Practica4

178095: Dorely Morales Santiago

11/7/2019

```
## Loading required package: lattice

## Loading required package: ggplot2

## Loading required package: combinat

##
## Attaching package: 'combinat'

## The following object is masked from 'package:utils':
##
##     combn

## Loading required package: fAsianOptions

## Loading required package: timeDate

## Loading required package: timeSeries

## Loading required package: fBasics

## Loading required package: fOptions

##
## Attaching package: 'prob'

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, union

## Registered S3 method overwritten by 'R.oo':
##   method        from
##   throw.default R.methodsS3
```

Practica 4 Probabilidad: 1 Elementos de Probabilidad

Lanzamiento de una moneda cargada Supóngase que $P(\{\text{sol}\}) = .7$ y $P(\{\text{águila}\}) = .3$ entonces:

```
probspace(tosscoin(1), probs = c(0.70, 0.30))
```

```
##   toss1 probs
## 1     H   0.7
## 2     T   0.3
```

Ejercicio: ¿cómo calcular la probabilidad anterior con la función urnsamples?

```r
probas<-probspace(urnsamples(1:2, size = 1),probs=c(0.7,0.3))
probas$out<- c("H", "T")
probas
```

```
##   out probs
## 1   H   0.7
## 2   T   0.3
```

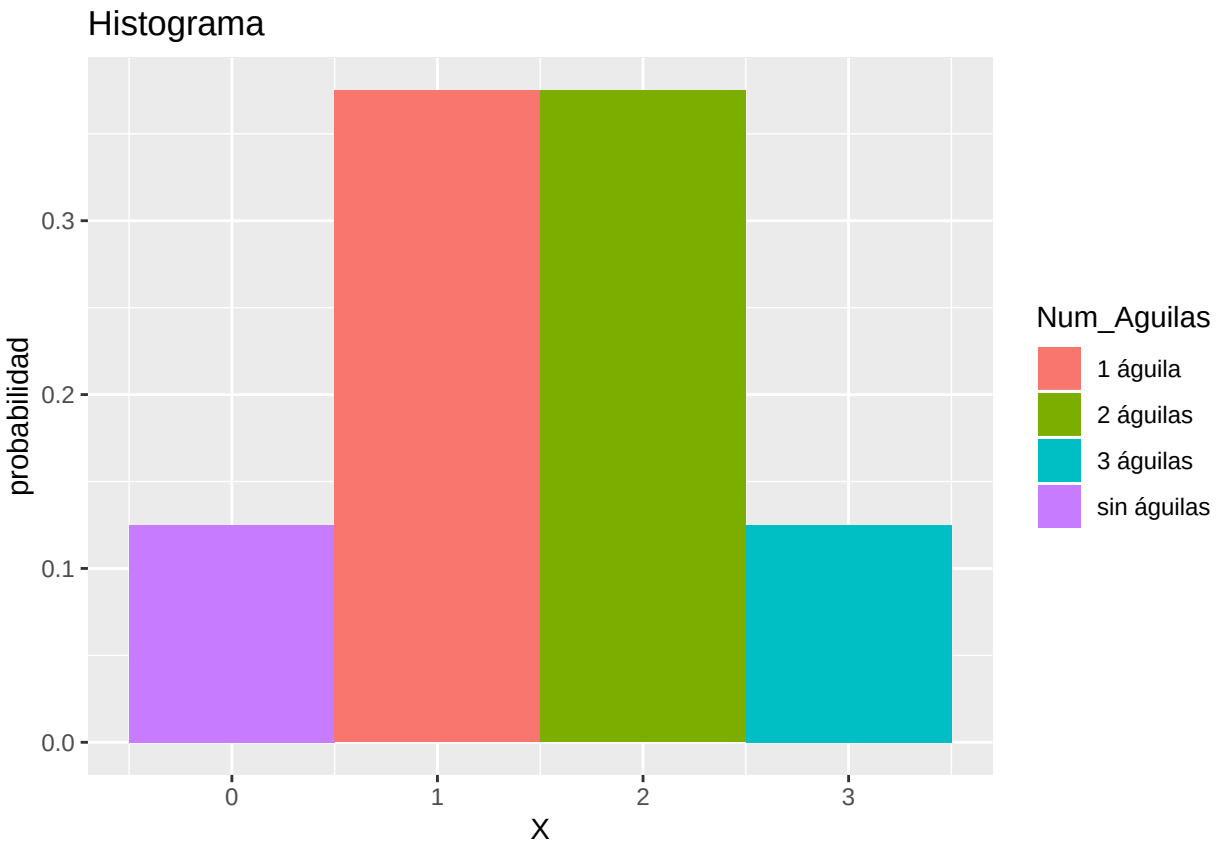
Practica 4 Probabilidad: 7 Variables aleatorias

Los siguientes ejercicios realizarlos en R y con ggplot2:

5) Encontrar los histogramas de probabilidad para las distribuciones asociadas a las siguientes variables aleatorias:

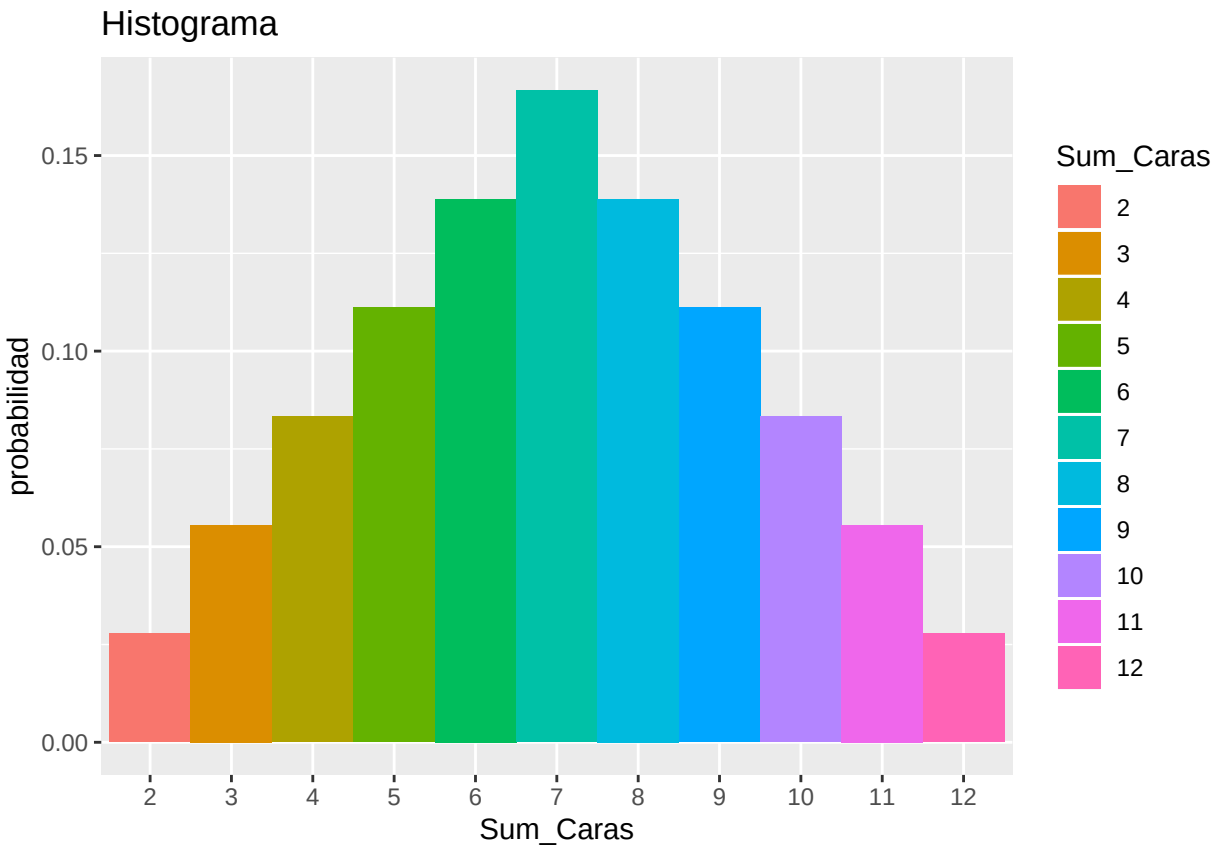
a) El número de águilas que aparecen cuando lanzamos tres monedas al aire.

```r
a0<-dbinom(0,3,.5)
a1<-dbinom(1,3,.5)
a2<-dbinom(2,3,.5)
a3<-dbinom(3,3,.5)

df = data.frame(Num_Aguilas = c('sin águilas', '1 águila', '2 águilas', '3 águilas'), X = c(0,1,2,3), pr

ggplot(data = df, aes(x=X, y=probabilidad, fill=Num_Aguilas)) +
geom_col(width=1) + #con el '+' añadimos capas a la gráfica
ggtitle('Histograma') #aes nos ayuda a mapear variables a objetos en la gráfica
```



b) La suma de los números de las caras cuando se lanzan dos dados.

```
S <- rolldie(2, nsides = 6, makespace = TRUE) #lanzamiento de un dado dos veces.
S <- addrv(S, U = X1+X2)
sumas<-marginal(S,vars = "U")

df = data.frame(Sum_Caras = factor(sumas$U), probabilidad=sumas$probs)
ggplot(data = df, aes(x=Sum_Caras, y=probabilidad, fill=Sum_Caras)) +
geom_col(width=1) + #con el '+' añadimos capas a la gráfica
ggtitle('Histograma') #aes nos ayuda a mapear variables a objetos en la gráfica
```



6) Se venden 8000 boletos para una rifa de $5000.00 y cada boleto cuesta $2.00.

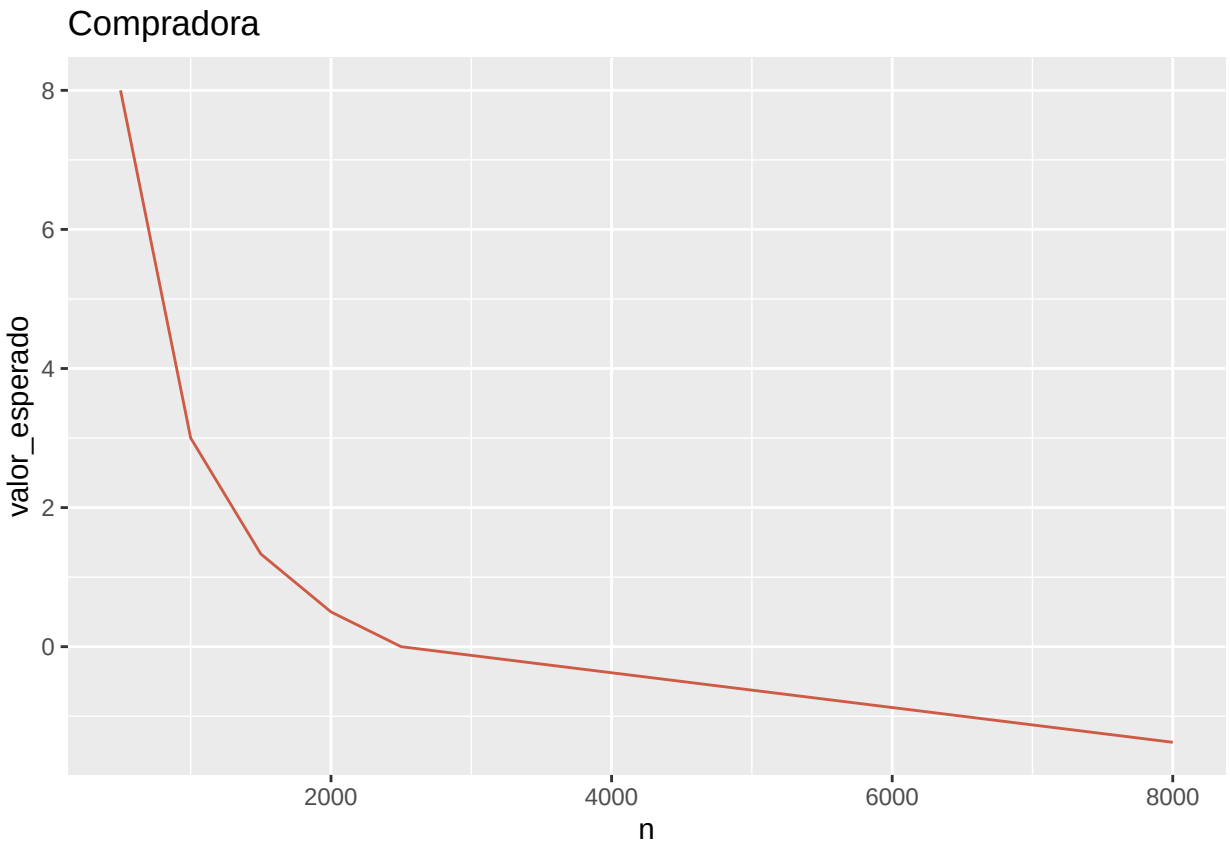
a) Encontrar la ganancia esperada del comprador de un boleto.

```
n=8000
premio=5000
costbol=2
gec=((1/n)*(premio-costbol))+((1-(1/n))*-costbol)
#La ganancia esperada es de:
gec
```

```
## [1] -1.375
```

b) Hacer la gráfica de la ganancia esperada que tiene una compradora en términos de un número de boletos n.

```
df_compradora <- data.frame(n=c(500,1000,1500,2000,2500,8000), valor_esperado=c(8,3,1.333,.5,0,-1.375))
ggplot(data=df_compradora, aes(x=n, y=valor_esperado)) +
geom_line(colour='coral3') +
ggtitle('Compradora')
```



c) ¿Cuál debería de ser el premio mínimo para que se pudiese garantizar "salir a mano" al comprar todos los boletos.

```
n=8000
costbol=2
#el valor esperado de la ganancia del comprador se debe igualar a cero y despejar el premio como incógn
premio=n*costbol*(1-(1/n))+2
#El premio mínimo debería ser de:
premio
```

```
## [1] 16000
```

Practica 4 Introducción: Core R

Secciones 1.3, 1.4, 1.7, 1.8, 1.9, 1.10, 1.16, 1.17 del libro de texto "Probability and Statistics with R" de M.D. Ugarte, A. F. Militino, A. T. Arnholt, 2ed.

Sección 1.1

```
#fija semilla para hacer reproducibles los resultados
set.seed(2019)
ruv <- runif(n = 5, min = 0, max = 1)
#Genera una muestra de 5 valores de una distribución uniforme [0,1] y las redondea a 3 decimales.
round(ruv, 3)
```

```
## [1] 0.770 0.713 0.303 0.618 0.050
```

```
#Calcula el resultado de la expresión:
(7 * 3) + 10/2 - 7^2 + sqrt(4)
```

```
## [1] -21
```

Sección 1.2

```
#Declara el vector x=5 de longitud 1
x <- 5
x
```

```
## [1] 5
```

```
#Declara el vector y=(11,3,91)
y <- c(11, 3, 91)     # combining the values into y
y
```

```
## [1] 11  3 91
```

```
#Suma de x, y
x + y
```

```
## [1] 16  8 96
```

```
# c(5, 5, 5) + y    es otra forma de hacer la misma suma
#declara el vector z=(1,2,4,8)
z <- c(1,2, 4, 8)
z
```

```
## [1] 1 2 4 8
```

```
#se añade una nueva entrada: 28 al vector y y se suma y+z
c(y, 28) + z
```

```
## [1] 12  5 95 36
```

Sección 1.3

Compara entrada a entrada si x<z. Los resultados los guarda en un vector llamado LogVec, luego imprime el tipo de dato del vector.

```
LogVec <- (x < z)
LogVec
```

```
## [1] FALSE FALSE FALSE  TRUE
```

```
typeof(LogVec)
```

```
## [1] "logical"
```

Sección 1.4

Declara los vectores lógicos X=(FALSE, TRUE, FALSE), Y=(TRUE, TRUE, TRUE) y los compara

```
X <- c(FALSE, TRUE, FALSE)
Y <- c(TRUE, TRUE, TRUE)

print("X intersección Y")
```

```
## [1] "X intersección Y"
```

```
X & Y   # Boolean X intersection Y
```

```
## [1] FALSE  TRUE FALSE
```

```
print("X unión Y")
```

```
## [1] "X unión Y"
```

```
X | Y   # Boolean X union Y
```

```
## [1] TRUE TRUE TRUE
```

```
print("¿X es igual a Y?")
```

```
## [1] "¿X es igual a Y?"
```

```
X == Y  # Boolean EQUALITY
```

```
## [1] FALSE  TRUE FALSE
```

```
print("¿X es distinto a Y?")
```

```
## [1] "¿X es distinto a Y?"
```

```
X != Y  # Boolean INEQUALITY
```

```
## [1]  TRUE FALSE  TRUE
```

```
print("X intersección Y sólo del primer elemento")
```

```
## [1] "X intersección Y sólo del primer elemento"
```

```
X && Y  # only looks at first element of each vector (intersection)
```

```
## [1] FALSE
```

```
print("X unión Y sólo del primer elemento")
```

```
## [1] "X unión Y sólo del primer elemento"
```

```
X || Y  # only looks at first element of each vector (union)
```

```
## [1] TRUE
```

Sección 1.7

Declara el Vector Numérico NV=(-2,-1, 0, 1, 2)

```
NV <- c(-2,-1, 0, 1, 2)
print("Dividimos NV entre sí mismo")
```

```
## [1] "Dividimos NV entre sí mismo"
```

```
NV/NV                # 0/0 is not a number (NaN)
```

```
## [1]   1   1 NaN   1   1
```

```
print("Sacamos raíz cuadrada de NV, pero no arroja resultados de negativos")
```

```
## [1] "Sacamos raíz cuadrada de NV, pero no arroja resultados de negativos"
```

```
sqrt(NV)
```

```
## Warning in sqrt(NV): Se han producido NaNs
```

```
## [1]      NaN      NaN 0.000000 1.000000 1.414214
```

```
print("Eleva el vector a la 2019")
```

```
## [1] "Eleva el vector a la 2019"
```

```
NV^2019  # very large and small numbers use -Inf and Inf
```

```
## [1] -Inf   -1    0    1  Inf
```

```
print("Omite el tercer elemento de NV")
```

```
## [1] "Omite el tercer elemento de NV"
```

```
NV[-3]
```

```
## [1] -2 -1  1  2
```

```
print("Extrae el primer y el quinto elemento de NV")
```

```
## [1] "Extrae el primer y el quinto elemento de NV"
```

```
NV[c(1, 5)]
```

```
## [1] -2  2
```

```
print("Genera el vector CV de las primeras y últimas 4 letras del alfabeto ")
```

```
## [1] "Genera el vector CV de las primeras y últimas 4 letras del alfabeto "
```

```
CV <- LETTERS[c(1, 2, 3, 4, 24,25,26)]
CV
```

```
## [1] "A" "B" "C" "D" "X" "Y" "Z"
```

```
print("Extrae las primeras 3 letras del vector CV")
```

```
## [1] "Extrae las primeras 3 letras del vector CV"
```

```
CV[c(1,2,3)]
```

```
## [1] "A" "B" "C"
```

Sección 1.8

Crea una secuencia de números y letras

```
print("Conteo regresivo desde 10")
```

```
## [1] "Conteo regresivo desde 10"
```

```
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
print("Secuencia de letras de la décima a la quinta")
```

```
## [1] "Secuencia de letras de la décima a la quinta"
```

```r
letters[10:5]
```

```
## [1] "j" "i" "h" "g" "f" "e"
```

```r
#series de números
seq(from = 1, to = 49, by = 3)
```

```
##  [1]  1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49
```

```r
seq(from = 5, by = 5, length.out = 10)
```

```
##  [1]  5 10 15 20 25 30 35 40 45 50
```

```r
seq(from = 23, to = 22, length.out = 5)
```

```
## [1] 23.00 22.75 22.50 22.25 22.00
```

Sección 1.9

```r
#vector de 5's con 20 entradas
rep(x = 5, times = 20)
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

```r
#vector lógico T/F con 5 entradas
rep(x = c(TRUE, FALSE), times = 10)
```

```
##  [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
## [12] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
```

```r
#De la A a la E con 3 repeticiones por letra
rep(x = letters[1:5], each = 3)
```

```
##  [1] "a" "a" "a" "b" "b" "b" "c" "c" "c" "d" "d" "d" "e" "e" "e"
```

```r
#conteo ascendente del 1 al 3 repitiendo los números 2, 3 y 4 veces
rep(x = 1:3, times = 2:4)
```

```
## [1] 1 1 2 2 2 3 3 3 3
```

```r
#repite serie del 1 al 5 hasta completar 20 elementos
rep(x = 1:5, length.out = 20)
```

```
##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

Sección 1.10

```r
#conteo regresivo del 3 al 1 repitiendo los números 1, 2 y 3 veces
FEV <- rep(x = 3:1, times = 1:3)
FEV
```

```
## [1] 3 2 2 1 1 1
```

```r
#Multiplica al vector por sí mismo
FEV * FEV
```

```
## [1] 9 4 4 1 1 1
```

```r
#Devuelve cada entrada si es mayor o igual a 4
FEV * FEV >= 4
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

```r
#Esta es otra forma de hacer lo mismo
subset(x = FEV, subset = FEV * FEV >=4)
```

```
## [1] 3 2 2
```

```r
#Aquí se devuelven los índices de los valores que sí son mayores o iguales a 4
which(FEV * FEV >=4)
```

```
## [1] 1 2 3
```

Sección 1.16

```r
#declara un vector numérico
v <- c(0, 1, 0,2, 2,3)
#convierte el vector en categórico
fv <- factor(v)
fv
```

```
## [1] 0 1 0 2 2 3
## Levels: 0 1 2 3
```

```r
#añade etiquetas a cada categoría
fv <- factor(v, levels = 0:3, labels = c("Facil","Media", "Dificil", "Imposible"))
fv
```

```
## [1] Facil     Media     Facil     Dificil   Dificil   Imposible
## Levels: Facil Media Dificil Imposible
```

Sección 1.17

```r
v2 <- c("Imposible", "Dificil", "Facil", "Imposible")
v2
```

```
## [1] "Imposible" "Dificil"   "Facil"     "Imposible"
```

```r
#Factor elimina las categorías duplicadas para crear los niveles
fv2 <- factor(v2)
fv2
```

```
## [1] Imposible Dificil   Facil     Imposible
## Levels: Dificil Facil Imposible
```

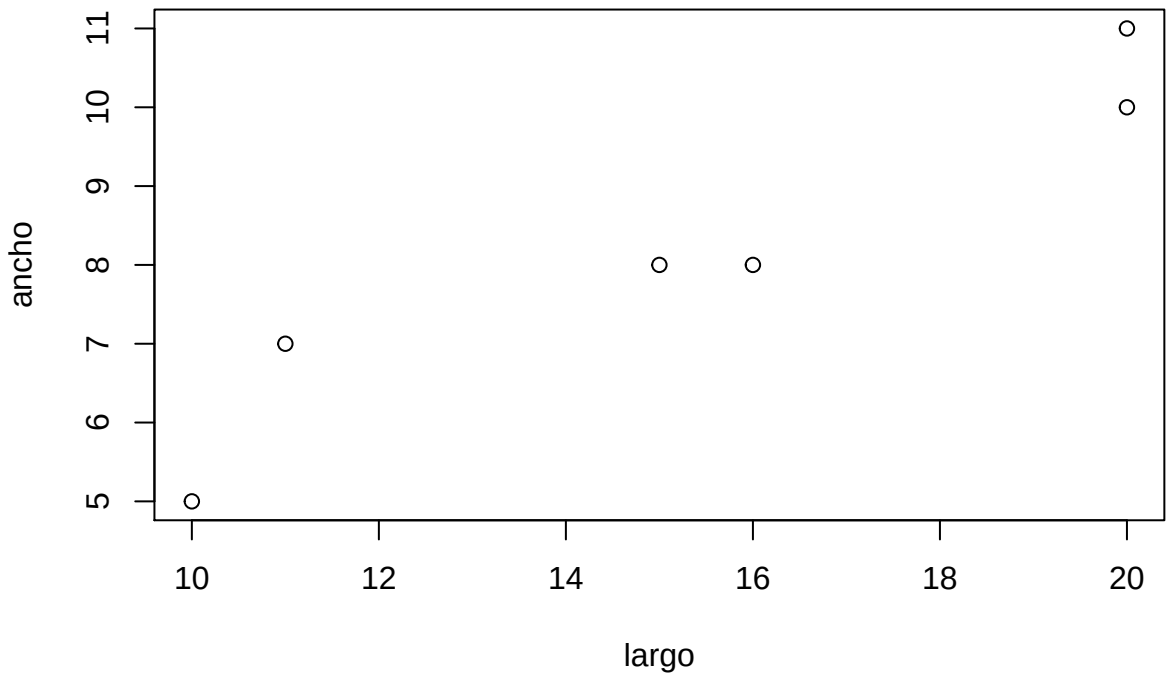
Practica 4 Introducción: base graphics y ggplot2

Revisar las siguientes referencias para graficar en el paquete base y ggplot2 de R

Gráficas x,y (`plot`: generic x-y plotting)

Dando x, y:

```r
largo<-c(10,20,11,15,16,20)
ancho<-c(5,10,7,8,8,11)
plot(x=largo, y=ancho)
```



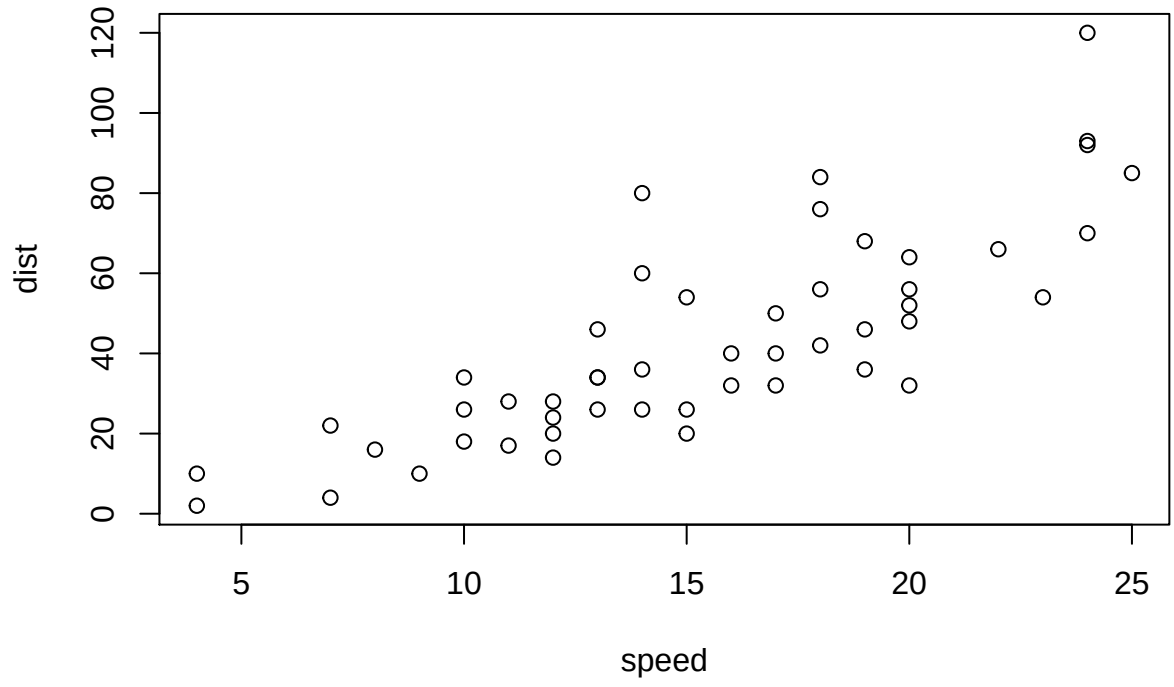
Dando un objeto que tiene dos columnas: Velocidad y Distancia, se toman automático como x,y:

```r
# ver el contenido de `cars`(una df ejemplo que viene con R)
head(cars)
```

```
##   speed dist
```

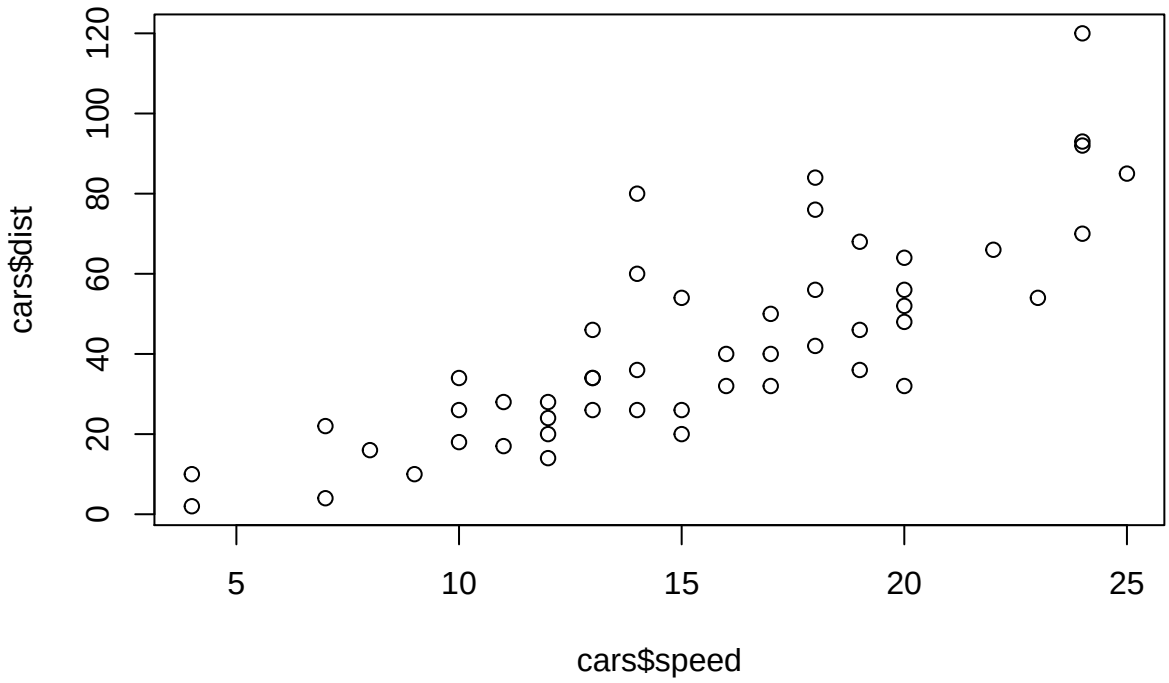
```
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```r
plot(cars)
```



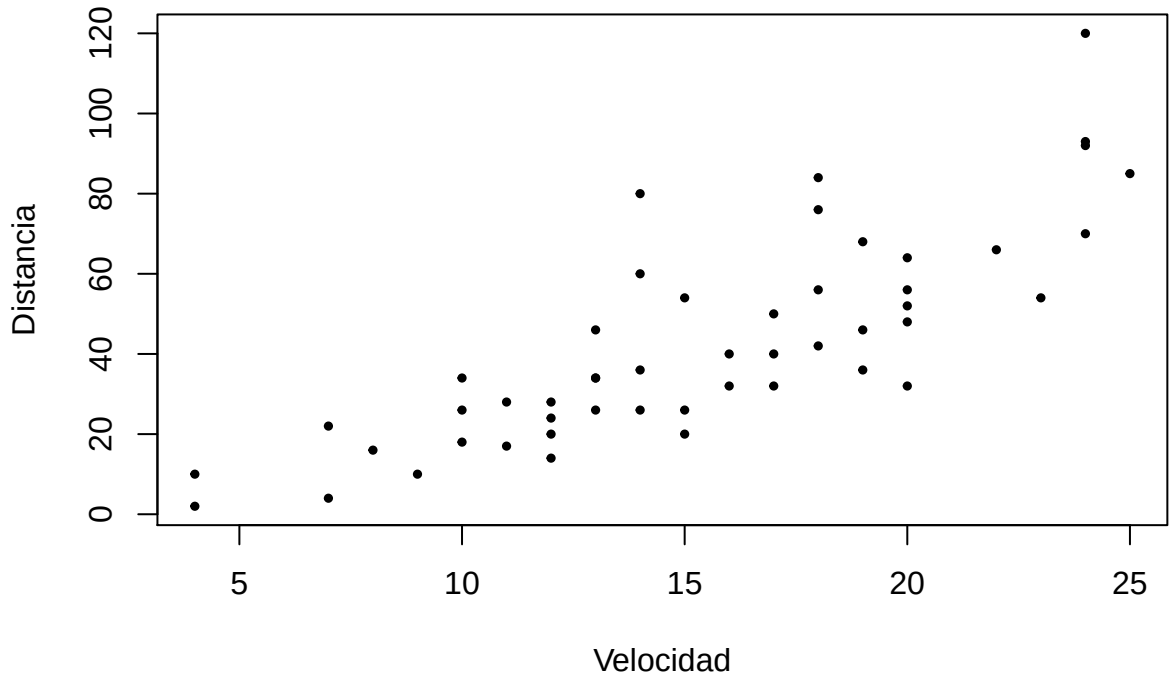
Si queremos especificar qué columnas serán x, y del objeto:

```r
# graficar vel vs distancia
plot(x=cars$speed, y=cars$dist)
```



Cambiar título de ejes e íconos:

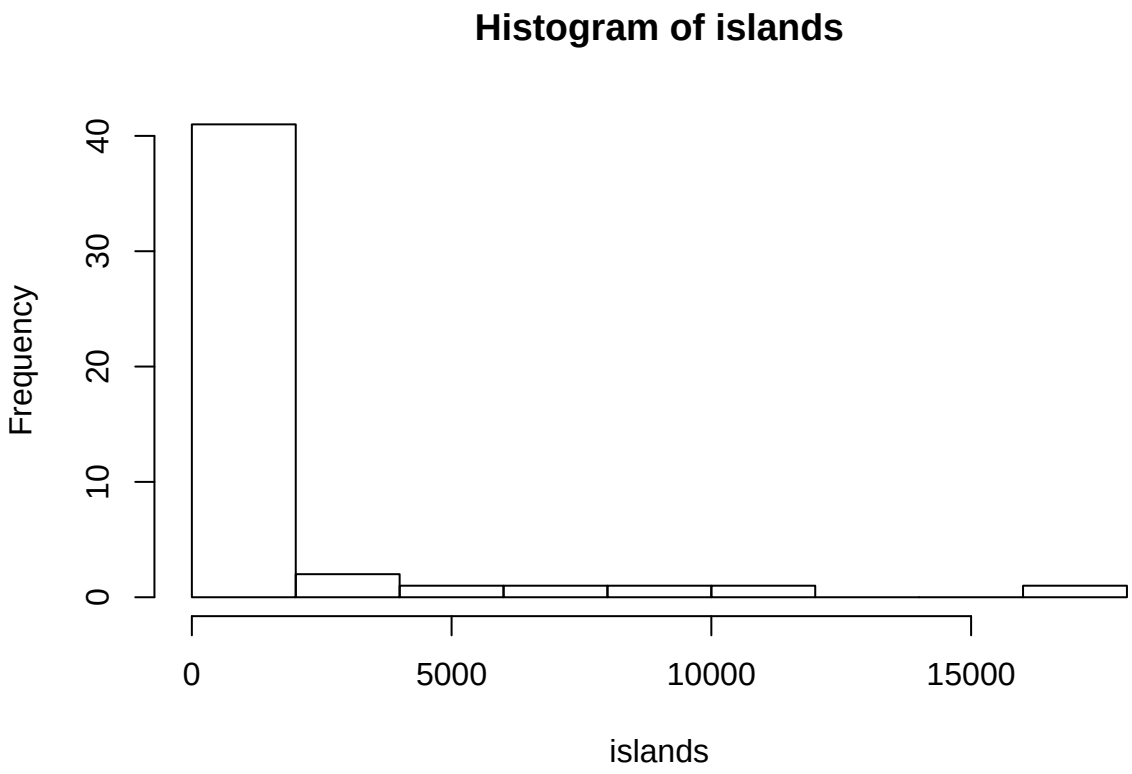
```
# graficar vel vs distancia
plot(x=cars$speed, y=cars$dist, xlab="Velocidad", ylab="Distancia", cex=0.5, pch=19)
```



Histogramas (`hist`: histograms)

Ejemplo con los datos islands (viene con R)

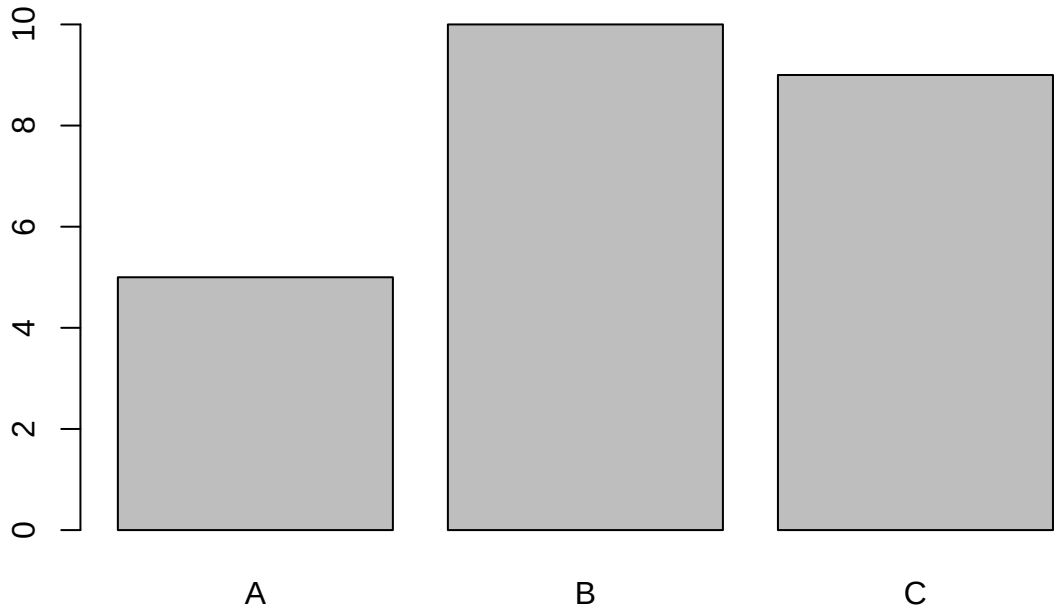
```
hist(islands)
```



Barplot (`barplot`: bar plots)

Ejemplo:

```
DNAcon<-data.frame(muestra=c("A", "B", "C"), concentracionADN=c(5,10,9))
barplot(DNAcon$concentracionADN, names.arg=DNAcon$muestra)
```



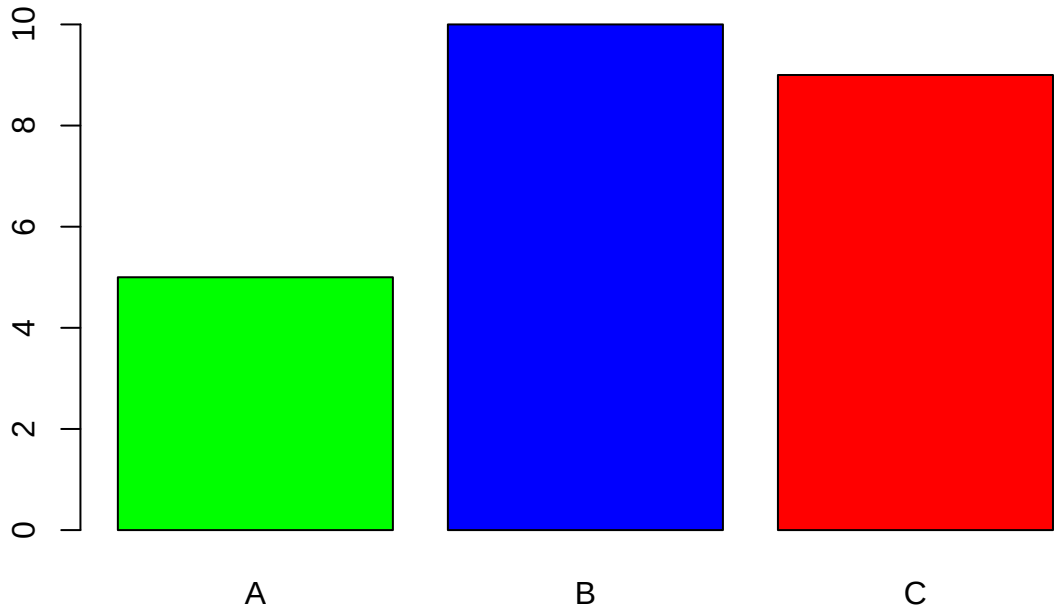
***Definir colores

Los colores que R ocupa para colorear algo están definidos en `palette` y pueden cambiarse

```
# Ver colores
palette()
```

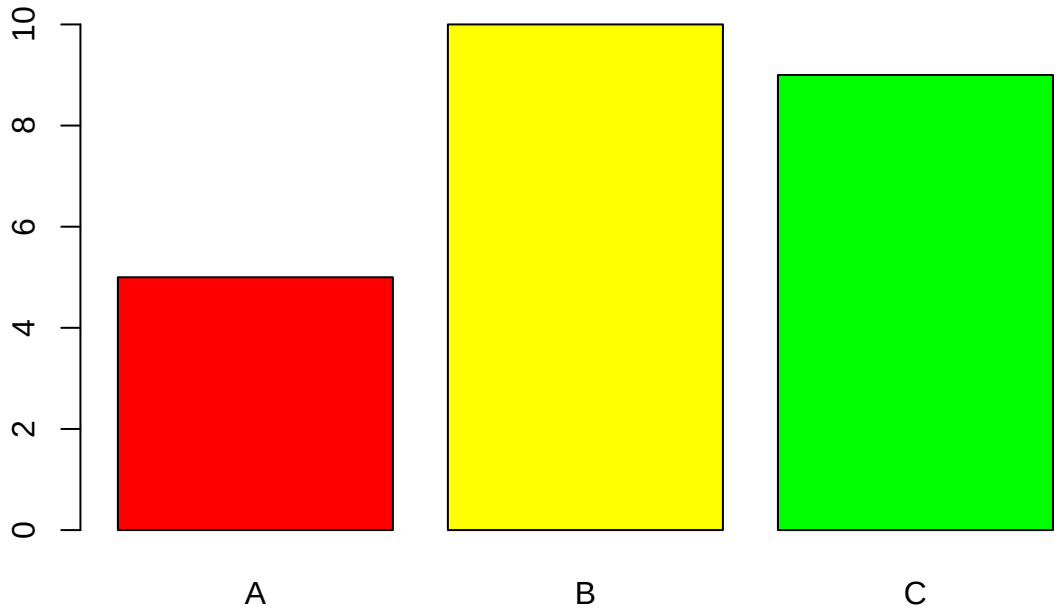
```
## [1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
## [8] "gray"
```

```
# Cambiar colores
palette(c("green", "blue", "red"))
# volver a graficar
barplot(DNAcon$concentracionADN, names.arg=DNAcon$muestra,col=DNAcon$muestra)
```



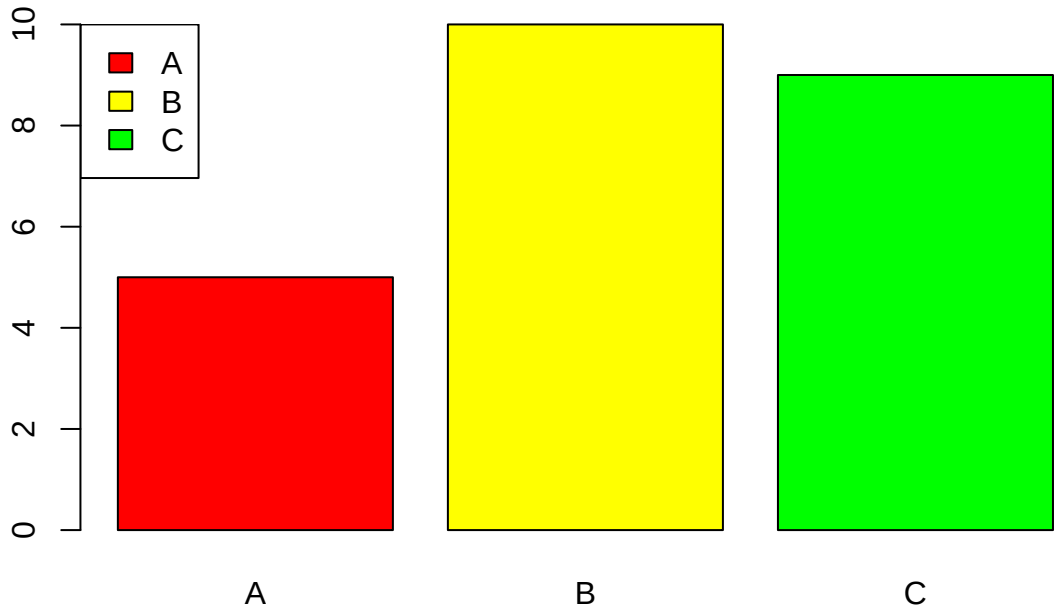
Además de manualmente, los colores se pueden definir via paletas predeterminadas:

```
# Cambiar palette a 6 colores del arcoiris
palette(rainbow(6))
# volver a graficar
barplot(DNAcon$concentracionADN, names.arg=DNAcon$muestra,col=DNAcon$muestra)
```



***Agregar una leyenda

```
# Graficar
barplot(DNAcon$concentracionADN, names.arg=DNAcon$muestra,col=DNAcon$muestra)
# Agregar leyenda
legend(x="topleft", legend=levels(DNAcon$muestra), fill=palette()[1:3])
```



Nota que `legend` es una función por si misma (i.e. NO un argumento de `plot`) que requiere que antes de correrlo se haya corrido `plot`. Es decir una vez que creamos una gráfica podemos **agregar sobre de esta** una leyenda. Lo mismo puede hacerse con la función `title`.

***ggplot2

Las gráficas que hemos visto hasta ahora pueden verse un poco feas de inicio y puede tomar un rato y mucho código arreglarlas a algo hermoso. `ggplot2` es un paquete que ahorra este trabajo y que ahora es ampliamente adoptado.

`ggplot2` construye gráficas "definiendo sus componentes paso a paso".

Para poder usar `ggplot2` se requiere que la data.frame esté en **formato largo**. Además de esos apuntes puedes revisar esto si te quedan dudas.

Términos importantes:

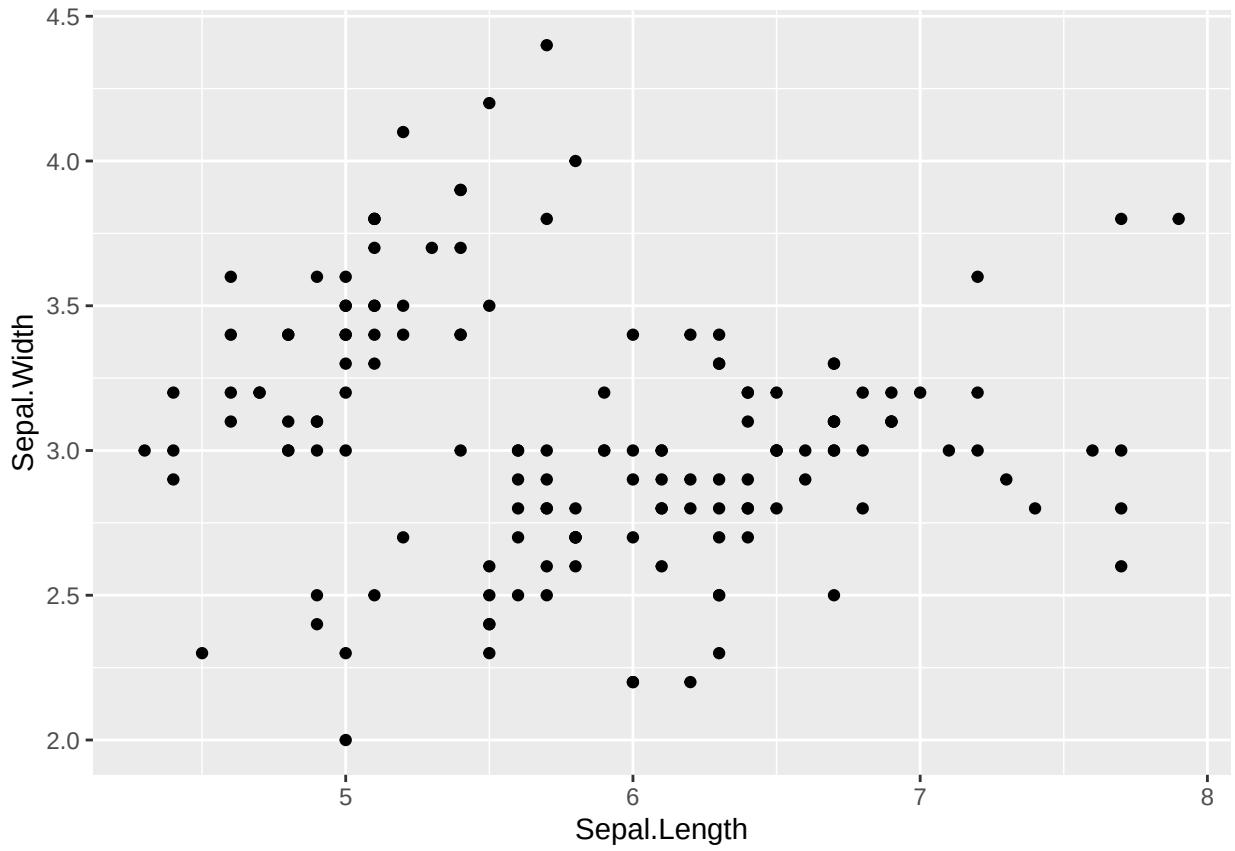
Ojo: Mucho mejor que ver la ayuda de cada función es ver la Documentación online de ggplot2 y este R Graphics Cookbook * `ggplot` la función principal donde se especifican el set de datos y las variables a graficar. * **geoms** "objetos geométricos" (el tipo de gráfica en cierto modo): - `geom_point()` - `geom_bar()` - `geom_density()` - `geom_line()` - `geom_area()` - `geom_histogram()` * **aes** los estéticos que pondremos: forma, transparencia (alpha), color, relleno, tipo de línea, etc. * **scales** para especificar si los datos se graficarán de forma continua, discreta, logarítmica. * **themes** para modificar los elementos de la gráfica no relacionados con los datos, como el tipo de letra y el color del fondo.

Gráficas de dispersión

```
# Examinar datos pre-cargados
head(iris)
```

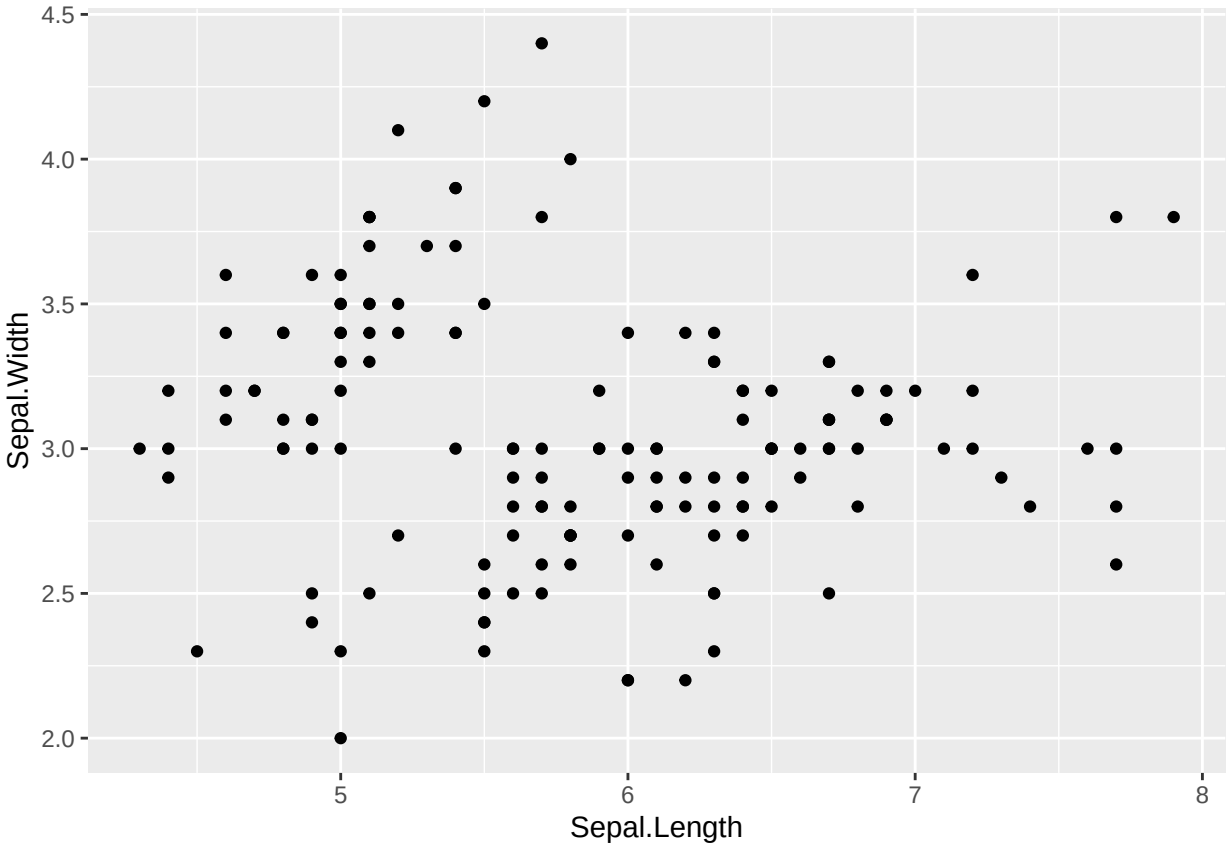
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# graficar
ggplot(data=iris, aes(x=Sepal.Length, y= Sepal.Width)) + geom_point()
```



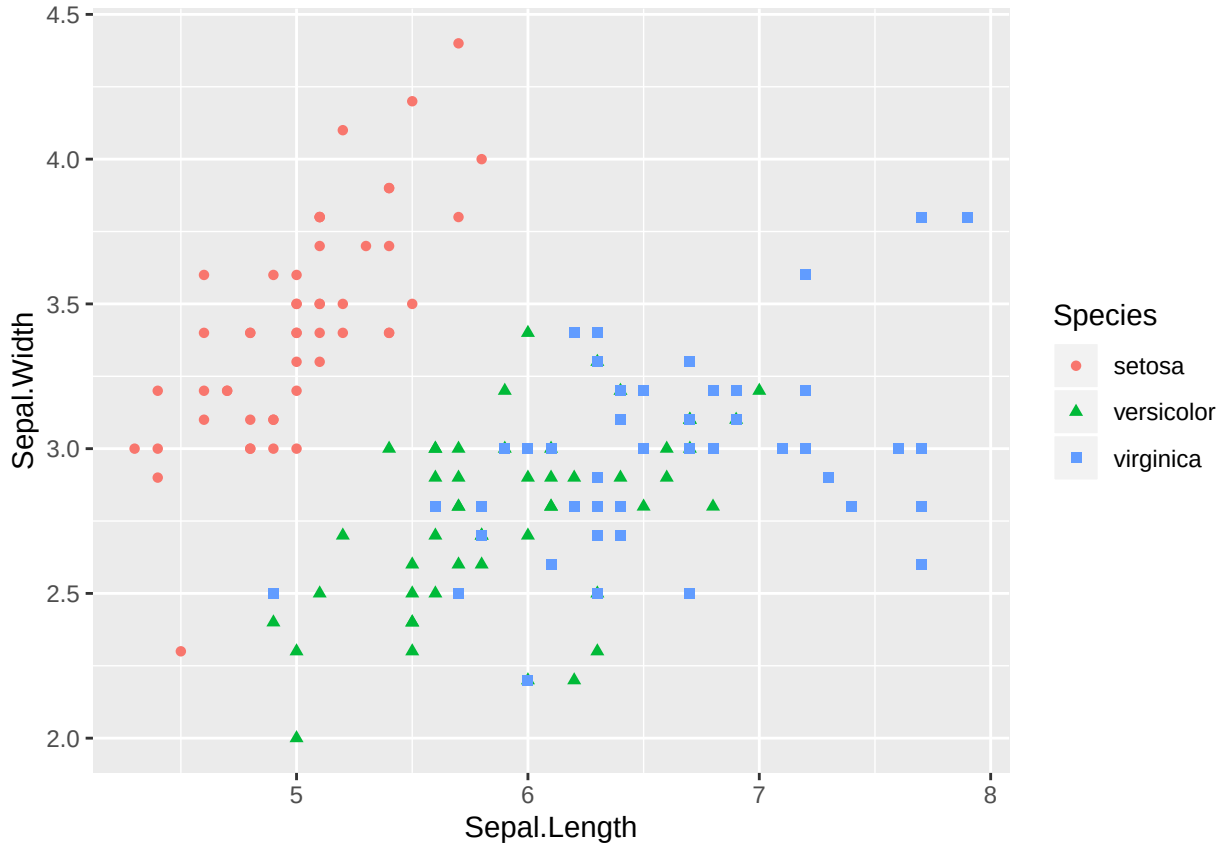
Pregunta: ¿Qué hace el símbolo +? Nota que el código anterior tmb puede escribirse así:

```
myplot<-ggplot(data=iris, aes(x=Sepal.Length, y= Sepal.Width))
myplot + geom_point()
```



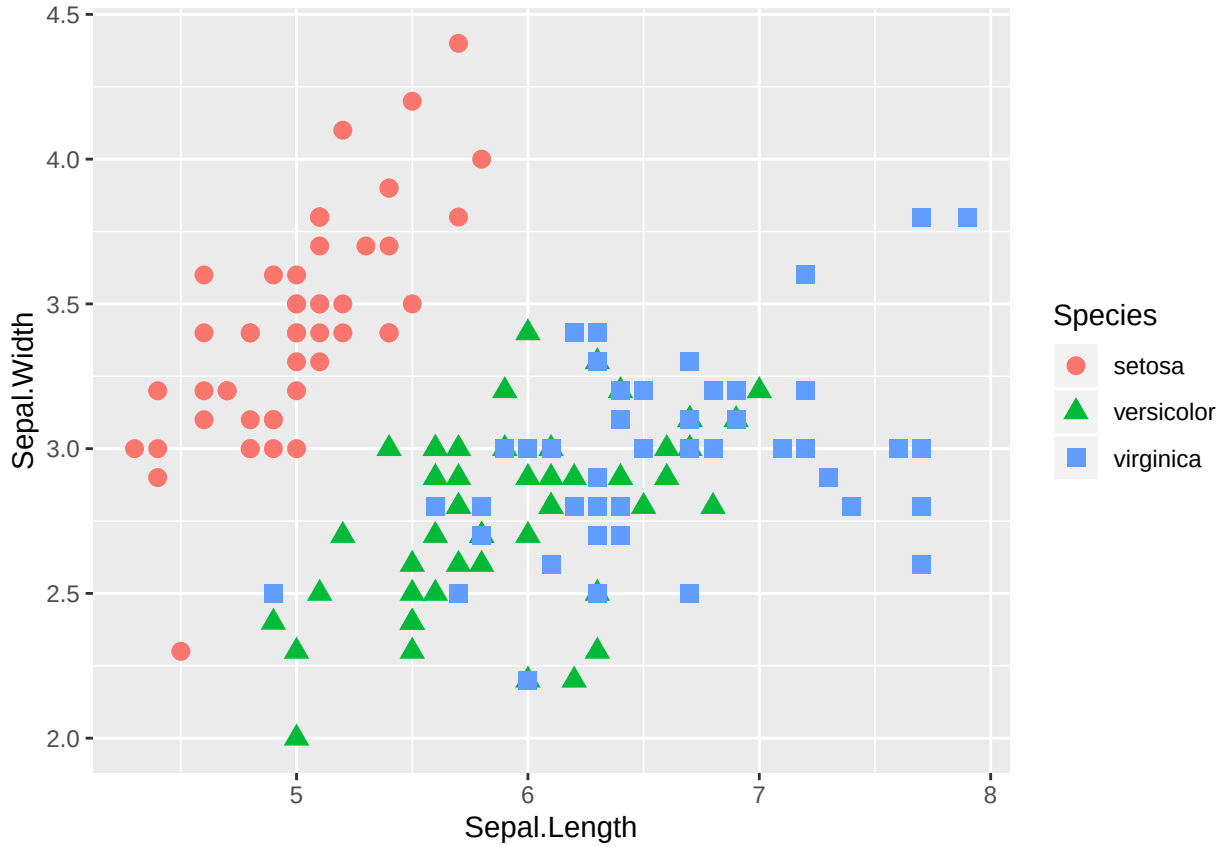
Los colores y formas se cambian en **aes**:

```
ggplot(data=iris, aes(x=Sepal.Length, y= Sepal.Width, color= Species, shape=Species)) + geom_point()
```



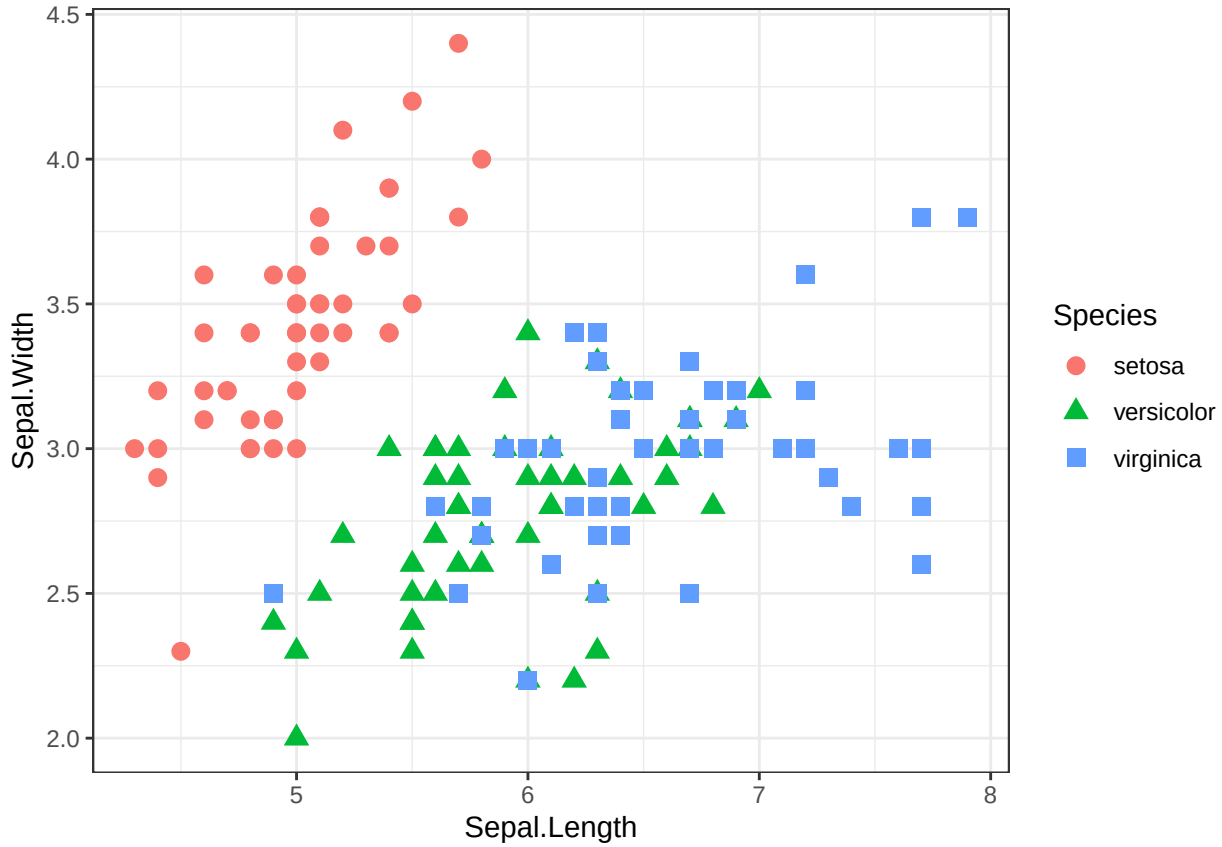
Ya sea en el aes de la función inicial o dentro de los geoms (Nota que el tamaño no es un aes, sino un argumento de geom_point)

```
ggplot(data=iris, aes(x=Sepal.Length, y= Sepal.Width)) +
    geom_point(aes(color= Species, shape=Species), size=3)
```



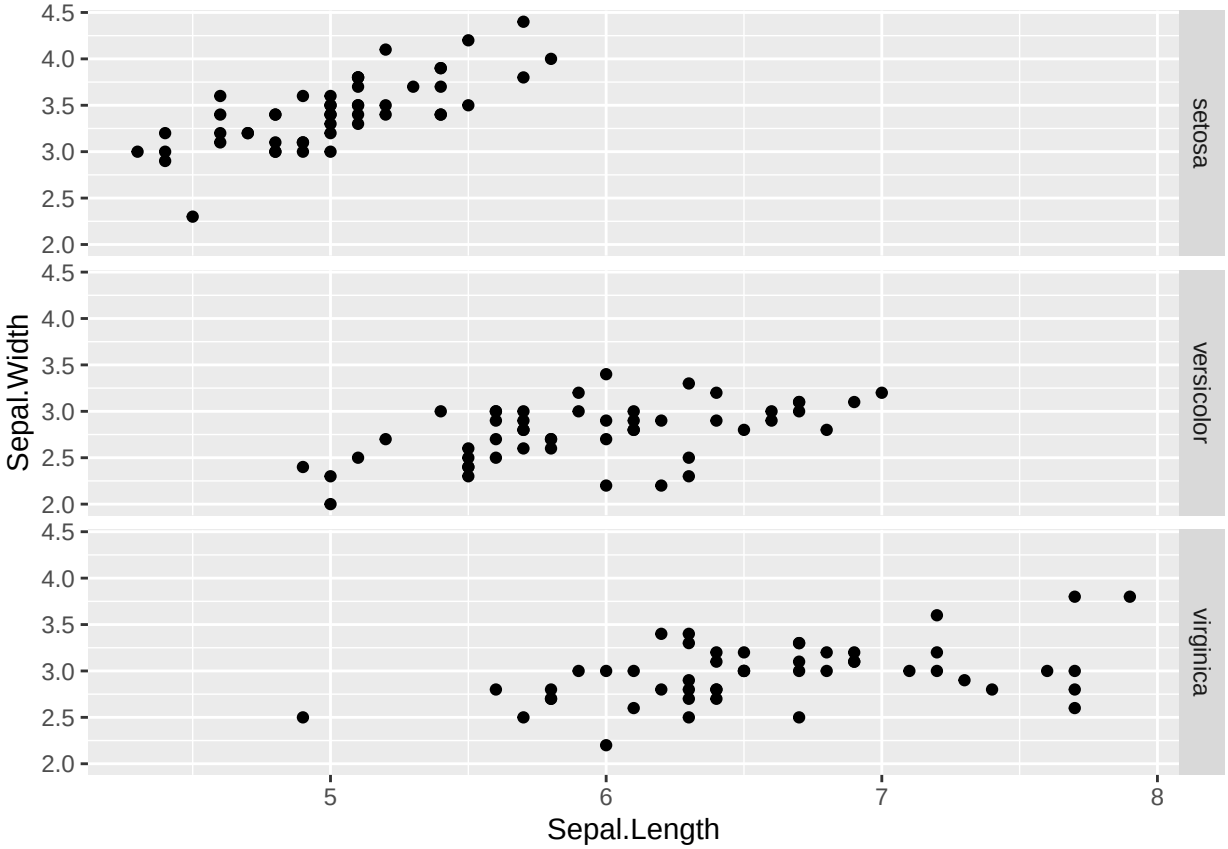
Si queremos quitar el fondo gris:

```
ggplot(data=iris, aes(x=Sepal.Length, y= Sepal.Width)) +
    geom_point(aes(color= Species, shape=Species), size=3) +
    theme_bw()
```

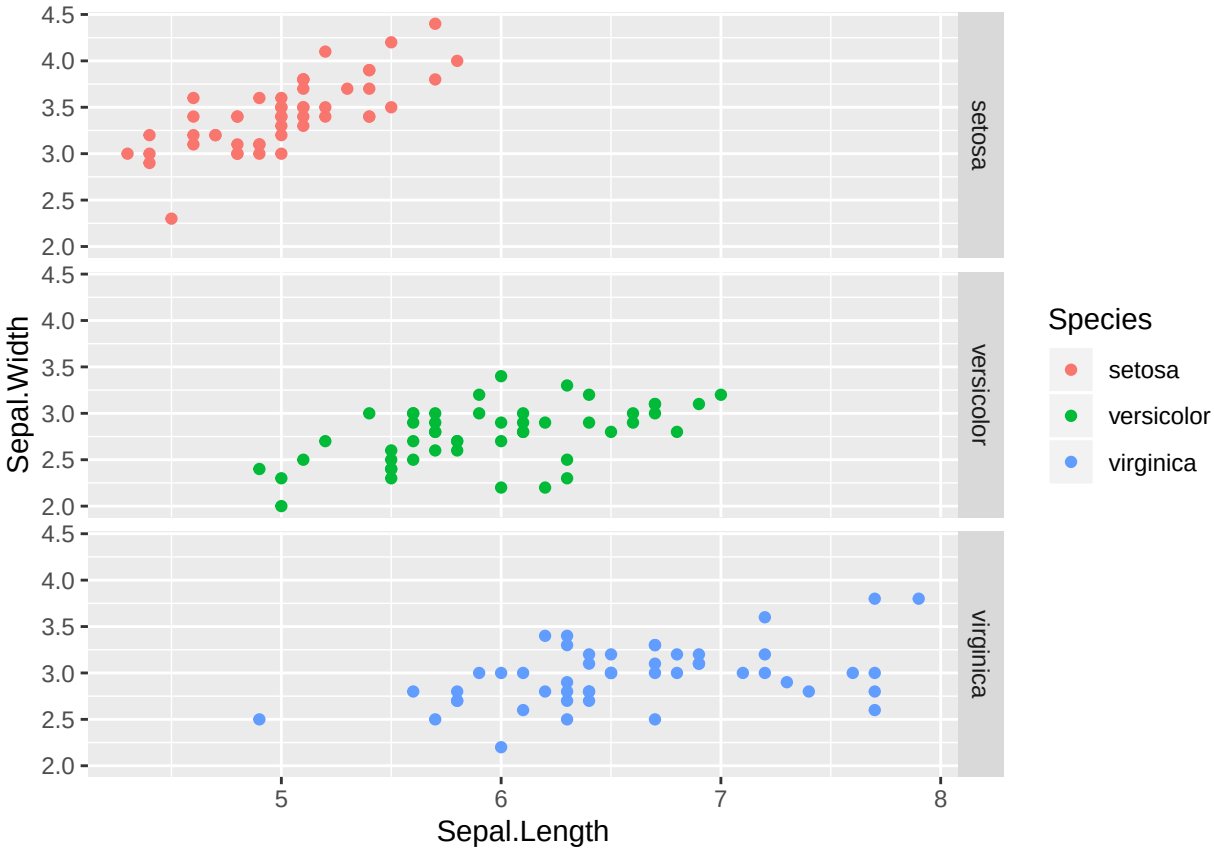


A veces queremos graficar en páneles separados la misma info para diferentes tratamientos o especies. Por ejemplo:

```
ggplot(data=iris, aes(x=Sepal.Length, y= Sepal.Width)) +
  geom_point() +
  facet_grid(Species ~ .)
```



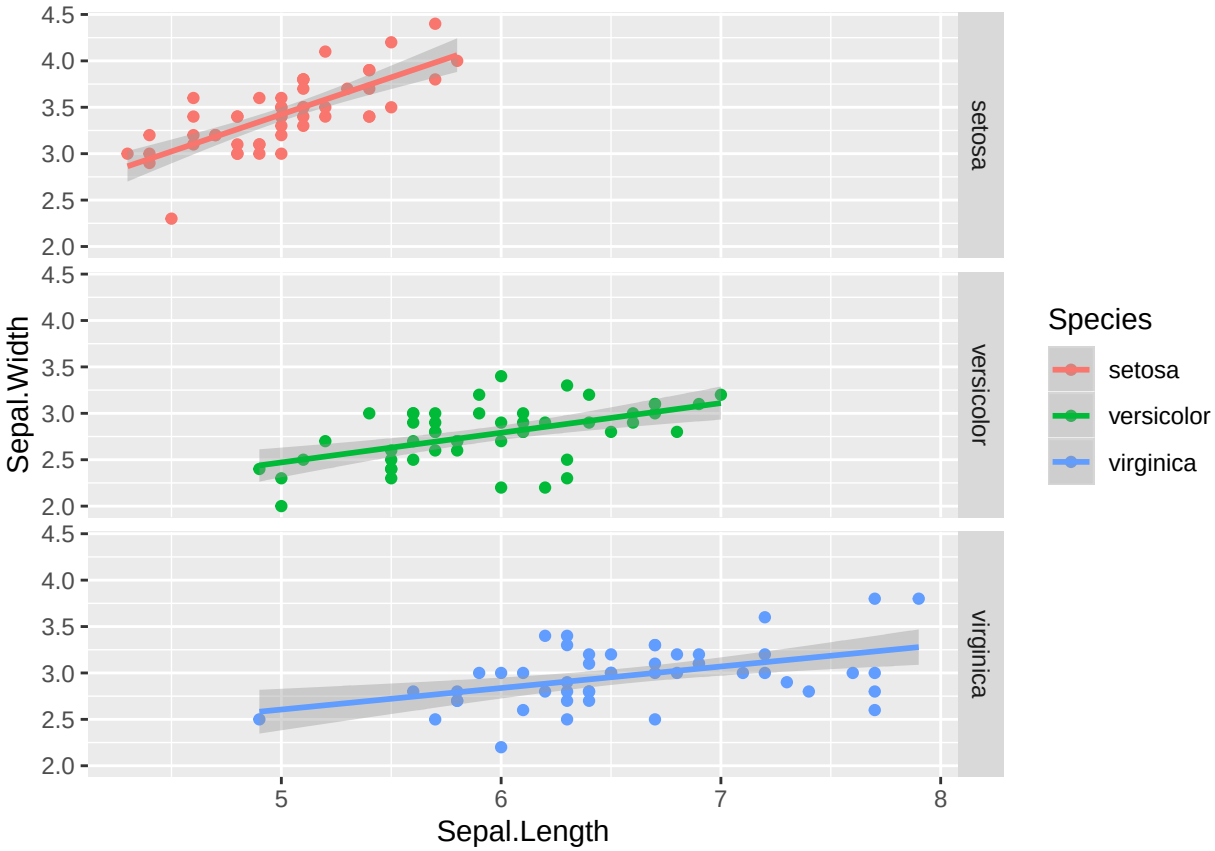
Ejercicio Pon color por especie a la gráfica anterior:



Repite la gráfica anterior pero para que se vea así:

También podemos agregar el resultado de un modelo matemático, como una regresión lineal:

```
ggplot(data=iris, aes(x=Sepal.Length, y= Sepal.Width, color=Species)) +
  geom_point() +
  facet_grid(Species ~ .) +
  geom_smooth(method="lm")
```



En `ggplot2` se construye la gráfica agregando diferentes capas o *layers*, diferentes tipos de capas que se trataran en este curso son:

1. **Aesthetic**: `aes()` propiedades de objetos por mapear en la gráfica (axis x, axis y, size, shape, color, fill).
2. **Geoms**: `geom_` Objetos geométricos. Estos objetos se dividen en objetos que mapean en una dimensión, dos dimensiones y tres dimensiones.
3. **Transformaciones estadísticas**: `stat_` Resúmen estadístico de variables.
4. **Facets**: `facet_` División de gráficas en diferentes paneles o subgráficas.
5. **Themes**: `theme()` Aspectos de la gráfica independientes de los datos, como: fuente, títulos, posición de leyendas y fondo.

Existen dos funciones para graficar: `ggplot` y `qplot`.

1. Gráficas de dispersión

Usaremos el conjunto de datos *mpg* que se incluye en R, puedes encontrar información de esta base de datos tecleando `?mpg`.

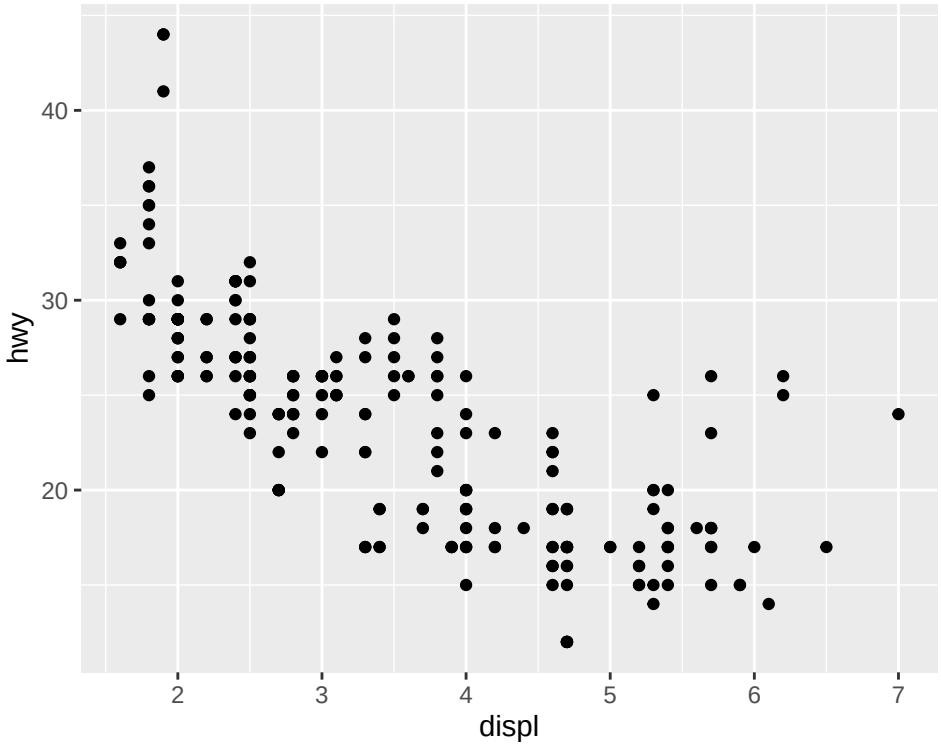
```
# estructura de la base
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and '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" ...
```

Para realizar una gráfica de dispersión:

1. Se debe especificar explícitamente que base de datos usamos, este es el primer argumento en la función `ggplot`
2. Dentro de *aes()* escribimos la variable que queremos graficar en cada eje.
3. Posteriormente se definen las geometrías, geoms, que controlan el tipo de gráfica. Así se agrega otra capa a la gráfica con el símbolo `(+)`.

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

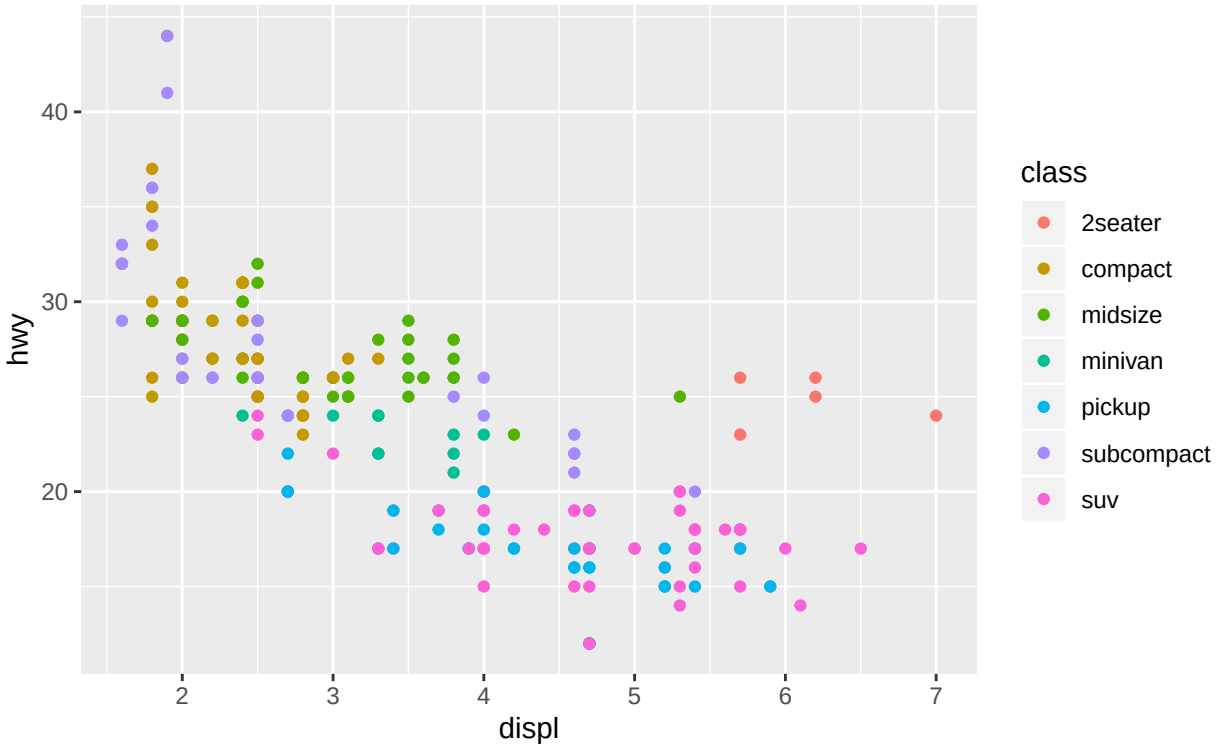


La forma paralela de graficar con `qplot` se presenta abajo.

```
qplot(displ, hwy, data = mpg, geom = 'point')
```

Podemos representar variables adicionales usando otras características estéticas (aesthetics) como forma, color o tamaño.

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



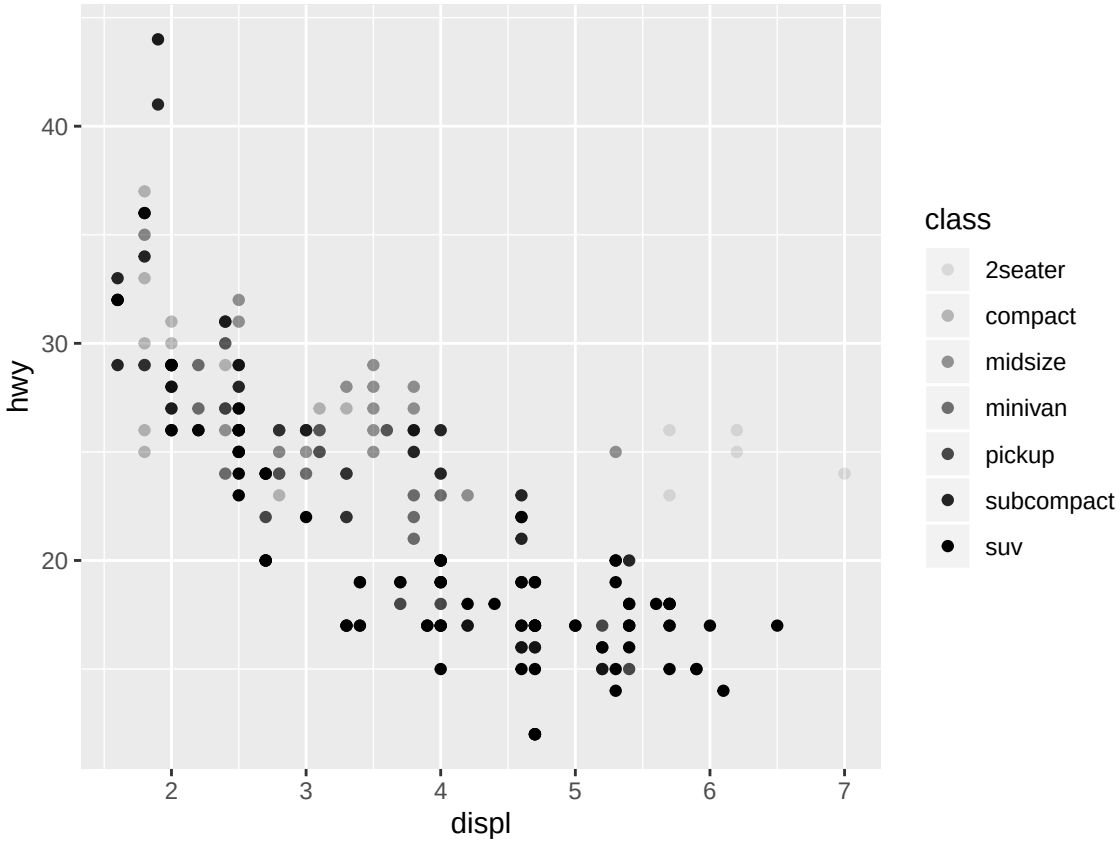
La forma paralela de graficar con `qplot` se presenta abajo.

```
qplot(displ, hwy, color = class, data = mpg, geom = 'point')
```

Ahora se prueba con otras variables estéticas.

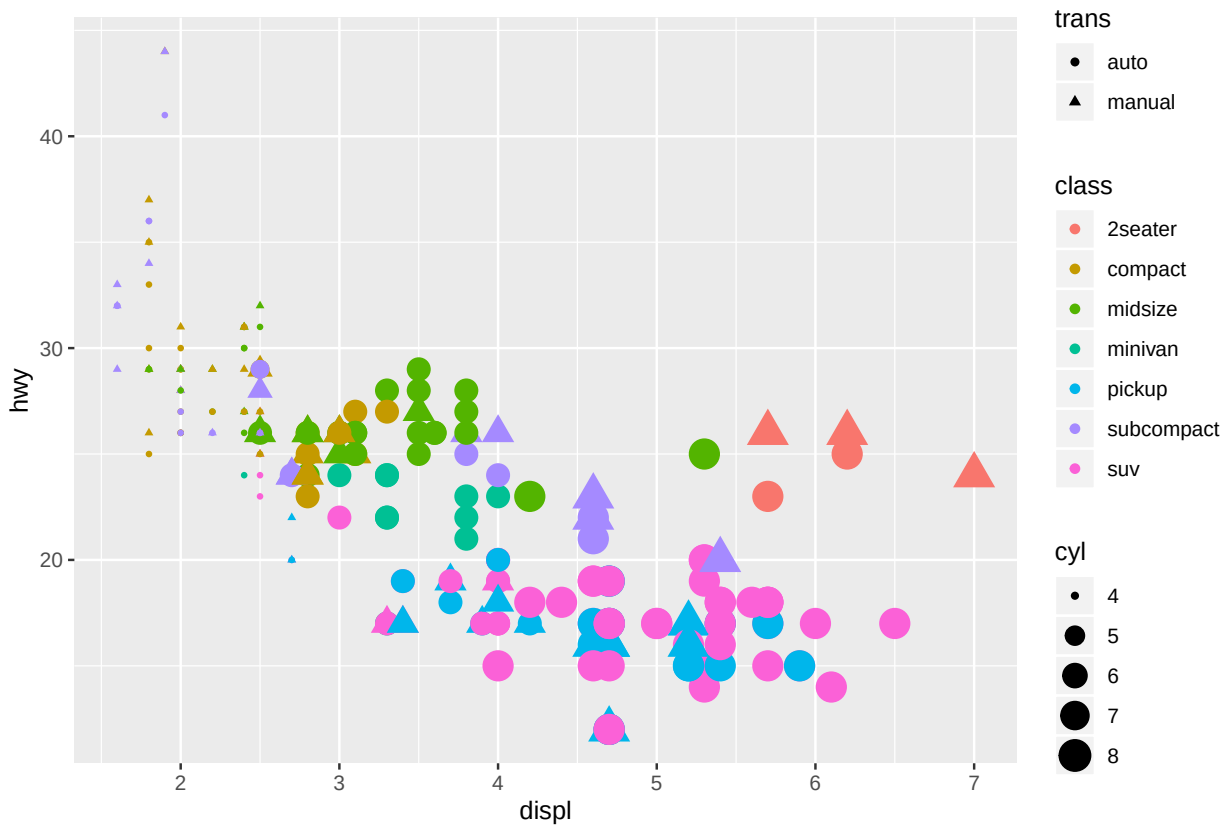
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```

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



Agregando valores estéticos podemos graficar hasta 5 variables en una sola gráfica.

```
mpg$trans <- str_replace( str_extract( mpg$trans, pattern = ".*[(]"), "[(]", "")
ggplot(mpg, aes(x = displ, y = hwy,
                color = class, shape = trans,
                size = cyl)) +
  geom_point()
```



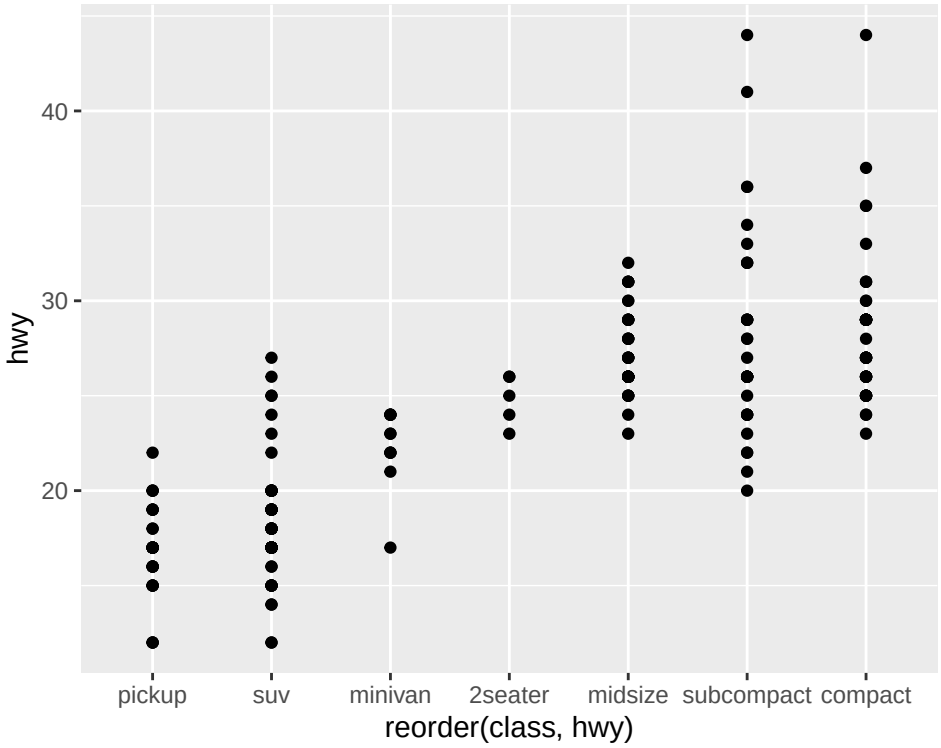
***Tipos de variables

El mapeo de las propiedades estéticas depende del tipo de variable, las variables discretas se mapean a distintas escalas que las variables continuas:

	Discreta	Continua
Color	Arcoiris de colores	Gradiente de colores
Tamaño	Escala discreta de tamaños	Mapeo lineal entre el radio y el valor
Forma	Distintas formas	No aplica

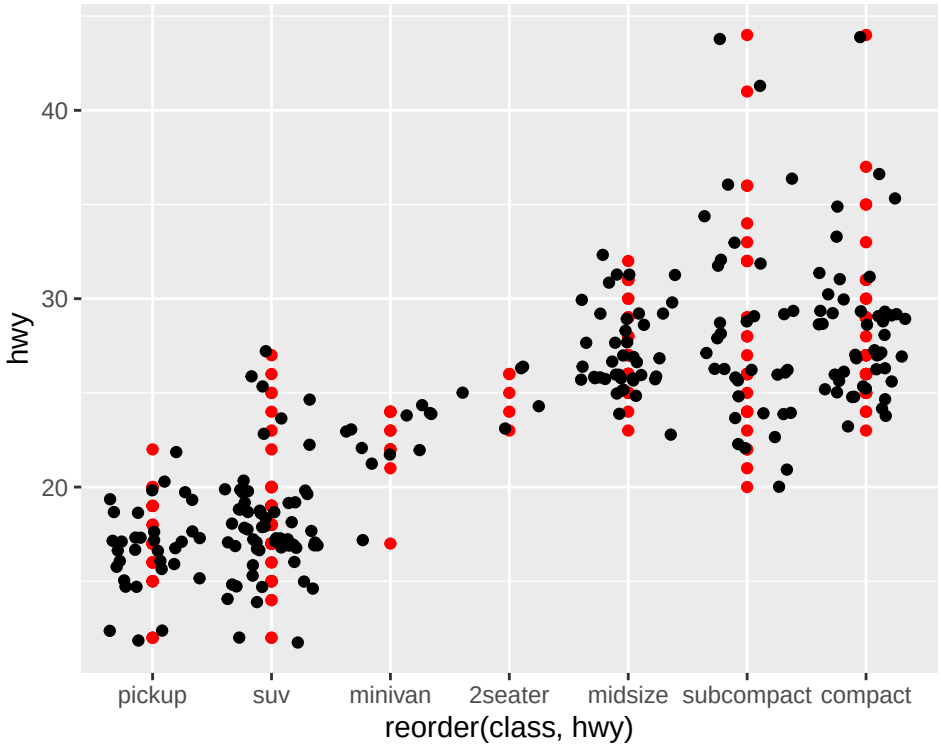
En la siguiente gráfica se muestra la variable categórica *class* reordenada en el eje x.

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



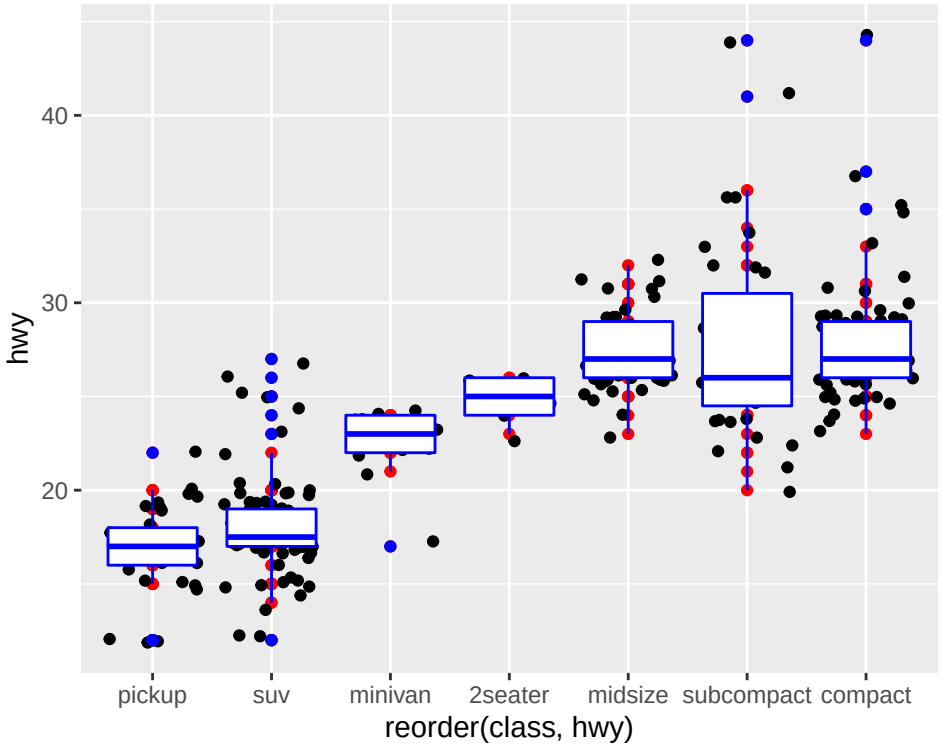
El problema con la gráfica es que no es muy informativa, por que no se observa variabilidad de los puntos. Una forma de corregirla es agregando *vibraciones* o *jitter*.

```
ggplot(mpg, aes(x = reorder(class, hwy), y = hwy)) +
  geom_point(color = 'red') +
  geom_jitter()
```



Se mencionó previamente que es posible graficar resúmenes estadísticos. En este ejemplo, agregaremos una capa más con un objeto `boxplot`.

```
ggplot(mpg, aes(x = reorder(class, hwy), y = hwy)) +
  geom_point(color = 'red') +
  geom_jitter() +
  geom_boxplot(color = 'blue')
```



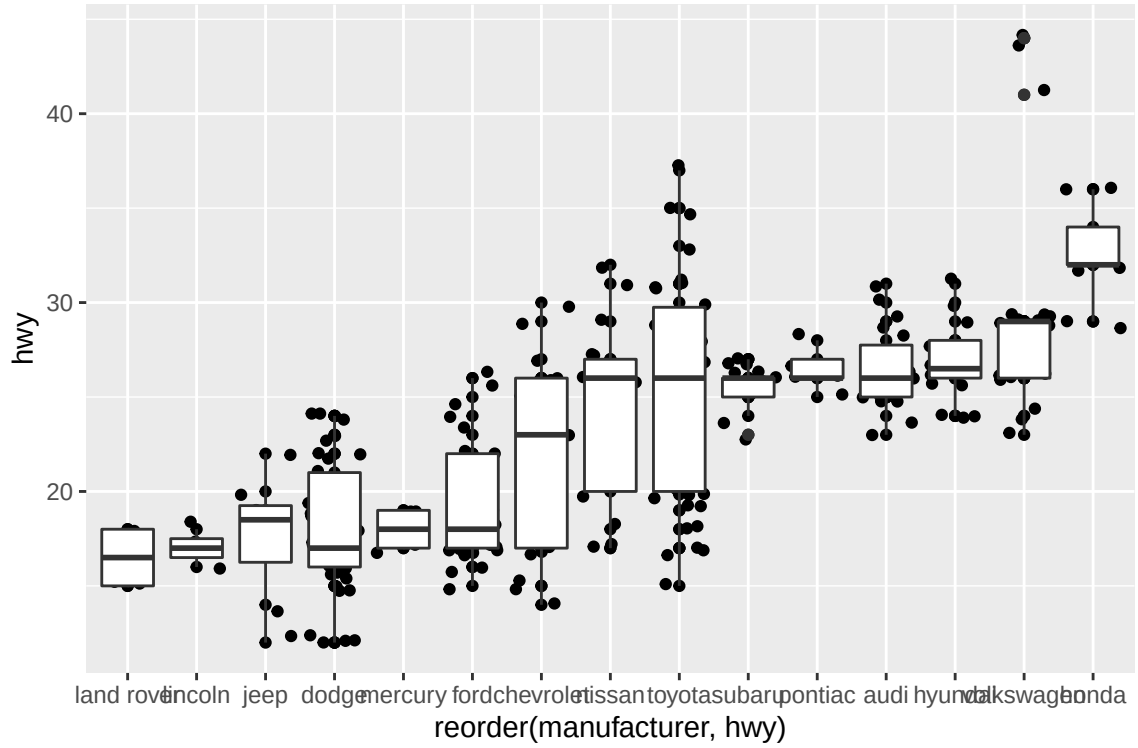
Nota que cada objeto de cada capa que se agrega tiene asignado un color diferente.

Modificar elementos

La gráfica tiene diferentes elementos, como: texto de los ejes, títulos de los ejes, títulos de las gráficas, leyendas de estéticas, etc. Una forma de modificar el tamaño, color, posición o rotación es mediante la función `theme()` y `element_text()`.

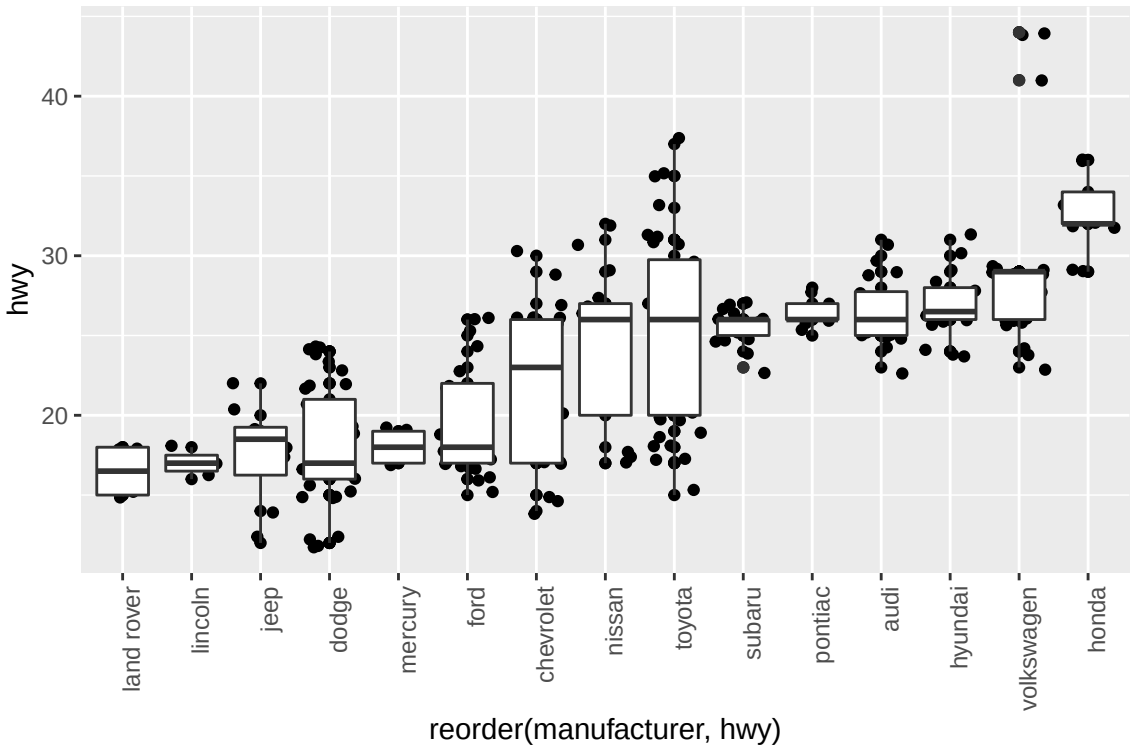
Por ejemplo, de la base **mpg** queremos observar el rendimiento de millas por galón en carretera **hwy** por cada fabricante **manufacturer**.

```
ggplot(mpg, aes(x = reorder(manufacturer, hwy), y = hwy)) +
  geom_point() +
  geom_jitter() +
  geom_boxplot()
```



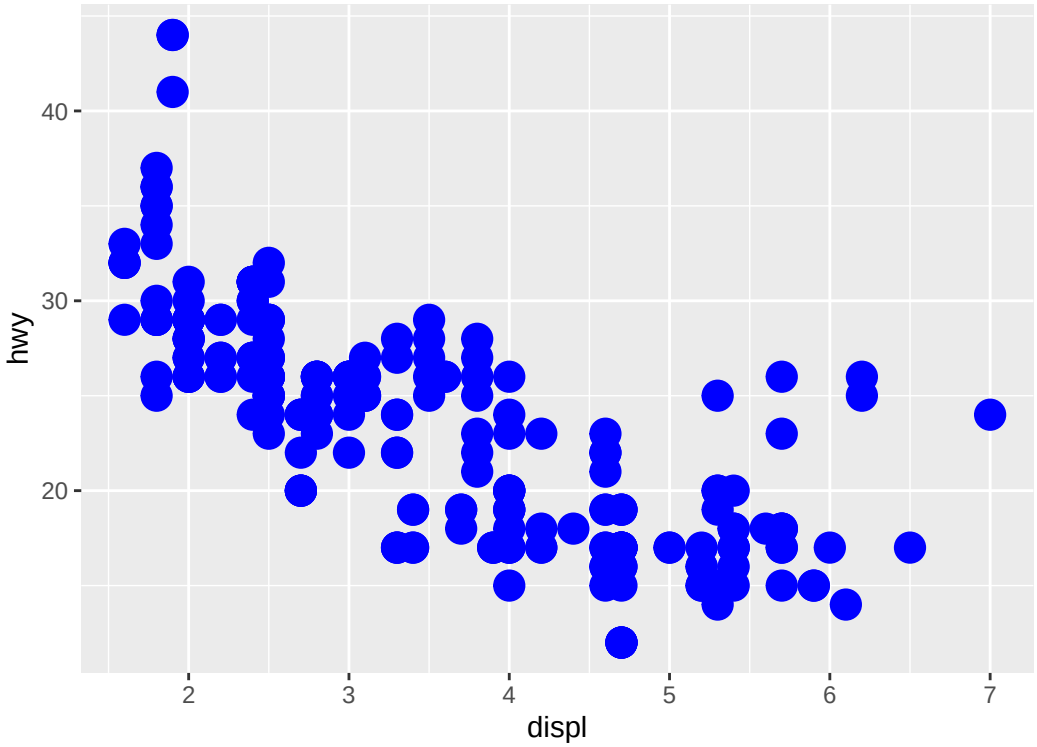
El problema con la gráfica anterior es la dificultad para leer los fabricante, por lo que se hace una rotación del texto del eje x.

```
ggplot(mpg, aes(x = reorder(manufacturer, hwy), y = hwy)) +
  geom_point() +
  geom_jitter() +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



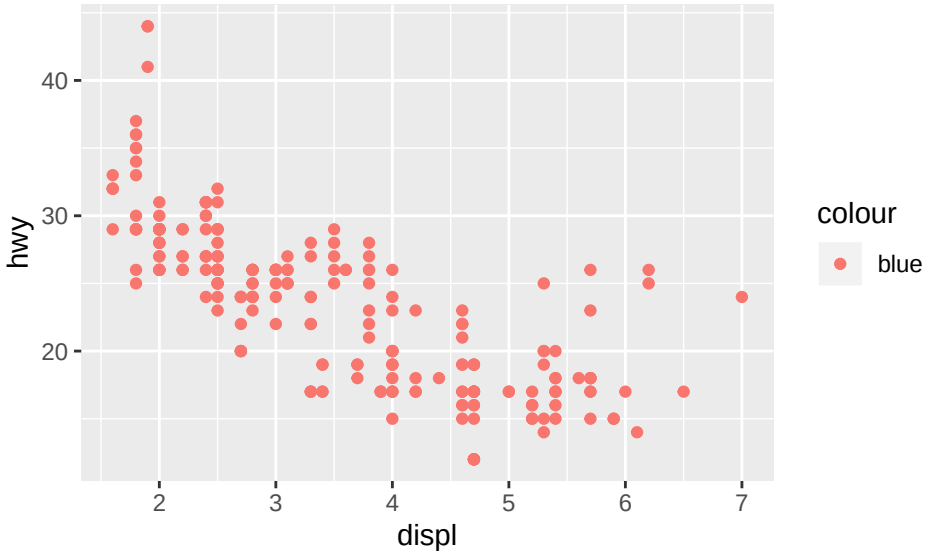
Dentro de cada **geom** es posible cambiar distintas características. Por ejemplo, el color y tamaño de cada punto de la gráfica de dispersión.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue", size = 5)
```



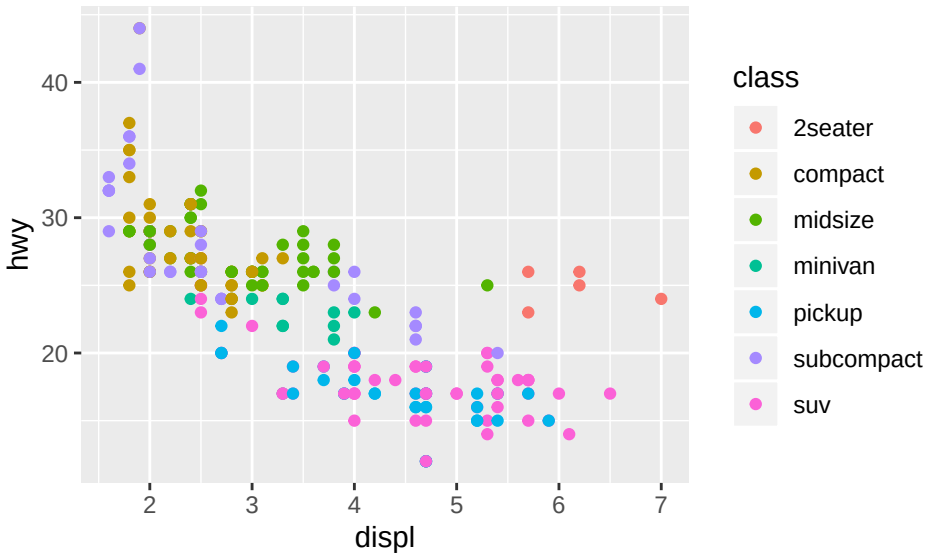
¿Por qué la siguiente gráfica no imprime los puntos en color azul?

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



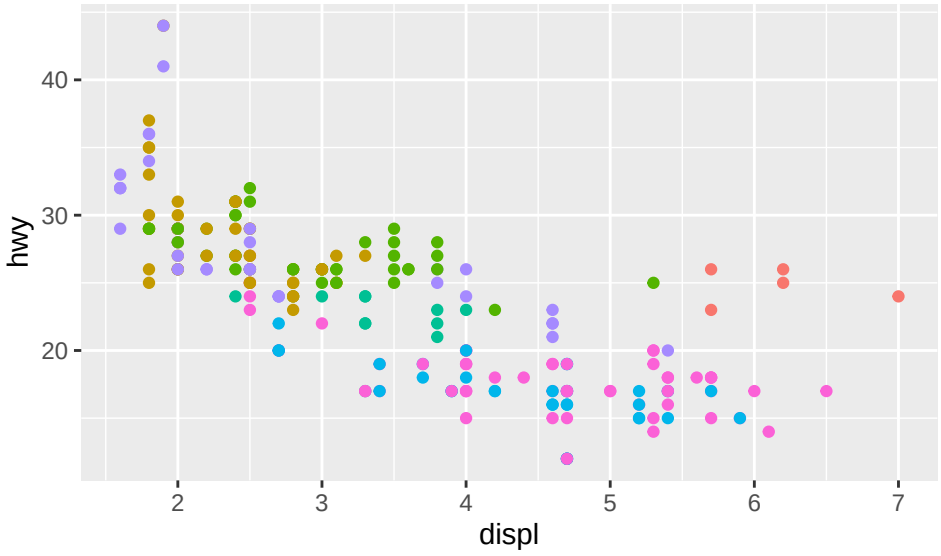
¿Qué sucede si en la siguiente gráfica agregas `legend.position = "none"` o `legend.position = "bottom"`.

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



¿Qué sucede si en el objeto geométrico de los puntos incluyes `show.legend = F`? No añade etiquetas

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class),show.legend = F) +
  theme()
```




```
#getwd()
setwd("C:/Users/dor31/Documents/Escuela/ITAM/CienciaDeDatos/Propedeutico/R_intro-master/R_intro-master/
# 1.
conapo <- read_csv('conapo.csv')
# 2.
str(conapo)
# 5.
tab.df <- conapo[conapo$CVE_ENT == '09', ]
ggplot(tab.df,
       aes(x = salarios_minimos, y = sin_electricidad)) +
```

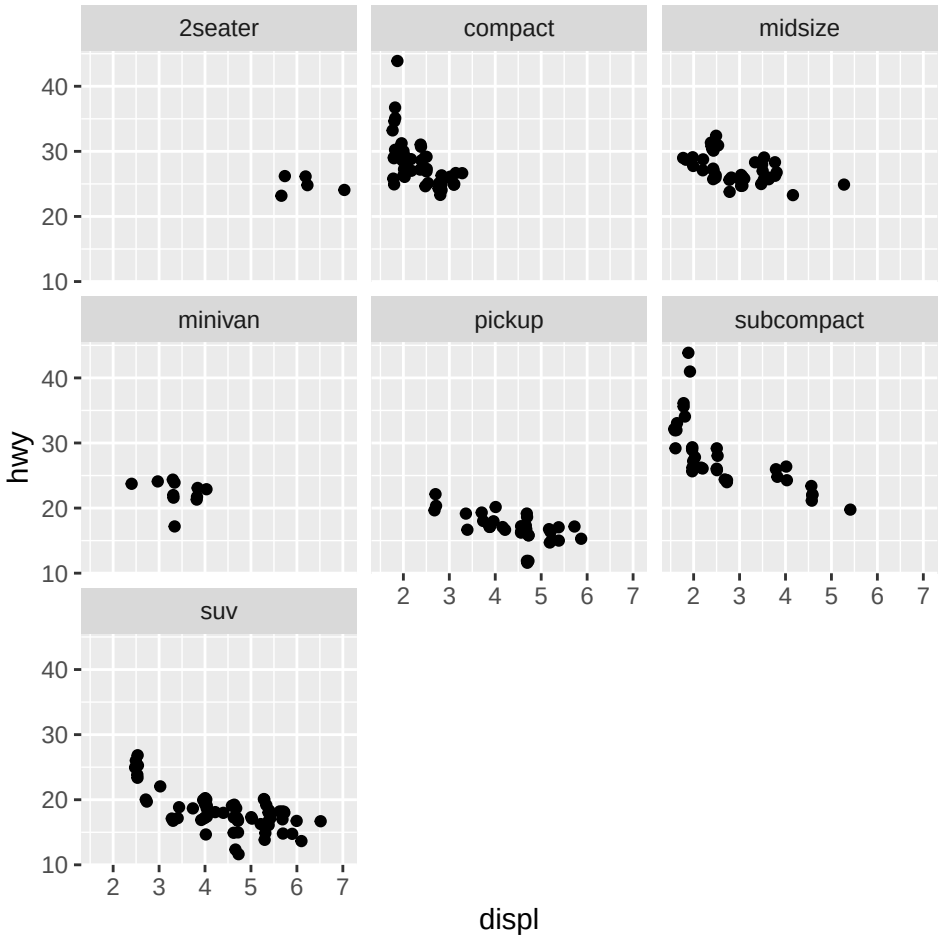
```
  geom_point( aes( color = analfabeta), size = 5) +
  geom_text(aes(label = NOM_MUN), check_overlap = T)
# 6.
ggplot(conapo, aes(x = reorder(NOM_ENT, analfabeta, median),
                   y = analfabeta)) +
  geom_point(size = .3) +
  geom_jitter(size = .2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
# 7.
ggplot(conapo, aes(x = reorder(NOM_ENT, analfabeta, median),
                   y = analfabeta)) +
  geom_boxplot() +
  geom_jitter(alpha = .2, size = .3) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab('Proporción de población analfabeta\nen el muncipio')+
  xlab('Entidad Federativa')
```

2. Gráficas de panel

El objetivo de las gráficas de panel es hacer varios múltiplos de una gráfica, donde cada múltiplo representa un subconjunto de los datos. Es una práctica muy útil para explorar relaciones condicionales.

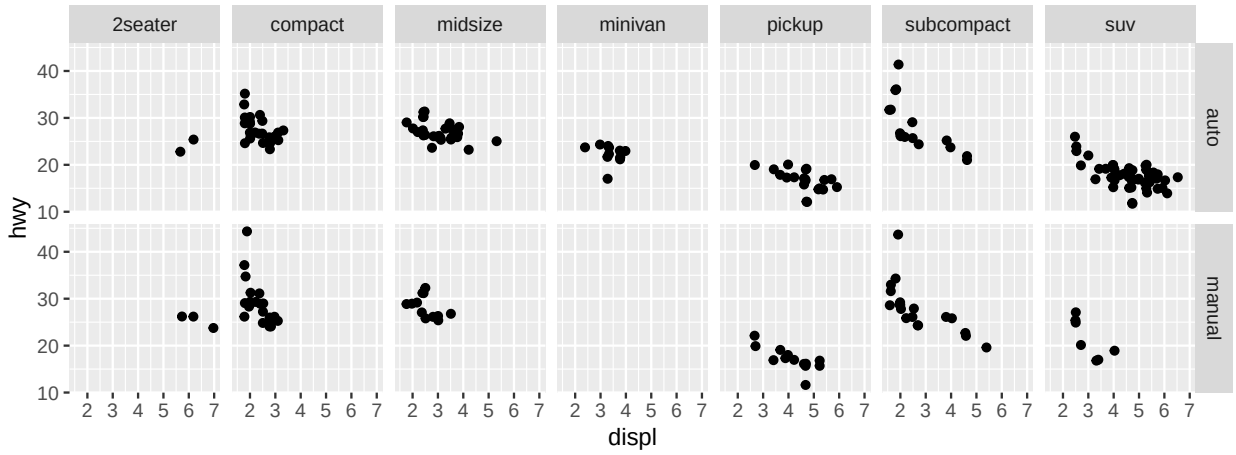
En ggplot una forma es usar la función `facet_wrap()` para hacer paneles dividiendo los datos de acuerdo a las categorías de una sola variable.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter() +
  facet_wrap(~ class)
```



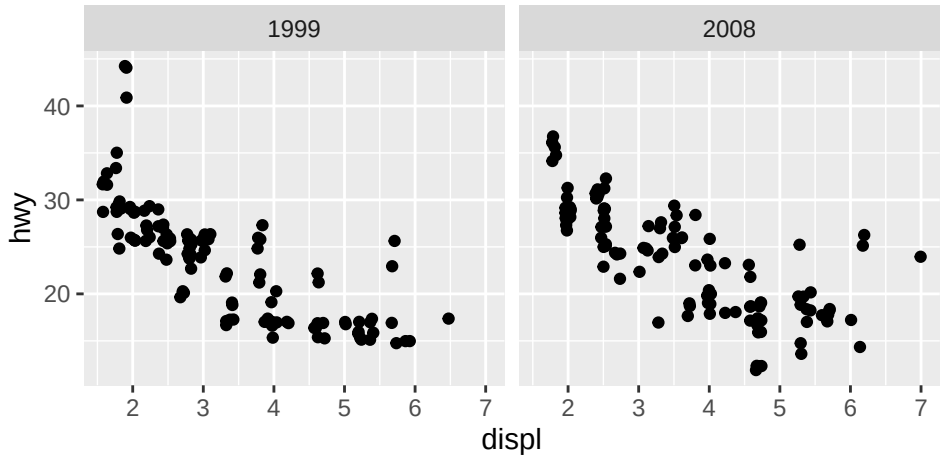
También podemos hacer una cuadrícula de 2 dimensiones usando `facet_grid(filas~columnas)`

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter() +
  facet_grid(trans~ class)
```



Ejercicio:

1. Recrea la siguiente gráfica. ¿Hay algún problema si la variable con la que generas los paneles es numérica?



2. ¿Qué gráfica se obtiene del siguientes código? ¿Qué hace `.`?

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter() +
  facet_grid(.~ class)
```

3. ¿Qué sucede si en el objeto de panel `facet_wrap()` de la siguiente gráfica agregas `scales = 'free_y'` y `nrow = 1`?

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter() +
  facet_wrap(~ class,scales = 'free_y',nrow = 1)
```

4. Lee los datos 'ingresos.csv'. Crea una variable con valor `TRUE` para los salarios que esten por arriba de la mediana y `FALSE` para los que no.
5. Grafica el género ordenado por la mediana de ingreso contra el ingreso y divide la gráfica por la variable binaria que creaste en el inciso 4.
6. Crea una variable de cortes de edad: menores o igual 35 años y mayores o igual a 35 años. (Tip: Usa la función `cut()`. Vale la pena revisar las funciones `cut_interval()` o `cut_number()`)
7. Repite la gráfica del inciso 5. pero ahora divide la gráfica por los rangos de edad que acabas de crear.

`Respuesta`:

```
setwd("C:/Users/dor31/Documents/Escuela/ITAM/CienciaDeDatos/Propedeutico/R_intro-master/R_intro-master/
# 1.
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter() +
  facet_wrap(~ year)
# 4.
ingresos <- read_csv("data/ingresos.csv")
ingresos$ingreso_med <- ingresos$ingreso > median(ingresos$ingreso)
# 5.
ggplot(ingresos,
       aes(x = reorder(genero, ingreso, median),
           y = ingreso)) +
  geom_point() +
```

```
  geom_jitter() +
  geom_boxplot()+
  facet_wrap(~ingreso_med) +
  xlab('Género') +
  ylab('Ingreso')
# 6.
ingresos$edad.cut <- cut(ingresos$edad, breaks = c(0,35,60))
# 7.
ggplot(ingresos,
       aes(x = reorder(genero, ingreso, median), y = ingreso)) +
  geom_point() +
  geom_jitter() +
  geom_boxplot()+
  facet_wrap(~edad.cut)+
  xlab('Género') +
  ylab('Ingreso')
```

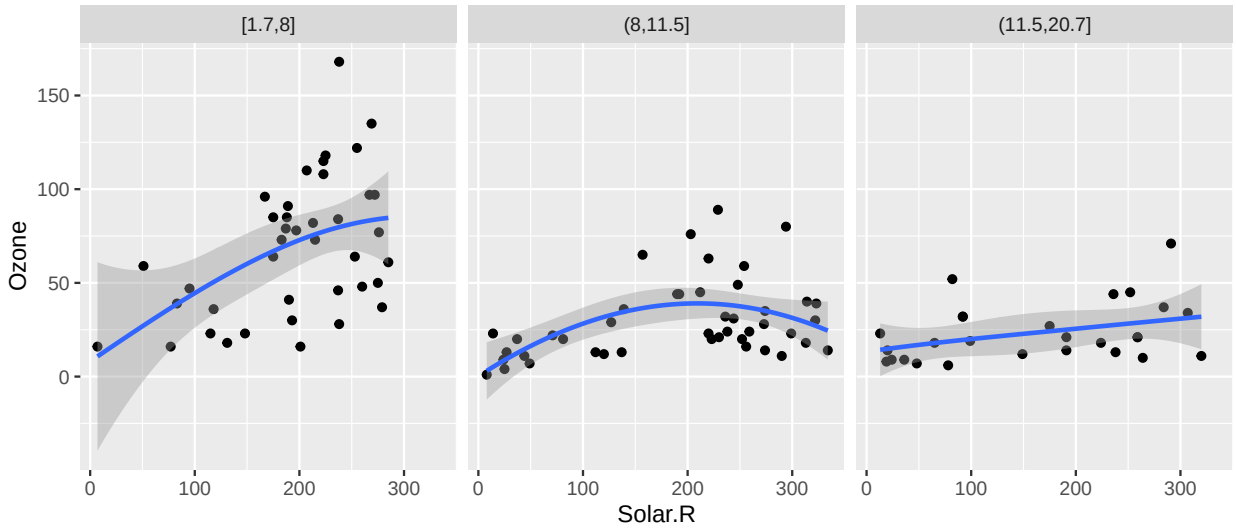
Otras Gráficas

***Suavizamientos

Al agregar suavizamientos se ajusta un modelo a los datos y en la grafica se imprimen las predicciones del modelo. La forma en que se agregan es con el objeto geométrico `geom_smooth()`.

En la siguiente gráfica para entender con mayor facilidad la gráfica de dispersión entre radiación solar y ozono se crea una nueva variable de la velocidad del viento y se agrega un suavizador (loess) por panel.

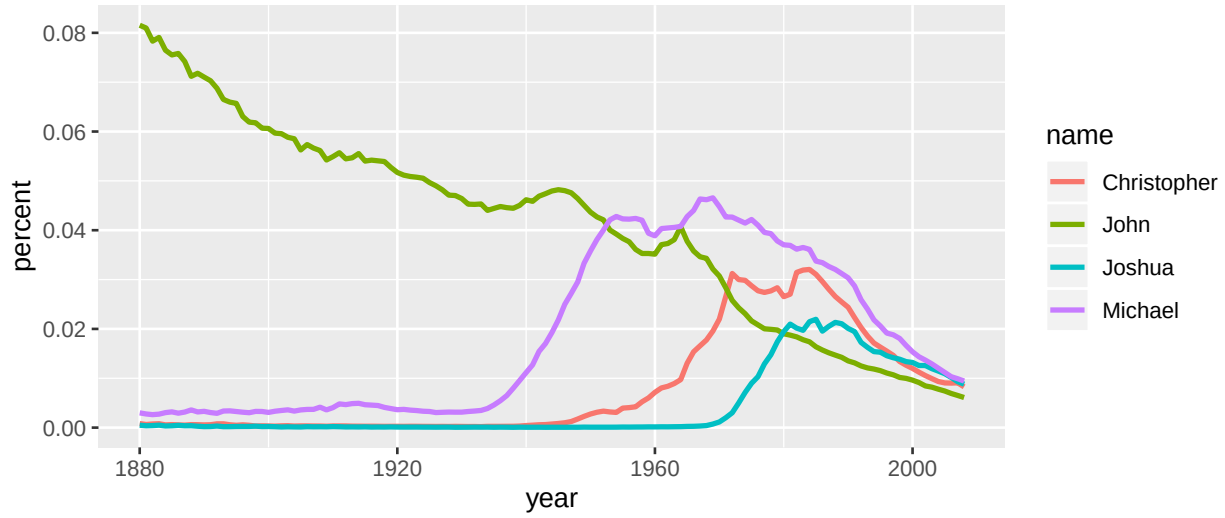
```
data(airquality)
airquality$Wind.cat <- cut_number(airquality$Wind, 3)
ggplot(airquality, aes(x = Solar.R, y = Ozone)) +
  geom_point() +
  facet_wrap(~ Wind.cat) +
  geom_smooth(span = 3)
```



***Series

En ocasiones se desea ver en alguna secuencia las variables. Supongamos que queremos ver la tendencia de los nombres *John*, *Michael*, *Joshua* y *Christopher* para niños. Para ello se genera un subconjunto de la base de datos.

```
setwd("C:/Users/dor31/Documents/Escuela/ITAM/CienciaDeDatos/Propedeutico/R_intro-master/R_intro-master/
bnames <- read_csv('data/bnames2.csv')
bnames$sex <- factor(bnames$sex, levels = 1:2, labels = c('boy', 'girl'))
sub.bnames <- bnames[bnames$name %in%
                       c("John", "Michael", "Joshua", "Christopher") &
                       bnames$sex == 'boy',]
ggplot(sub.bnames, aes(x = year, y = percent,
                       color = name, group = name)) +
  geom_line(size = 1)
```



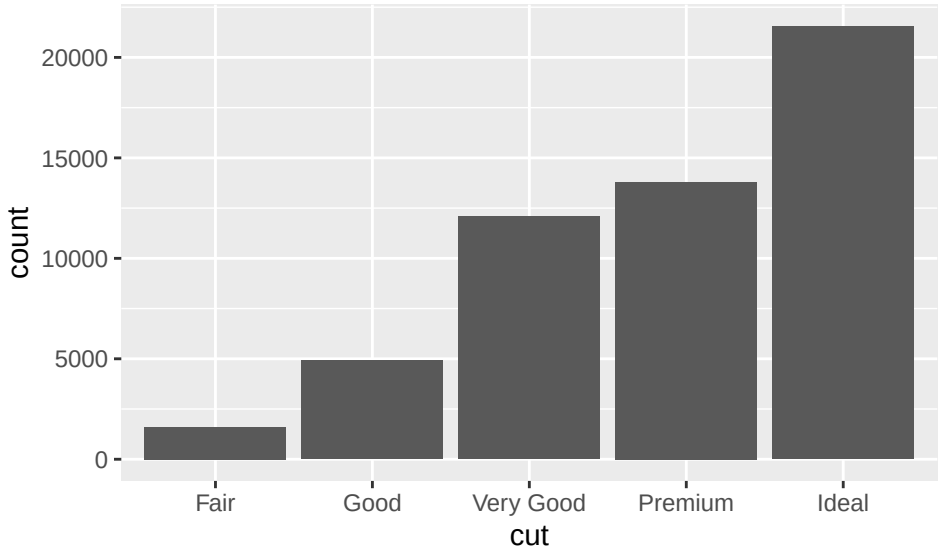
En esta gráfica se presenta el valor de estética `group` que permite agrupar los datos por nombre **name** y para cada nombre crea una línea con color y tipo (linetype) único.

***Barras

Las gráficas de barras se generan con el objeto `geom_bar()`. Este es uno de los objetos que hacen transformaciones estadísticas, en este caso en particular conteos.

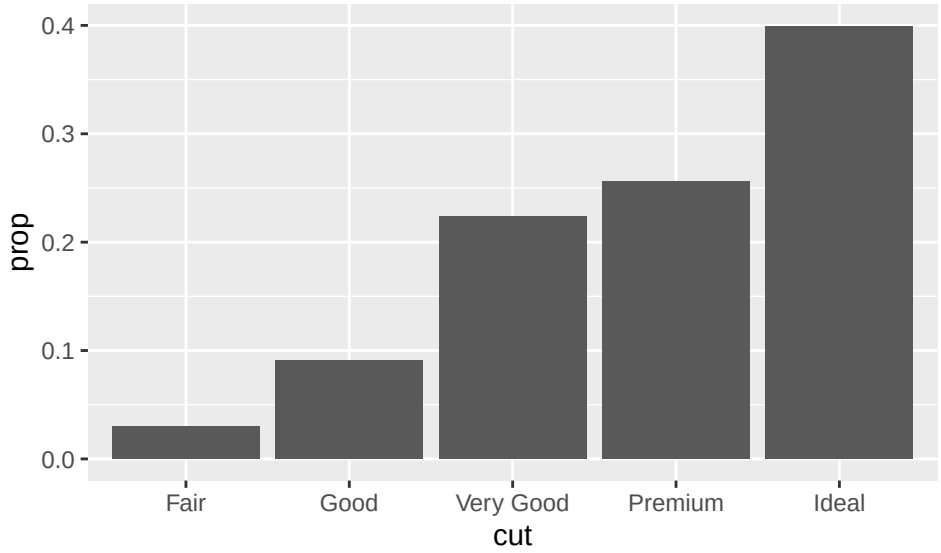
En la siguiente gráfica se muestra el conteo de diamantes por cada calidad de corte **cut**.

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



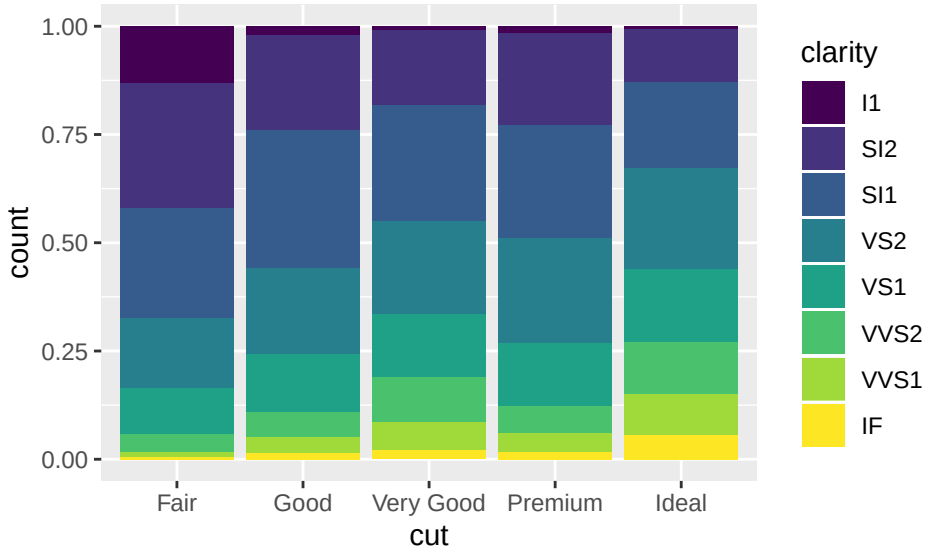
Es posible cambiar el valor estadístico de conteos, que es el estadístico default, a los conteos proporcionales con la especificación entre `..prop..` como se muestra abajo. Observar el eje y.

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



Ahora, también es posible obtener la distribución proporcional de cada medida de claridad del diamante por corte **cut**.

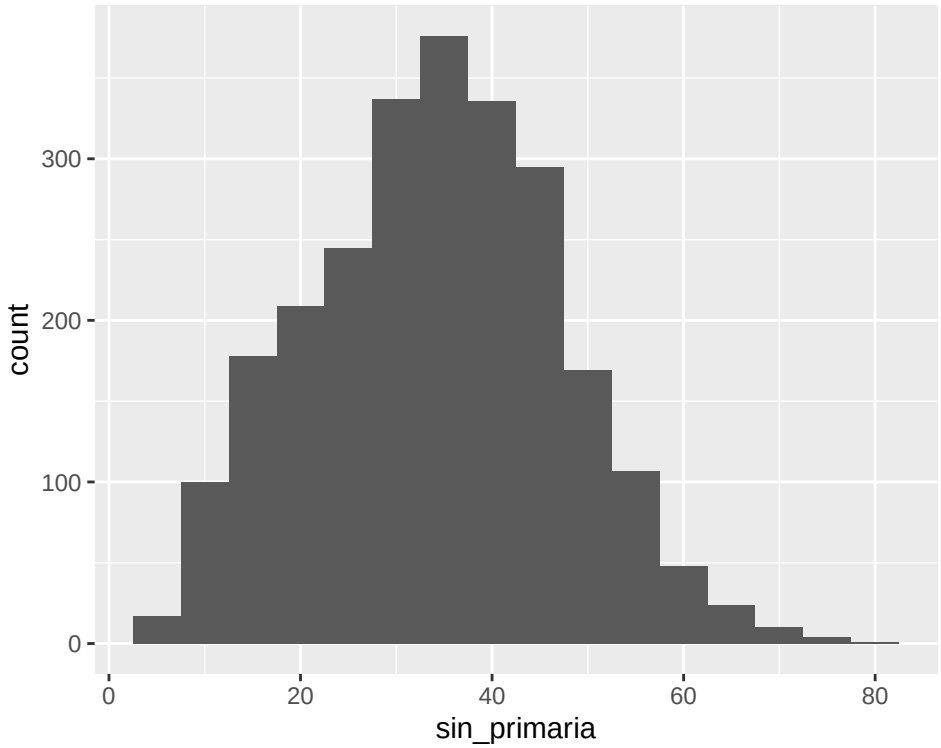
```
ggplot(data = diamonds,
       aes(x = cut, fill = clarity)) +
  geom_bar( position = "fill")
```



***Histogramas

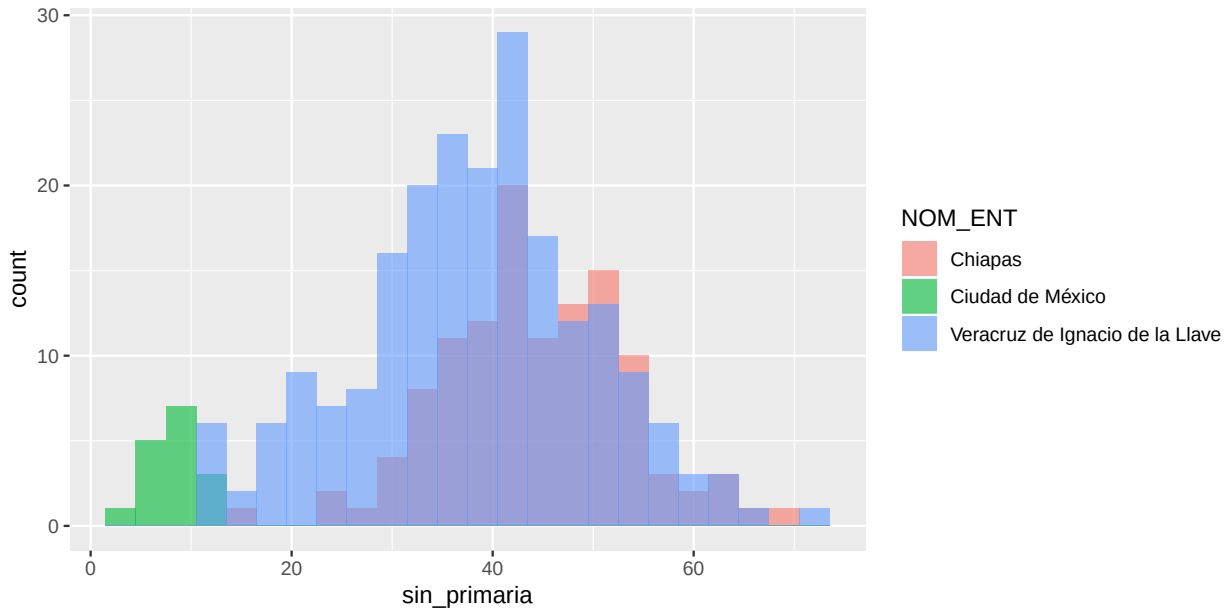
Los histogramas muestran la distribución de una variable numérica. El objeto geométrico para crear histogramas es `geom_histogram()`. La forma en que funciona este objeto es cortando la variable y cuenta el número de observaciones en cada corte. La forma de controlar el ancho de los cortes es con el argumento `binwidth`, es recomendable jugar con diferentes anchos de corte.

```
setwd("C:/Users/dor31/Documents/Escuela/ITAM/CienciaDeDatos/Propedeutico/R_intro-master/R_intro-master/
conapo <- read_csv('data/conapo.csv')
ggplot(data = conapo, aes( x= sin_primaria)) +
  geom_histogram(binwidth =  5)
```



También se pueden sobre poner distintas distribución dependiendo de variables categóricos.

```
sub <- conapo[conapo$CVE_ENT %in% c("09", "07", "30"),]
ggplot(data = sub, aes( x= sin_primaria, fill = NOM_ENT)) +
  geom_histogram(alpha = .6,
                 position = 'identity',
                 binwidth = 3)
```



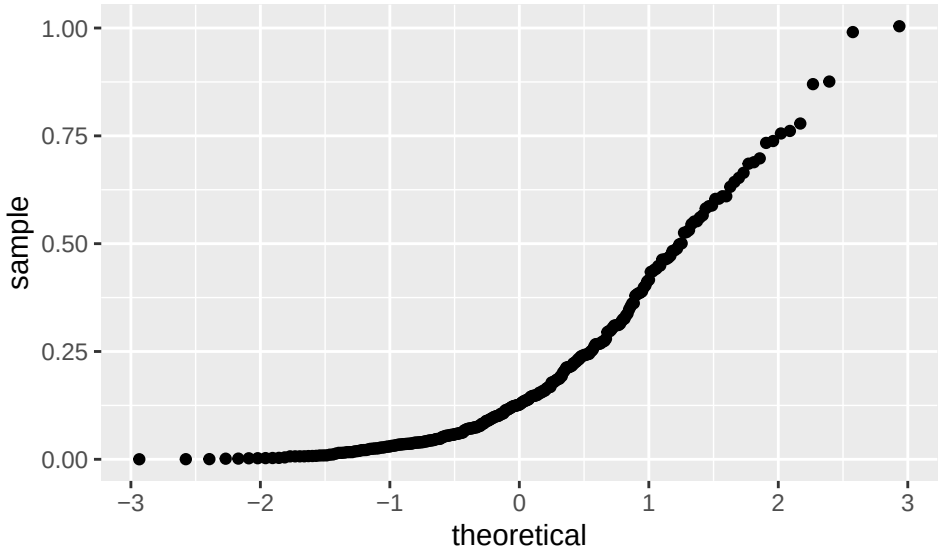
Nota: También se recomiendan los objetos `geom_density()` y `geom_freqpoly()` muestran distribuciones de variables.

***Información estadística

Los siguientes objetos resumen información estadística con distintas transformaciones. Es por esto que no son objetos geométricos, son objetos estadísticos y se llaman como `stat_`:

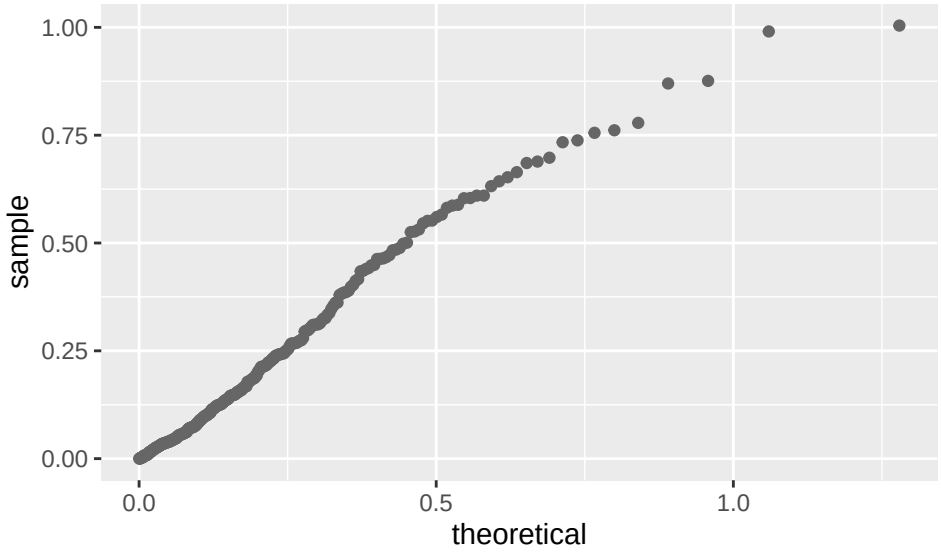
- **QQ-Plot:** `stat_qq` Realiza una gráfica cuantil-cuantil de una muestra y la distribución teórica. Por default compara los cuantiles teóricos de la distribución normal.

```
tab <-  data.frame(
  simulación = rexp(300, rate = 5)
)
ggplot(tab, aes(sample = simulación)) +
      stat_qq()
```



En el siguiente ejemplo se modifica para comparar con los cuantiles teóricos de la distribución exponencial.

```
ggplot(tab, aes(sample = simulación)) +
      stat_qq(distribution = qexp, dparams = 5,
              color = 'gray40')
```

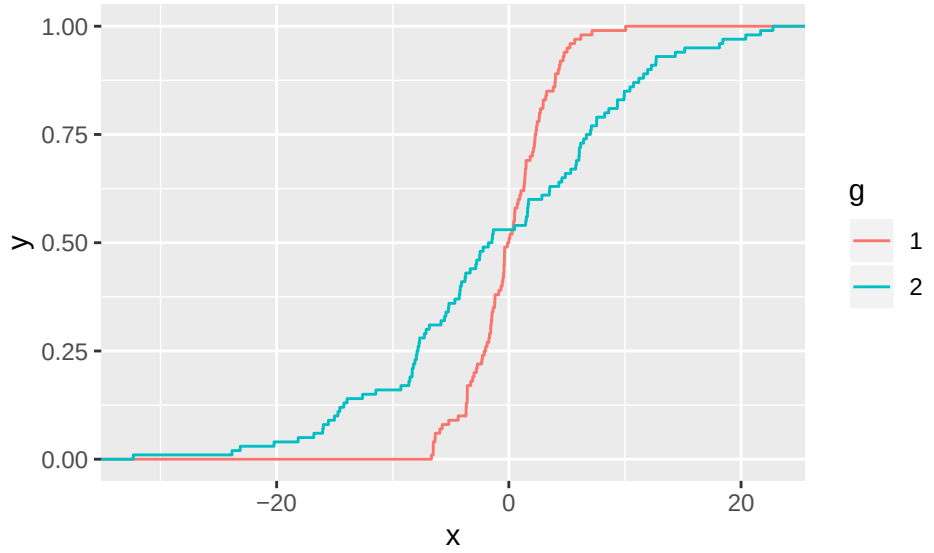


- **Distribución acumulada:** `stat_qq` Este objeto realiza las transformaciones necesarias para presentar la distribución acumulada de una muestra.

En la siguiente gráfica se compara la distribución de simulaciones normales con distintos parámetros.

```
df <- data.frame(x = c(rnorm(100, 0, 3),
                       rnorm(100, 0, 10)
                       ),
                 g = gl(2, 100))
```

```
ggplot(df, aes(x, colour = g)) +
  stat_ecdf()
```

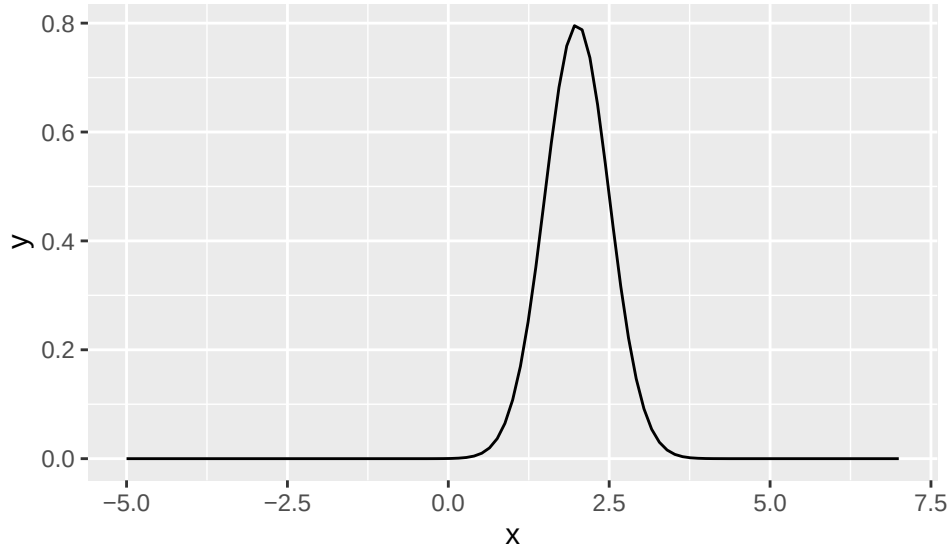


***Funciones

Este objeto permite graficar funciones en un rango determinado. El objeto estadístico es `stat_function()` que tiene como argumento una función.

La siguiente gráfica muestra una función predeterminada del objeto estadístico `dnorm`, que es la distribución normal con media 2 y desviación 0.5.

```
ggplot(data.frame(x = c(-5, 7)), aes(x)) +
  stat_function(fun = dnorm, args = list(mean = 2, sd = .5))
```



También es posible gráficar cualquier función, como en el siguiente ejemplo:

```r
FunSin <- function(x){
  sin(x)^2
}
ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = FunSin, size = 1, color = 'salmon')
```