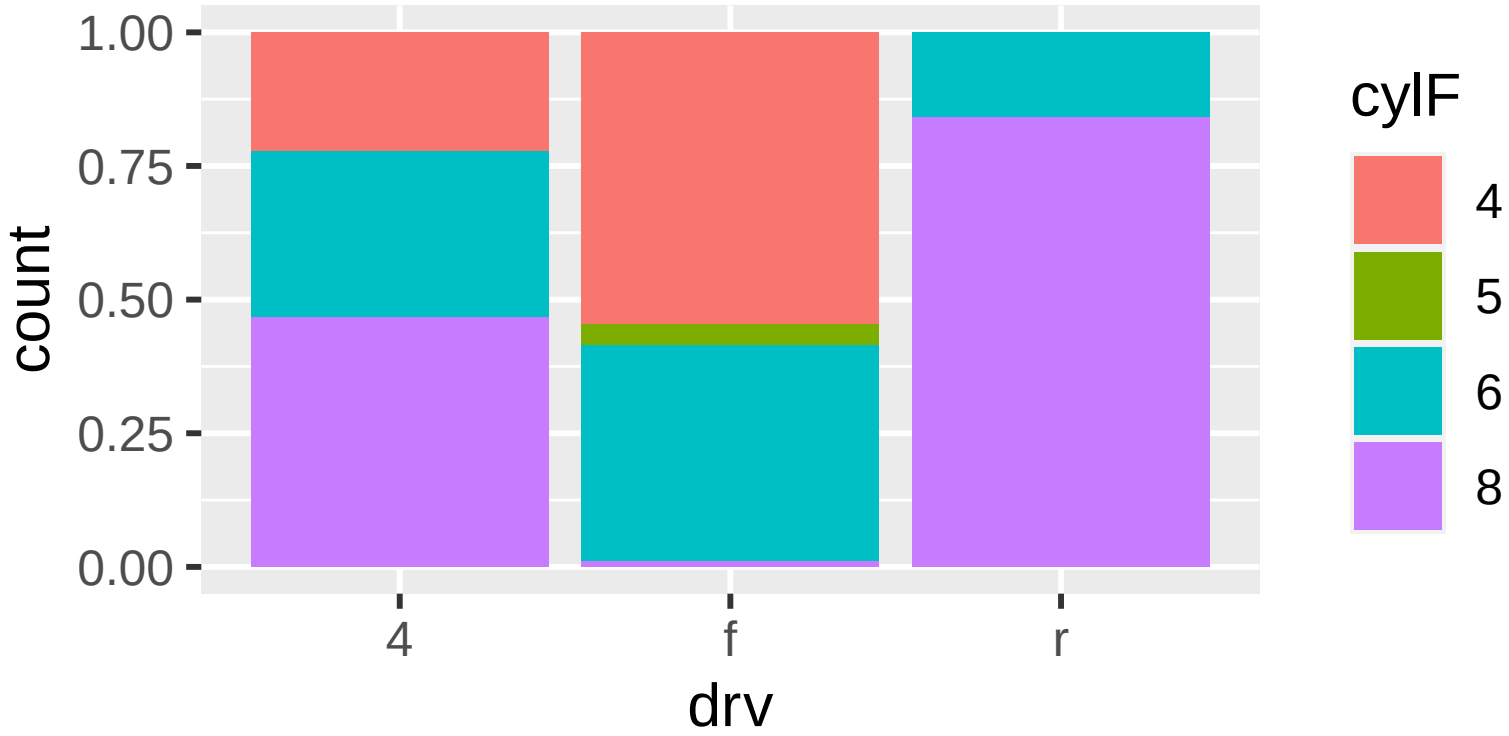
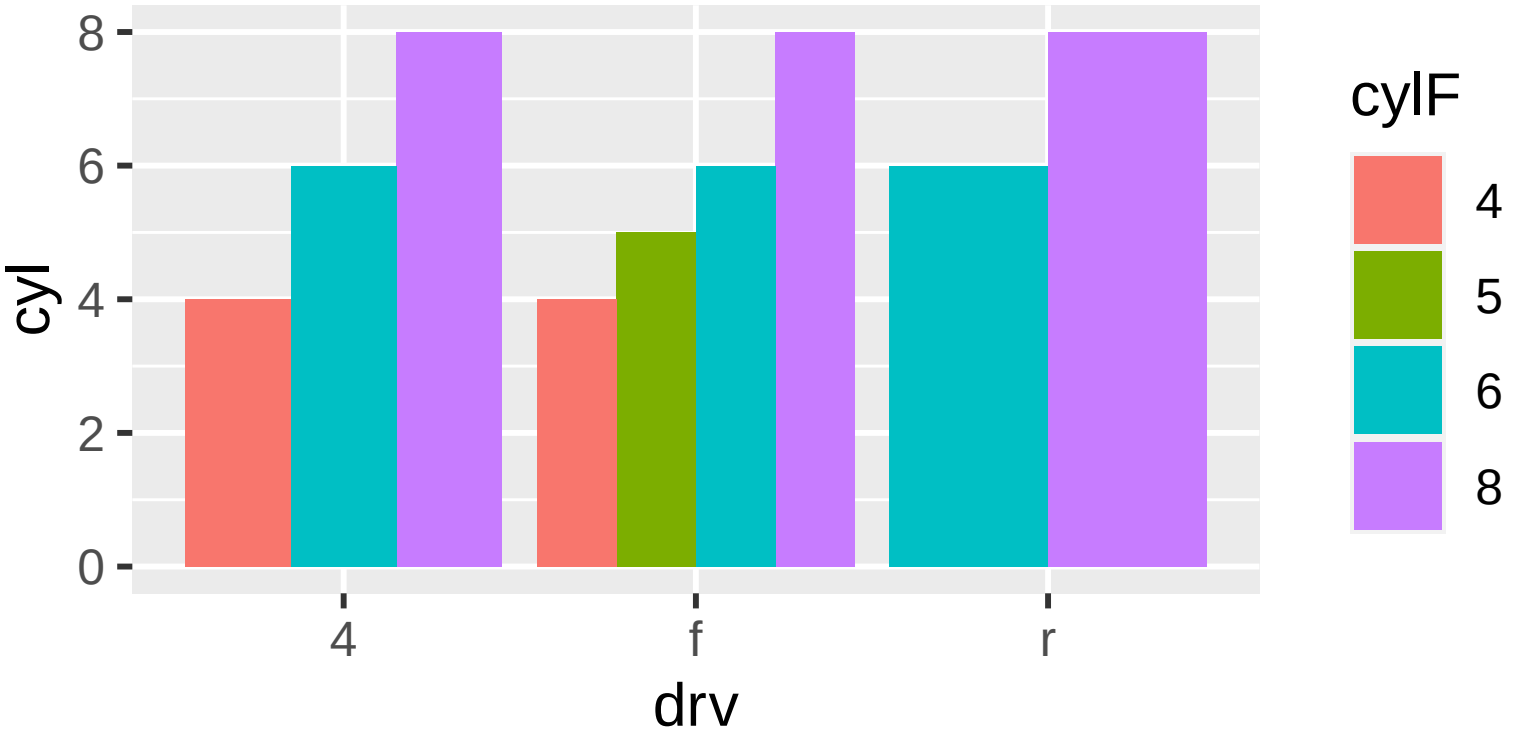


Gráfico de barras

Para que las barras tengan una altura que no sea necesariamente la frecuencia absoluta, se le agrega una variable estética y, mientras que se fija la transformación estadística como stat="identity":

```
GG.Bar+
  geom_bar(aes(fill=cylF,y=cyl),
           position = "dodge",
           stat="identity")
```

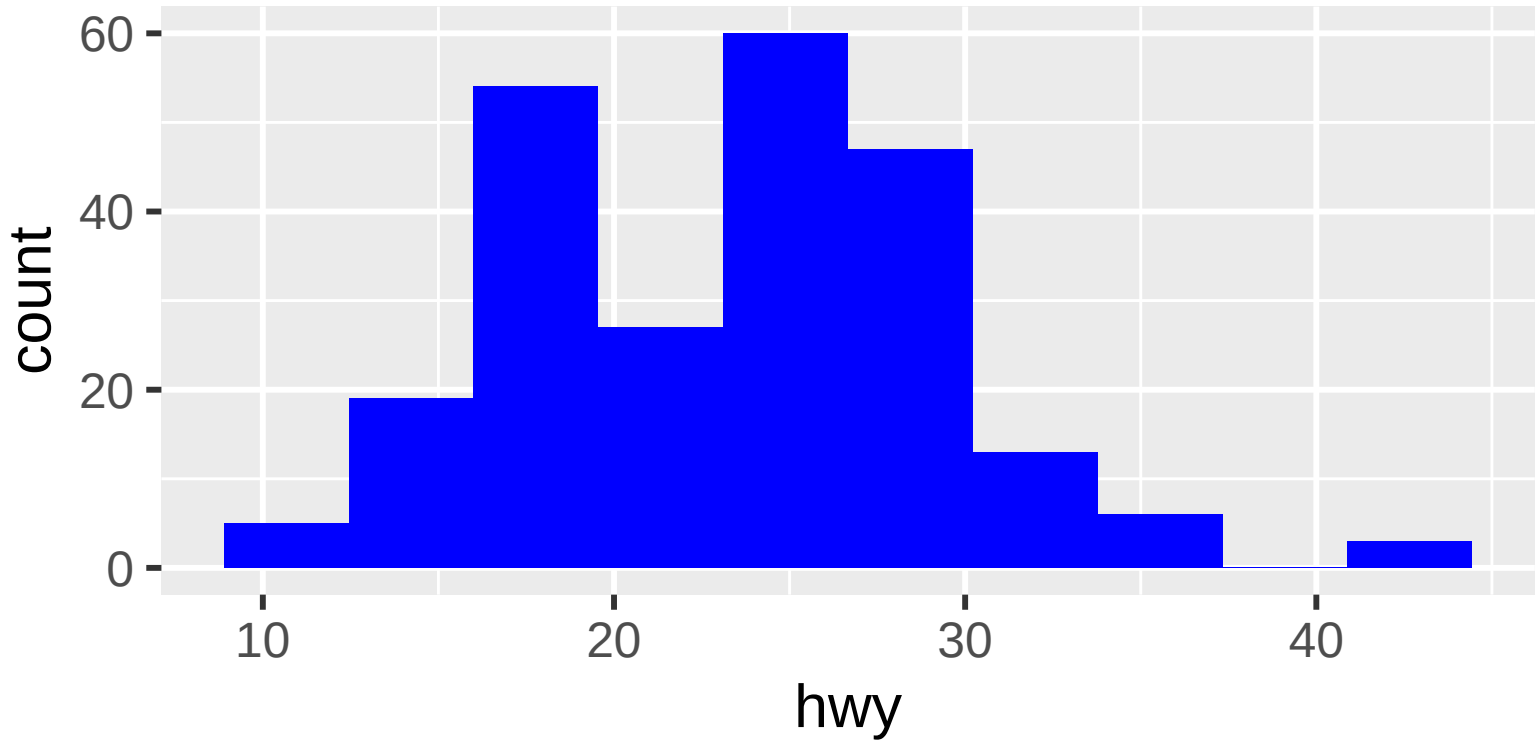


Análisis de variables cuantitativas

Histogramas

Cuando la variable es continua, el equivalente al gráfico de barras es el histograma, en el que se divide el rango de las variables en una cantidad de intervalos (bins) y se cuenta la cantidad de datos en cada intervalo.

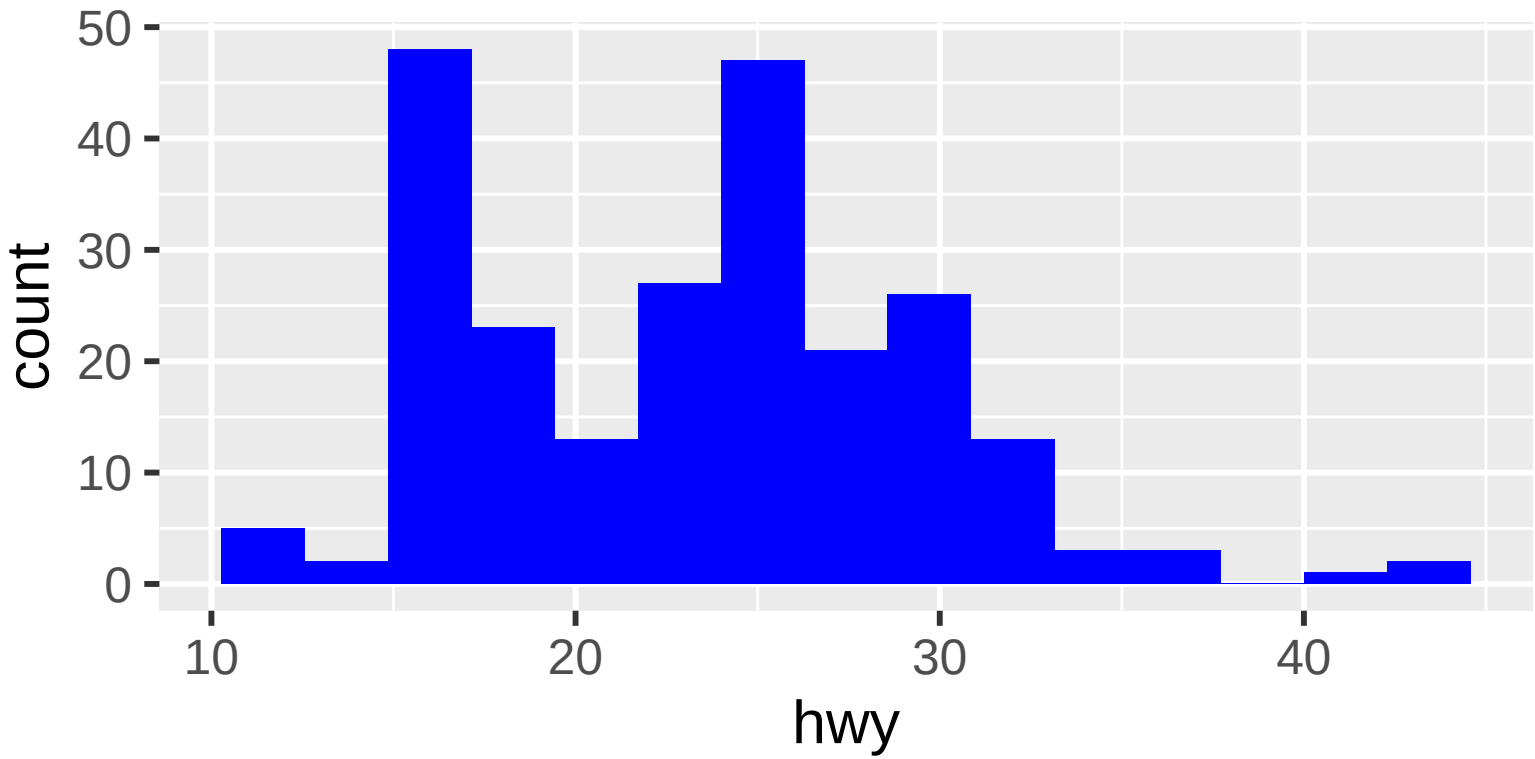
```
GG.Hist=ggplot(data=mpg,aes(x=hwy))
GG.Hist+
  geom_histogram(fill="blue",bins=10)
```



Histogramas

Cuando la variable es continua, el equivalente al gráfico de barras es el histograma, en el que se divide el rango de las variables en una cantidad de intervalos (bins) y se cuenta la cantidad de datos en cada intervalo.

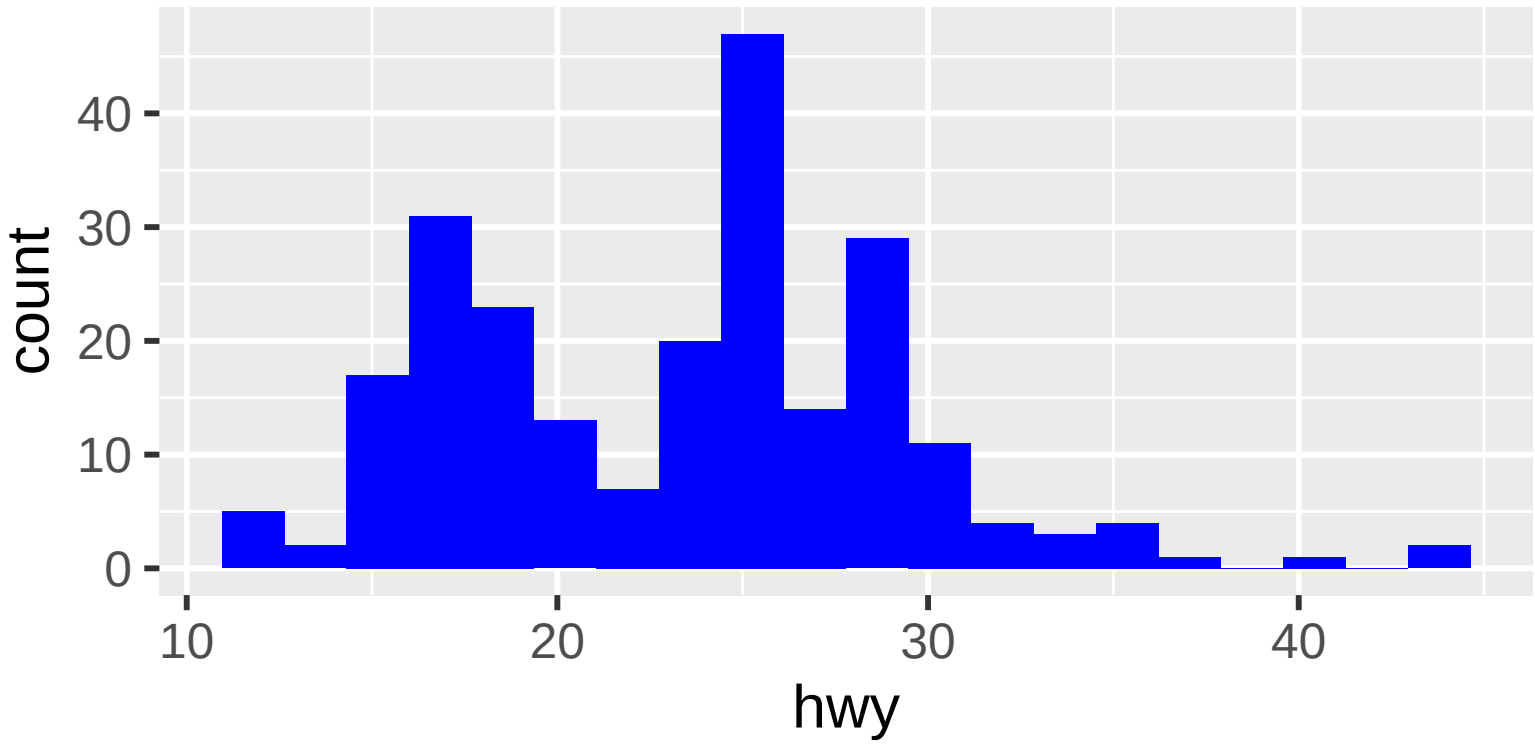
```
GG.Hist+
  geom_histogram(fill="blue",bins=15)
```



Histogramas

Cuando la variable es continua, el equivalente al gráfico de barras es el histograma, en el que se divide el rango de las variables en una cantidad de intervalos (bins) y se cuenta la cantidad de datos en cada intervalo.

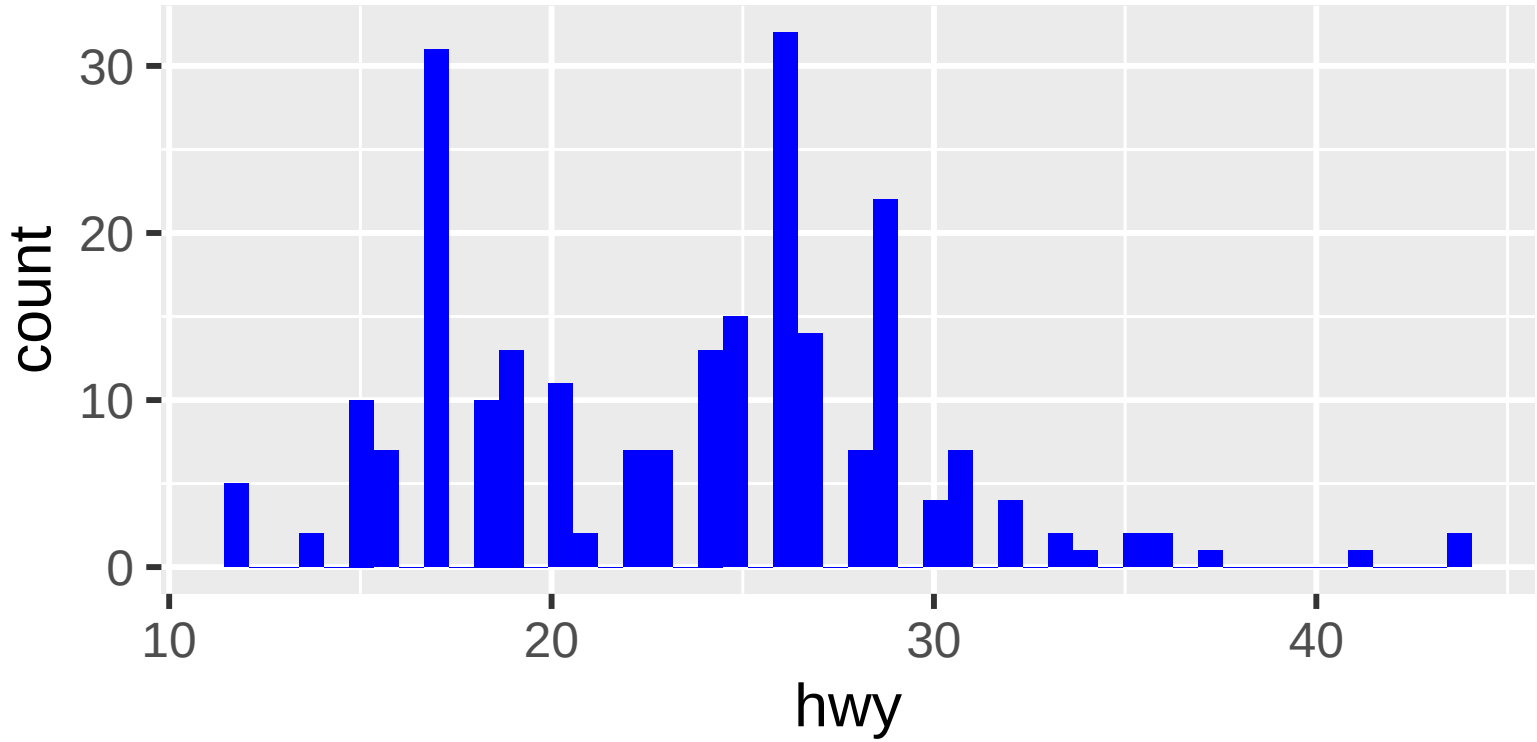
```
GG.Hist+
  geom_histogram(fill="blue",bins=20)
```



Histogramas

Cuando la variable es continua, el equivalente al gráfico de barras es el histograma, en el que se divide el rango de las variables en una cantidad de intervalos (bins) y se cuenta la cantidad de datos en cada intervalo.

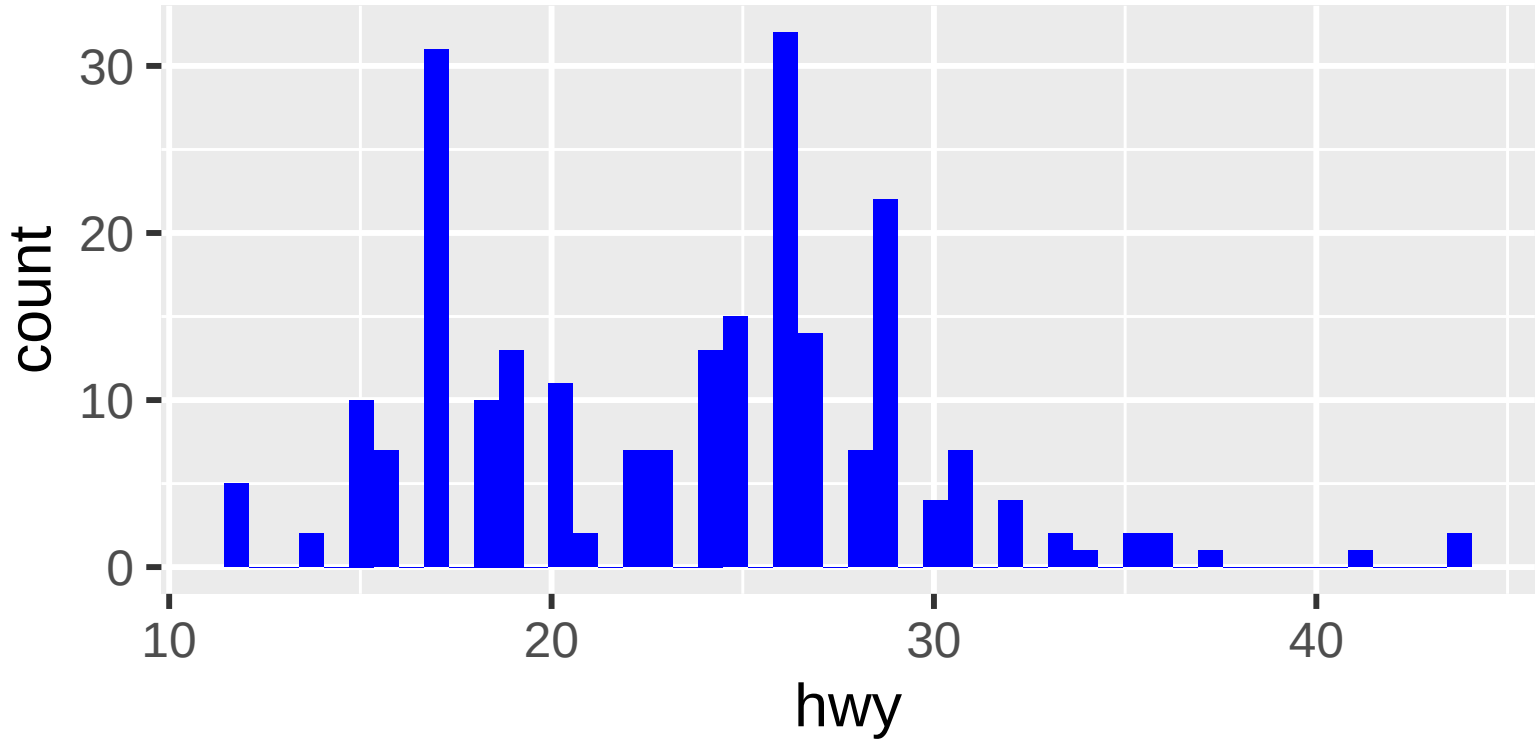
```
GG.Hist+
  geom_histogram(fill="blue",bins=50)
```



Histogramas

Cuando la variable es continua, el equivalente al gráfico de barras es el histograma, en el que se divide el rango de las variables en una cantidad de intervalos (bins) y se cuenta la cantidad de datos en cada intervalo.

```
GG.Hist+
  geom_histogram(fill="blue",bins=50)
```

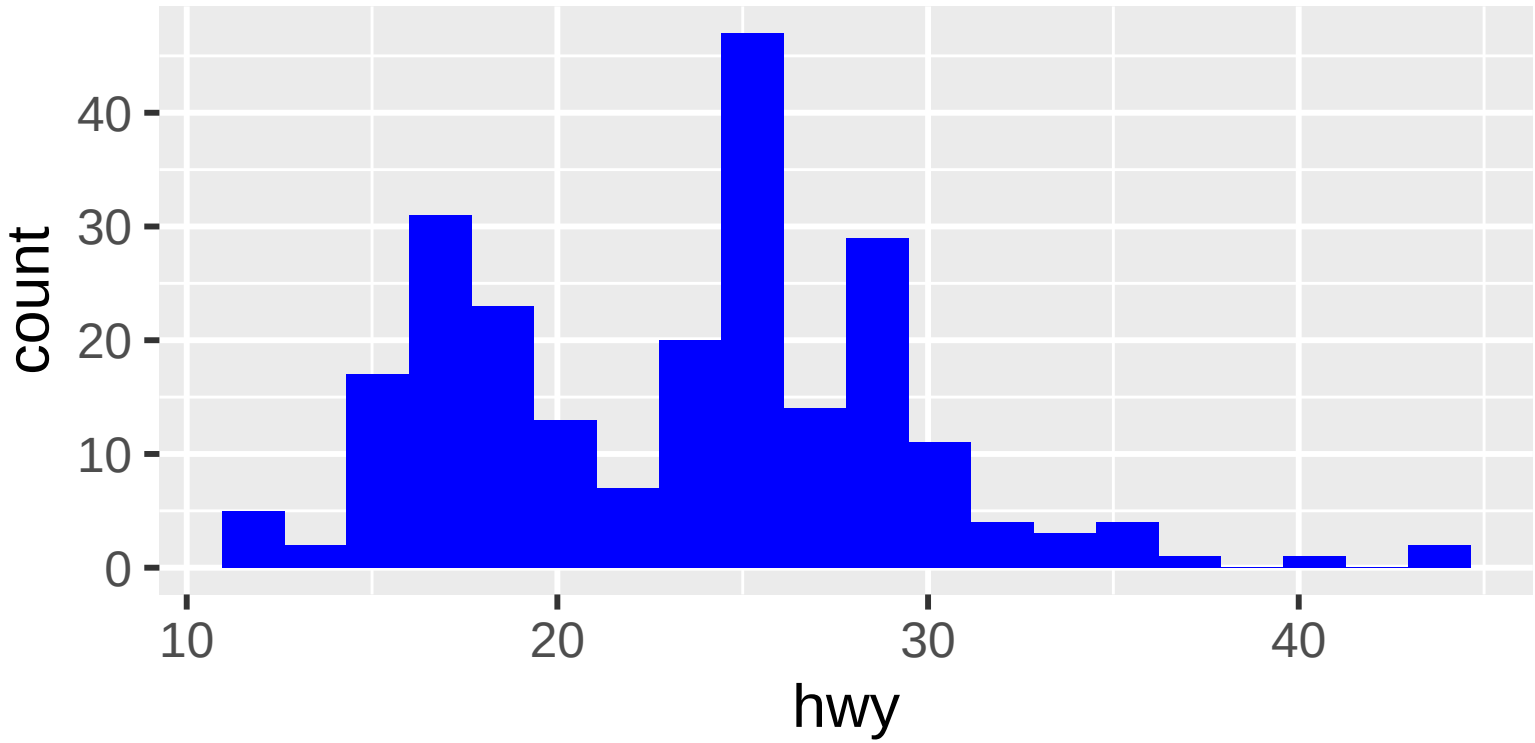


Cuando son muchos intervalos, se empieza a perder la apreciación de cómo se distribuye la variable.

Histogramas

Por lo tanto, eligiremos 20 como la cantidad de intervalos:

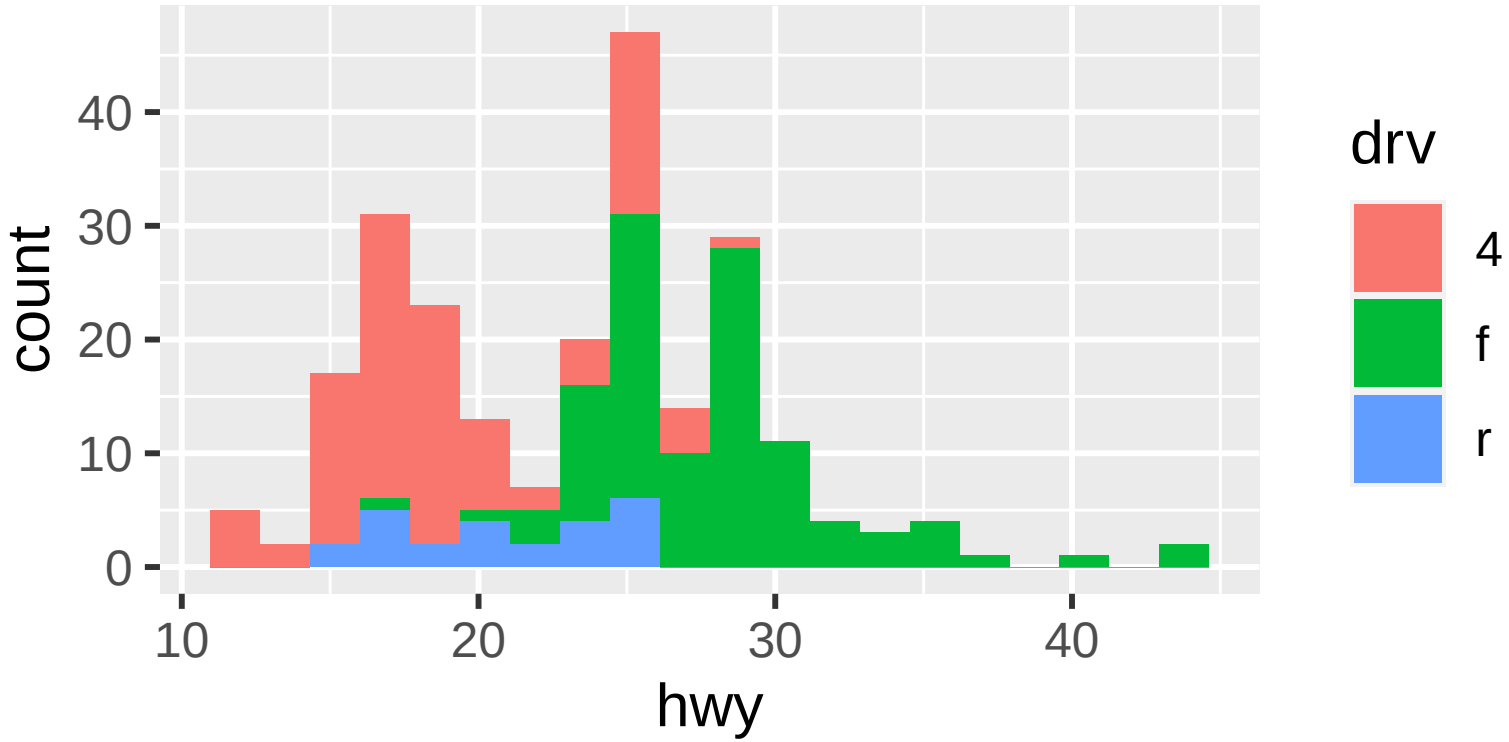
```
GG.Hist+
  geom_histogram(fill="blue",bins=20)
```



Histogramas

Además, podemos agregarle con la variable fill, para apreciar cuántas de cada variable de un factor están en cada intervalo:

```
GG.Hist+
  geom_histogram(aes(fill=drv),bins=20)
```

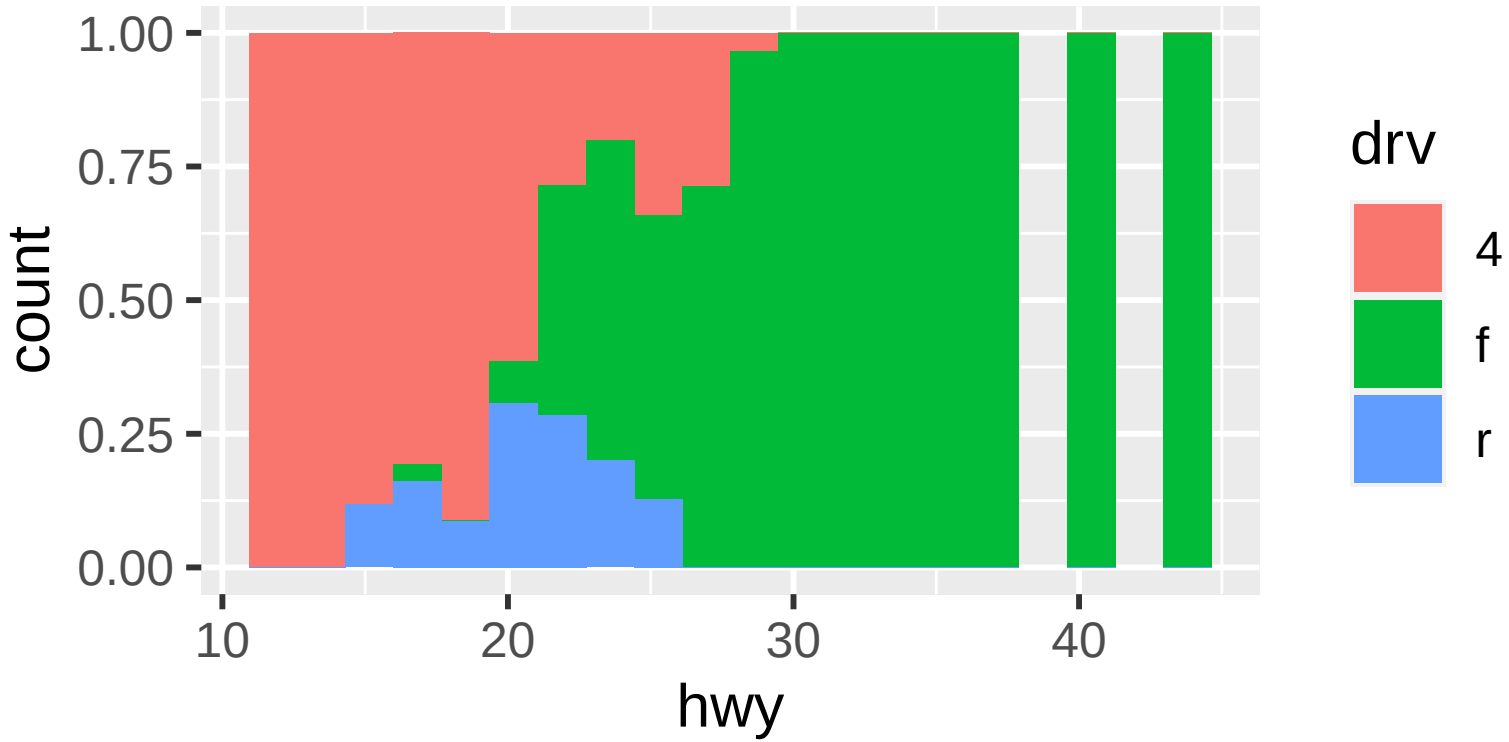


Histogramas

Al igual que para el gráfico de barras, se puede "llenar" el eje y con position="fill":

```
GG.Hist+
  geom_histogram(aes(fill=drv),
                 bins=20,
                 position = "fill")
```

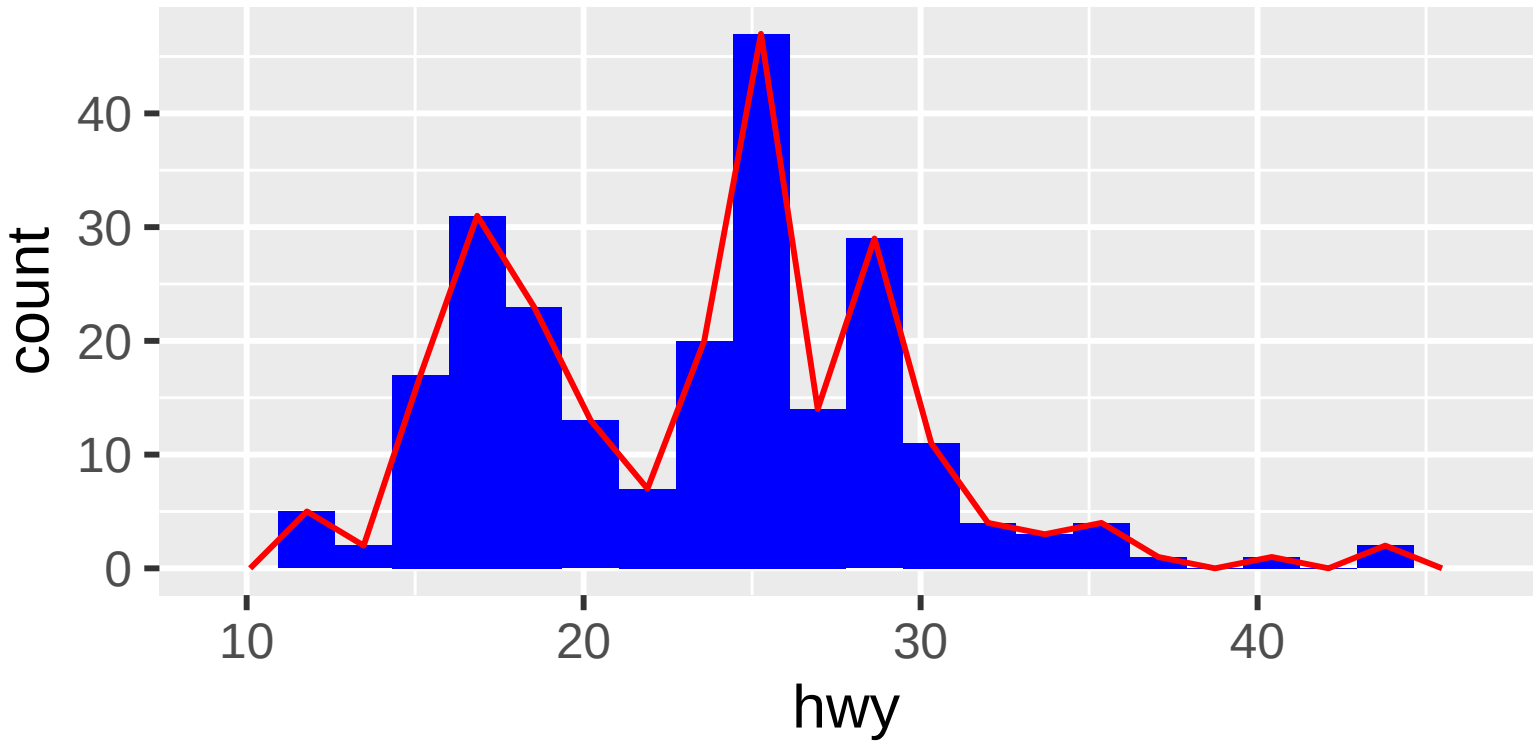
```
## Warning: Removed 6 rows containing missing values (geom_bar).
```



Histogramas

Para terminar de apreciar la forma de la distribución, se puede agregar lo que se llama un polígono de frecuencias

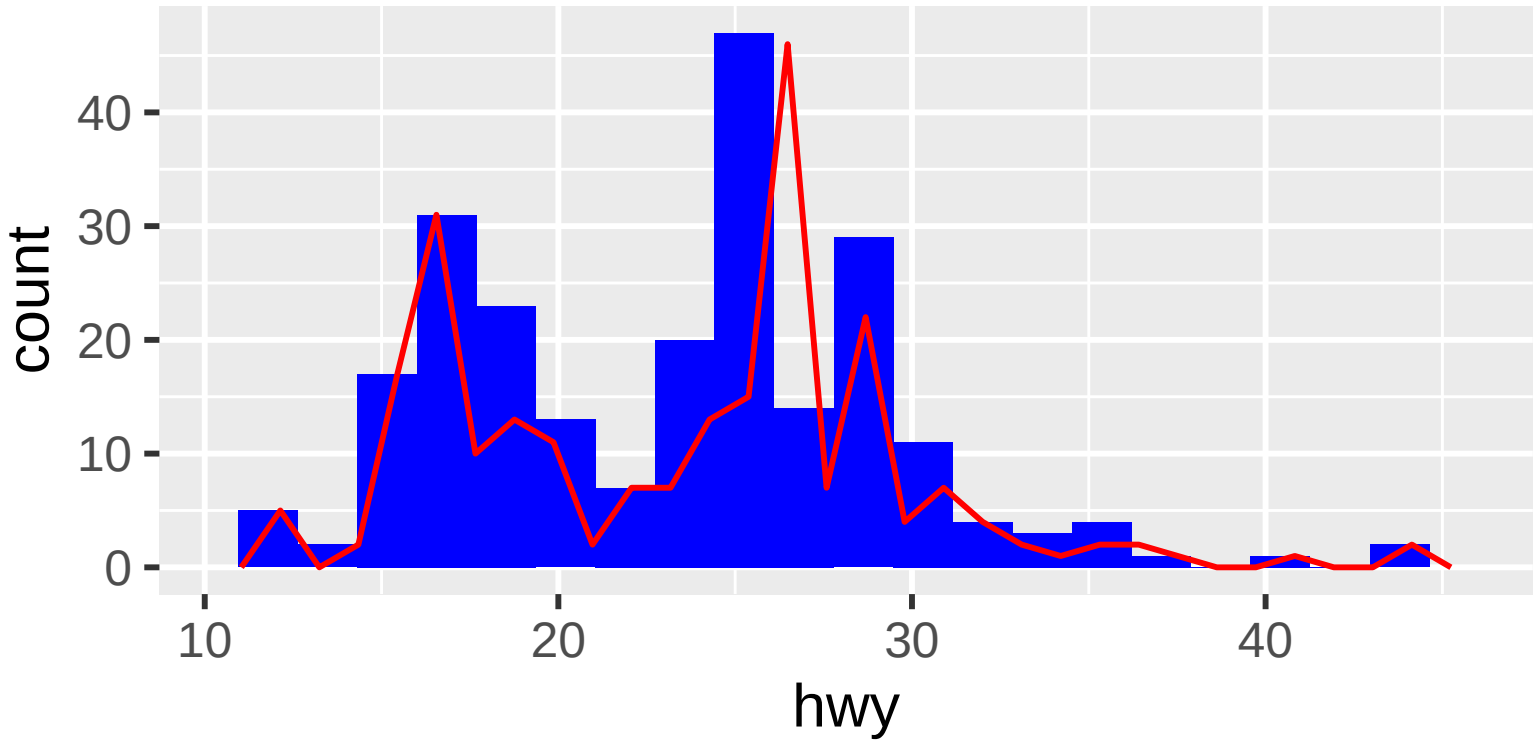
```
GG.Hist+
    geom_histogram(fill="blue",bins=20)+
    geom_freqpoly(color="red",bins=20)
```



Histogramas

Notar que si no se especifican la misma disposición de los intervalos no coinciden el histograma con el polígono de frecuencias:

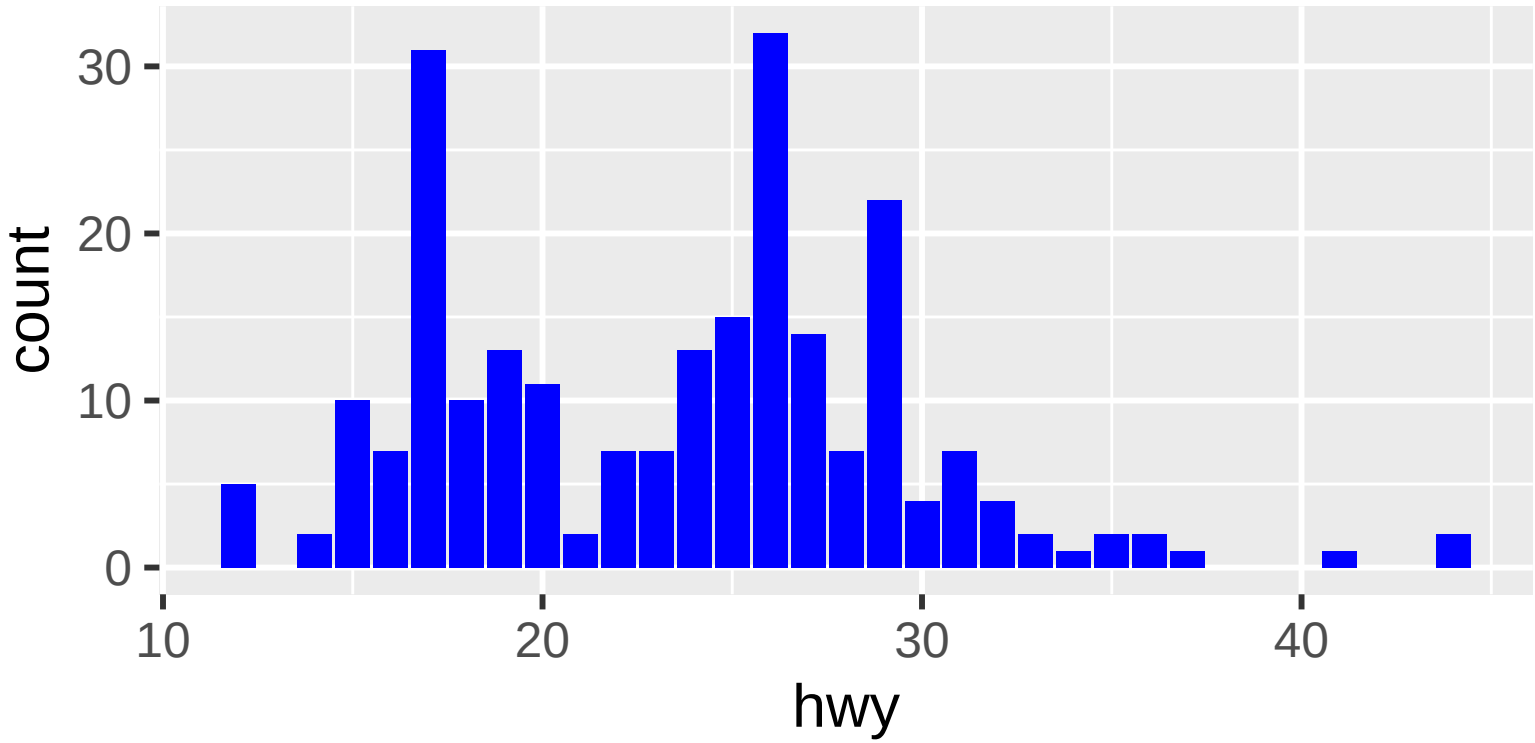
```
GG.Hist+
  geom_histogram(fill="blue",bins=20)+
  geom_freqpoly(color="red",bins=30)
```



Histogramas

Notar que si se usa el comando geom_bar, las barras quedan separadas, ya que está pensado para factores o variables discretas. Sin embargo, las barras deberían ser contiguas ya que se considera que la variable es continua:

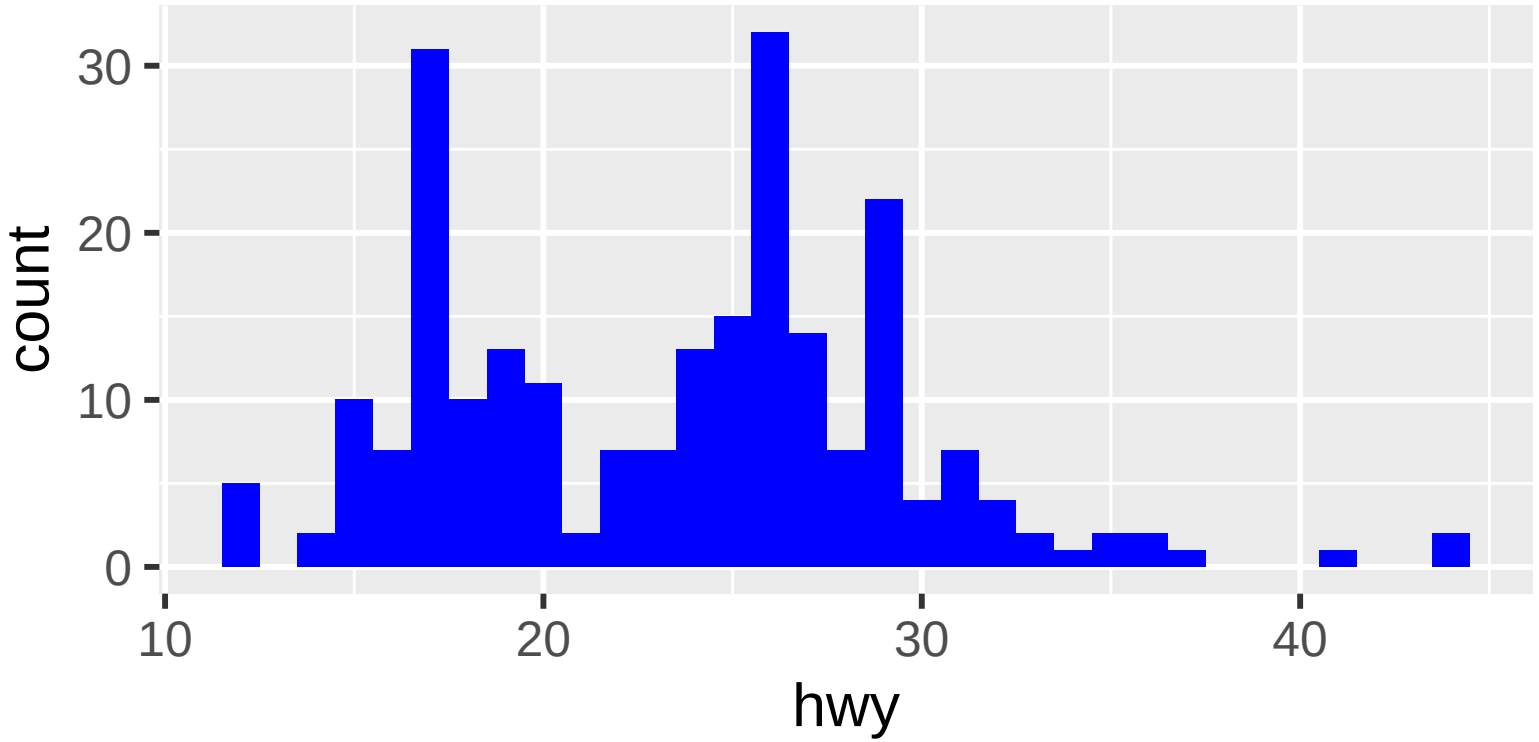
```
ggplot(mpg,aes(x=hwy))+
  geom_bar(fill="blue")
```



Histogramas

Esto se puede arreglar ajustando el ancho (width):

```
ggplot(mpg,aes(x=hwy))+
  geom_bar(fill="blue",width=1)
```



Histogramas

Por último, ggplot se va a quejar si se usa una variable categórica como variable a describir:

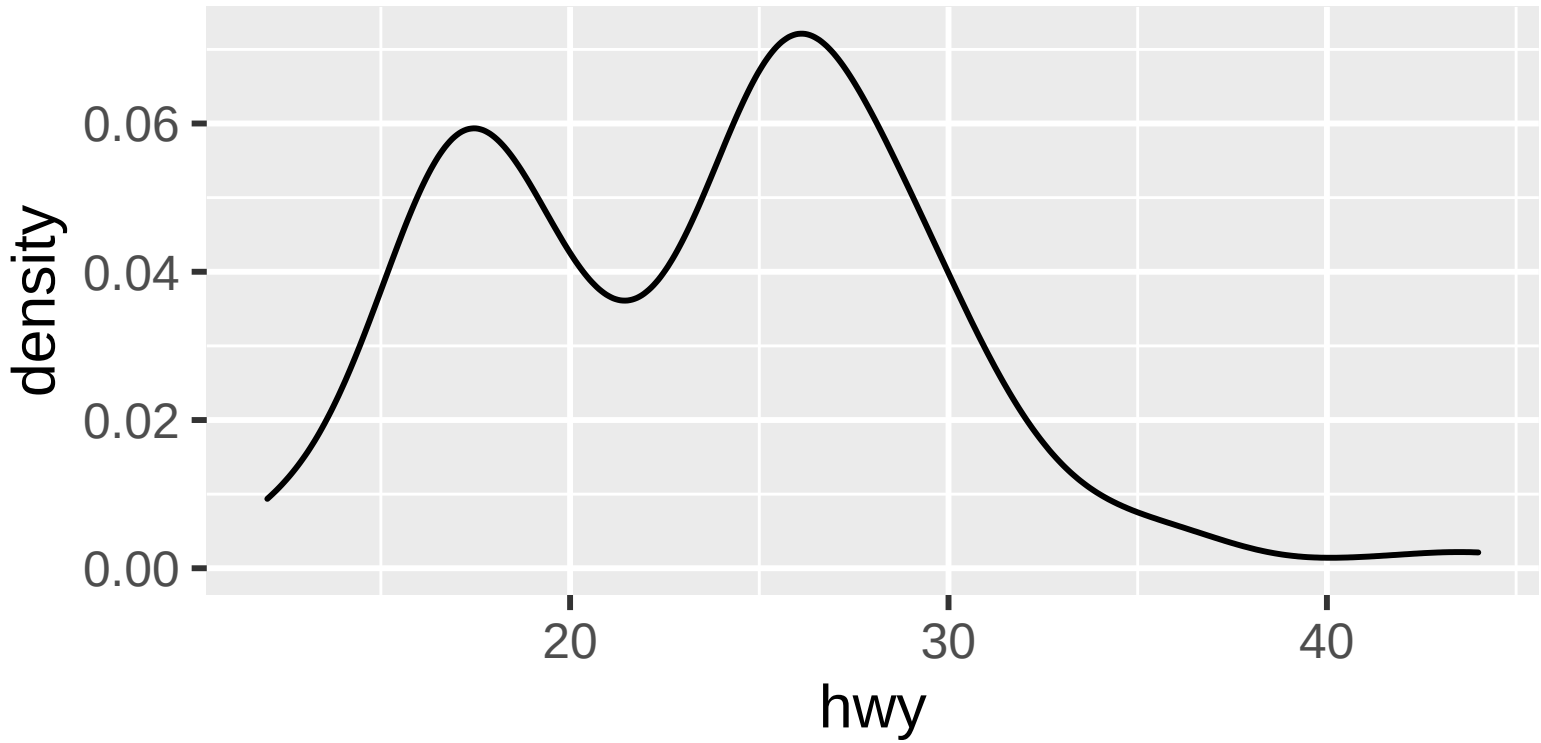
```
ggplot(mpg,aes(x=drv))+
  geom_histogram()
```

```
## Error: StatBin requires a continuous x variable: the x variable is discre
```

Density plot

Una alternativa al histograma es el comando geom_density en el que se estima la densidad de la variable considerada continua:

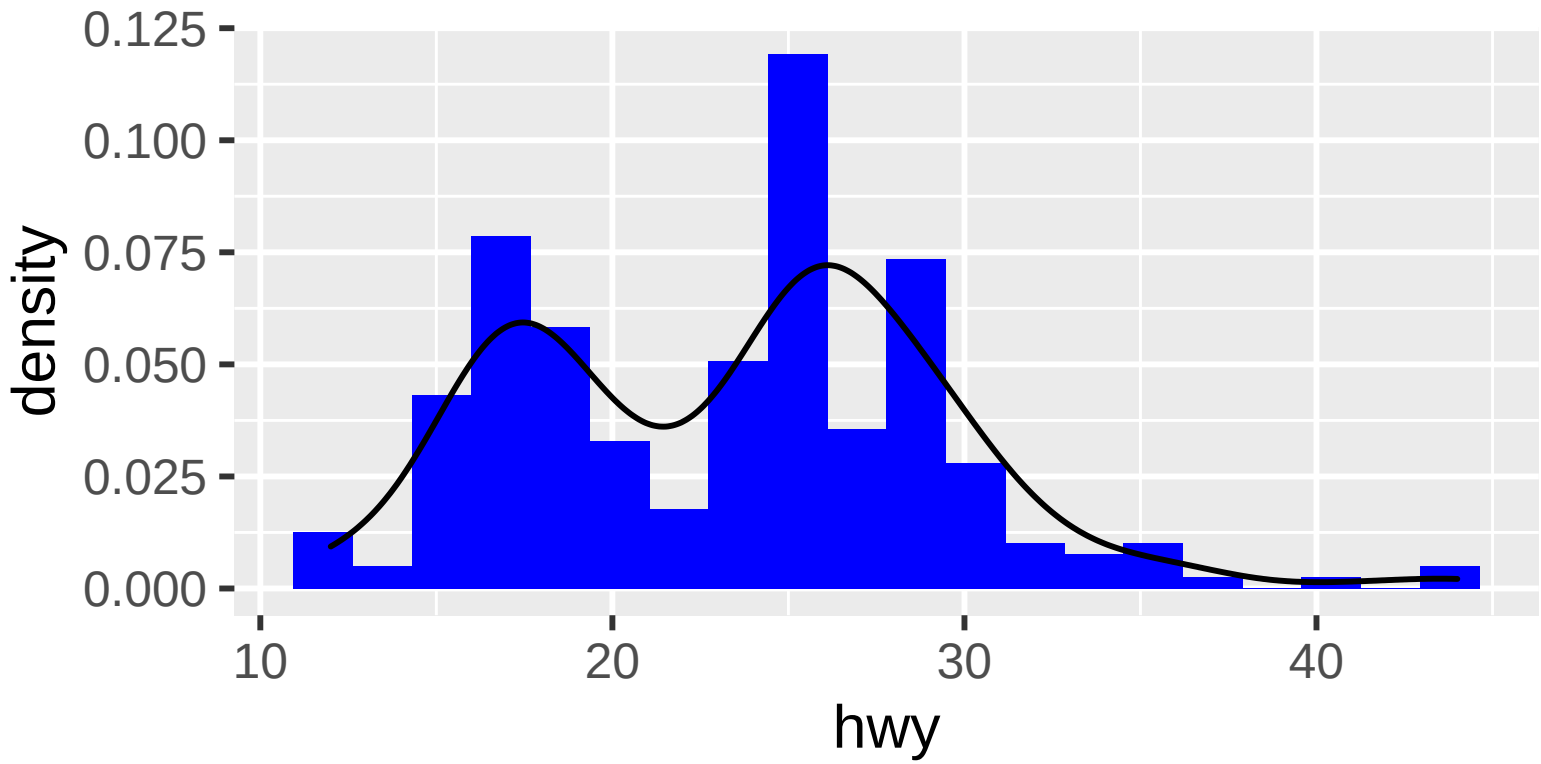
```
ggplot(mpg,aes(x=hwy))+
  geom_density()
```



Density plot

Notar que cuando se cuentan las frecuencias relativas en cada intervalo, (y=..density..) se puede comparar el histograma con la estimación de la función de densidad:

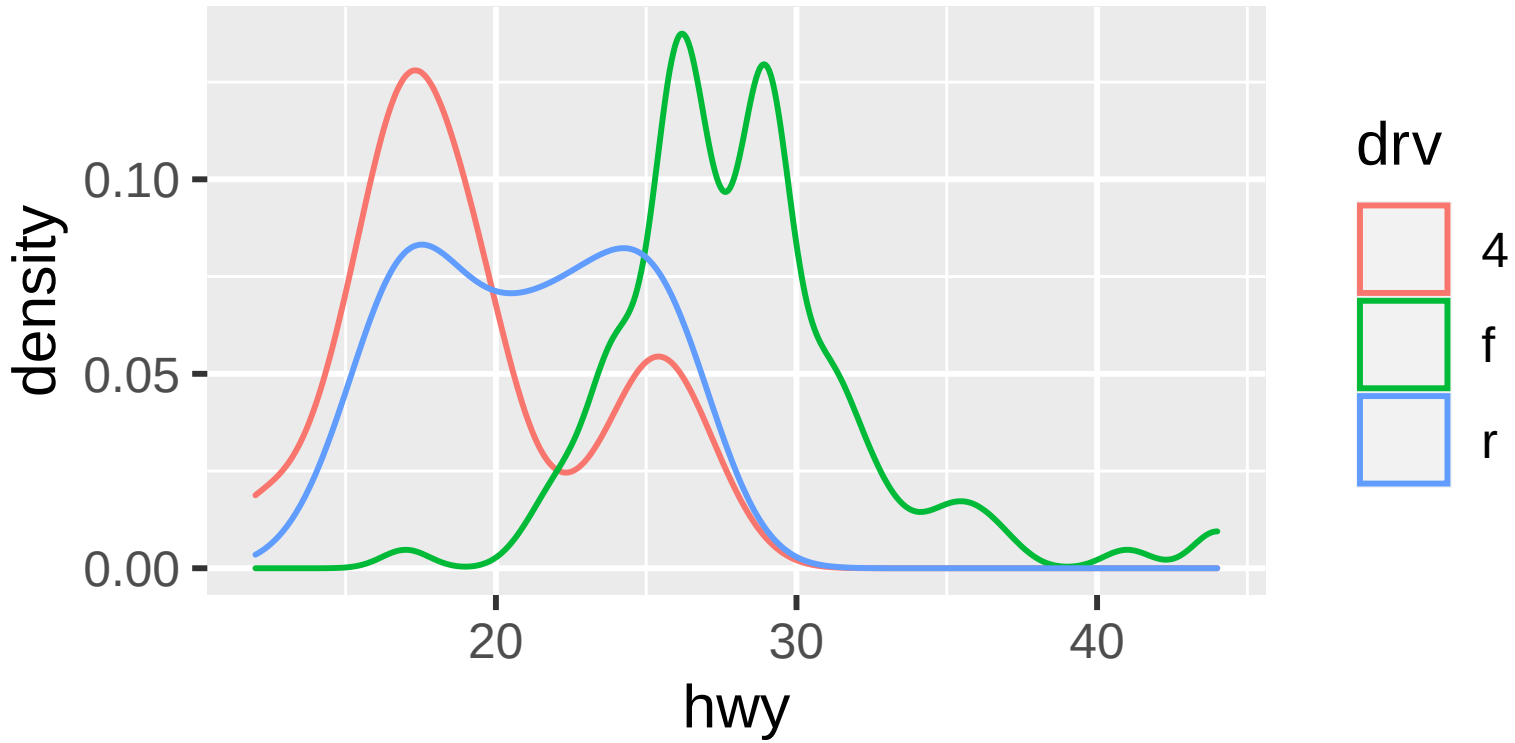
```
ggplot(mpg,aes(x=hwy))+
  geom_histogram(aes(y=..density..),fill="blue",bins=20)+
  geom_density()
```



Density plot

Podemos además utilizar algun factor para comparar en un mismo plano las densidades estimadas:

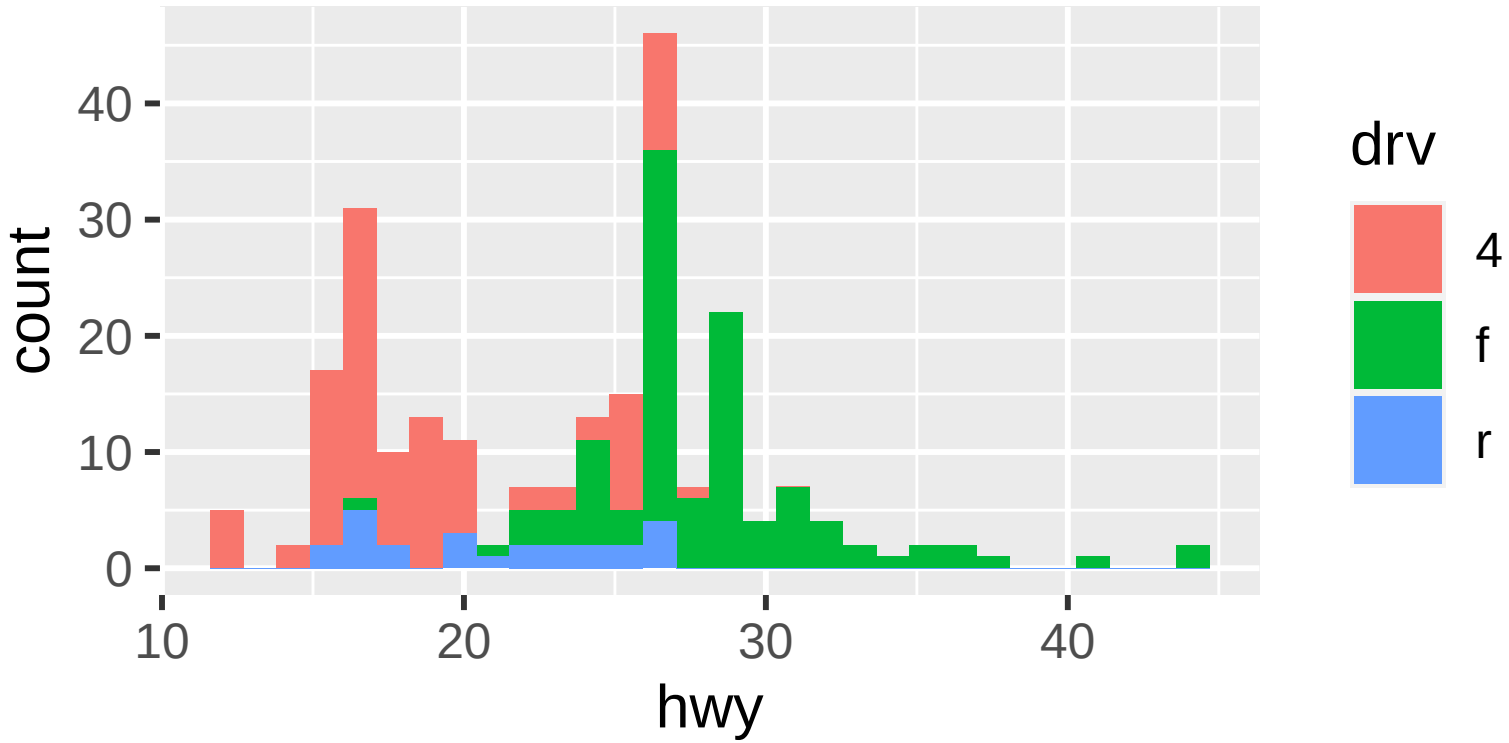
```
ggplot(mpg,aes(x=hwy,color=drv))+
  geom_density()
```



Density plot

Notar que esto simplifica mucho el caso en el que se quieren comparar histogramas, ya que se superponen:

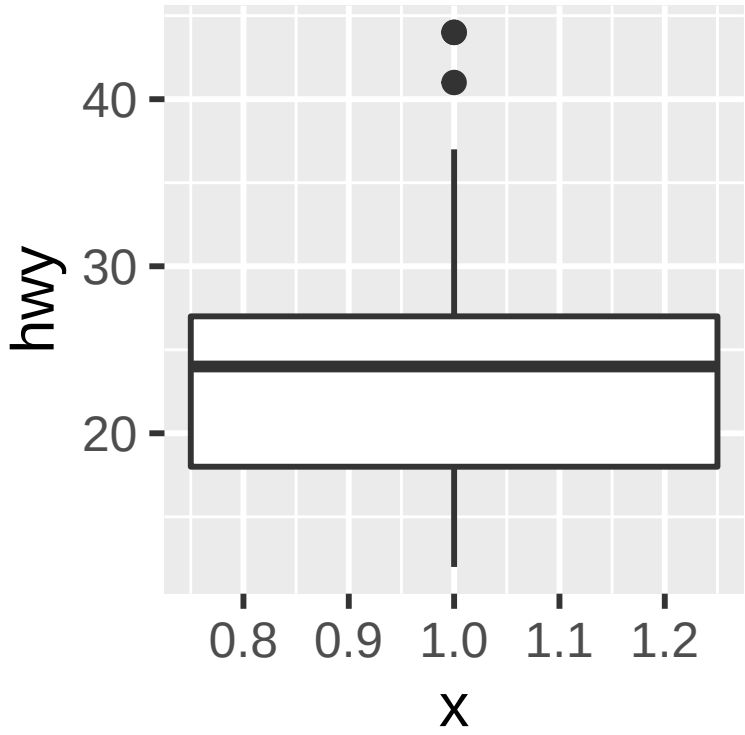
```
ggplot(mpg,aes(x=hwy,fill=drv))+
  geom_histogram()
```



Boxplots

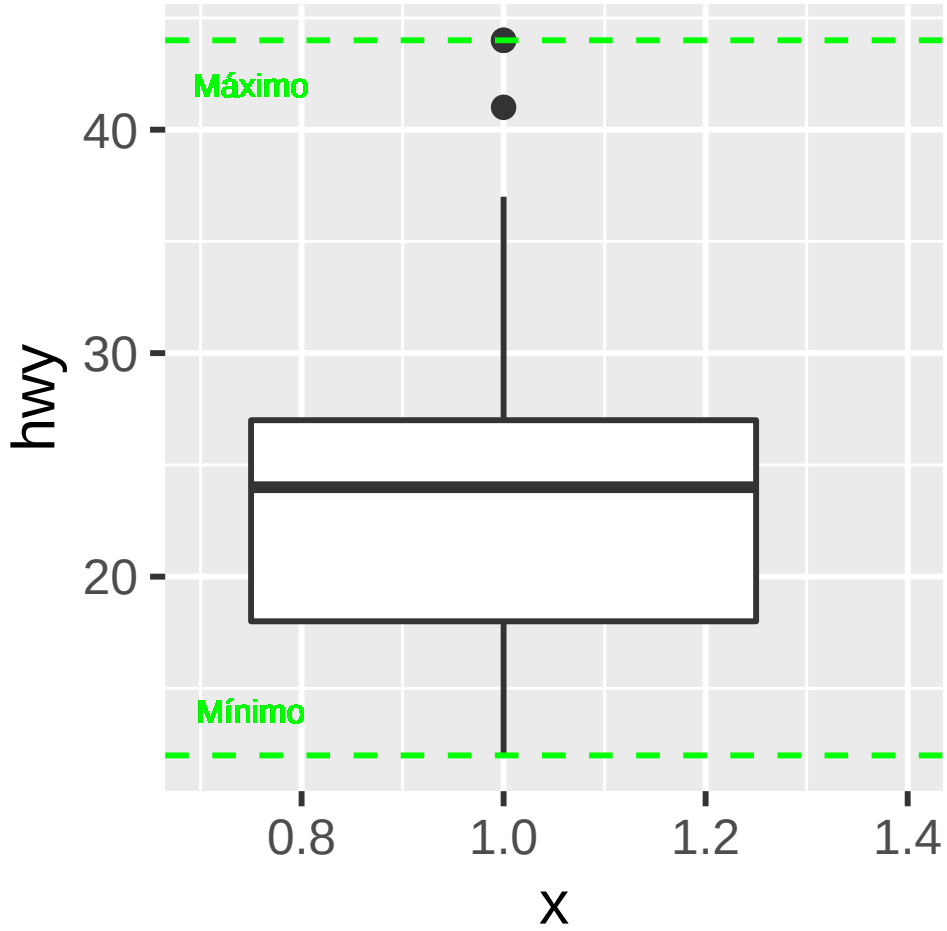
Otra forma de analizar cómo se distribuye una variable continua es con un boxplot:

```
GG.Box=ggplot(mpg,aes(x=1,y=hwy))+
  geom_boxplot(width=0.5)
GG.Box
```



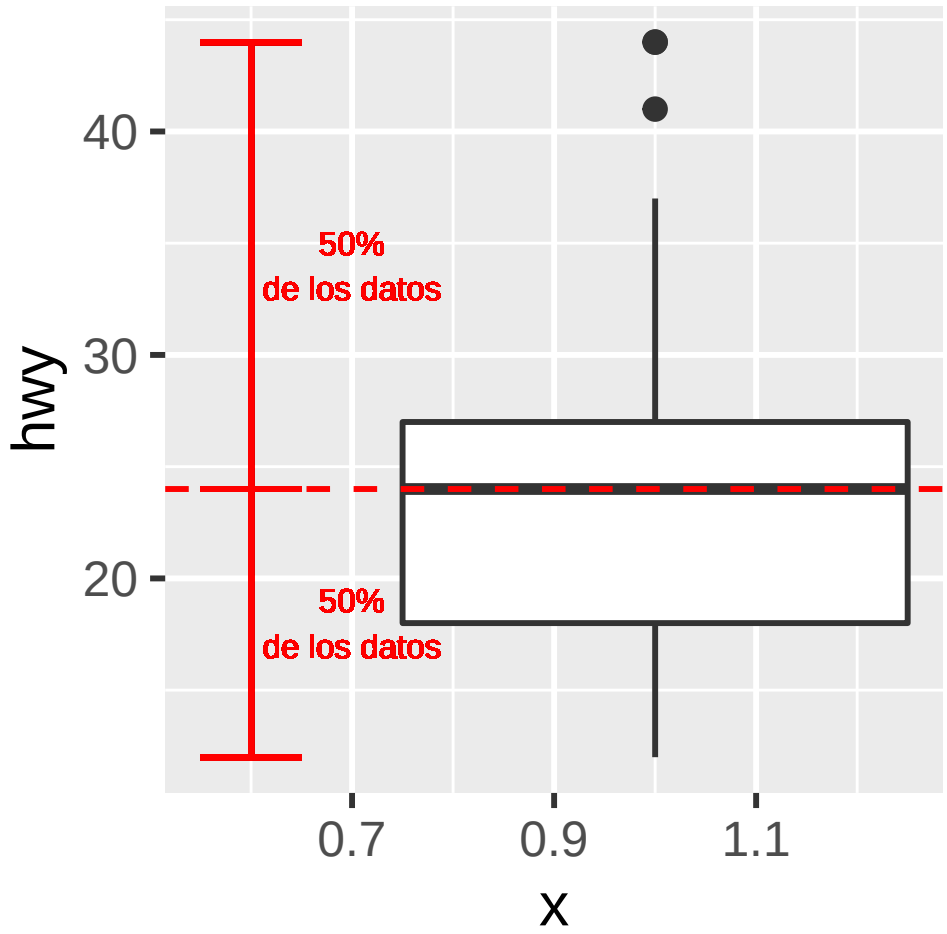
Boxplots

El boxplot se extiende entre el máximo y el mínimo de la variable:



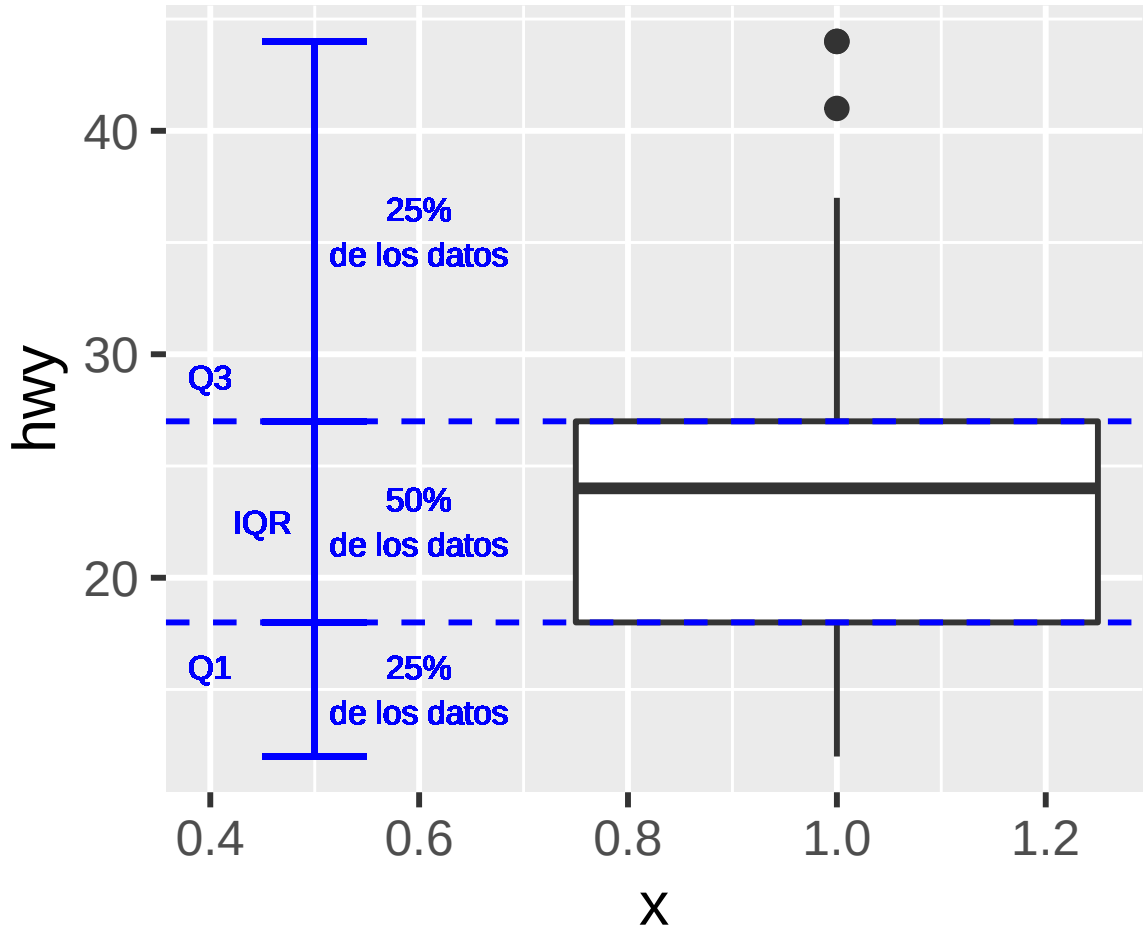
Boxplots

La línea central de la caja está a la altura del valor que divide la cantidad de datos (ordenados) en dos:



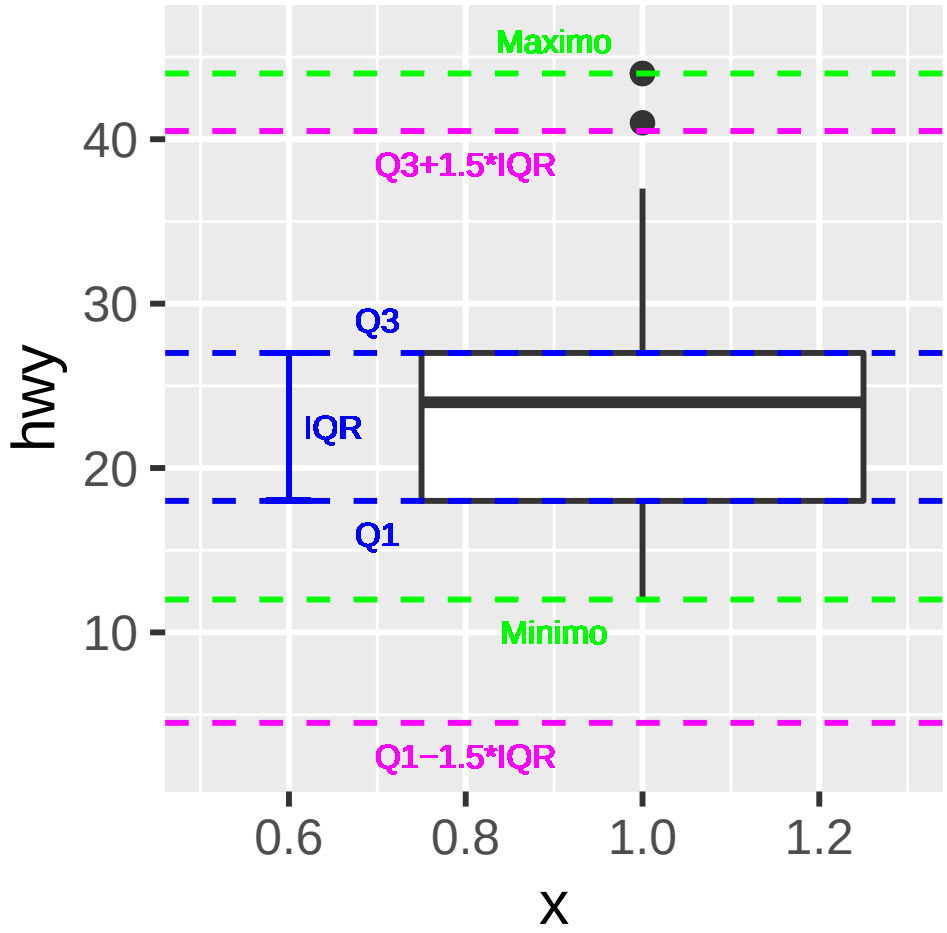
Boxplots

Los límites de la caja central (los cuartiles Q1 y Q3) se extienden de forma de comprender el 50% central de los datos, la caja tiene una longitud IQR (rango intercuartil):



Boxplots

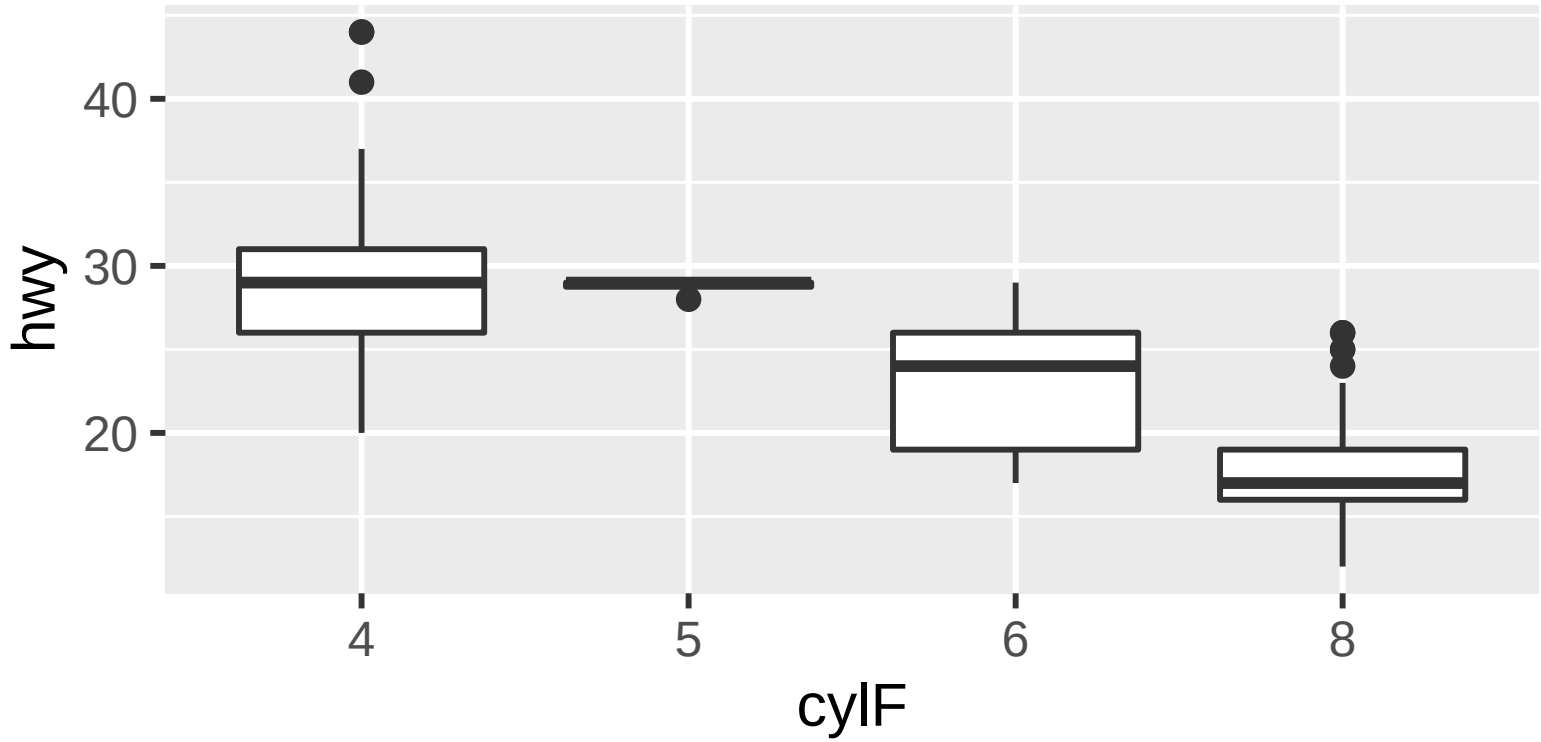
Cuando una variable excede los límites de Tukey (Q1-1.5*IQR,Q3+1.5*IQR) se consideran valores atípicos o outliers y se grafican con puntos para diferenciarlos:



Boxplots

Además, se pueden establecer diferentes boxplots en paralelo usando la variable x (idealmente discreta):

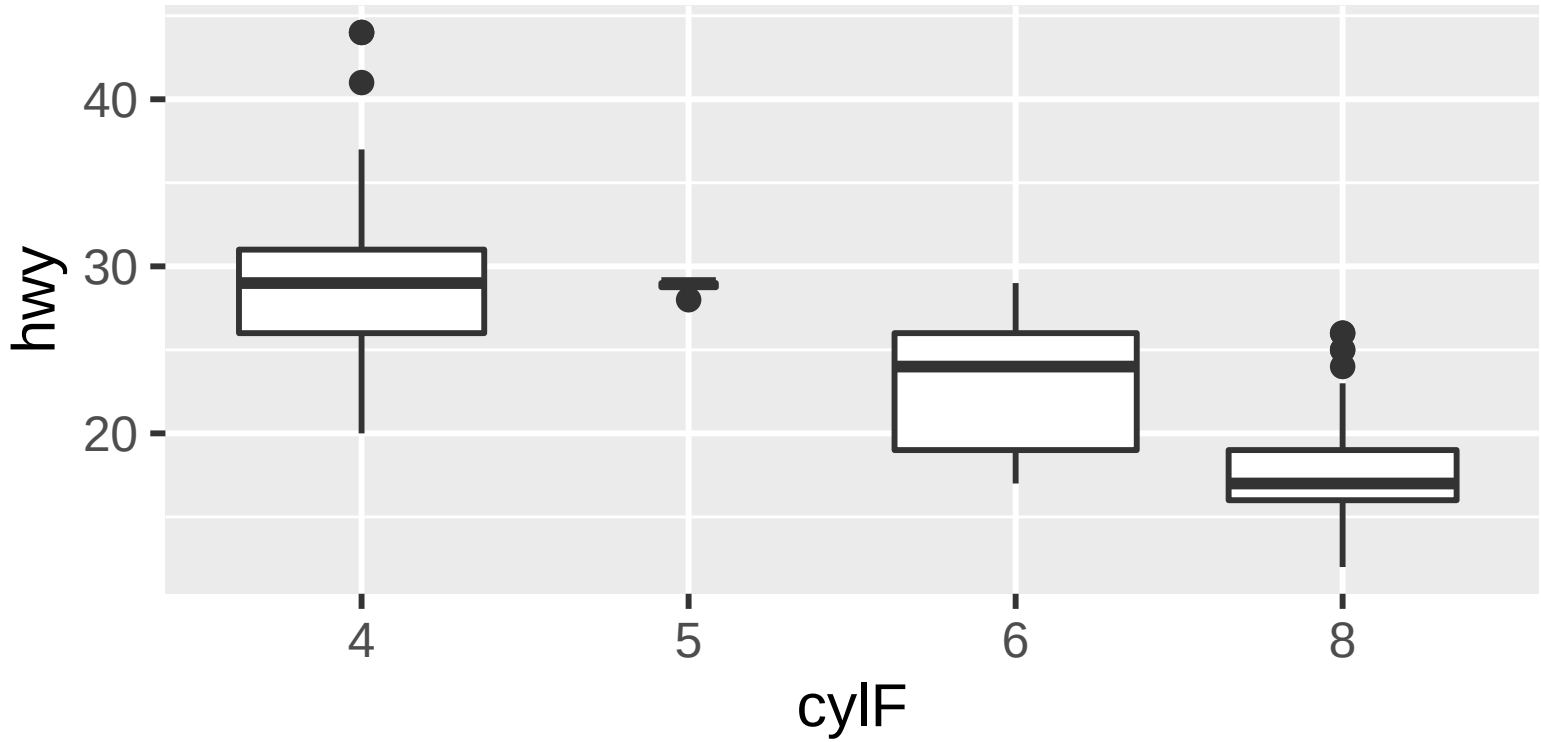
```
GG.ParBox=ggplot(mpg,aes(x=cylF,y=hwy))
GG.ParBox+
  geom_boxplot()
```



Boxplots

Cuando los boxplots están en paralelo para ser comparados, se puede agregar el comando varwidth para priorizar los boxplots con mayor cantidad de datos:

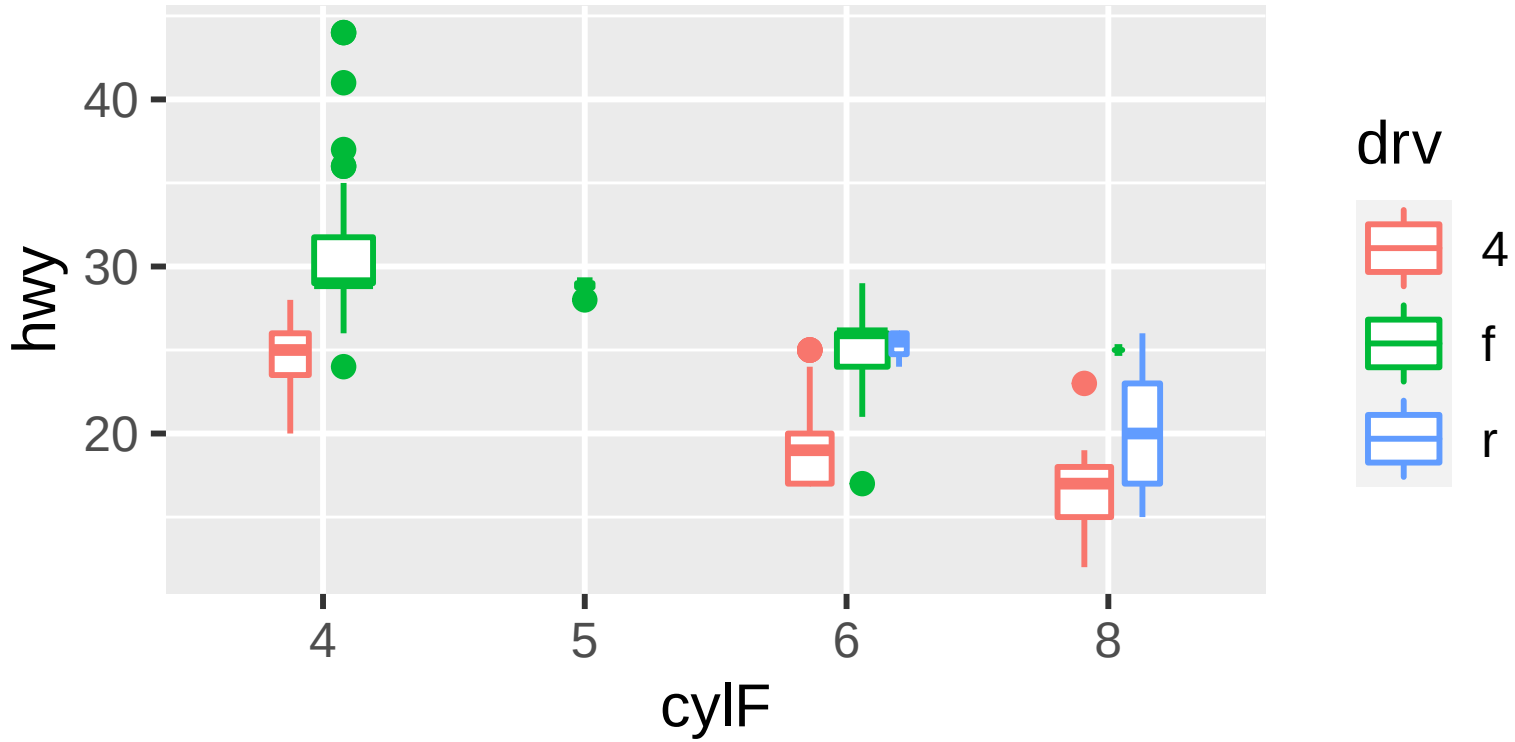
```
GG.ParBox+
  geom_boxplot(varwidth = T)
```



Boxplots

Además, se les puede agregar colores según otro factor, pero usando position="dodge" para que no se superpongan:

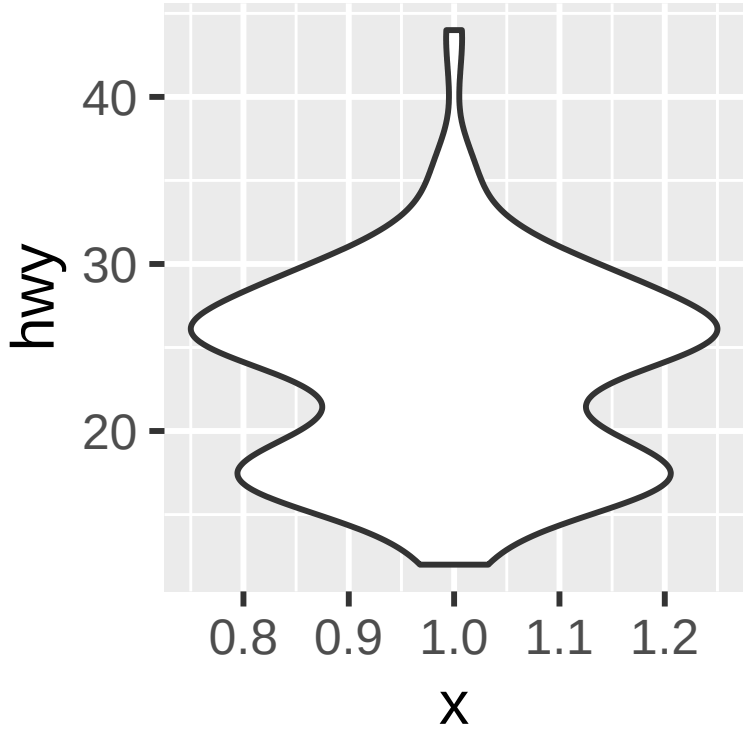
```
GG.ParBox+
  geom_boxplot(aes(color=drv),varwidth = T,position = "dodge")
```



Violin plots

Otra alternativa al boxplot es el violin plot, que también muestra cómo se distribuyen los datos:

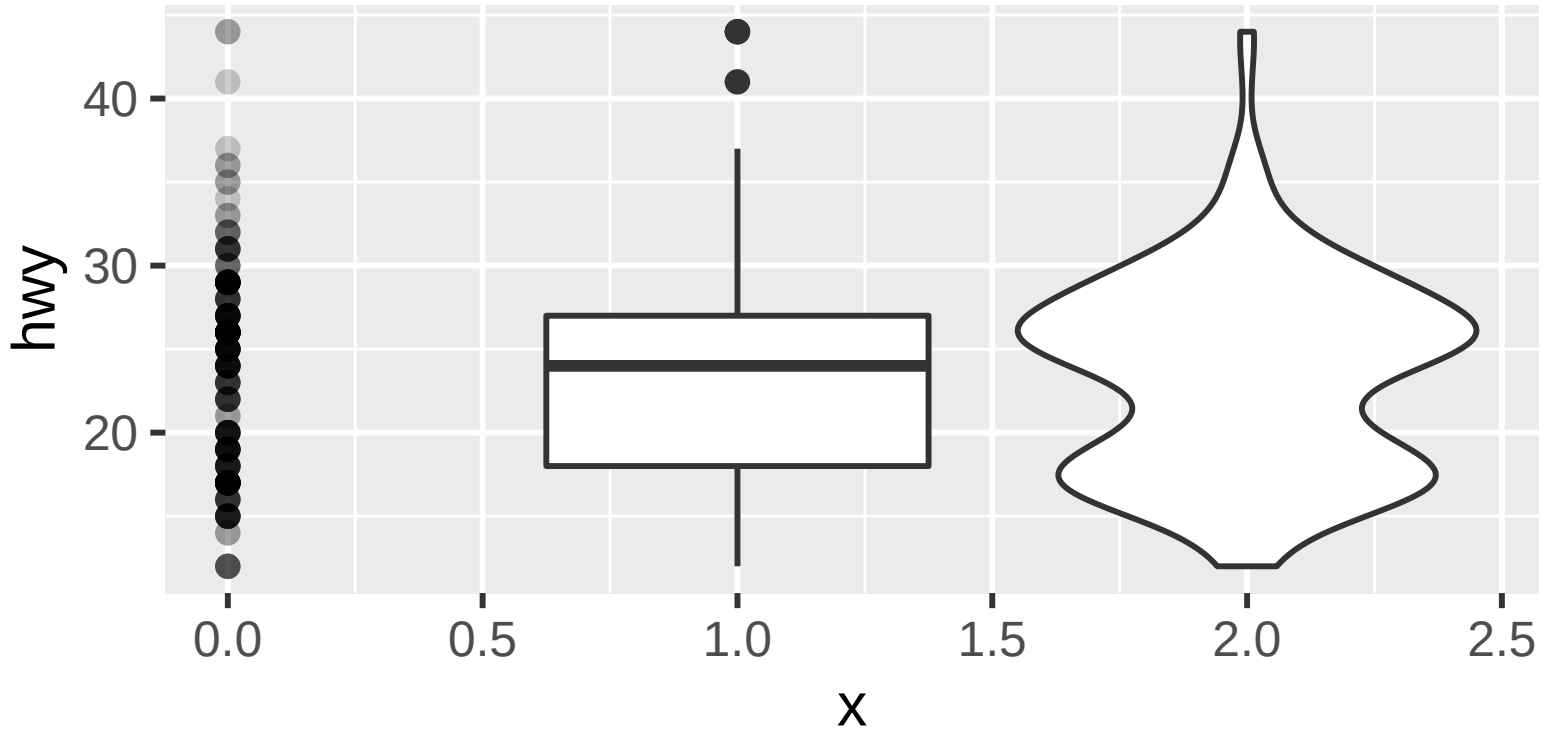
```
ggplot(mpg,aes(x=1,y=hwy))+
  geom_violin(width=0.5)
```



Violin plots

Notar que se pueden ver en paralelo cómo se distribuyen tanto los datos, como el boxplot y el violin plot:

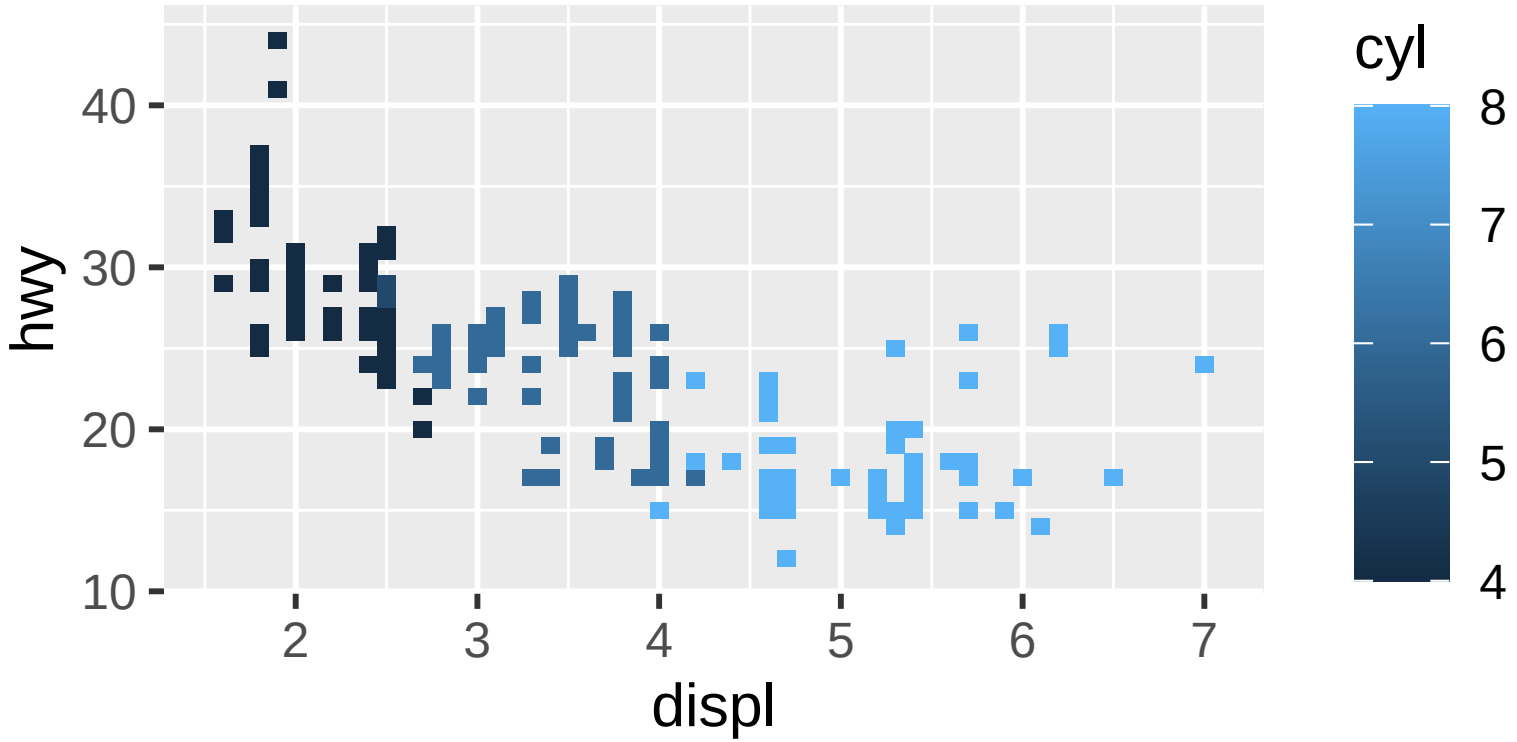
```
ggplot(mpg,aes(y=hwy))+
  geom_point(aes(x=0),alpha=0.2)+
  geom_boxplot(aes(x=1))+
  geom_violin(aes(x=2))
```



Mapas de calor

Similar a los casos vistos con geom_bin2d y geom_hex, se puede realizar un mapa de calor que no necesariamente cuente la cantidad de datos por región del plano. Esto se hace incorporando una variable al comando geom_tile:

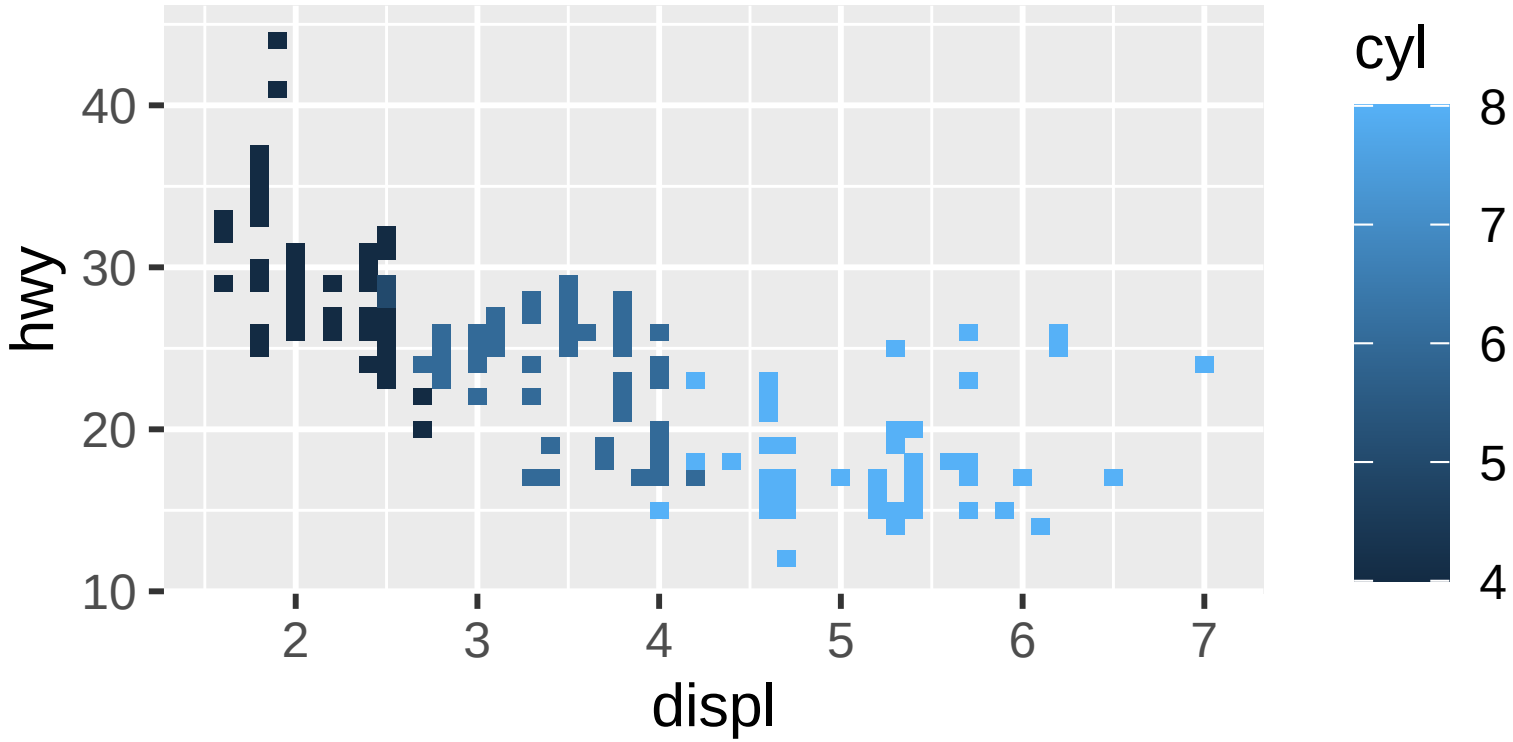
```
GG.Plano+
  geom_tile(aes(fill=cyl))
```



Mapas de calor

Similar a los casos vistos con geom_bin2d y geom_hex, se puede realizar un mapa de calor que no necesariamente cuente la cantidad de datos por región del plano. Esto se hace incorporando una variable al comando geom_tile:

```
GG.Plano+
  geom_tile(aes(fill=cyl))
```



Cuando los datos completan el plano se visualiza mucho mejor.

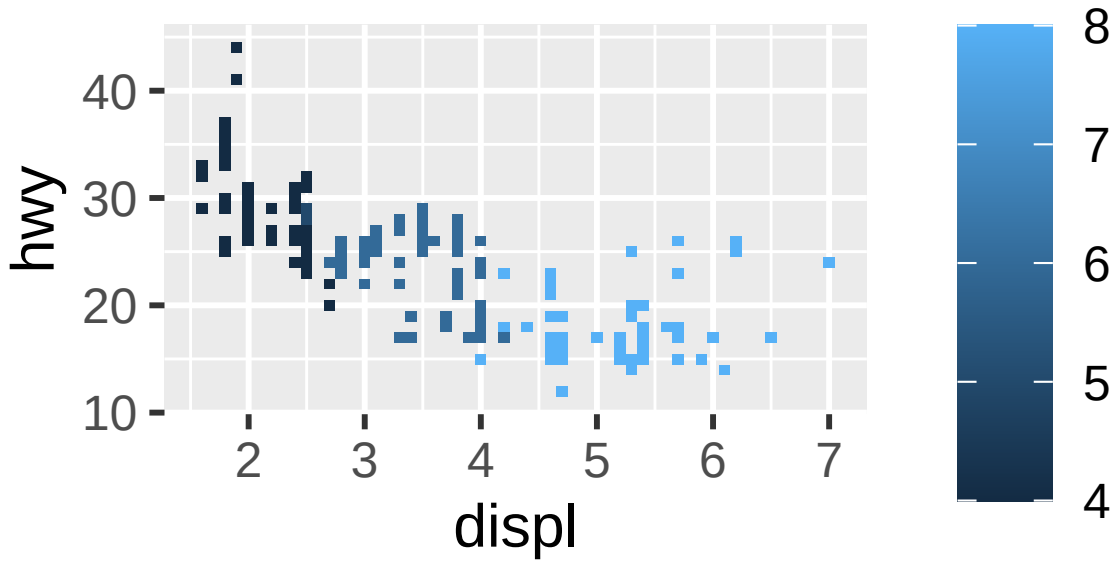
Mapas de calor

Otra alternativa es el comando geom_raster:

```
GG.Plano+
  geom_raster(aes(fill=cyl))
```

```
## Warning: Raster pixels are placed at uneven horizontal intervals and will
## shifted. Consider using geom_tile() instead.

## Warning: Raster pixels are placed at uneven vertical intervals and will b
## shifted. Consider using geom_tile() instead.
```



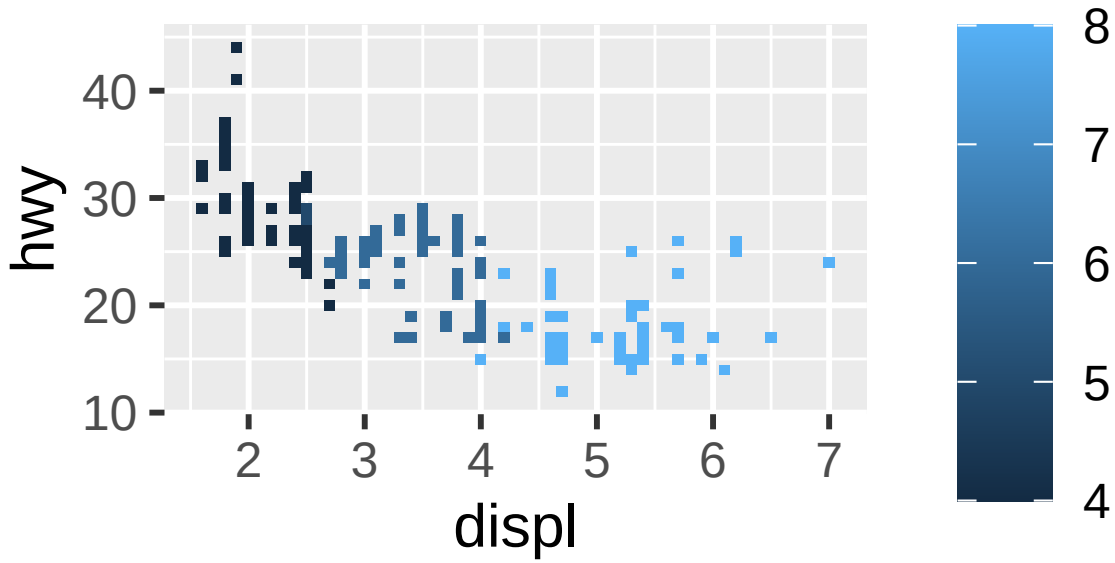
Mapas de calor

Otra alternativa es el comando geom_raster:

```
GG.Plano+
  geom_raster(aes(fill=cyl))
```

```
## Warning: Raster pixels are placed at uneven horizontal intervals and will
## shifted. Consider using geom_tile() instead.
```

```
## Warning: Raster pixels are placed at uneven vertical intervals and will b
## shifted. Consider using geom_tile() instead.
```



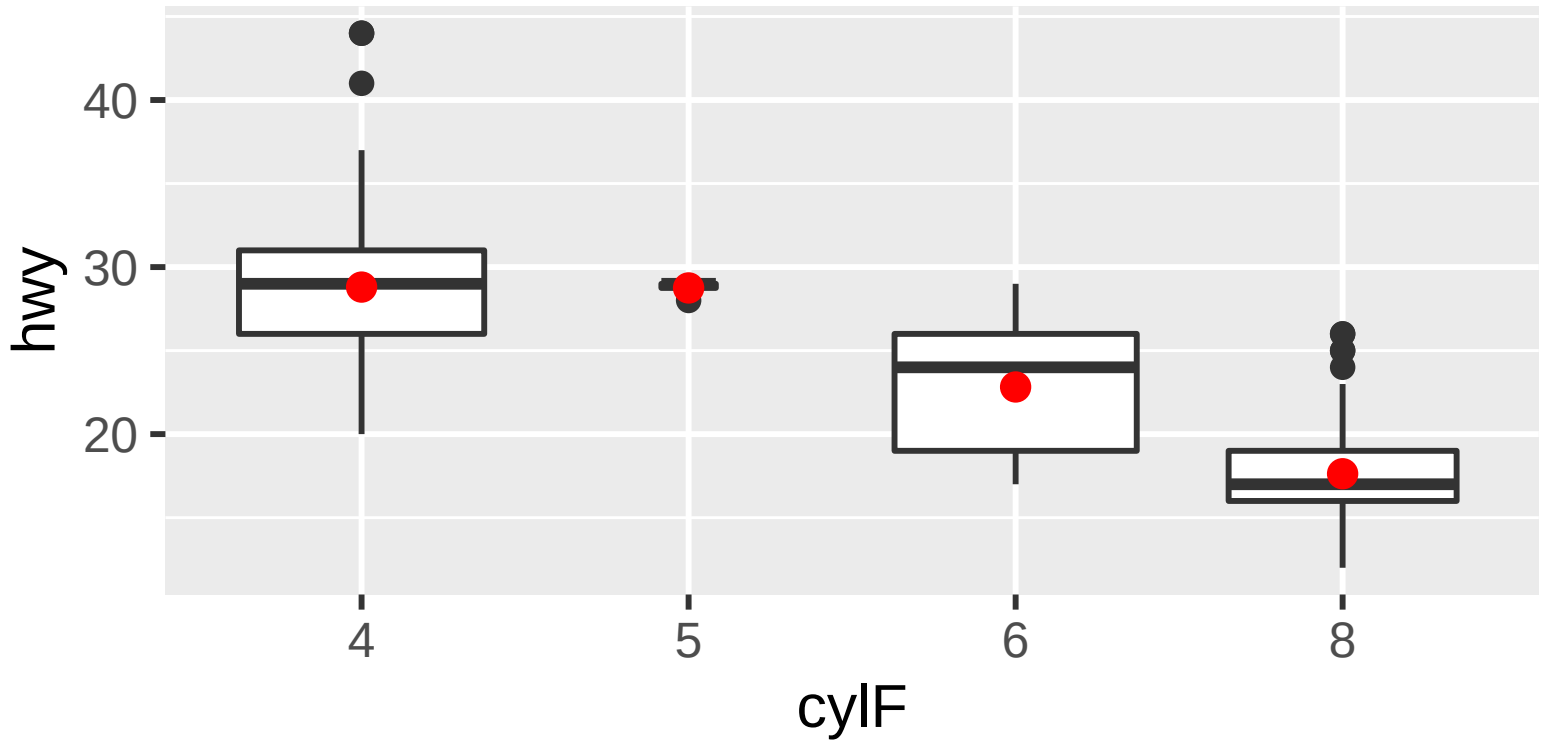
Nuevamente, cuando los datos completan el plano se visualiza mucho mejor.

Stats

stat_summary

Usando comandos del tipo stat, se pueden agregar objetos aplicando transformaciones estadísticas a las variables. Por ejemplo, es muy común que algunos boxplots tengan un punto representando la media (fun="mean", por default, calcula la media de la variable y para cada valor de x), ya que no siempre coincide con la mediana:

```
GG.ParBox+
  geom_boxplot(varwidth = T)+
  stat_summary(geom="point",fun = "mean",
               color="red",size=2)
```

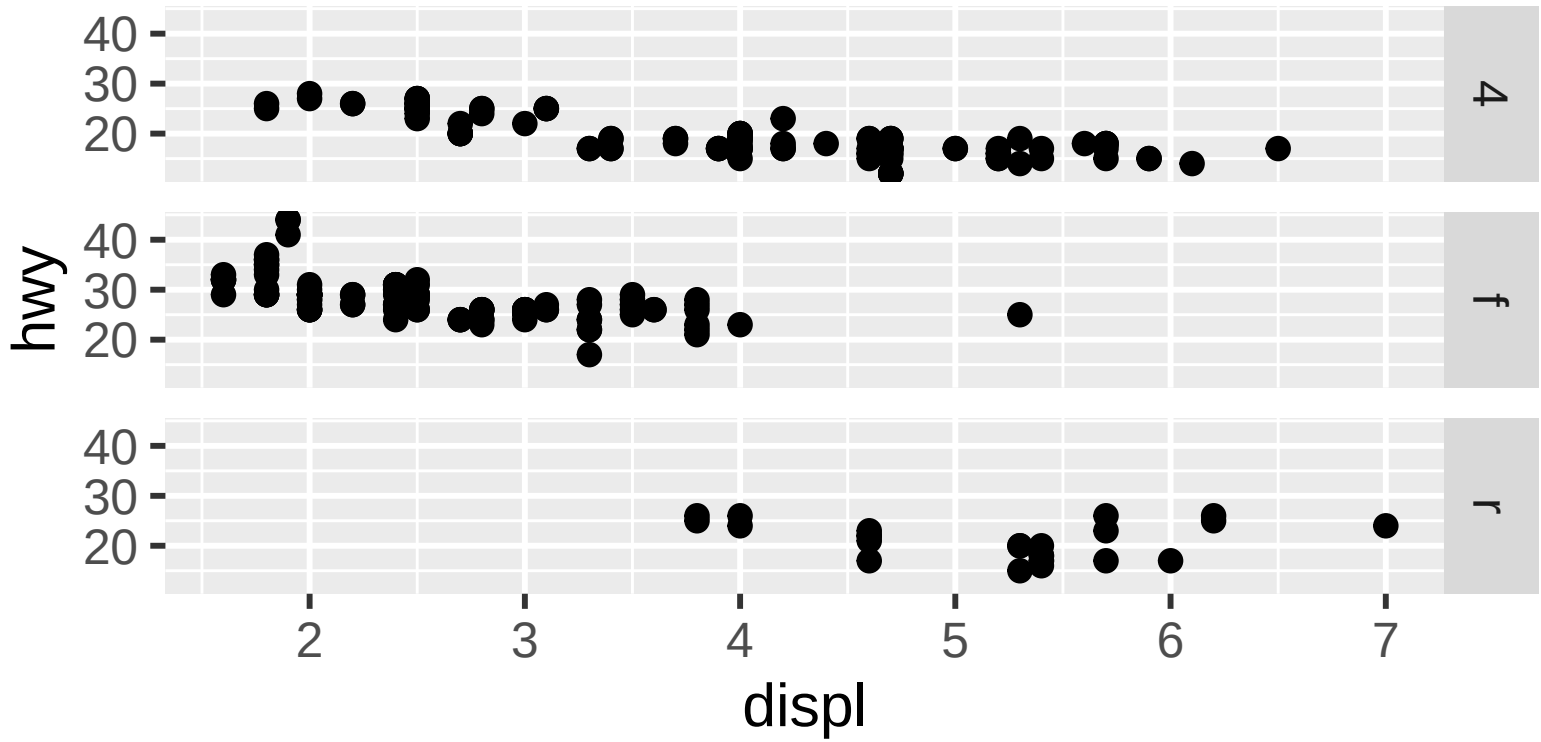


Paneles

Facet_grid

Una vez establecidas las variables estéticas, se pueden añadir nuevas perspectivas con el comando facet_grid, que divide los datos según una variable categórica. En este caso, usamos la variable drv. El símbolo ~ se puede considerar como "en función de" y el punto "." se usa para especificar que cuando se divide en paneles, no se usa ninguna otra variable para la categorización:

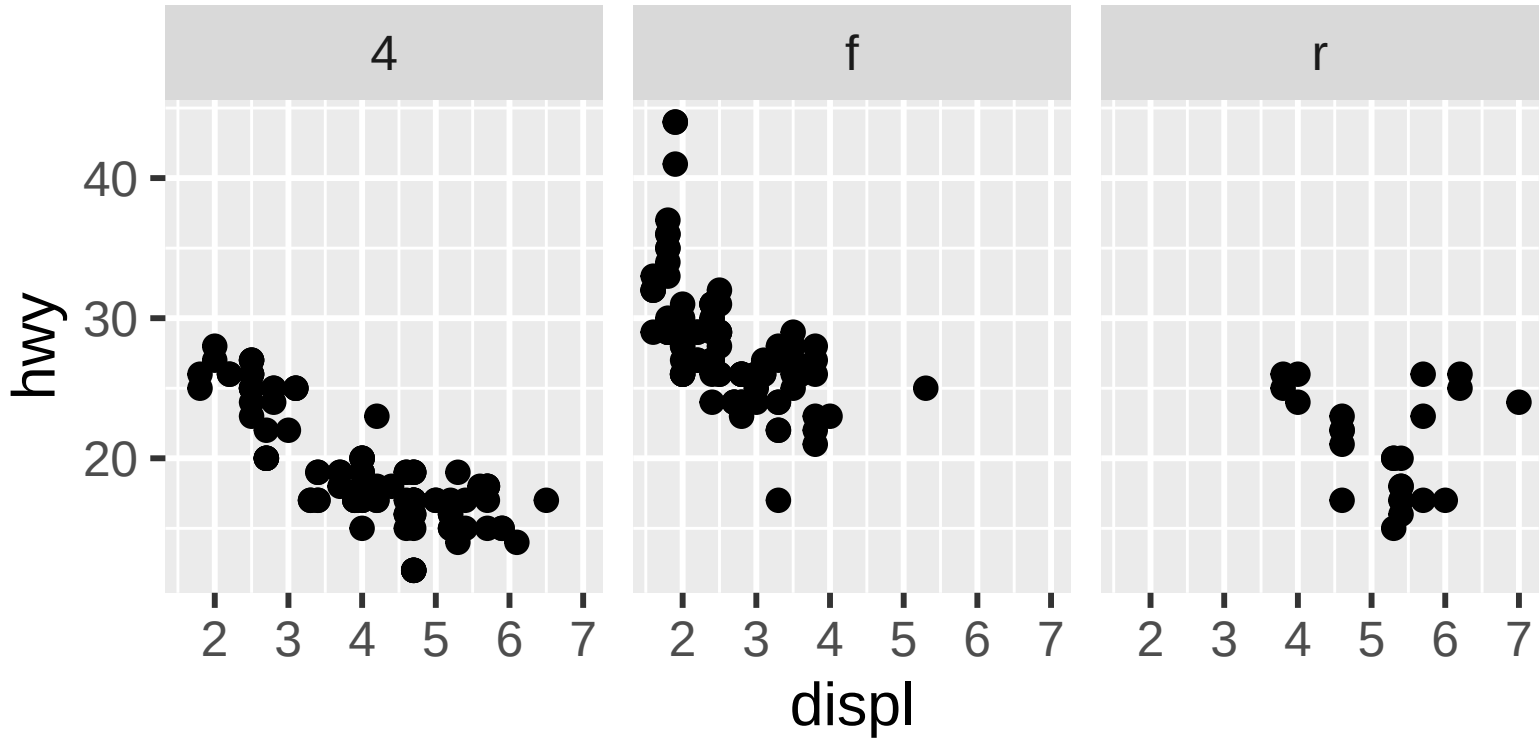
```
GG.Scatter+
  facet_grid(drv~.)
```



Facet_grid

Cuando se invierte el orden de las variables respecto del símbolo ~, se genera una disposición horizontal de los paneles:

```
GG.Scatter+
  facet_grid(.~drv)
```



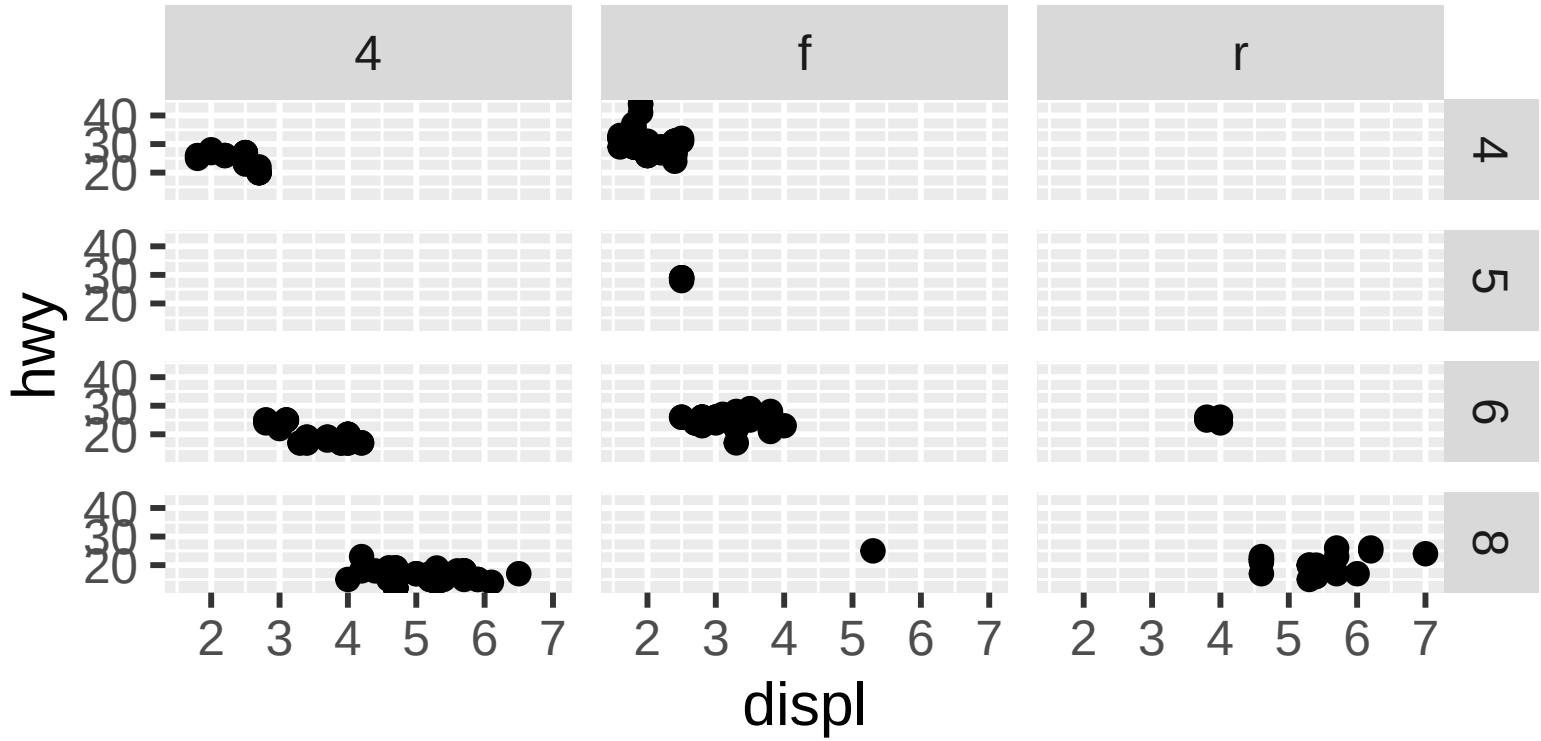
Facet_grid

El símbolo $\sim$ se puede pensar de modo que lo que está a la izquierda del símbolo es la variable "dependiente" y lo que está a la derecha es la variable "independiente". Por eso, cuando no se especifican más variables, cuando la variable está a la izquierda de $\sim$, hay una disposición vertical de los paneles (eje y) y cuando la variable está a la derecha de $\sim$ la disposición es horizontal (eje x).

Facet_grid

Se puede usar más de un factor:

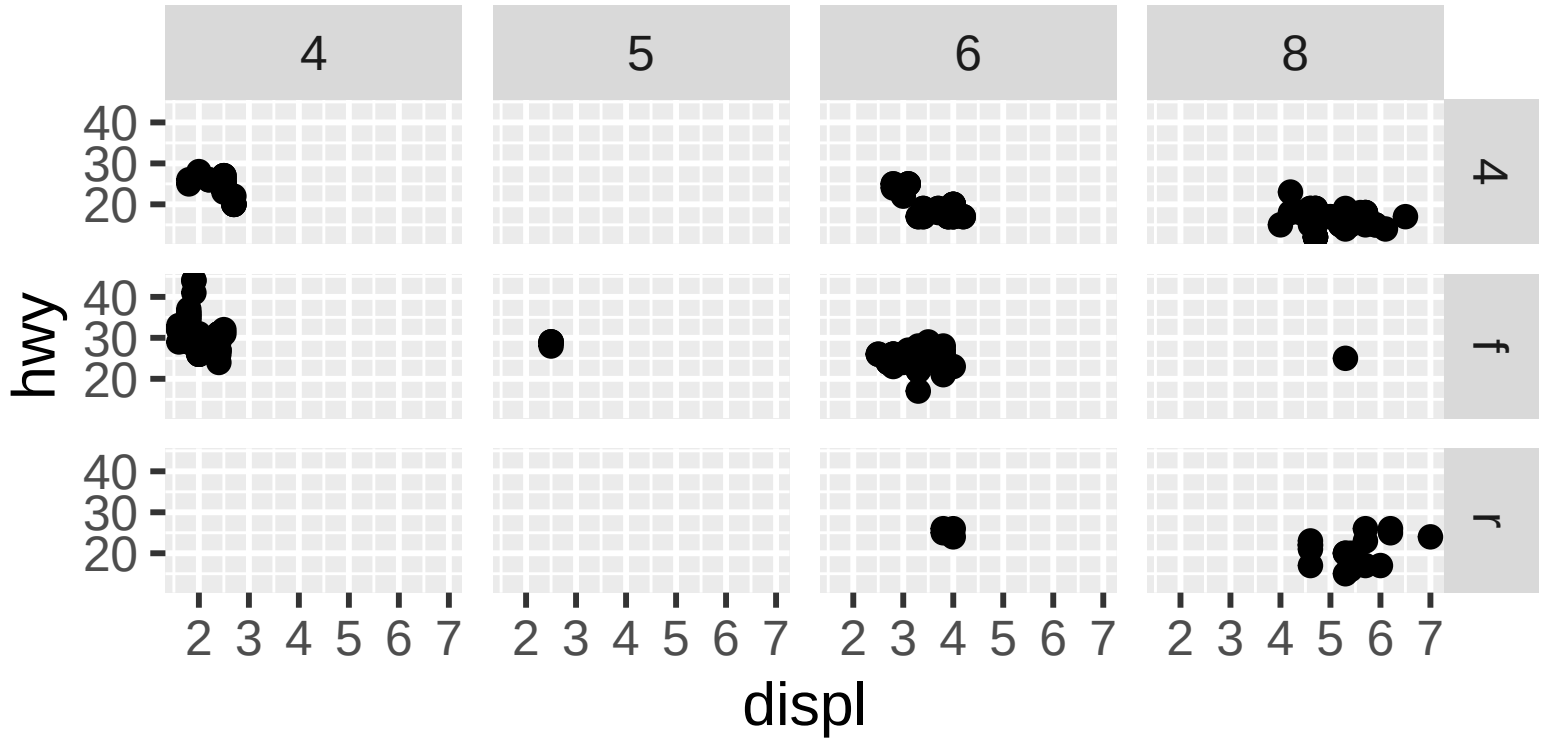
```
GG.Scatter+
  facet_grid(cyl~drv)
```



Facet_grid

Se puede usar más de un factor:

```
GG.Scatter+
  facet_grid(drv~cyl)
```

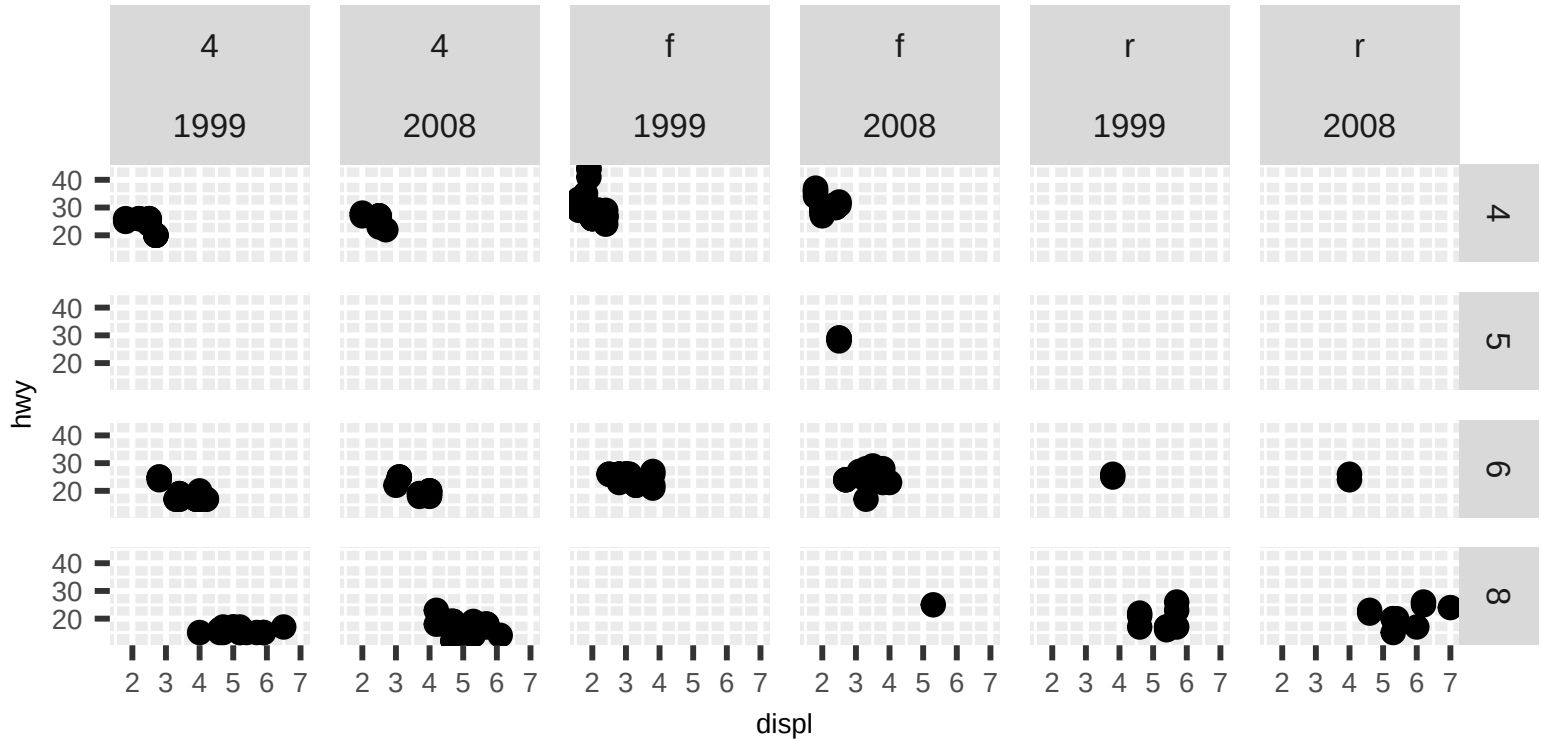


Facet_grid

Más aún, se puede usar más de dos factores:

```
GG.fac=GG.Scatter+
  facet_grid(cyl~drv+year)
```

```
GG.fac
```

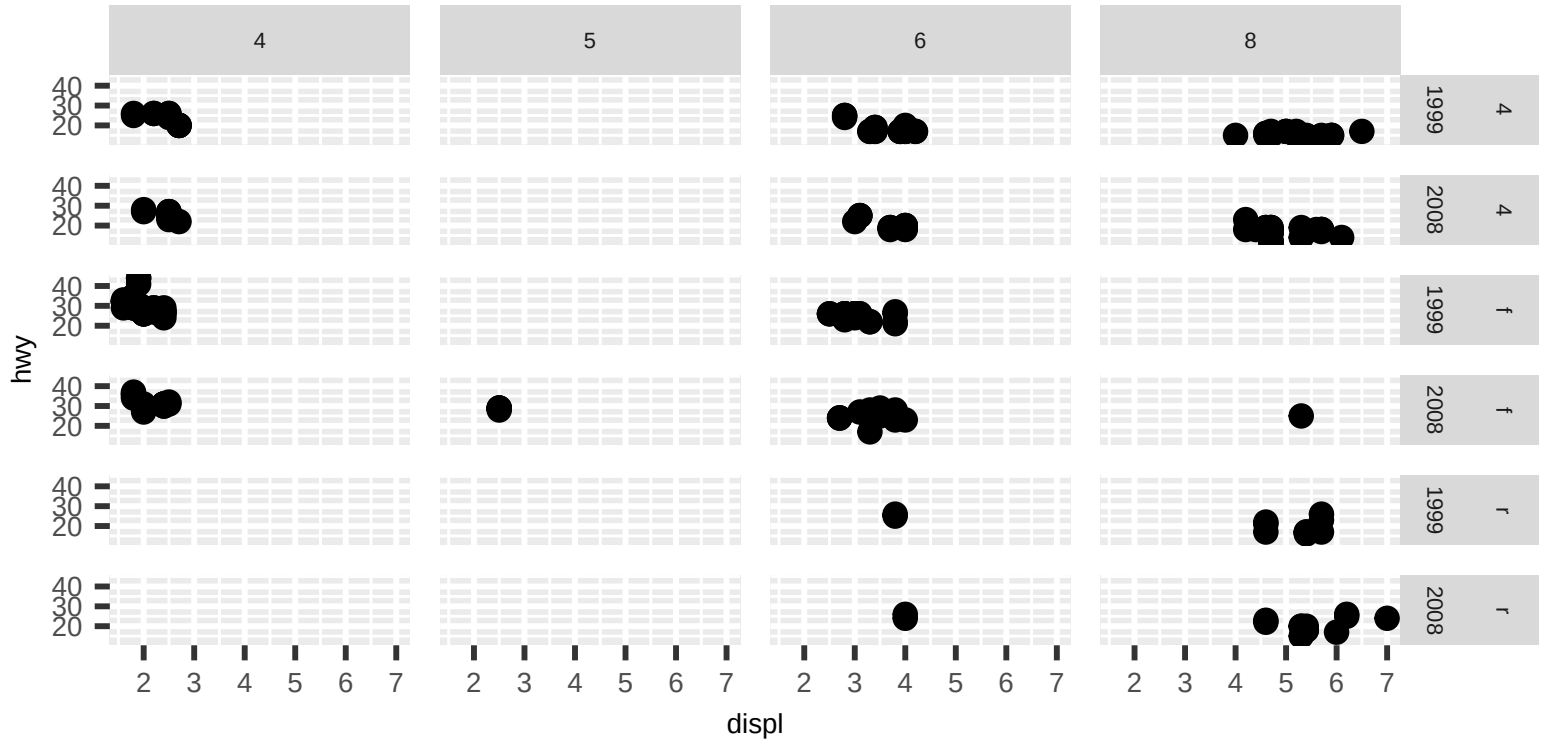


Facet_grid

Más aún, se puede usar más de dos factores:

```
GG.fac=GG.Scatter+
  facet_grid(drv+year~cyl)
```

```
GG.fac
```

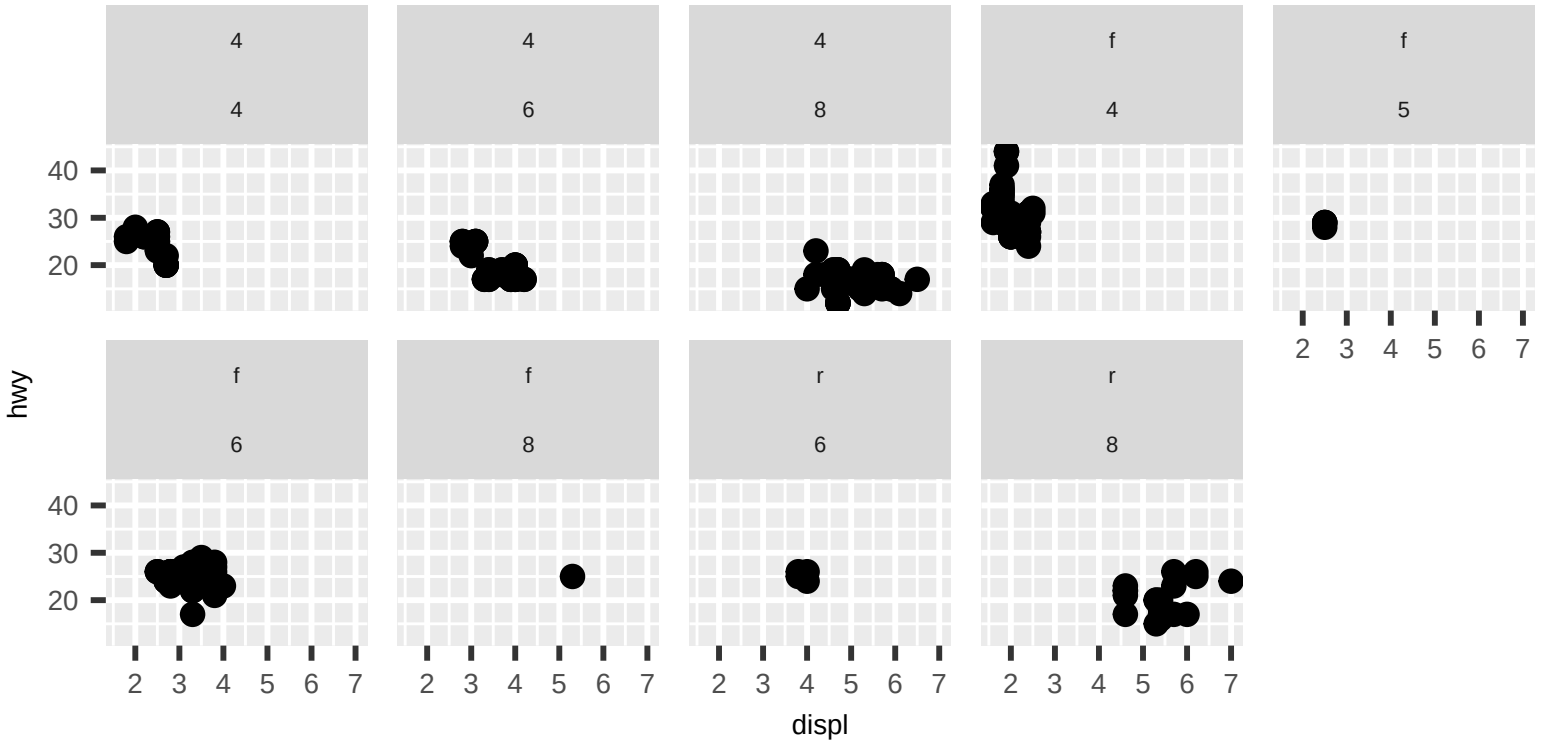


Facet_wrap

Notar que quedan muchos paneles sin datos, ya que no hay intersecciones entre los distintos niveles de los factores. El comando facet_wrap elimina los paneles vacíos:

```
GG.fac=GG.Scatter+
  facet_wrap(drv~cyl,nrow=2)
```

```
GG.fac
```

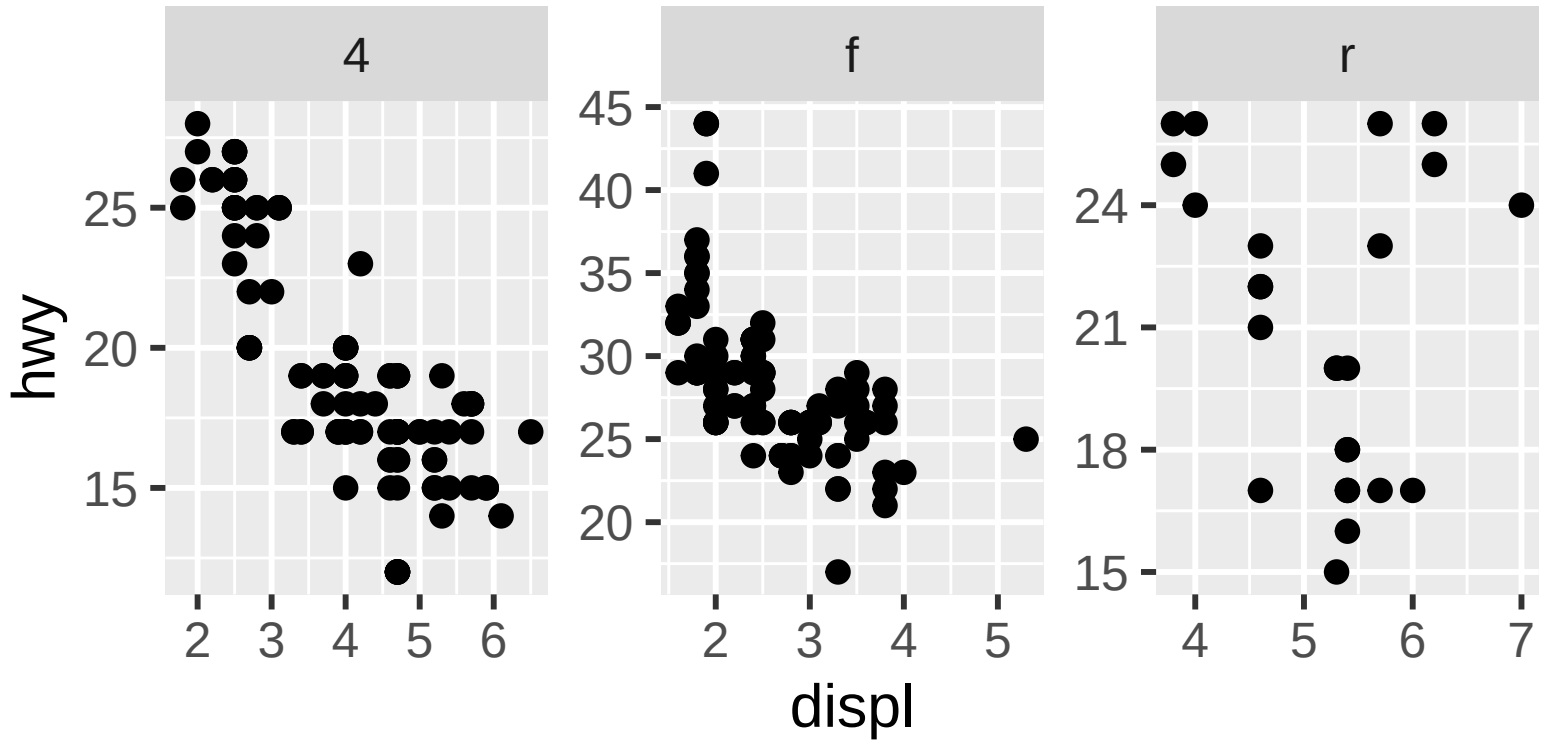


Paneles

Para cambiar las escalas de los paneles, se agrega scales="free"

```
GG.fac=GG.Scatter+
  facet_wrap(drv~.,scales = "free")

GG.fac
```



Escalas

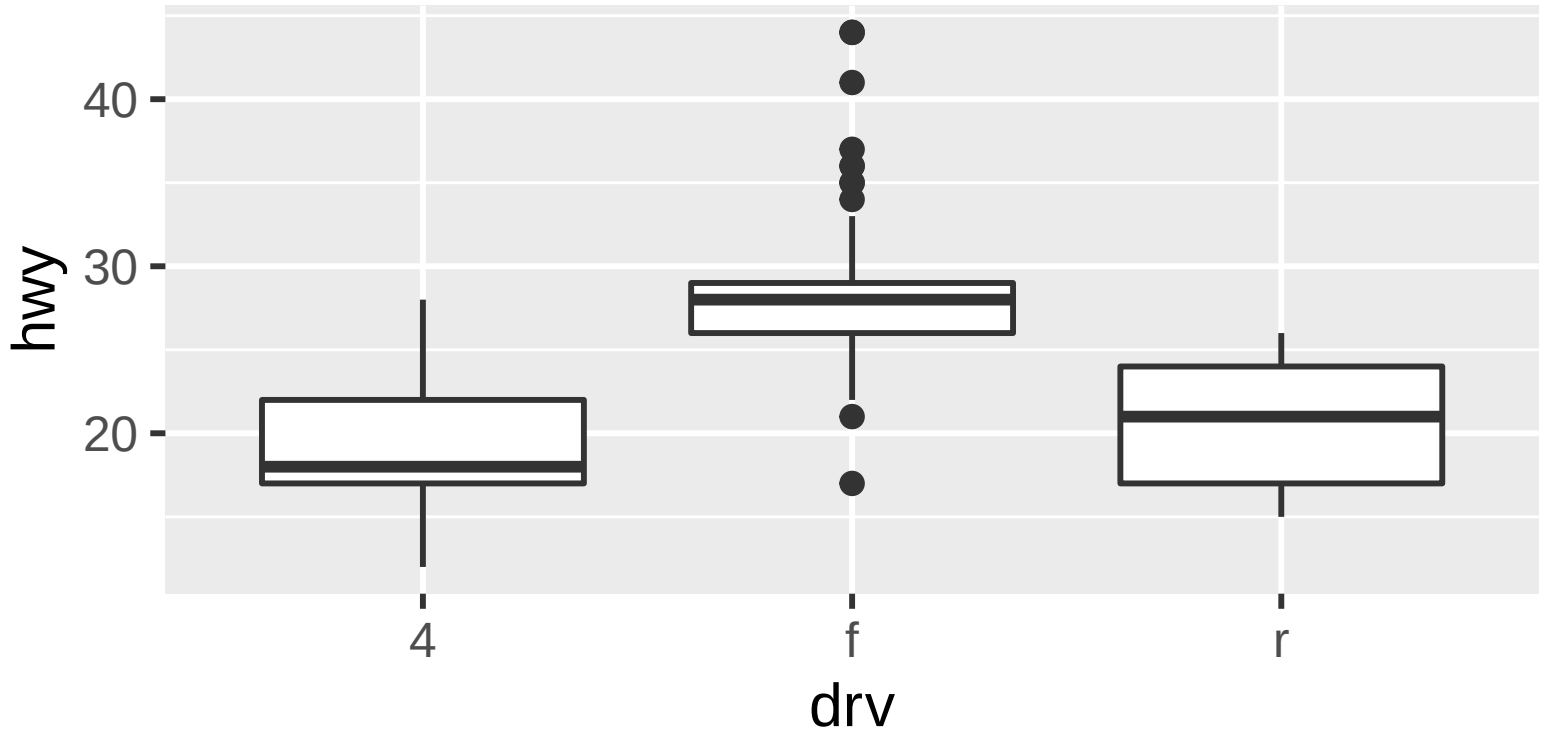
Generalidades

- Las escalas se usan para intervenir el gráfico en todas las expresiones de las variables estéticas.
- Para eso, siempre se hace la distinción de si la variable es considerada continua o discreta.

Ejes

Por ejemplo, volvamos a ver un diagrama de boxplots paralelos:

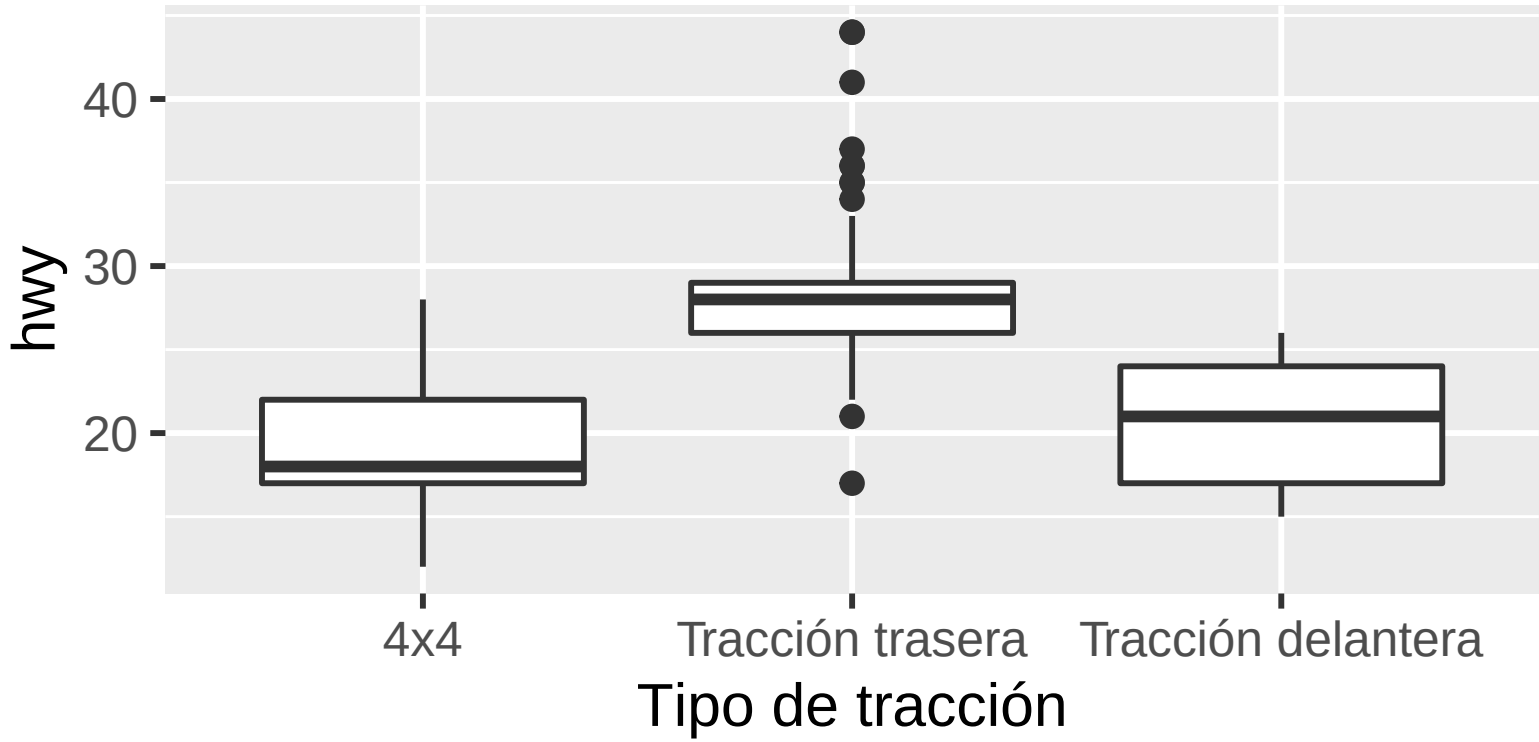
```
GG.ParBox=ggplot(mpg,aes(x=drv,y=hwy))+
  geom_boxplot()
GG.ParBox
```



Ejes

En este caso, el eje x es discreto y el eje y es continuo. Por lo tanto, se pueden cambiar características de las variables especificando dichas categorías:

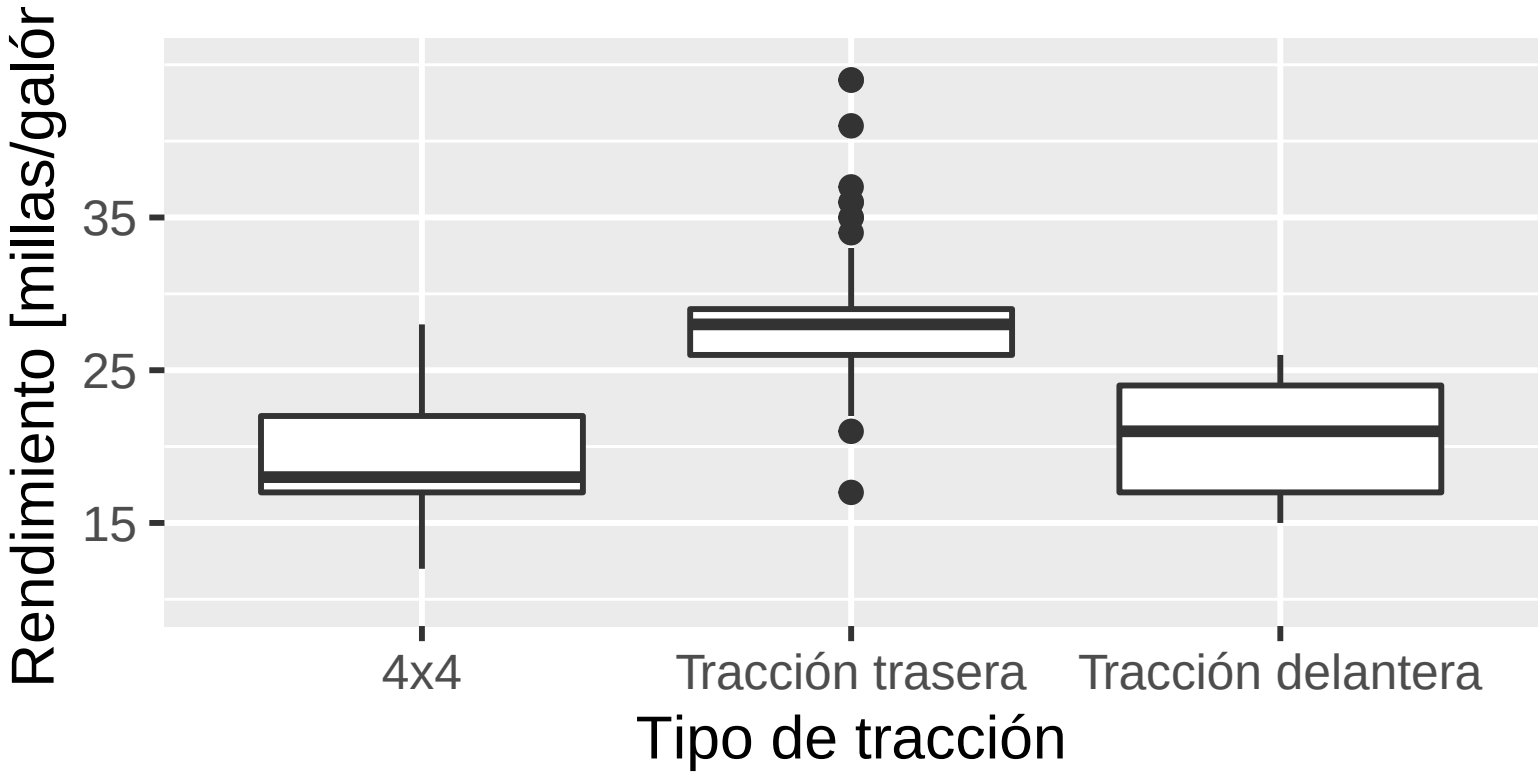
```
GG.ParBox=GG.ParBox+
  scale_x_discrete(name= "Tipo de tracción",
                   labels=c("4x4","Tracción trasera","Tracción delantera"))
GG.ParBox
```



Ejes

En este caso, el eje x es discreto y el eje y es continuo. Por lo tanto, se pueden cambiar características de las variables especificando dichas categorías:

```
GG.ParBox=GG.ParBox+
  scale_y_continuous(name= "Rendimiento [millas/galón]",
                     breaks=c(15,25,35),limits=c(10,45))
GG.ParBox
```



Ejes

Si no se puede coercionar el tipo de datos a una continua, el ggplot tira un error:

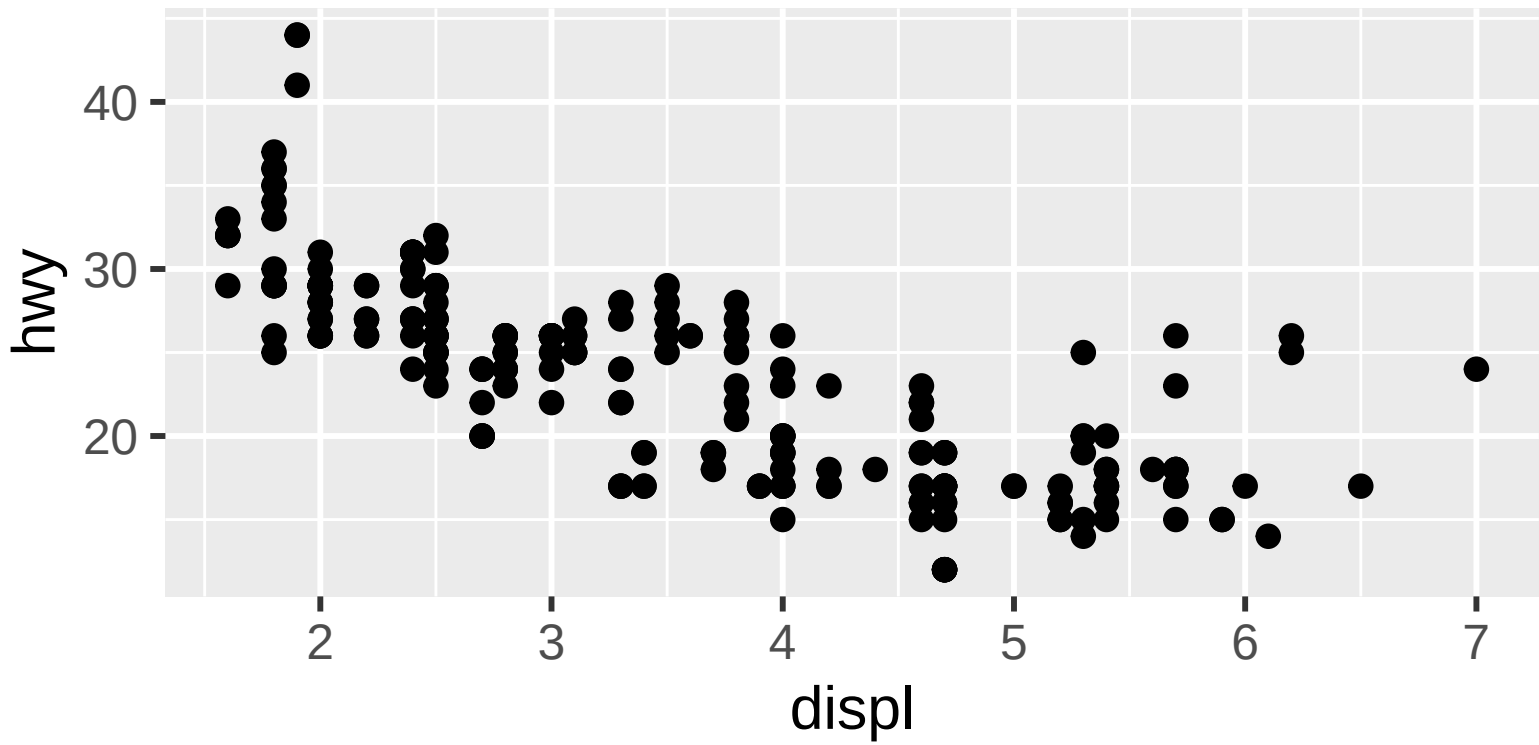
```
GG.ParBox+
  scale_x_continuous(name= "Rendimiento [millas/galón]",
                     breaks=c(15,25,35))
```

```
## Error: Discrete value supplied to continuous scale
```

Ejes

Volvamos a ver cómo queda el scatter plot de displ vs. hwy (Ambas continuas):

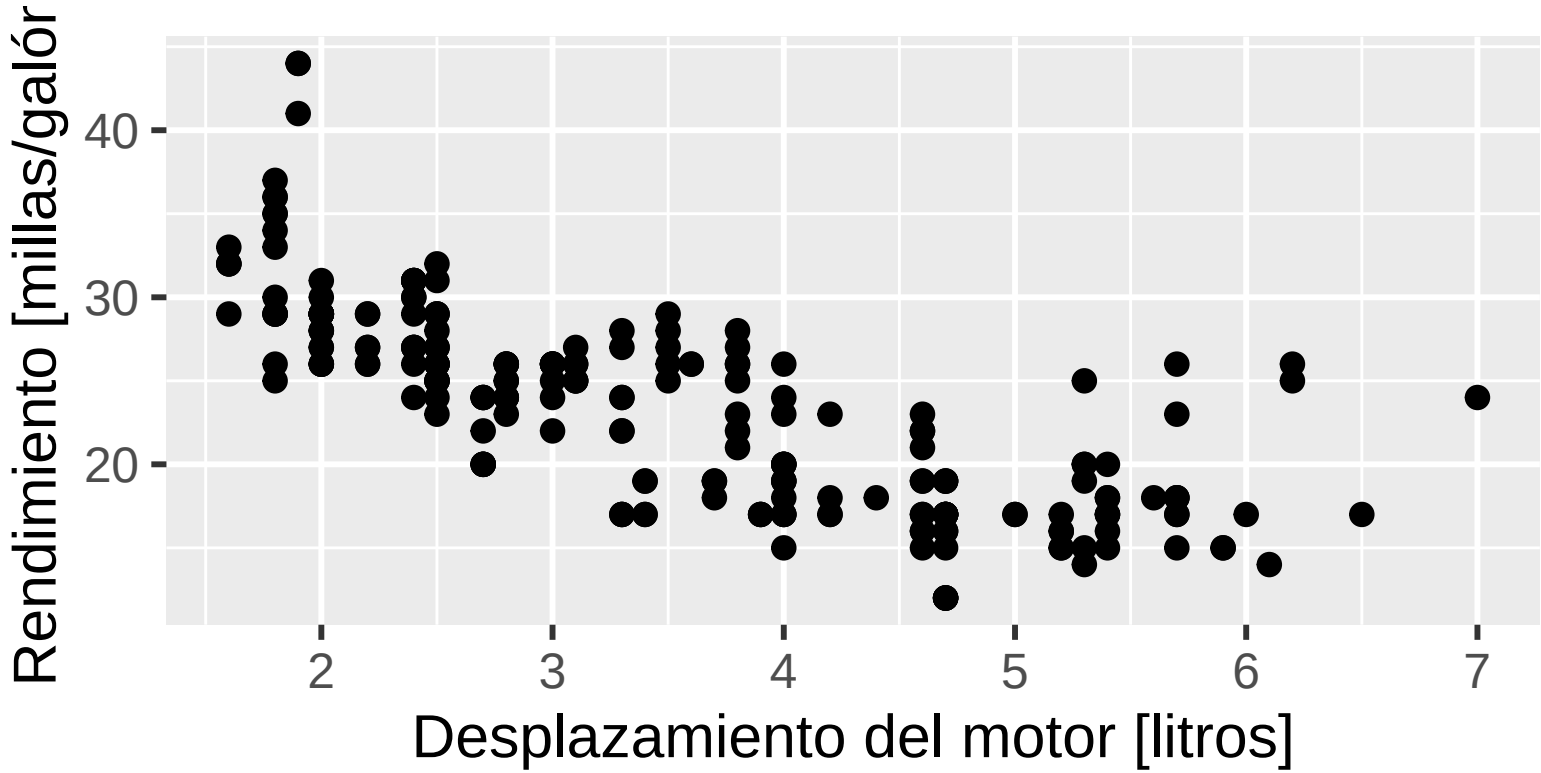
```
GG.Scatter
```



Ejes

Primero, cambiar los nombres de los ejes es más fácil con xlab e ylab:

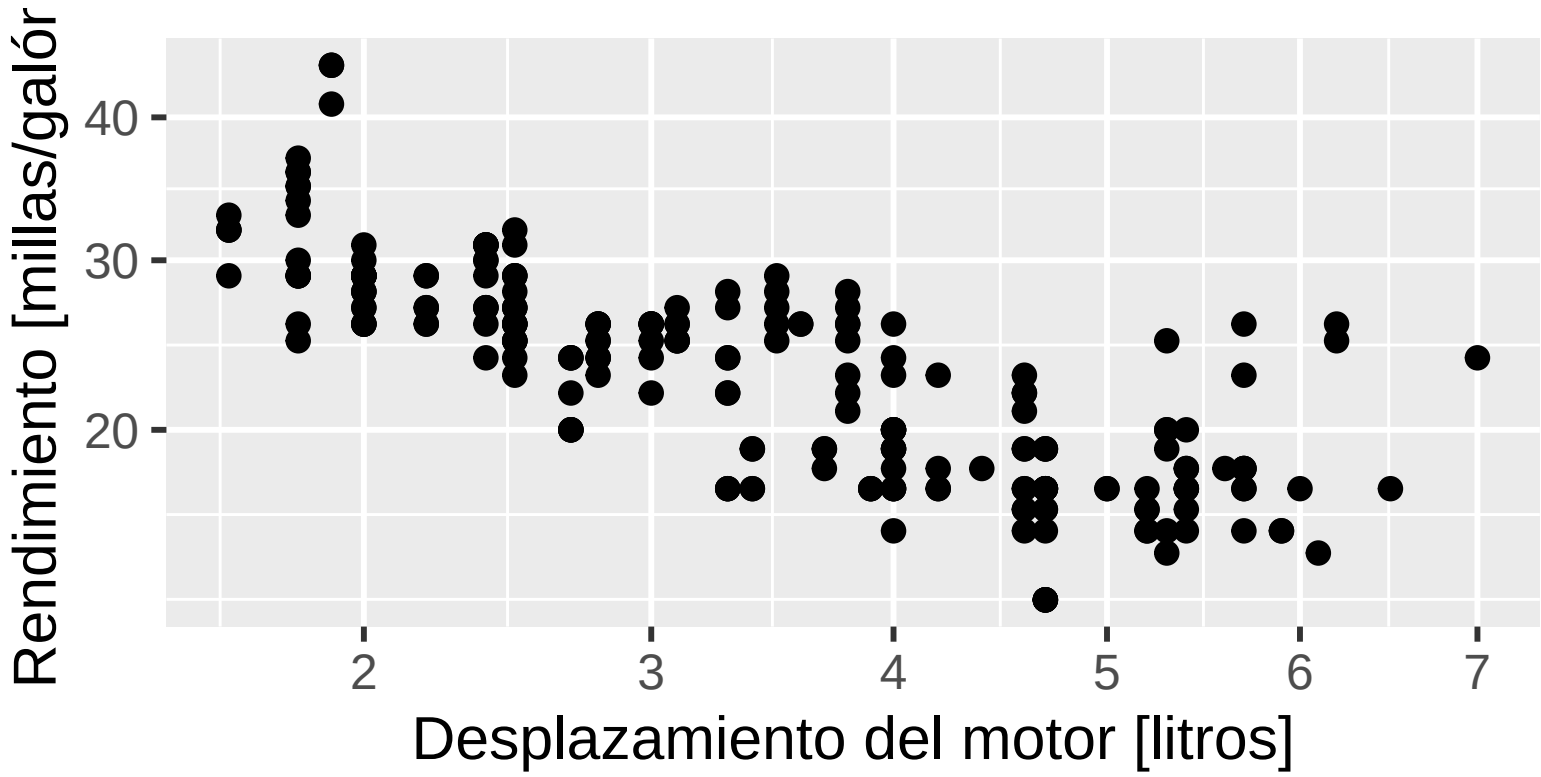
```
GG.Scatter=GG.Scatter+
  xlab("Desplazamiento del motor [litros]")+
  ylab("Rendimiento [millas/galón]")
GG.Scatter
```



Ejes

Se pueden aplicar transformaciones de las variables de forma que el gráfico resulte más alineado, en este caso, se puede tomar la raíz cuadrada de cada eje:

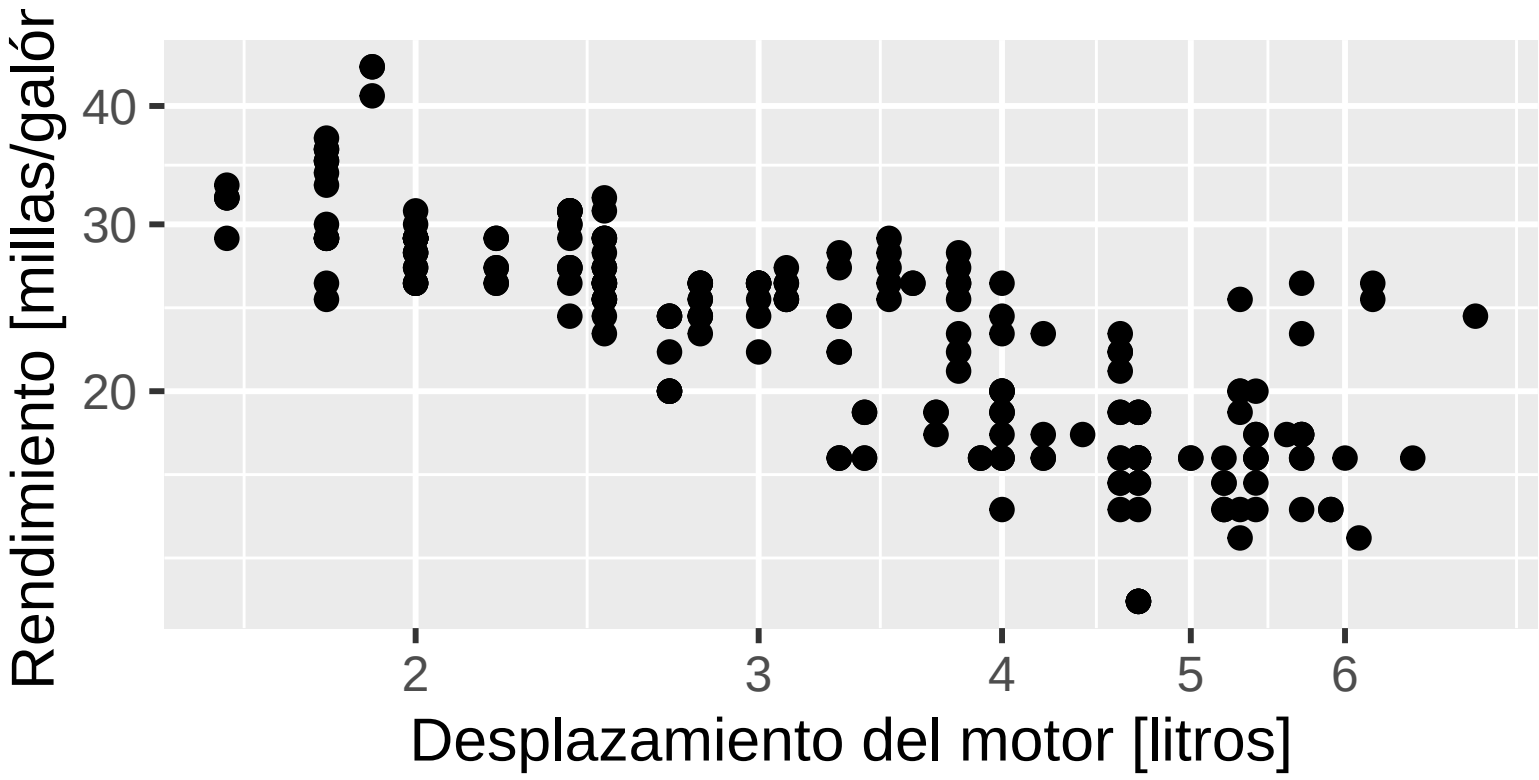
```
GG.Scatter+
  scale_x_continuous(trans = "sqrt")+
  scale_y_continuous(trans = "sqrt")
```



Ejes

Algunas transformaciones ya tienen su comando de atajo:

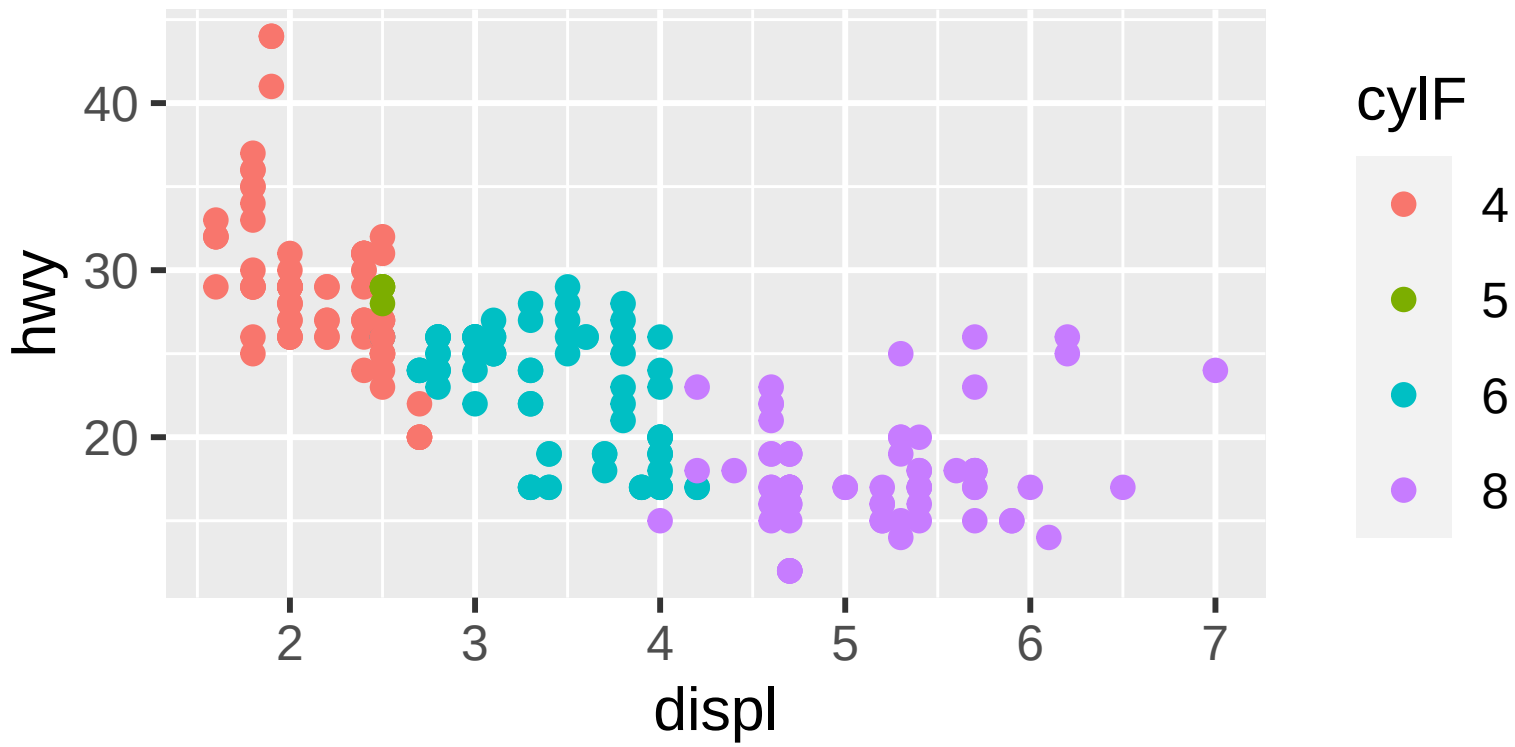
```
GG.Scatter+
  scale_x_log10(breaks=2:6)+
  scale_y_log10(breaks=10*(2:4))
```



Colores

Podemos volvemos al gráfico inicial y darle color por la variable cylF:

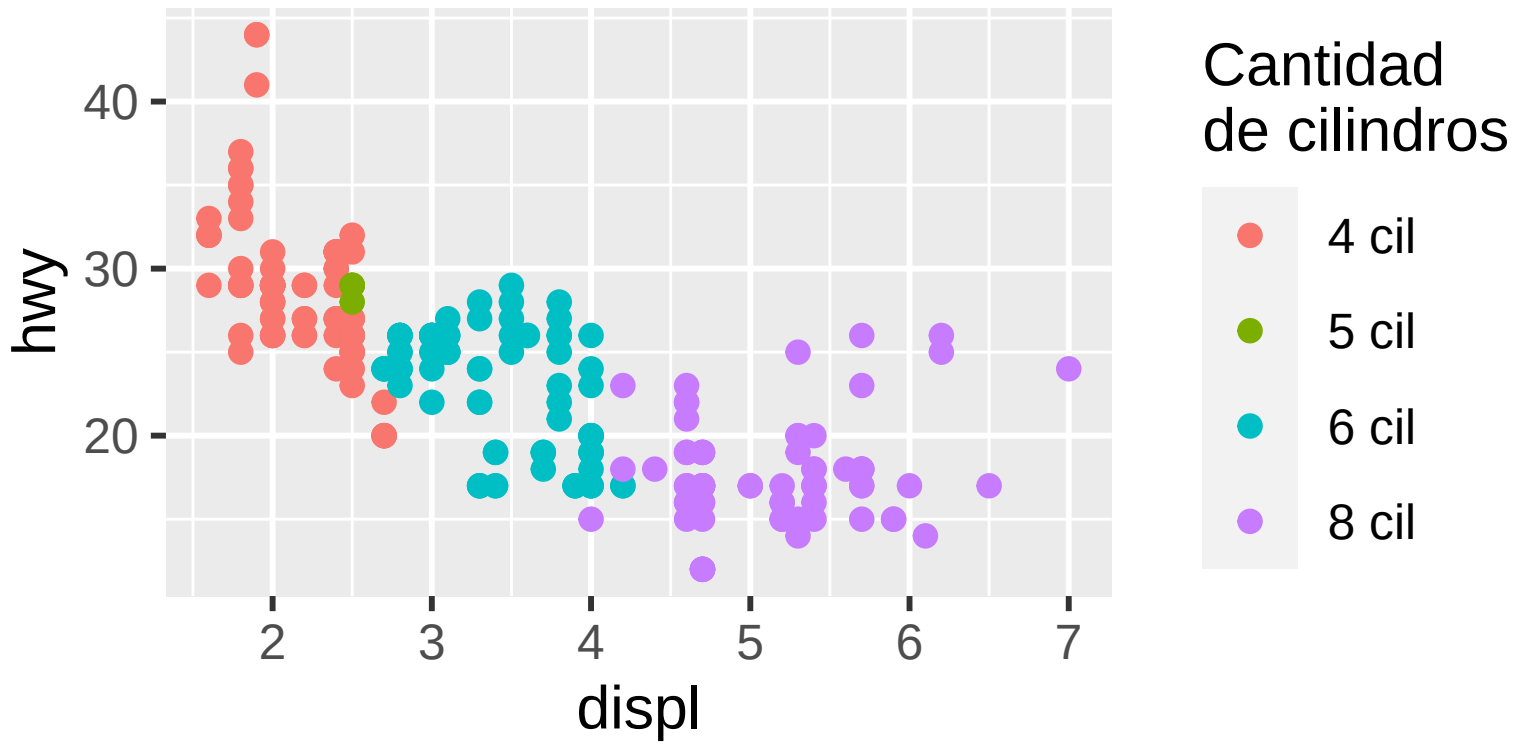
```
GG.Plano+
  geom_point(aes(color=cylF))
```



Colores

Para cambiar los valores de la variable color, hay que considerarla discreta:

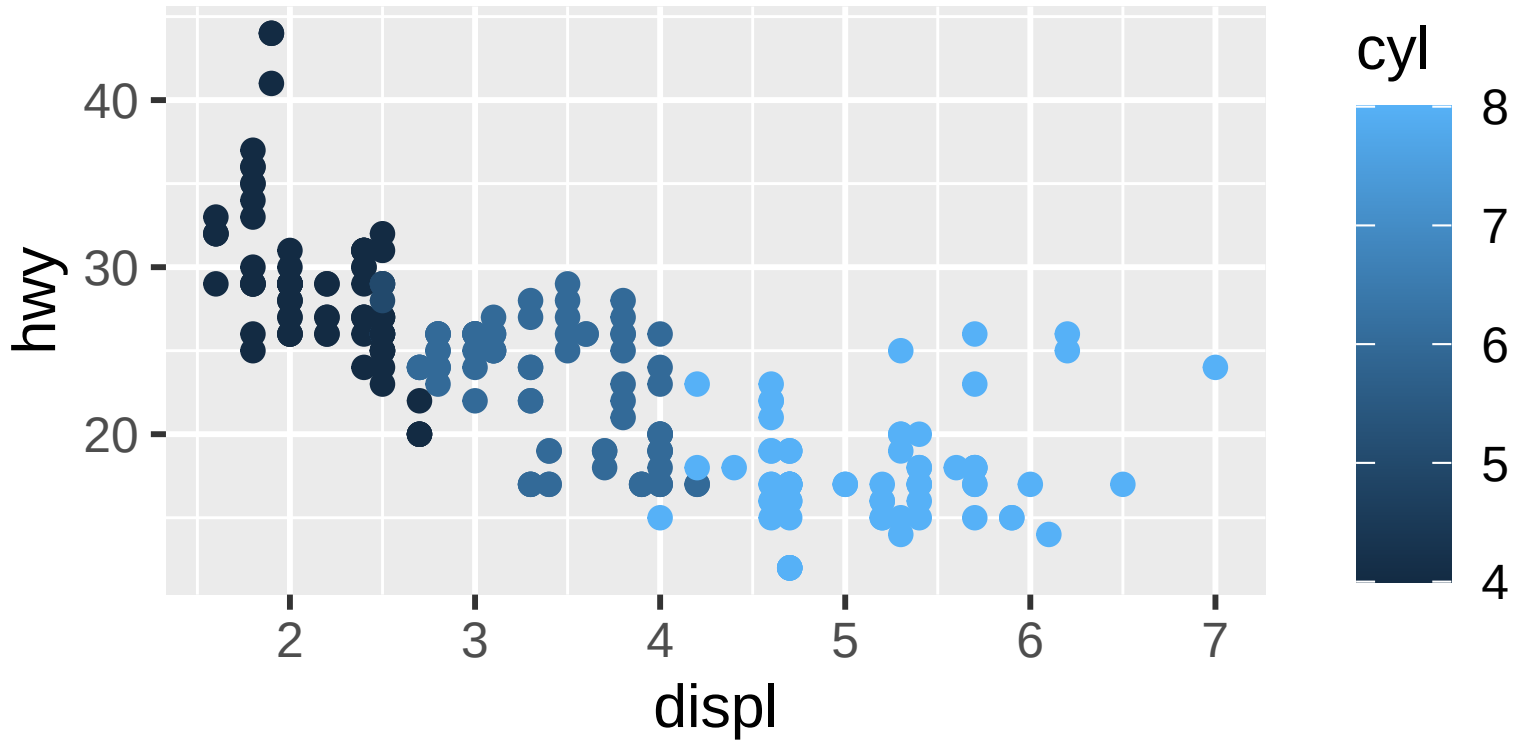
```
GG.Plano+
  geom_point(aes(color=cylF))+
  scale_color_discrete(name="Cantidad\nde cilindros",
                       labels=c("4 cil","5 cil","6 cil","8 cil"))
```



Colores

Si usamos la versión continua de cyl:

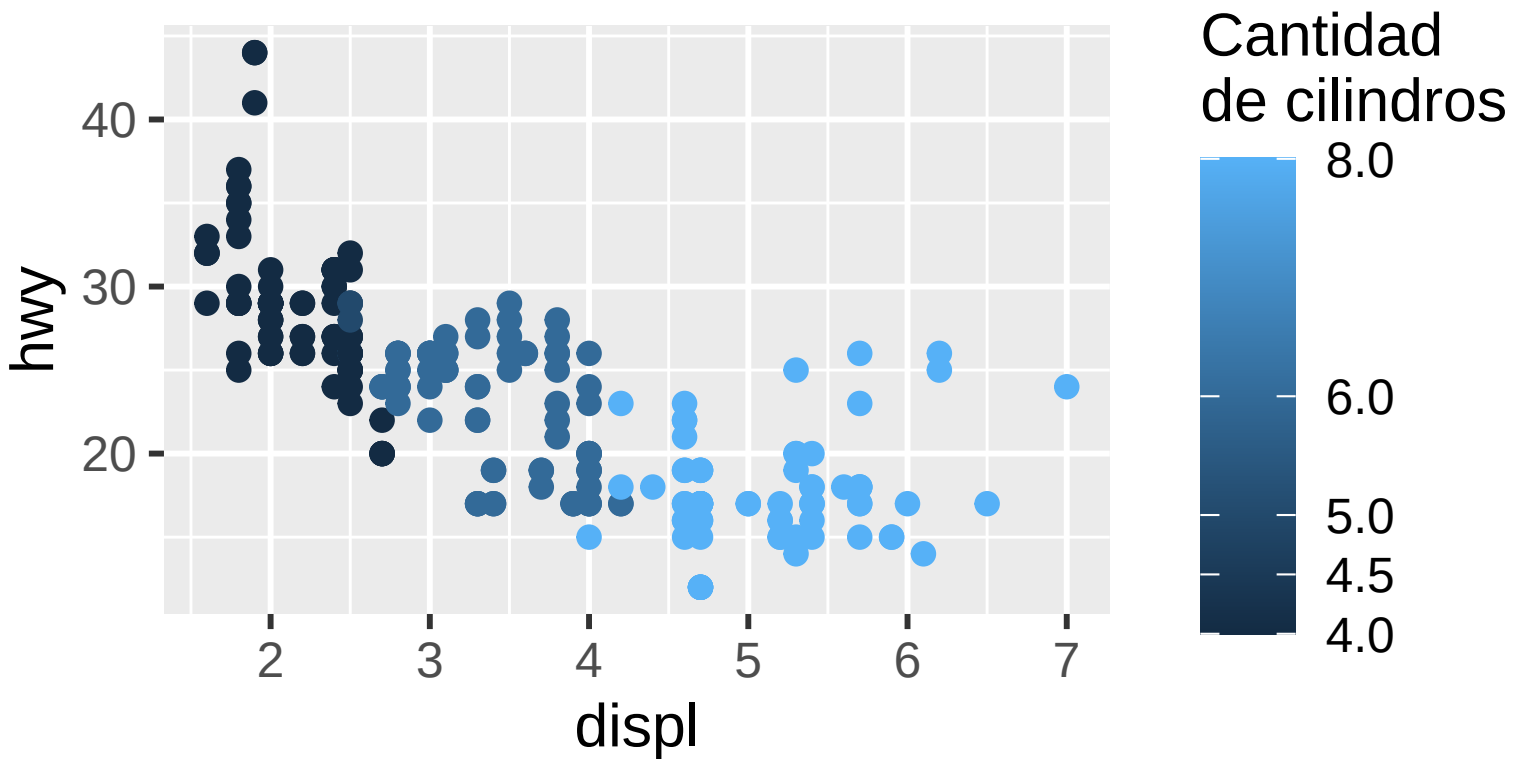
```
GG.Plano+
  geom_point(aes(color=cyl))
```



Colores

Ahora hay que considerarla continua:

```
GG.Plano+
  geom_point(aes(color=cyl))+
  scale_color_continuous(name="Cantidad\nde cilindros",
                         breaks=c(4,4.5,5,6,8))
```

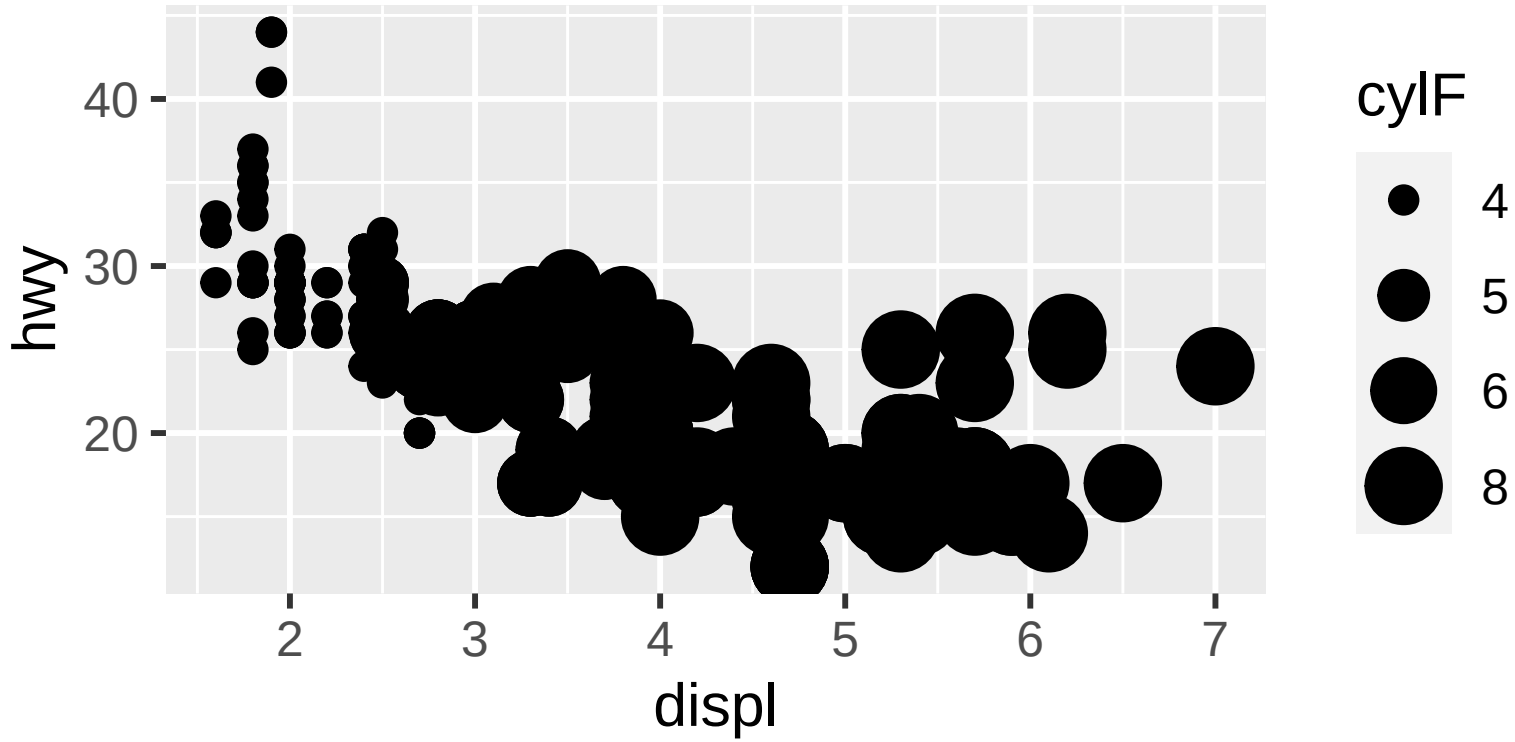


Tamaño

Podemos volvemos al gráfico inicial y darle color por la variable cylF:

```
GG.Plano+
  geom_point(aes(size=cylF))
```

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

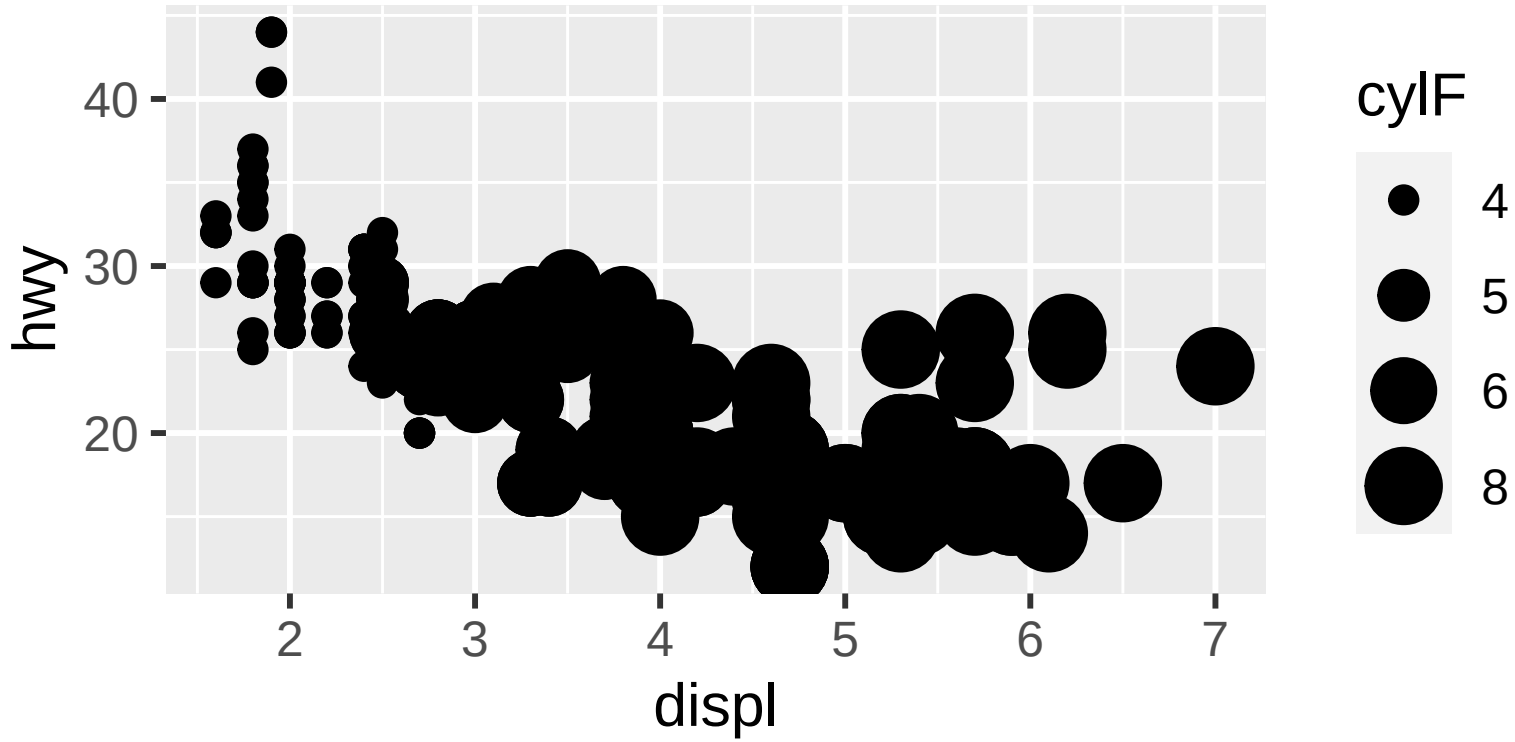


Tamaño

Podemos volvemos al gráfico inicial y darle color por la variable cylF:

```
GG.Plano+
  geom_point(aes(size=cylF))
```

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



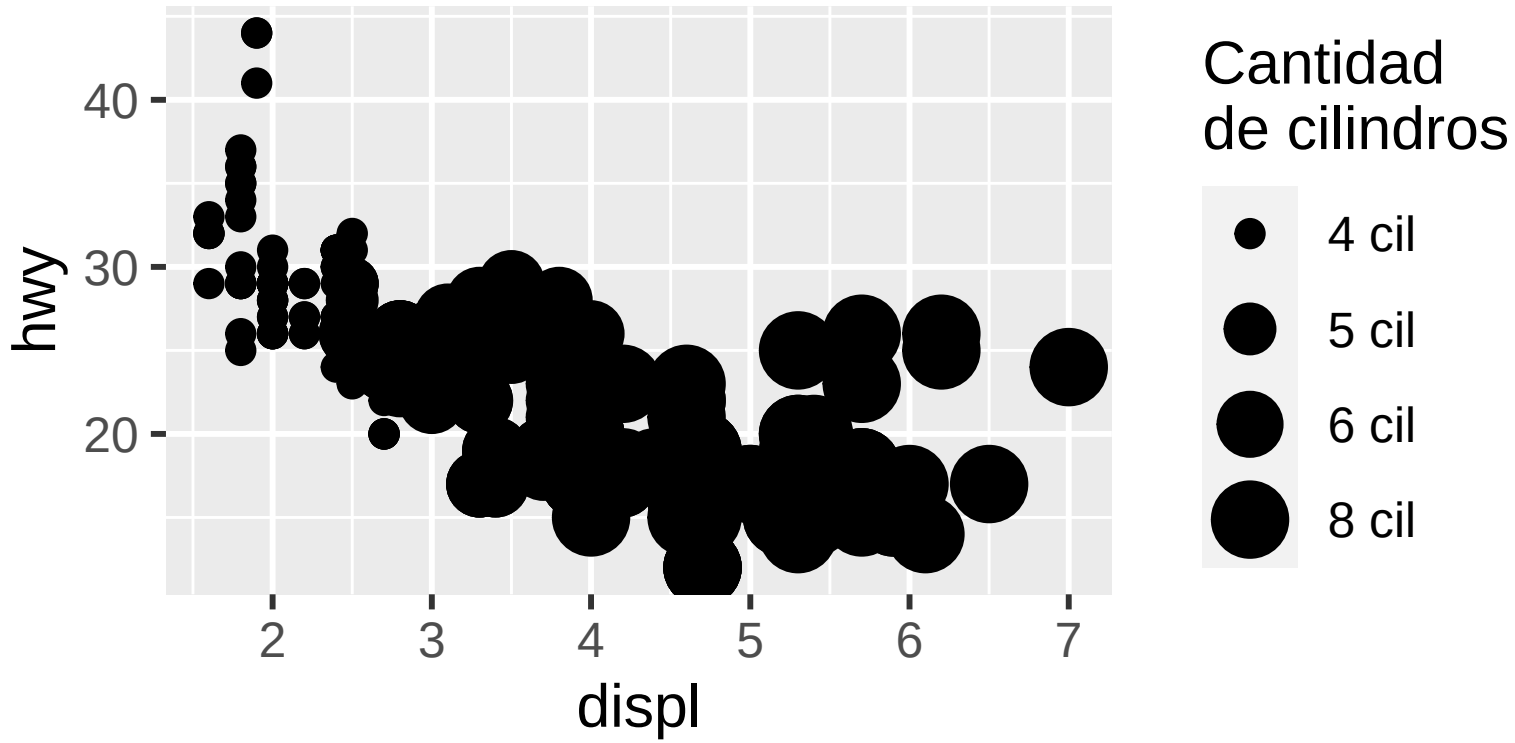
Notar que ggplot desaconseja usar un factor para el tamaño.

Tamaño

Para cambiar los valores de la variable color, hay que considerarla discreta:

```
GG.Plano+
  geom_point(aes(size=cylF))+
  scale_size_discrete(name="Cantidad\nde cilindros",
                      labels=c("4 cil","5 cil","6 cil","8 cil"))
```

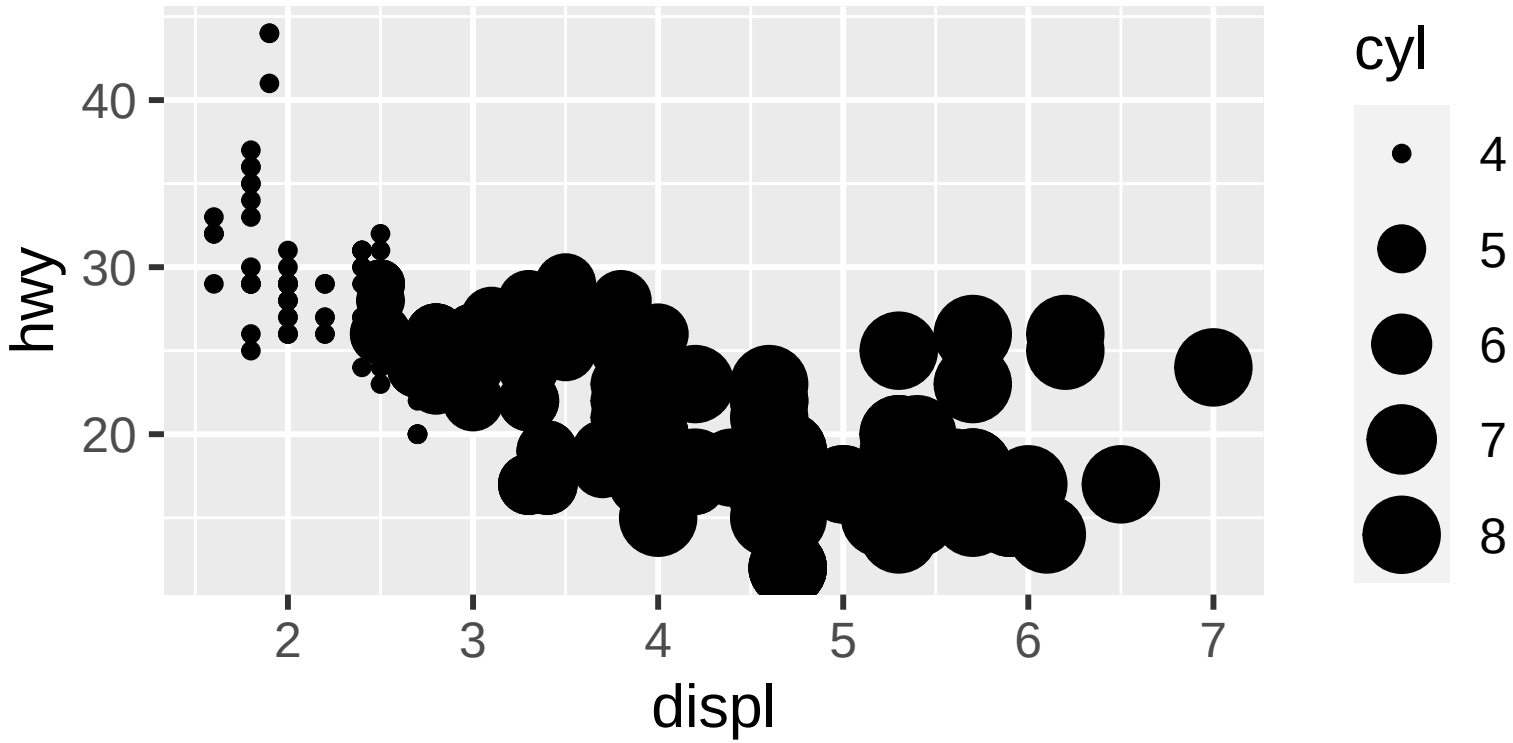
```
## Warning: Using size for a discrete variable is not advised.
```



Tamaño

Si usamos la versión continua de cyl:

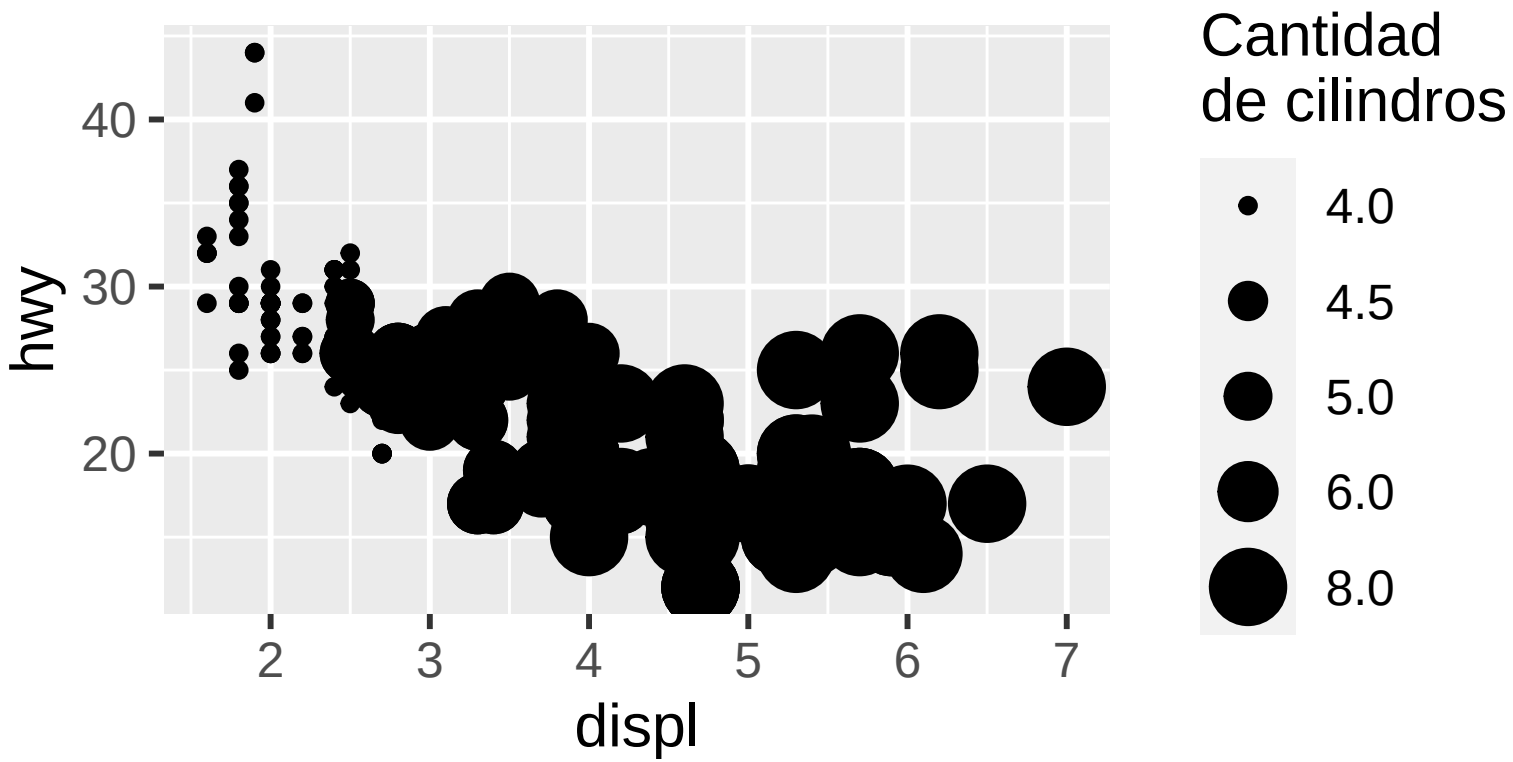
```
GG.Plano+
  geom_point(aes(size=cyl))
```



Tamaño

Ahora hay que considerarla continua:

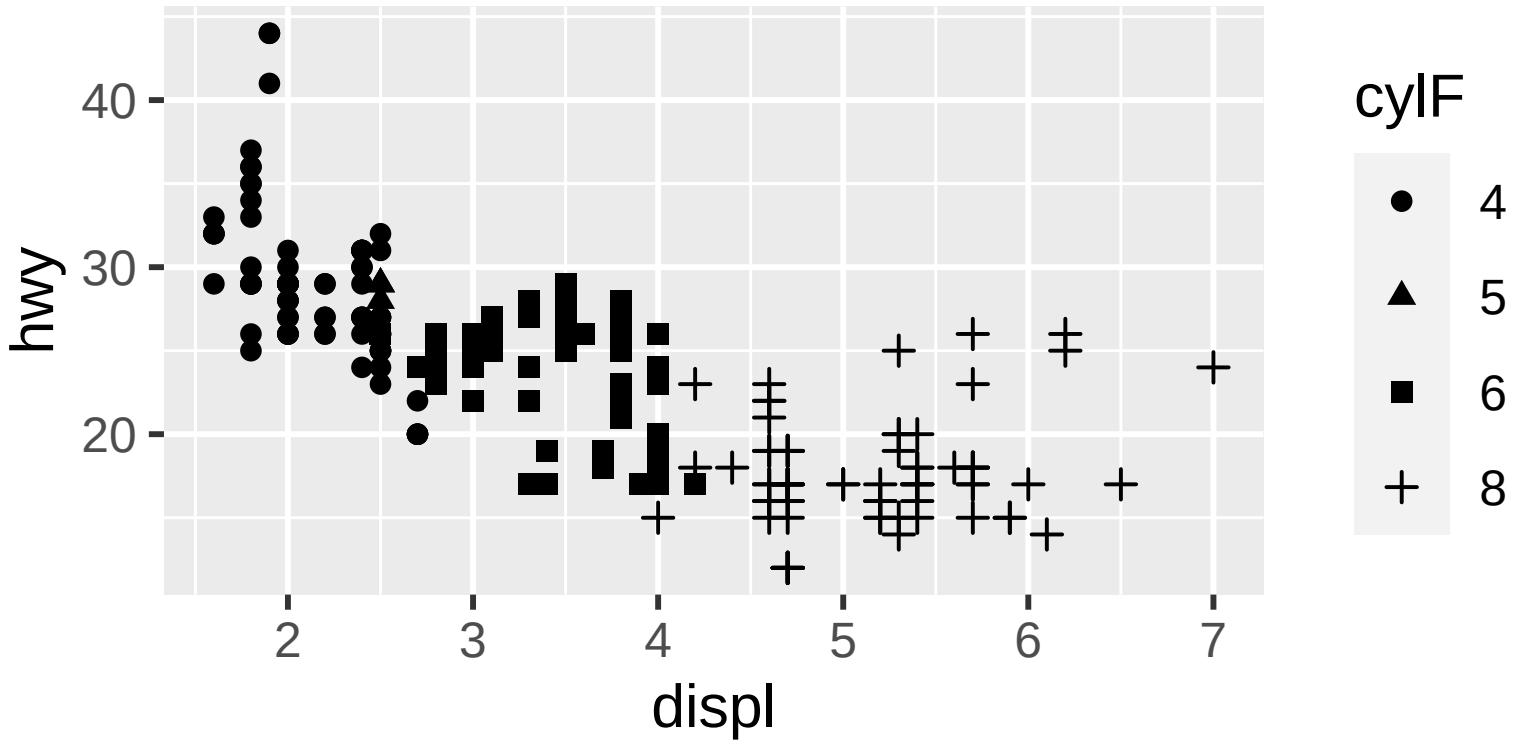
```
GG.Plano+
  geom_point(aes(size=cyl))+
  scale_size_continuous(name="Cantidad\nde cilindros",
                        breaks=c(4,4.5,5,6,8))
```



Forma

Le damos forma por la variable cylF:

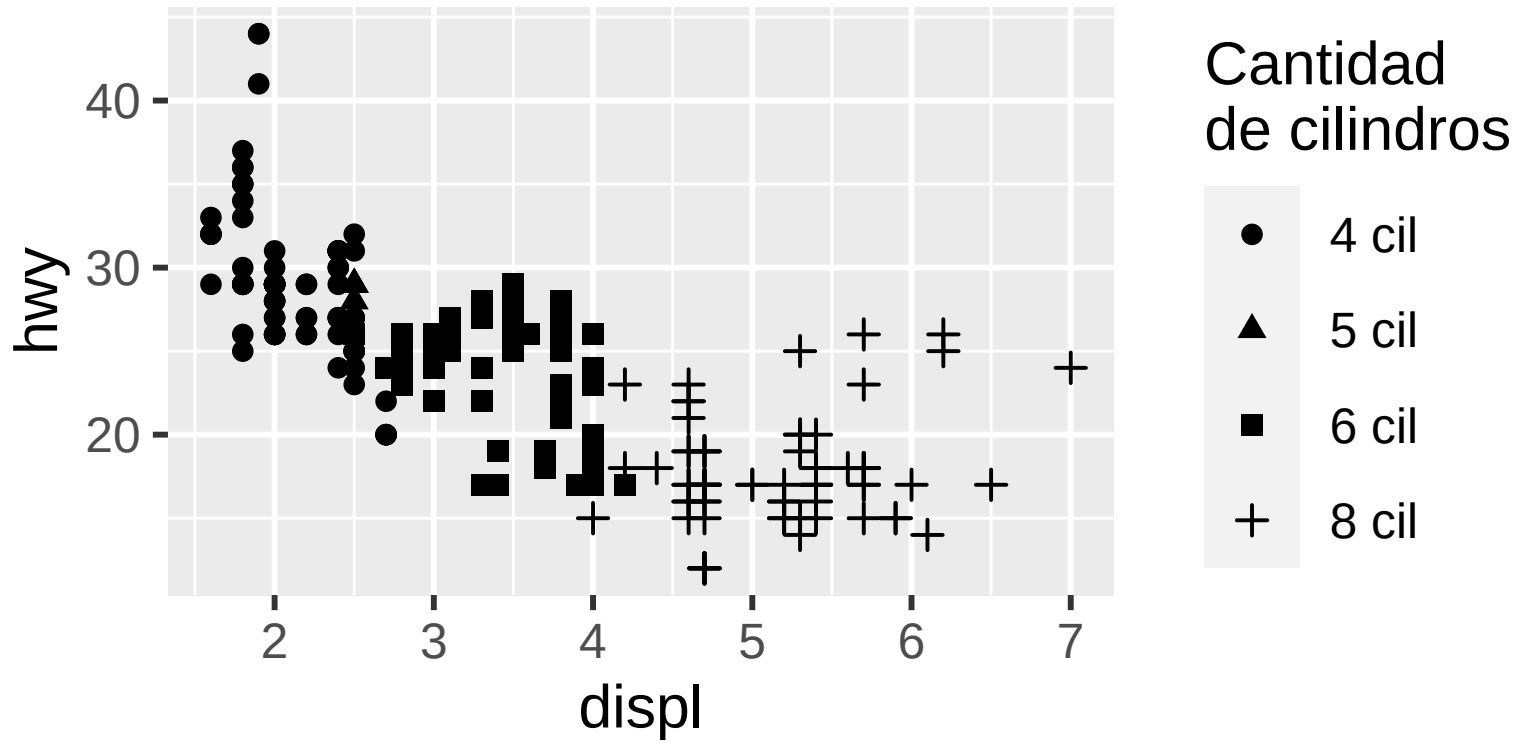
```
GG.Plano+
  geom_point(aes(shape=cylF))
```



Forma

Para cambiar los valores de la variable forma, hay que considerarla discreta:

```
GG.Plano+
  geom_point(aes(shape=cylF))+
  scale_shape_discrete(name="Cantidad\nde cilindros",
                       labels=c("4 cil","5 cil","6 cil","8 cil"))
```



Forma

En este caso, ggplot no acepta variables continuas para darle forma (tiene sentido):

```
GG.Plano+
  geom_point(aes(shape=cyl))
```

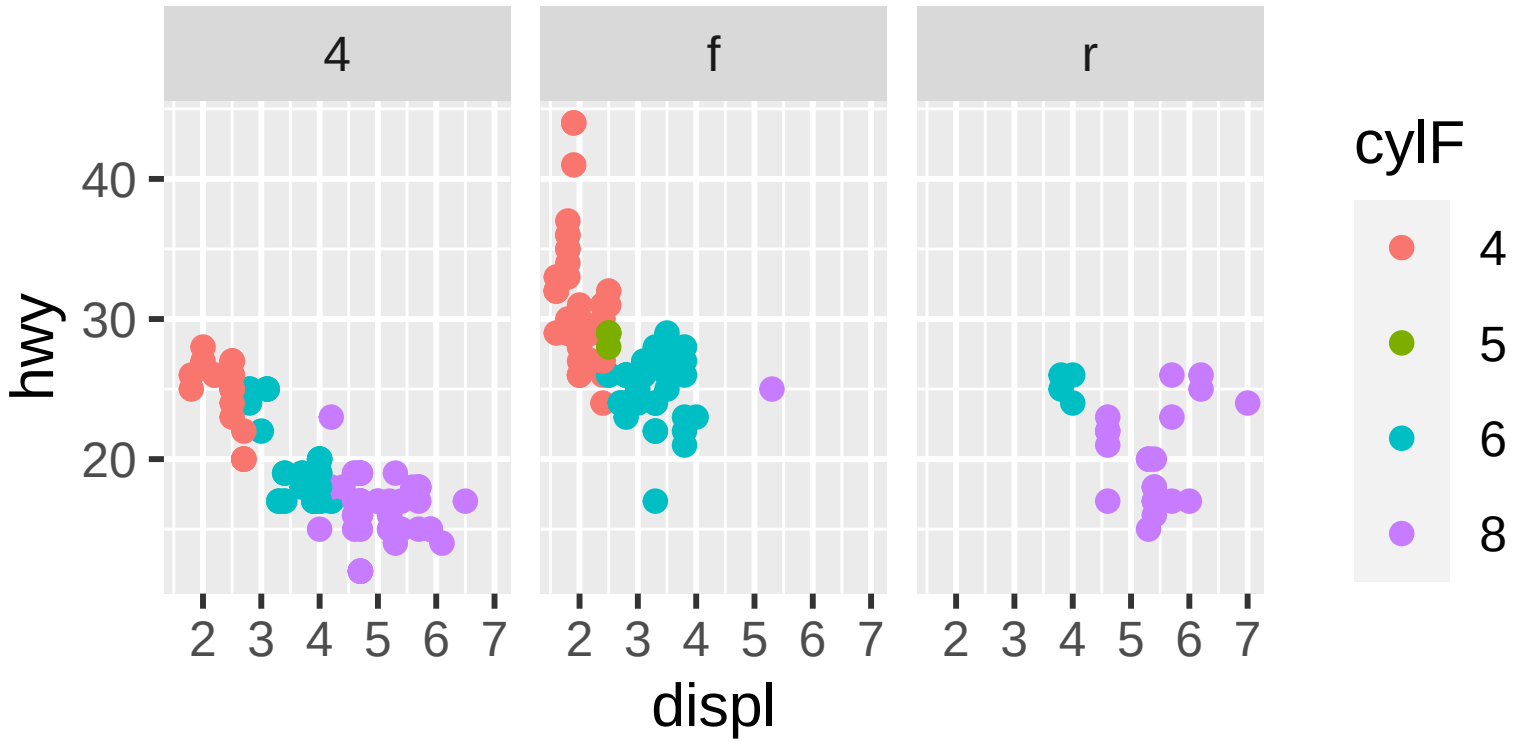
```
## Error: A continuous variable can not be mapped to shape
```

Tema

Generalidades

El tema se encarga de cuestiones de formato. Consideremos:

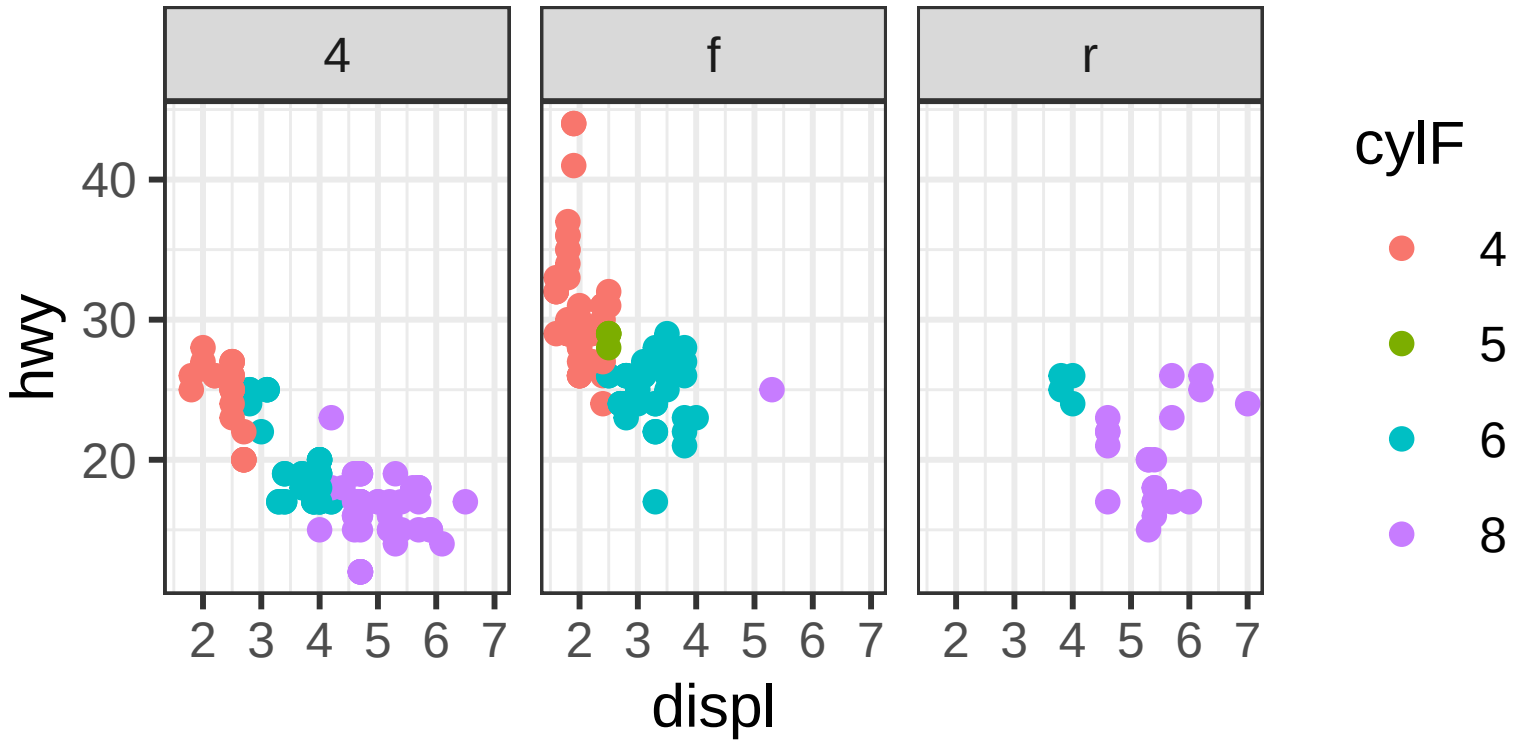
```
GG.Scatter=ggplot(mpg,aes(x=displ,y=hwy,color=cylF))+
  geom_point()+
  facet_wrap(.~drv)
GG.Scatter
```



Generalidades

Algunos temas tienen formatos globales. Por ejemplo, theme_bw tiene un formato para todos los componentes del gráfico:

```
GG.Scatter+
  theme_bw()
```



Generalidades

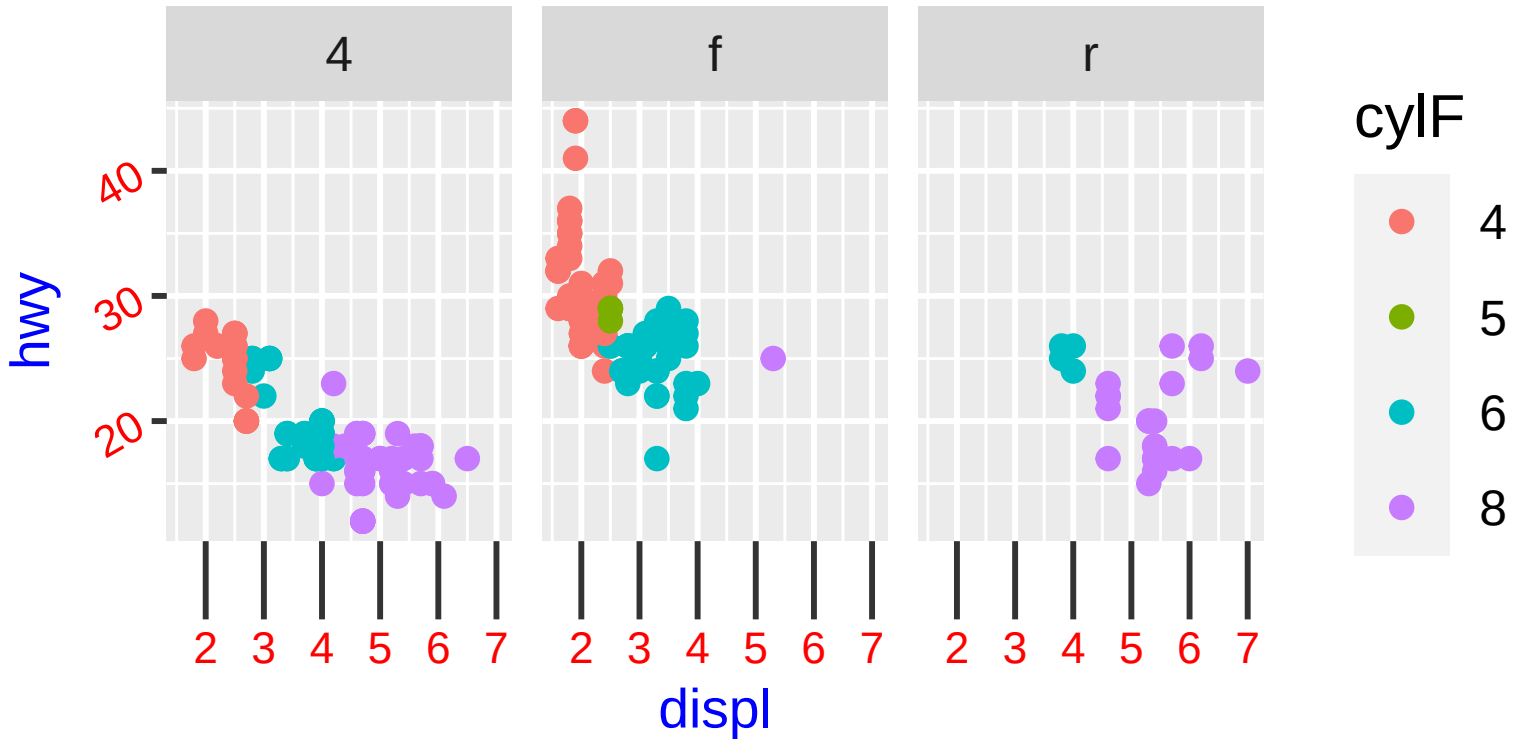
Luego, podríamos dividir el formato en cuestiones de:

- Ejes
- Leyendas
- Paneles

Ejes

Para cambiar cosas de los ejes, agregamos el comando theme() y al ingresar axis nos aparecen las opciones:

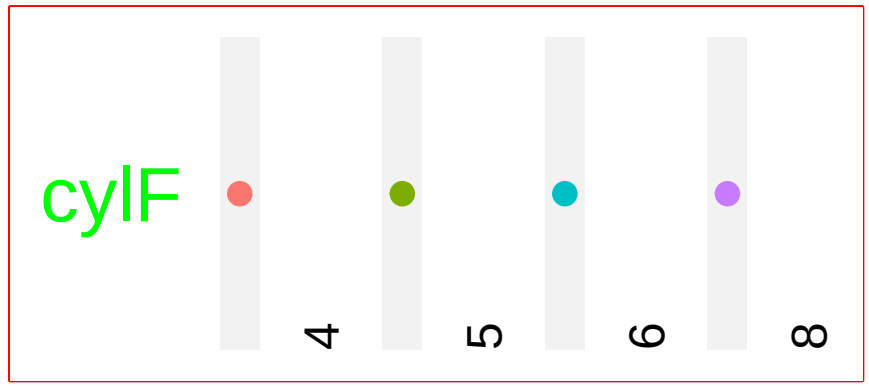
```
GG.Scatter+
  theme(axis.title = element_text(color="blue",size = 10),
        axis.text = element_text(color="red",size=8),
        axis.text.y = element_text(angle=30),
        axis.ticks.length.x = unit(0.5,"cm"))
```

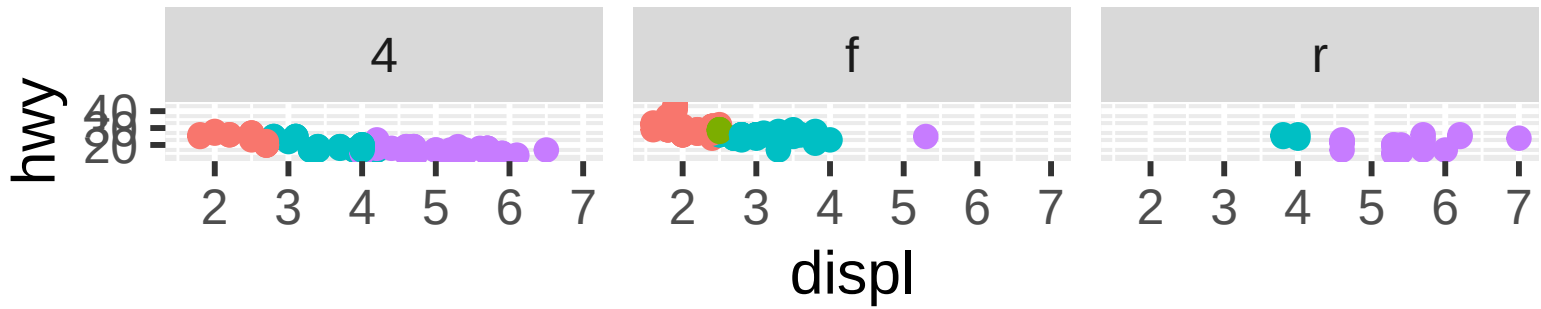


Leyenda

Para cambiar cosas de las leyendas, agregamos el comando theme() y al ingresar legend nos aparecen las opciones:

```
GG.Scatter+
  theme(legend.position = "top",
        legend.title = element_text(color="green",size=14),
       legend.text = element_text(angle=90),
       legend.key.height = unit(2,"cm"),
       legend.key.width = unit(0.25,"cm"),
       legend.box.background = element_rect(color="red"))
```

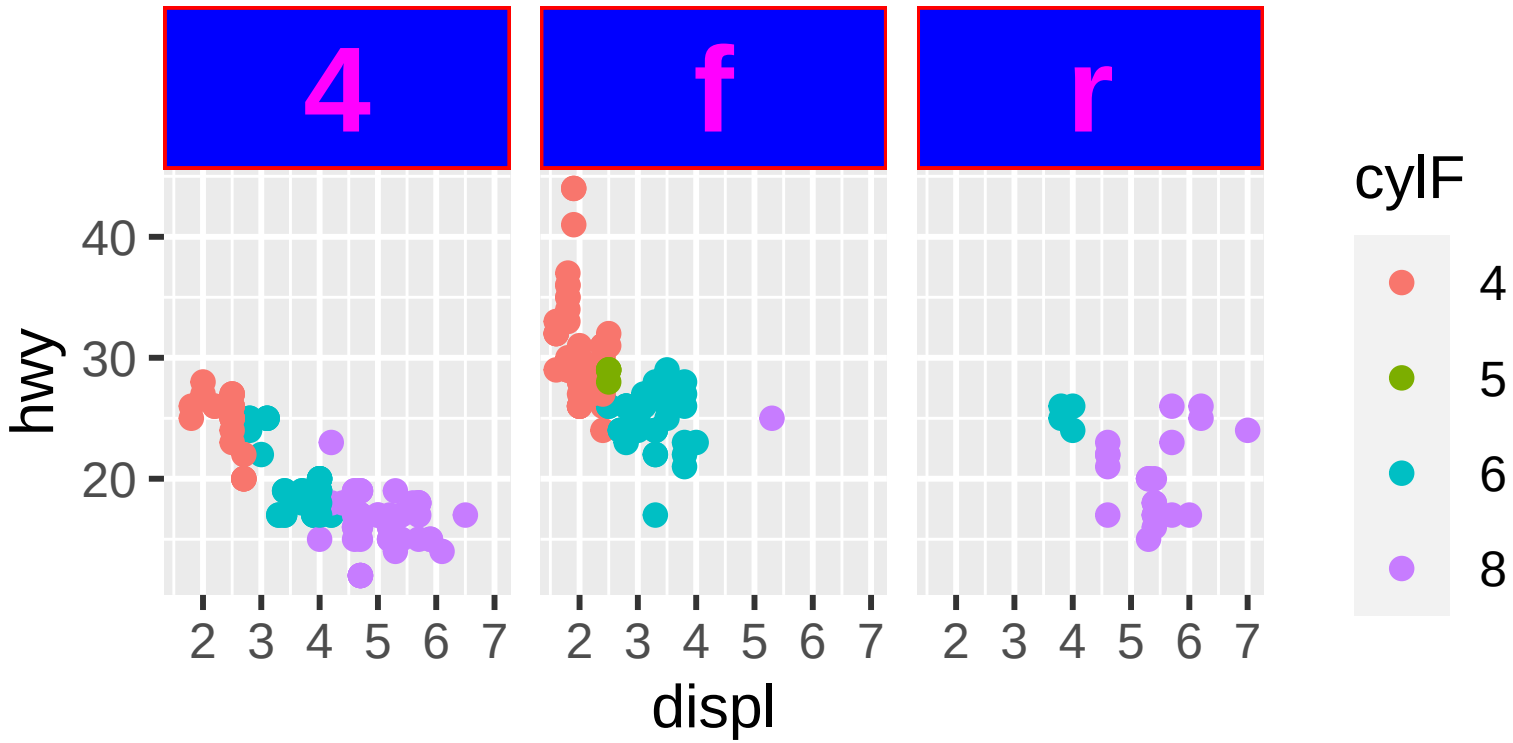




Paneles

Para cambiar cosas de las leyendas, agregamos el comando theme() y al ingresar strip (para el nombre) nos aparecen las opciones:

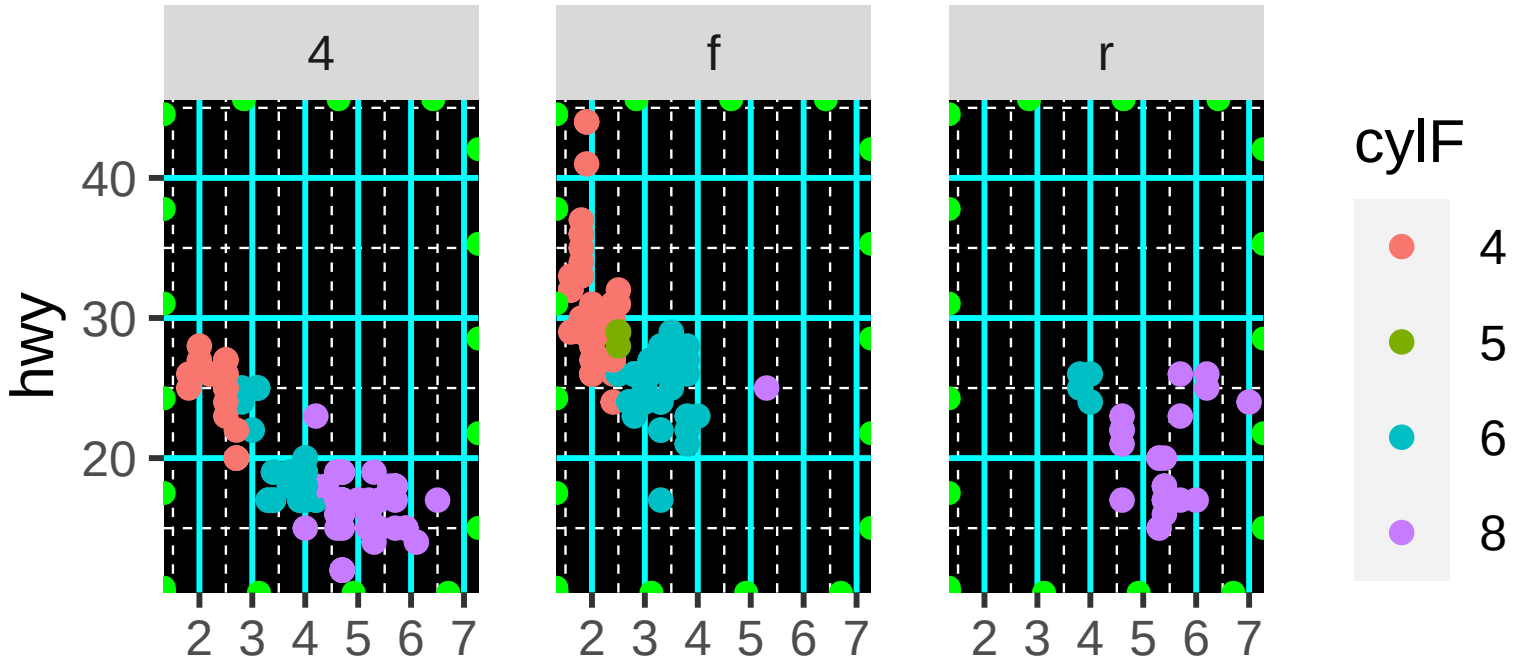
```
GG.Scatter+
  theme(strip.background = element_rect(fill="blue",color="red"),
        strip.text = element_text(size=22,color="magenta",
                                  face="bold"))
```



Paneles

Para cambiar cosas de las leyendas, agregamos el comando theme() y al ingresar panel (para el panel total) nos aparecen las opciones:

```
GG.Scatter+
  theme(panel.spacing = unit(0.5,"cm"),
        panel.border = element_rect(
          fill = NA,color="green",
        linetype = "dotted",size=2),
        panel.grid.major = element_line(size=0.5,color="cyan"),
        panel.grid.minor = element_line(size=0.2,
              linetype = "dashed",color="white"),
        panel.background = element_rect(fill = "black"))
```



Coordenadas

Coordenadas polares

ggplot tiene algunos tipos de coordenadas (más relevantes a la hora de hacer mapas), pero vamos a usar sólo las coordenadas polares. Empezamos con una base inventada:

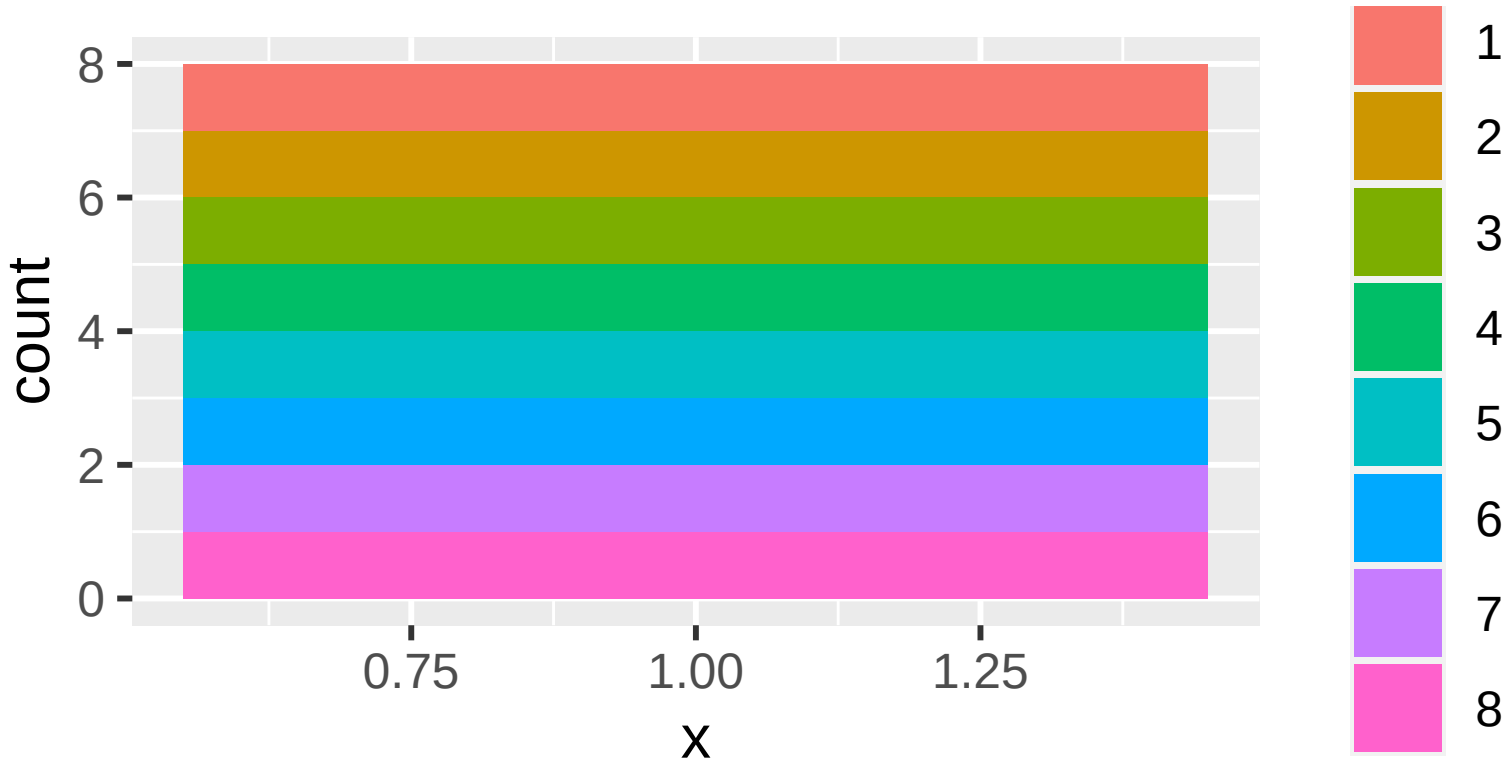
```
Base.Cart=data.frame(x=rep(1,8),y=factor(1:8),c=factor(c(rep(1,6),2,3)))
Base.Cart
```

```
##   x y c
## 1 1 1 1
## 2 1 2 1
## 3 1 3 1
## 4 1 4 1
## 5 1 5 1
## 6 1 6 1
## 7 1 7 2
## 8 1 8 3
```

Coordenadas polares

Vamos a hacer un gráfico de barras:

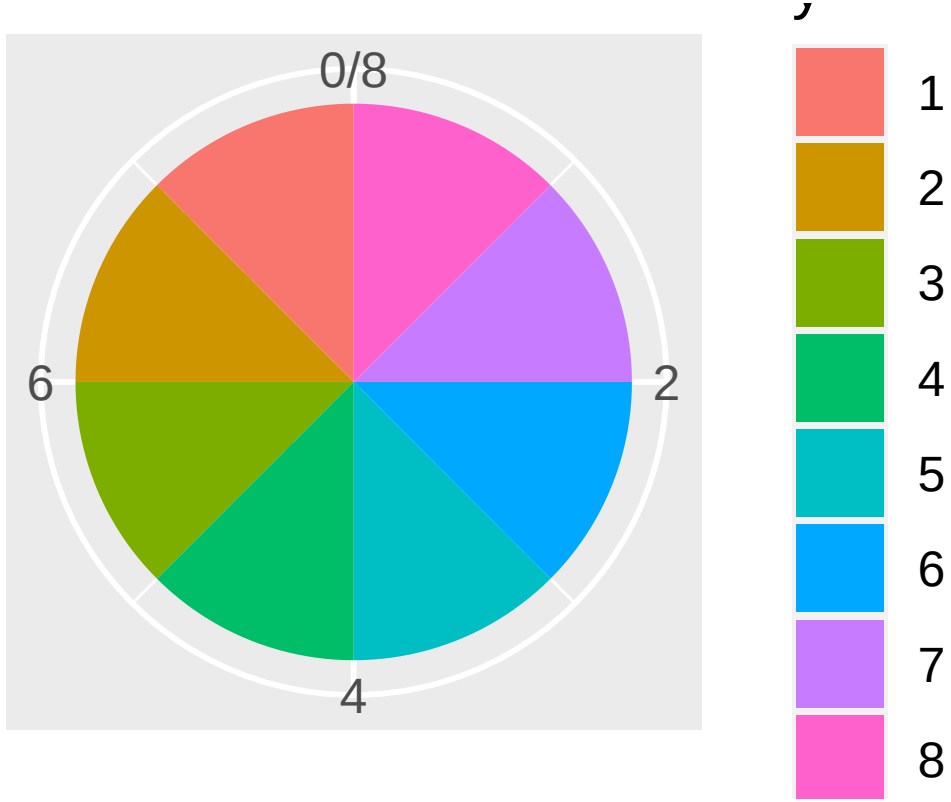
```
Bar.Cart=ggplot(data = Base.Cart,aes(x=x,fill=y))+
  geom_bar()
Bar.Cart
```



Coordenadas polares

Removiendo el nombre de los ejes y pasandolo a coordenadas polares, tenemos el clásico gráfico de torta:

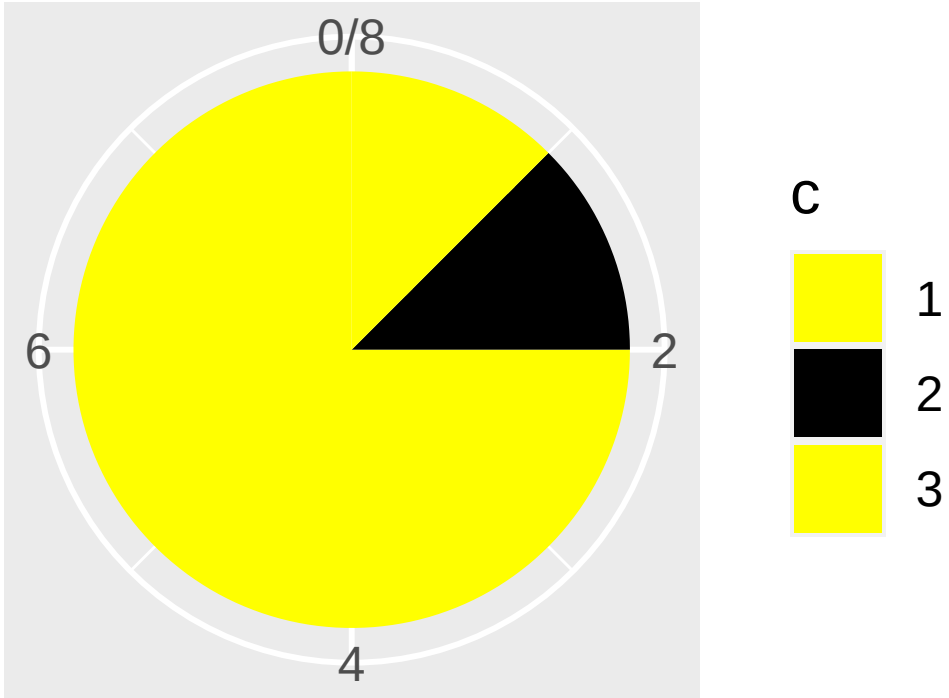
```
Bar.Cart=Bar.Cart+
  scale_x_discrete(NULL,expand = c(0,0))+
  scale_y_continuous(NULL,expand = c(0,0))
Bar.Cart+
  coord_polar(theta="y")
```



Coordenadas polares

Y bueno, también podemos hacer un pacman:

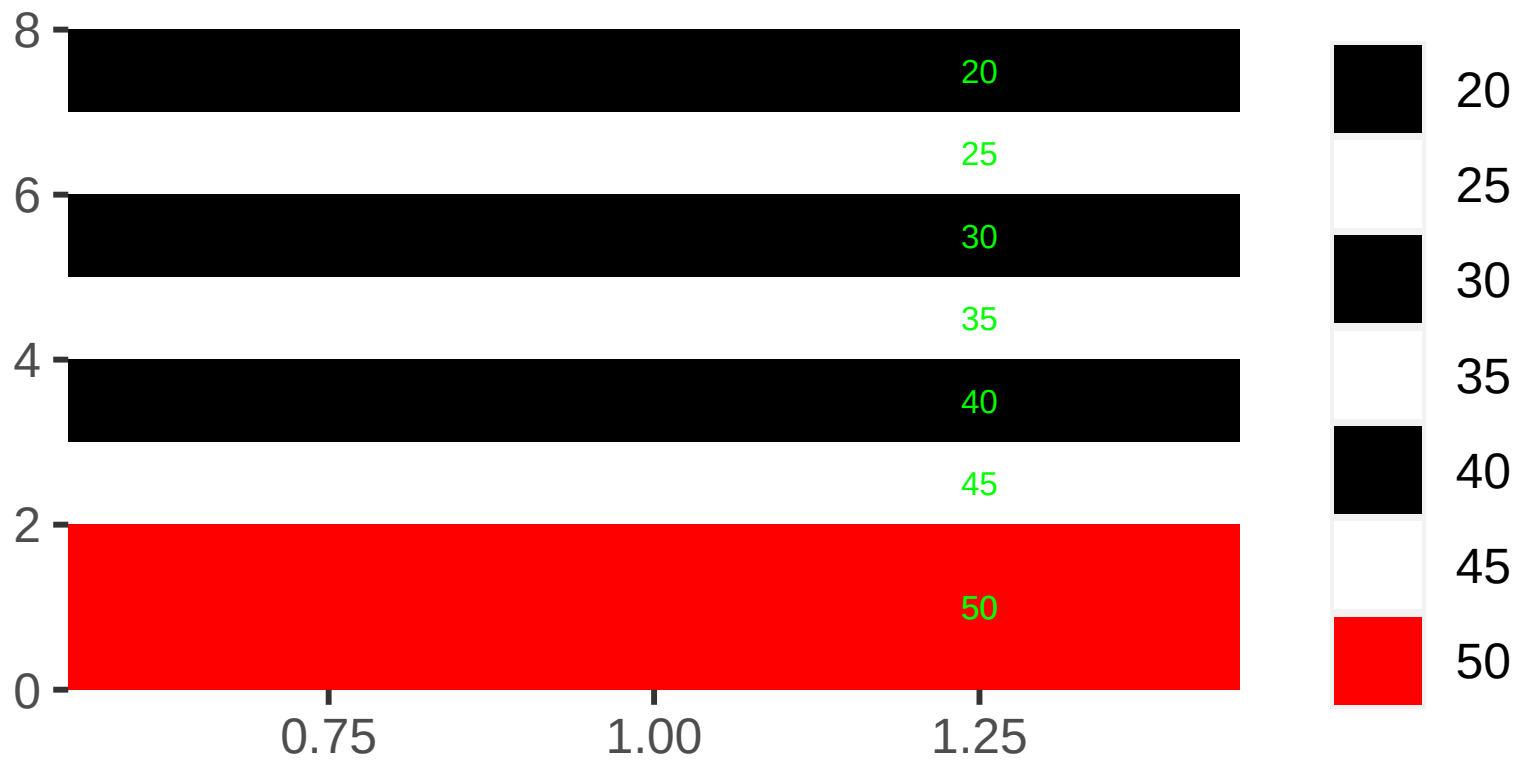
```
Bar.Cart2=ggplot(data = Base.Cart,aes(x=x,fill=c))+
  geom_bar()+
  scale_x_discrete(NULL,expand = c(0,0))+
  scale_y_continuous(NULL,expand = c(0,0))+
  scale_fill_manual(values = c("yellow","black","yellow"))
Bar.Cart2+
  coord_polar(theta = "y")
```



Coordenadas polares

Por último, también podemos hacer una tabla de tiro al blanco:

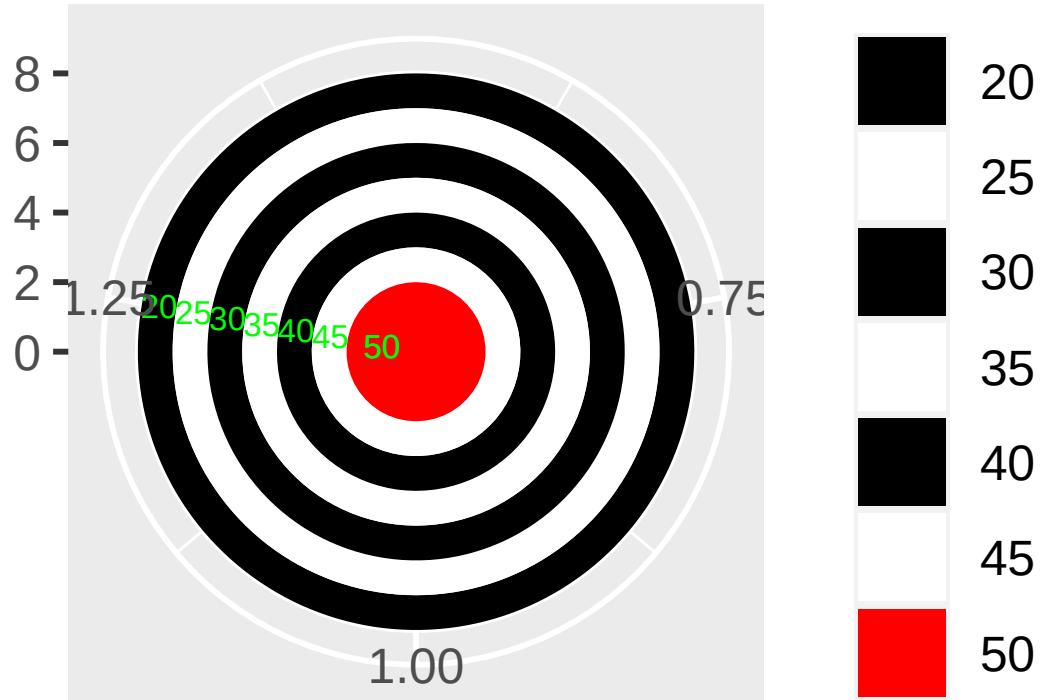
```
Base.Cart=data.frame(x=rep(1,8),y=factor(c(50,50,45,40,35,30,25,20)))
Bar.Cart=ggplot(Base.Cart,aes(x=x,fill=y))+
  geom_bar()+
  scale_fill_manual(name=NULL,
                    values = c("black","white","black","white","black","whi
  scale_x_continuous(NULL,expand = c(0,0))+
  scale_y_continuous(NULL,expand = c(0,0))
Bar.Cart
```



Coordenadas polares

Agregando coordenadas polares, obtenemos el tablero del tiro al blanco.

```
Bar.Cart+
  coord_polar(theta = "x")
```



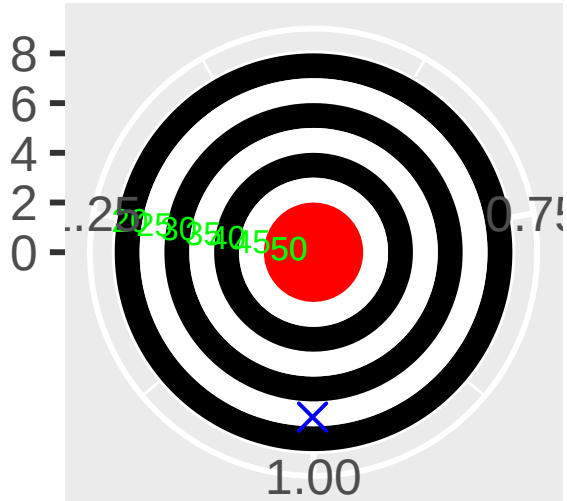
Coordenadas polares

Sorteamos un número al azar en el diagrama inicial y luego volvemos a pasar a coordenadas polares:

```
Intento1=data.frame(x=runif(1,0.5,1.5),y=8*runif(1));Intento1
```

```
##          x        y
## 1 1.000852 6.631981
```

```
Bar.Cart=Bar.Cart+
  geom_point(data = Intento1,aes(x=x,y=y),color="blue",
             size=2,shape=4,inherit.aes = F)+theme(legend.position = "none")
Bar.Cart+coord_polar(theta = "x")
```



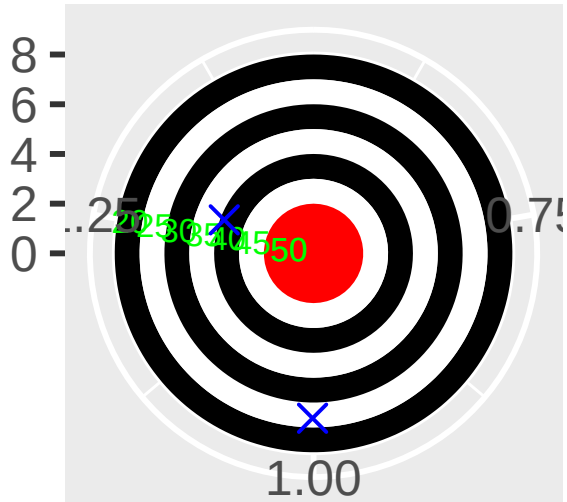
Coordenadas polares

Sorteamos un número al azar en el diagrama inicial y luego volvemos a pasar a coordenadas polares:

```
Intento2=data.frame(x=runif(1,0.5,1.5),y=8*runif(1))
Intento2
```

```
##          x        y
## 1 1.276446 3.838503
```

```
Bar.Cart=Bar.Cart+
  geom_point(data = Intento2,aes(x=x,y=y),color="blue",
             size=2,shape=4,inherit.aes = F)+theme(legend.position = "none")
Bar.Cart+coord_polar(theta = "x")
```



Coordenadas polares

Sorteamos un número al azar en el diagrama inicial y luego volvemos a pasar a coordenadas polares:

```
Intento3=data.frame(x=runif(1,0.5,1.5),y=8*runif(1))
Intento3
```

```
##          x        y
## 1 1.151344 3.456364
```

```
Bar.Cart=Bar.Cart+
  geom_point(data = Intento3,aes(x=x,y=y),color="blue",
             size=2,shape=4,inherit.aes = F)+theme(legend.position = "none")
Bar.Cart+coord_polar(theta = "x")
```