Taller sobre el lenguaje R

Clase 4: Simulación y Regresión lineal

Lic. Lucio José Pantazis

April 07, 2021

Simulación

Probabilidades

- Se llama a un experimento aleatorio a un experimento cuyo resultado no podemos predecir, aunque podemos tener una idea de cuáles son los resultados posibles. Ejemplos: tiradas de monedas o dados, repartija de una mano de truco, resultados de un sorteo, etc.
- En estos casos, si bien no podemos predecir el resultado, podemos evaluar cuán frecuente es cada resultado posible, apelando a la cantidad de veces que ocurriría en una gran cantidad (n) de repeticiones del mismo experimento. Por ejemplo, si quisiéramos evaluar cuán frecuente es que salga una cara en una tirada de monedas, luego de n ensayos podríamos utilizar como medida la frecuencia relativa:

$$\frac{\#\text{ caras en } n \text{ tiradas}}{n} \approx P(\text{la moneda sale cara})$$

Simulación de probabilidades

- La simulación nos permite crear múltiples ocurrencias de un proceso cuyo comportamiento es desconocido. Por ejemplo, podríamos simular n=100 tiradas de una moneda con el comando sample:

```
n=100;
Tiradas=sample(c("cara","ceca"),size=100,replace = T);
head(Tiradas)
```

```
## [1] "ceca" "cara" "cara" "ceca" "ceca" "cara"
```

- Al tener a nuestra disposición muchas repeticiones del mismo experimento, podríamos evaluar la frecuencia relativa de la ocurrencia de una cara en la tirada de una moneda:

```
sum(Tiradas=="cara")/n
```

```
## [1] 0.52
```

- El resultado es similar a la probabilidad de ocurrencia: 0.5.

Tiradas de dados

- Otro ejemplo es la suma de dados al tirar dos dados.

```
n=100;
Resultados=matrix(sample(1:6,size = 2*n,replace = T),ncol = 2)
Resultados=as.data.frame(Resultados);
names(Resultados)=c("Tirada 1","Tirada 2")
Resultados$Suma=rowSums(Resultados);head(Resultados)
```

```
##   Tirada 1 Tirada 2 Suma
## 1        4        5    9
## 2        4        2    6
## 3        3        2    5
## 4        2        4    6
## 5        6        2    8
## 6        4        2    6
```

Tiradas de dados

Con estos resultados, podemos estimar la probabilidad de que la suma de los dados tome cada resultado posible entre 2 y 12:

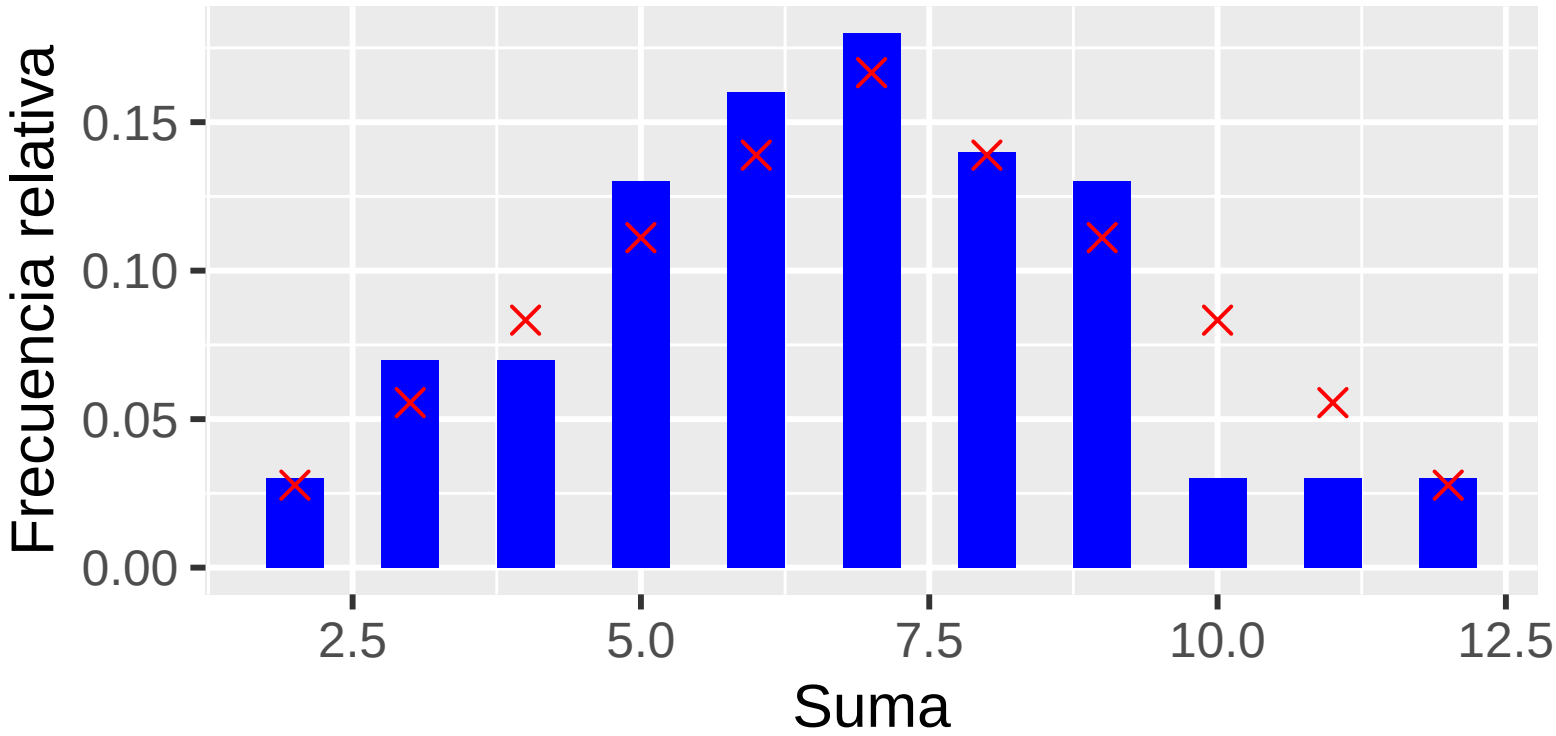
```
Freqs=Resultados %>%
  group_by(Suma) %>%
  summarise(rFreq=n()/n)
Freqs
```

```
## # A tibble: 11 x 2
##     Suma rFreq
##    <dbl> <dbl>
##  1     2  0.03
##  2     3  0.07
##  3     4  0.07
##  4     5  0.13
##  5     6  0.16
##  6     7  0.18
##  7     8  0.14
##  8     9  0.13
##  9    10  0.03
## 10    11  0.03
## 11    12  0.03
```

Tiradas de dados

A estos se les puede sumar la probabilidad de ocurrencia de cada suma:

```
Freqs$Probs=c(1:6,5:1)/36
Freqs %>%
  ggplot(aes(x=Suma))+
  geom_bar(aes(y=rFreq),fill="blue",stat="identity",width = 0.5)+
  geom_point(aes(y=Probs),color="red",shape=4,size=2)+
  ylab("Frecuencia relativa")
```



Mazo de cartas

Recordemos de la primera clase cuando armamos el Mazo, la función Mezcla y Repartir y Truco:

```
Mazo=data.frame(
  Palo=rep(c("Espada","Basto","Oro","Copa"),each=10),
  Numero=rep(c(1:7,10:12),4))
head(Mazo,n=3)
```

```
##     Palo Numero
## 1 Espada      1
## 2 Espada      2
## 3 Espada      3
```

```
Truco=function(Mazo,K,jug=0){
  Mez=Mezcla(Mazo)
  Rep=Repartir(Mez,K,jug)
  return(Rep)
}
Truco(Mazo,2)$Jugador1
```

```
##     Palo Numero
## 1   Copa     10
## 2 Espada      4
## 3  Basto      7
```

Mazo de cartas

Podemos repetir varias veces la mano de truco y por ejemplo, contar la cantidad de veces que sale una flor:

```
n=200;Flores=logical(0)
Flor=function(Mano){
  EsFlor=length(unique(Mano$Palo))==1
  return(EsFlor)
}
for(i in 1:n){
  Actual=Truco(Mazo,1)
  EsFlor=Flor(Actual$Jugador1)
  Flores=c(Flores,EsFlor)
}
sum(Flores)/n
```

```
## [1] 0.045
```

```
4*10*9*8/(40*39*38)
```

```
## [1] 0.048583
```

Mazo de cartas

Podemos repetir varias veces la mano de truco y por ejemplo, contar la cantidad de veces que sale una flor de Espadas:

```
n=200;FloresEsp=logical(0)
FlorEspadas=function(Mano){
  EsFlor=length(unique(Mano$Palo))==1
  if(EsFlor){
    EsFlorEsp=unique(Mano$Palo)=="Espada"
  }else{
    EsFlorEsp=FALSE
  }
  return(EsFlorEsp)
}
for(i in 1:n){
  Actual=Truco(Mazo,1)
  EsFlorEsp=FlorEspadas(Actual$Jugador1)
  FloresEsp=c(FloresEsp,EsFlorEsp)
}
sum(FloresEsp)/n
```

```
## [1] 0.015
```

```
10*9*8/(40*39*38)
```

```
## [1] 0.01214575
```

Sorteo

A "La Champions Liga" clasificaron a cuartos de final 2 equipos alemanes, 2 equipos españoles, 2 equipos franceses, un equipo inglés y un equipo italiano, dando lugar a la siguiente base de equipos:

```
Base=data.frame(Pais=c("Alemania","Alemania","España","España",
                       "Francia","Francia","Inglaterra","Italia"),
                Equipo=c("Bayern Munich","Leipzig",
                         "Atlético de Madrid","Barcelona",
                         "Lyon","Paris Saint Germain",
                         "Manchester city","Atalanta"))
Base
```

```
##         Pais                Equipo
## 1   Alemania        Bayern Munich
## 2   Alemania              Leipzig
## 3     España  Atlético de Madrid
## 4     España            Barcelona
## 5    Francia                 Lyon
## 6    Francia Paris Saint Germain
## 7 Inglaterra     Manchester city
## 8     Italia             Atalanta
```

Sorteo

Podemos usar la función Mezcla para determinar el sorteo de los partidos, agregando a qué partido y llave pertenecen, y si definen de local o visitante:

```
Sorteo=function(Base){
  BaseSort=Mezcla(Base)
  BaseSort$Llave=rep(c("Llave1","Llave2"),each=4)
  BaseSort$Partido=rep(c("Partido1","Partido2",
                         "Partido3","Partido4"),each=2)
  BaseSort$Define=rep(c("Local","Visitante"),4)
  return(BaseSort)
}
Llaves=Sorteo(Base)
head(Llaves,n=4)
```

```
##         Pais          Equipo  Llave  Partido    Define
## 7 Inglaterra Manchester city Llave1 Partido1     Local
## 1   Alemania   Bayern Munich Llave1 Partido1 Visitante
## 8     Italia        Atalanta Llave1 Partido2     Local
## 4     España       Barcelona Llave1 Partido2 Visitante
```

Sorteo

Aquí vemos el sorteo completo:

```
Llaves
```

```
##         Pais                Equipo  Llave  Partido    Define
## 7 Inglaterra     Manchester city Llave1 Partido1     Local
## 1   Alemania       Bayern Munich Llave1 Partido1 Visitante
## 8     Italia             Atalanta Llave1 Partido2     Local
## 4     España            Barcelona Llave1 Partido2 Visitante
## 2   Alemania              Leipzig Llave2 Partido3     Local
## 3     España  Atlético de Madrid Llave2 Partido3 Visitante
## 6    Francia Paris Saint Germain Llave2 Partido4     Local
## 5    Francia                Lyon Llave2 Partido4 Visitante
```

Sorteo

Los sorteos deportivos llevan a muchas especulaciones, y siempre están los análisis de las posibilidades pero nunca de las probabilidades. Sin embargo, calcular las probabilidades de cosas muy específicas no siempre es fácil y podríamos usar la simulación para ver, por ejemplo, qué probabilidad hay de que los dos equipos españoles se enfrenten entre sí:

```
EnfEsp=function(Base){
  BaseEsp=Base %>%
    filter(Pais=="España")
  Enf=length(unique(BaseEsp$Partido))==1
  return(Enf)
}
n=100;EspEnfs=logical(0);
for(i in 1:n){
  SortActual=Sorteo(Base)
  EsEnfEsp=EnfEsp(SortActual)
  EspEnfs=c(EspEnfs,EsEnfEsp)
}
sum(EspEnfs)/n
```

```
## [1] 0.13
```

```
1/7
```

```
## [1] 0.1428571
```

Sorteo

Podemos ver probabilidades mucho más complejas, como la probabilidad simultánea de que los españoles se enfrenten y los alemanes NO se enfrenten, los franceses pertenezcan a distintas llaves y que el Atalanta defina de local, usando las siguientes funciones:

```
NoEnfAle=function(Base){
  BaseAle=Base %>%
    filter(Pais=="Alemania")
  NoEnf=length(unique(BaseAle$Partido))>1
  return(NoEnf)
}
FranLlDist=function(Base){
  BaseFr=Base %>%
    filter(Pais=="Francia")
  SoloFin=length(unique(BaseFr$Llave))>1
  return(SoloFin)
}
AtaLoc=function(Base){
  BaseAta=Base %>%
    filter(Equipo=="Atalanta")
  AtaLocal=BaseAta$Define=="Local"
  return(AtaLocal)
}
```

Sorteo

Para estimar la probabilidad, corremos una simulación:

```
n=1000;CumpleConds=logical(0);
for(i in 1:n){
  SortActual=Sorteo(Base)
  EsEnfEsp=EnfEsp(SortActual)
  EsNoEnfAle=NoEnfAle(SortActual)
  EsFranLlDist=FranLlDist(SortActual)
  EsAtaLoc=AtaLoc(SortActual)
  ActCond=EsEnfEsp & EsNoEnfAle & EsFranLlDist & EsAtaLoc
  CumpleConds=c(CumpleConds,ActCond)
}
sum(CumpleConds)/n
```

```
## [1] 0.036
```

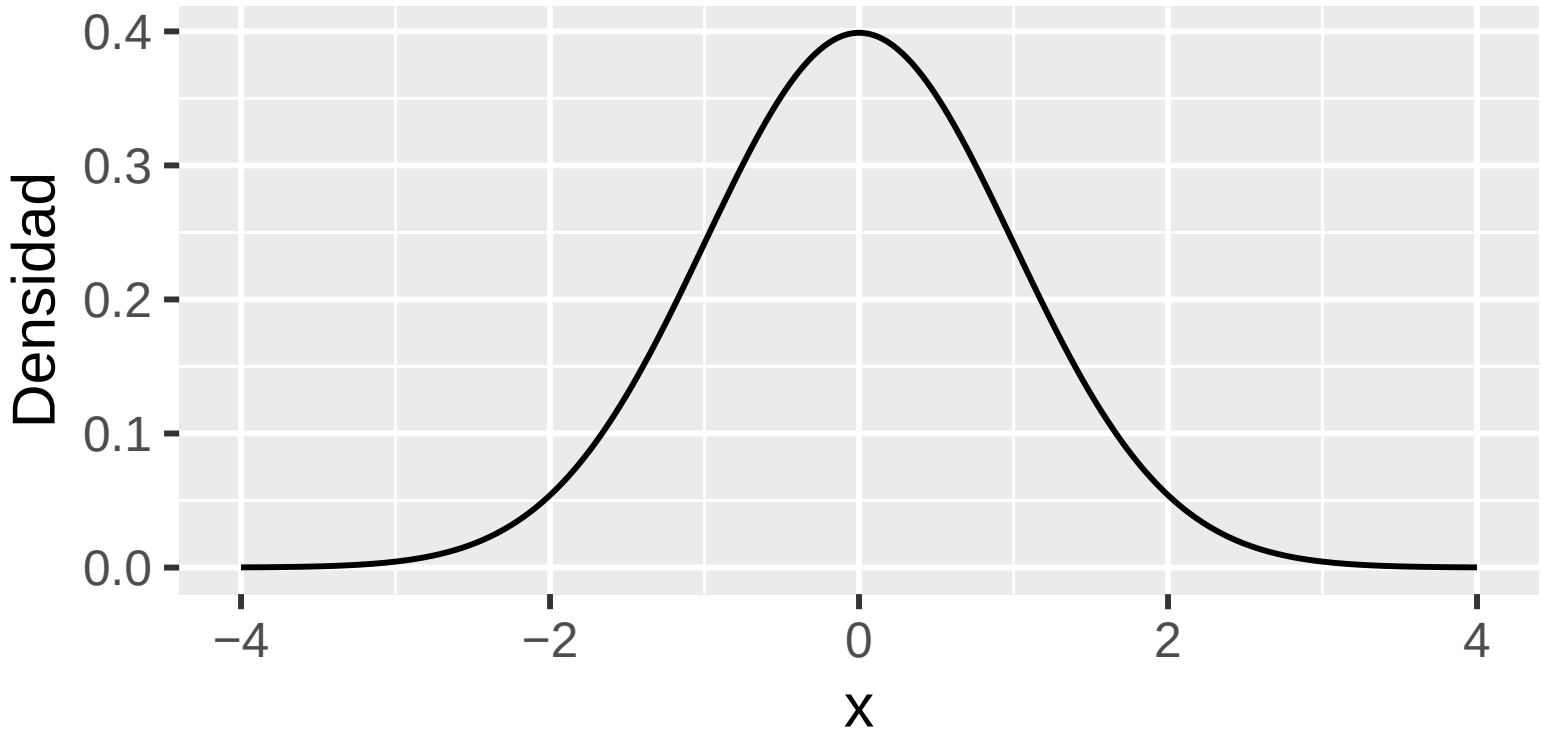
Distribuciones

Distribución normal

A veces las variables cuantitativas tienen una estructura respecto a los valores que toma y la frecuencia con las que lo hacen.

Una de las más famosas en la estadística es la distribución normal, que tiene forma acampanada:

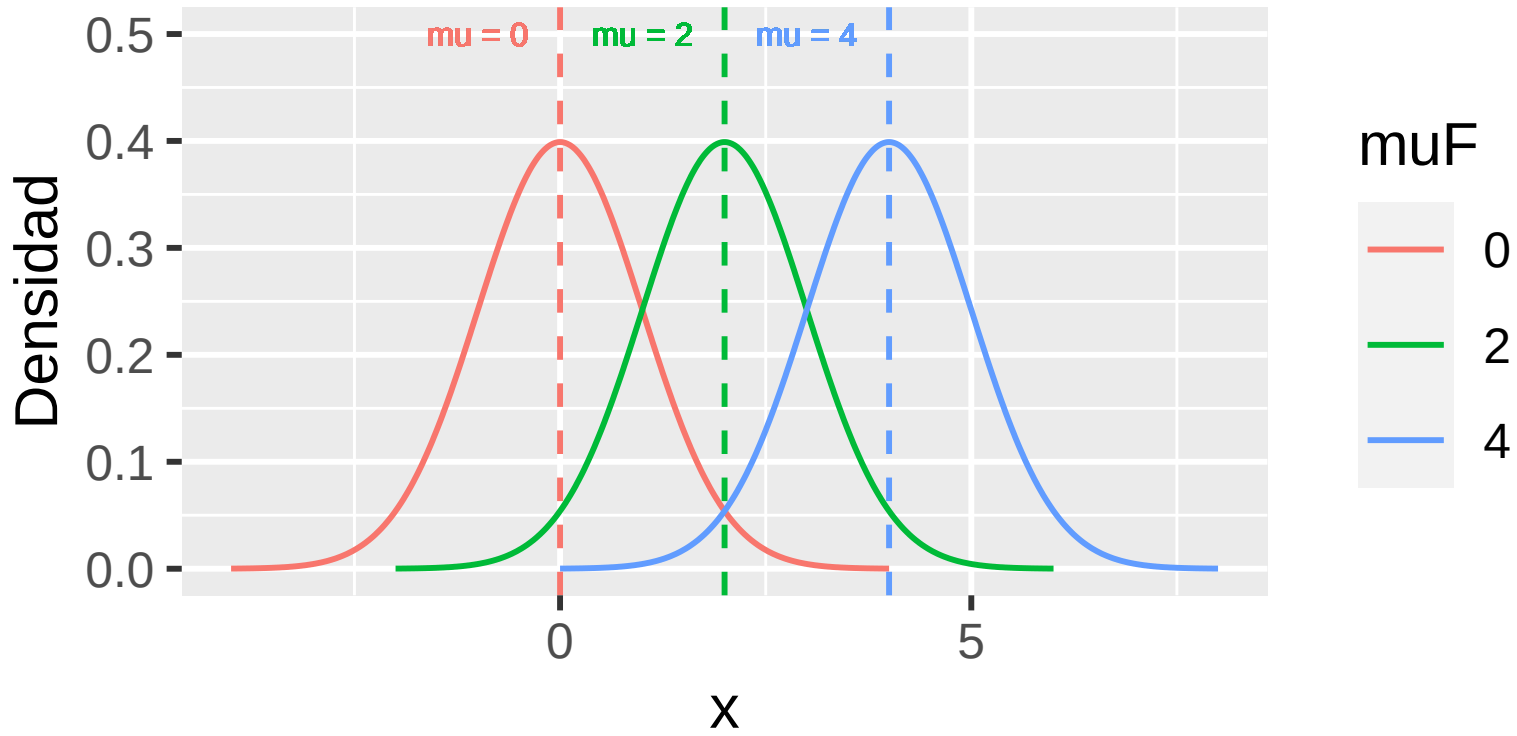
```
D=data.frame(x=seq(-4,4,by=0.01))
D$f=dnorm(D$x)
D %>%
  ggplot(aes(x=x,y=f))+
  geom_line()+ylab("Densidad")
```



Distribución normal

La distribución normal tiene una media μ (llamada por el R como "mean", default=0) que marca el centro de los datos:

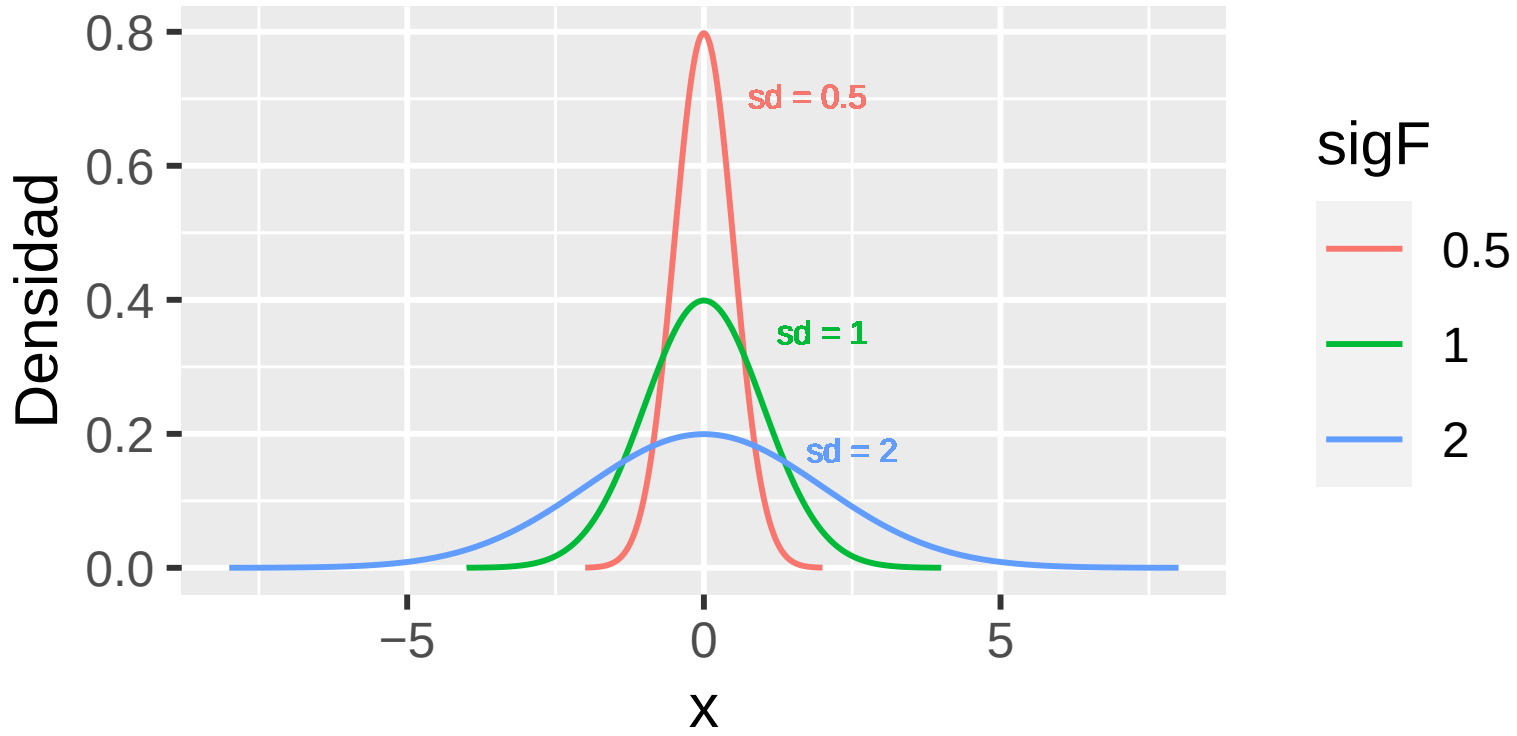
```
dx=seq(-4,4,by=0.01)
D=data.frame(mu=rep(2*(0:2),each=801))
D$x=dx+D$mu;D$f=dnorm(D$x,mean=D$mu);D$muF=factor(D$mu)
D %>% ggplot(aes(x=x,y=f,color=muF))+geom_line()+ylab("Densidad")+
  geom_vline(aes(xintercept=mu,color=muF),linetype="dashed",show.legend = F)
  geom_text(aes(x=mu,y=0.5,color=muF,label=paste("mu =",mu)),show.legend = F
```



Distribución normal

La distribución normal tiene un desvío standard σ (llamada por el R como "sd", default=1) que marca la variabilidad de los datos respecto de la media:

```
dx=seq(-4,4,by=0.01)
D=data.frame(sig=rep(c(0.5,1,2),each=801))
D$x=dx*D$sig;D$f=dnorm(D$x,sd=D$sig);D$sigF=factor(D$sig)
D %>% ggplot(aes(x=x,y=f,color=sigF))+geom_line()+ylab("Densidad")+
  geom_text(aes(x=sig/2,y=dnorm(sig/2,sd=sig),color=sigF,label=paste("sd =",
```



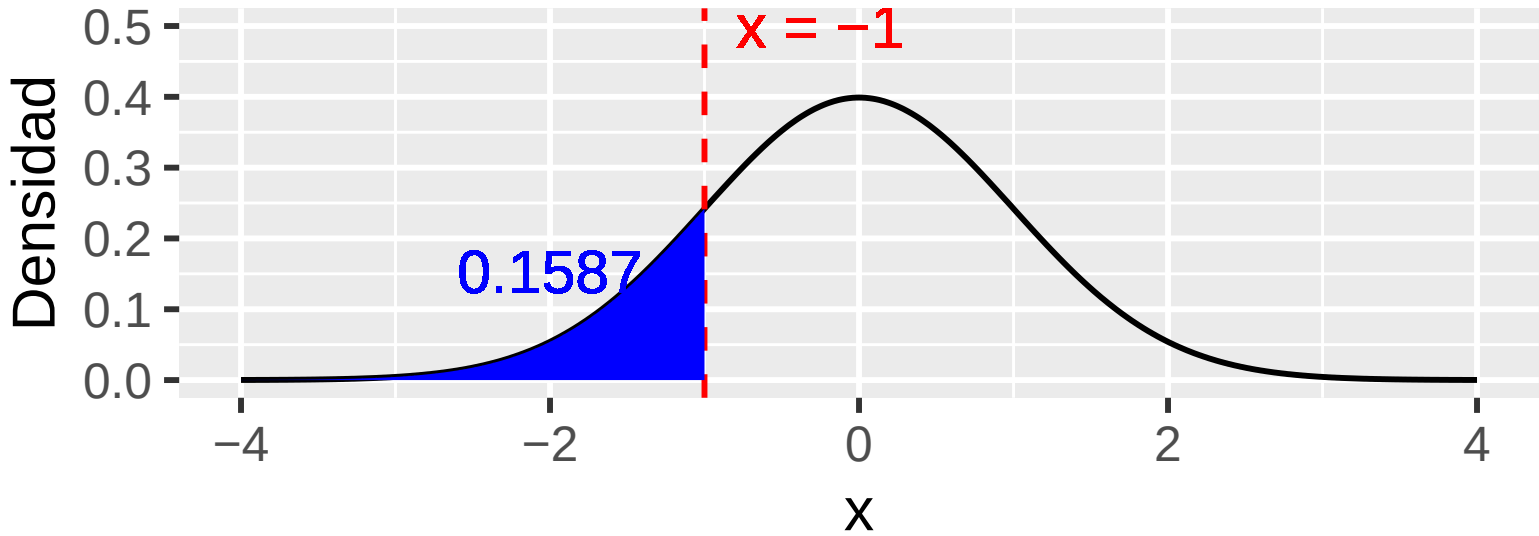
Distribución normal

Para cualquier distribución (en este caso la normal), el comando "p____(x)" (en este caso, "pnorm") calcula la probabilidad de que esa variable tome un valor menor o igual a x. Tomemos por ejemplo, x=-1:

```
Pmin1=pnorm(-1);Pmin1
```

```
## [1] 0.1586553
```

```
D=data.frame(x=seq(-4,4,by=0.01));D$f=dnorm(D$x);SubD=D %>% filter(x<=-1)
D %>% ggplot(aes(x=x,y=f))+geom_line()+ ylab("Densidad")+
  geom_vline(aes(xintercept=-1),color="red",linetype="dashed")+
  geom_area(data = SubD,aes(x=x,y=f),fill="blue")+
  geom_text(aes(x=-2,y=dnorm(-2),label=signif(Pmin1,4)),color="blue",nudge_y
  geom_text(aes(x=-1,y=0.5,label="x = -1"),color="red",nudge_x = 0.75)
```



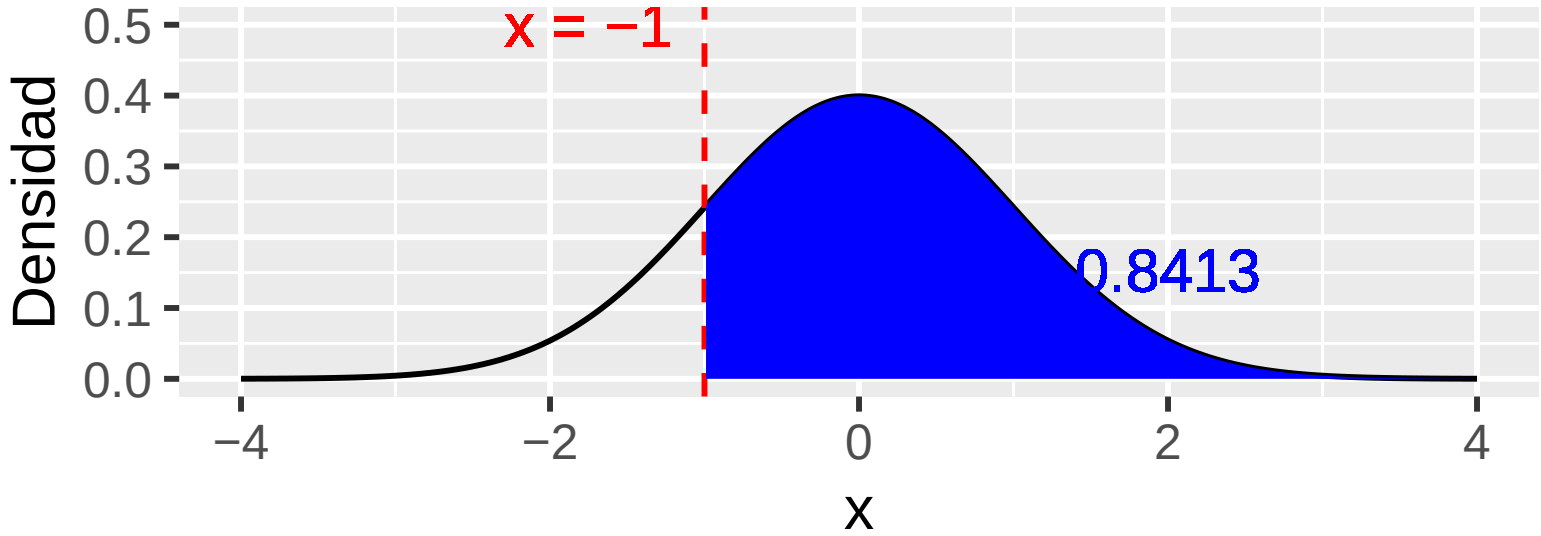
Distribución normal

En caso de querer la probabilidad de que la variable tome un valor mayor a x, se agrega lower.tail=F:

```
Pmin1=pnorm(-1,lower.tail = F);Pmin1
```

```
## [1] 0.8413447
```

```
D=data.frame(x=seq(-4,4,by=0.01));D$f=dnorm(D$x);SubD=D %>% filter(x>-1)
D %>% ggplot(aes(x=x,y=f))+geom_line()+ ylab("Densidad")+
  geom_vline(aes(xintercept=-1),color="red",linetype="dashed")+
  geom_area(data = SubD,aes(x=x,y=f),fill="blue")+
  geom_text(aes(x=2,y=dnorm(2),label=signif(Pmin1,4)),color="blue",nudge_y =
  geom_text(aes(x=-1,y=0.5,label="x = -1"),color="red",nudge_x = -0.75)
```



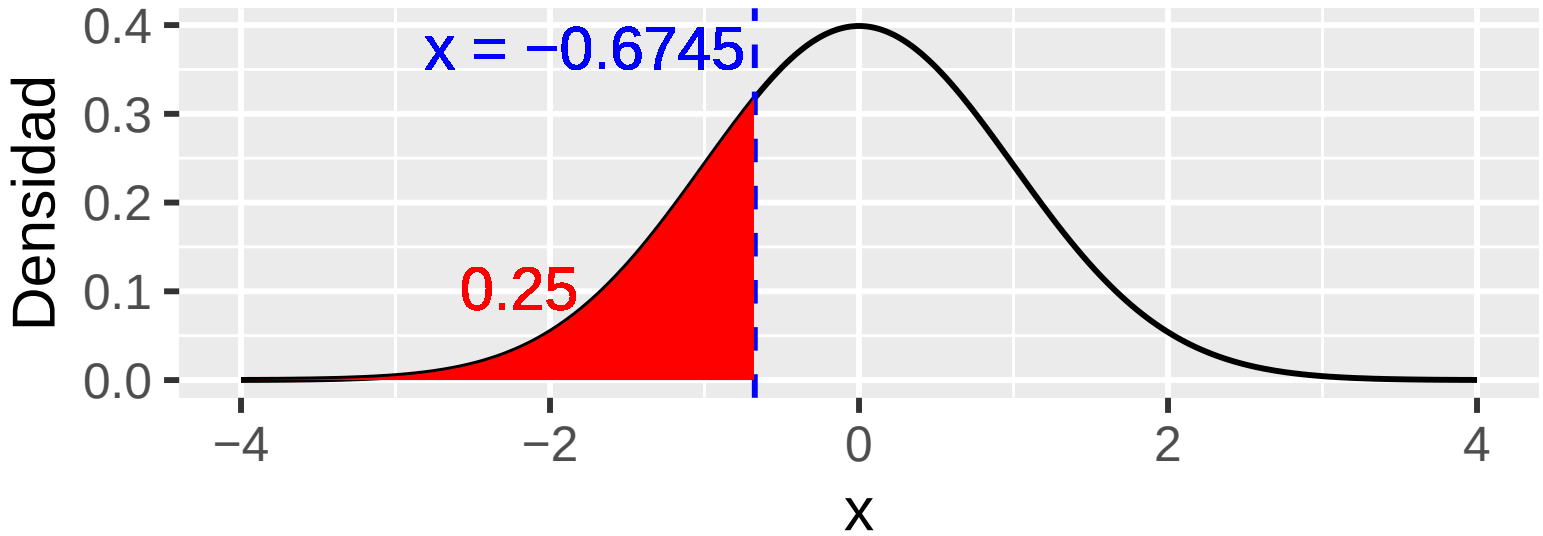
Distribución normal

Para cualquier distribución (en este caso la normal), el comando "q____(a)" (en este caso, "qnorm") calcula el valor de la variable para el cual el área encerrada es a. Tomemos por ejemplo, a=0.25:

```
Q1=qnorm(0.25);Q1
```

```
## [1] -0.6744898
```

```
D=data.frame(x=seq(-4,4,by=0.01));D$f=dnorm(D$x);SubD=D %>% filter(x<=Q1)
D %>% ggplot(aes(x=x,y=f))+geom_line()+ ylab("Densidad")+
  geom_vline(aes(xintercept=Q1),color="blue",linetype="dashed")+
  geom_area(data = SubD,aes(x=x,y=f),fill="red")+
  geom_text(aes(x=Q1,y=0.375,label=paste("x =",signif(Q1,4))),color="blue",n
  geom_text(aes(x=-2,y=dnorm(-2),label="0.25"),color="red",nudge_y = 0.05,nu
```



Distribución normal

Para cualquier distribución (en este caso la normal), el comando "d____(x)" (en este caso, "dnorm") calcula el valor de la densidad de probabilidad en el punto x. Por ejemplo, para x=1:

```
dnorm(1)
```

```
## [1] 0.2419707
```

Para una distribución normal estándard (con media 0 y desvío 1), la densidad de probabilidad es:

$$f(x) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2}}$$

Veamos que coincide con lo obtenido por el comando dnorm:

```
1/sqrt(2*pi)*exp(-1^2/2)
```

```
## [1] 0.2419707
```

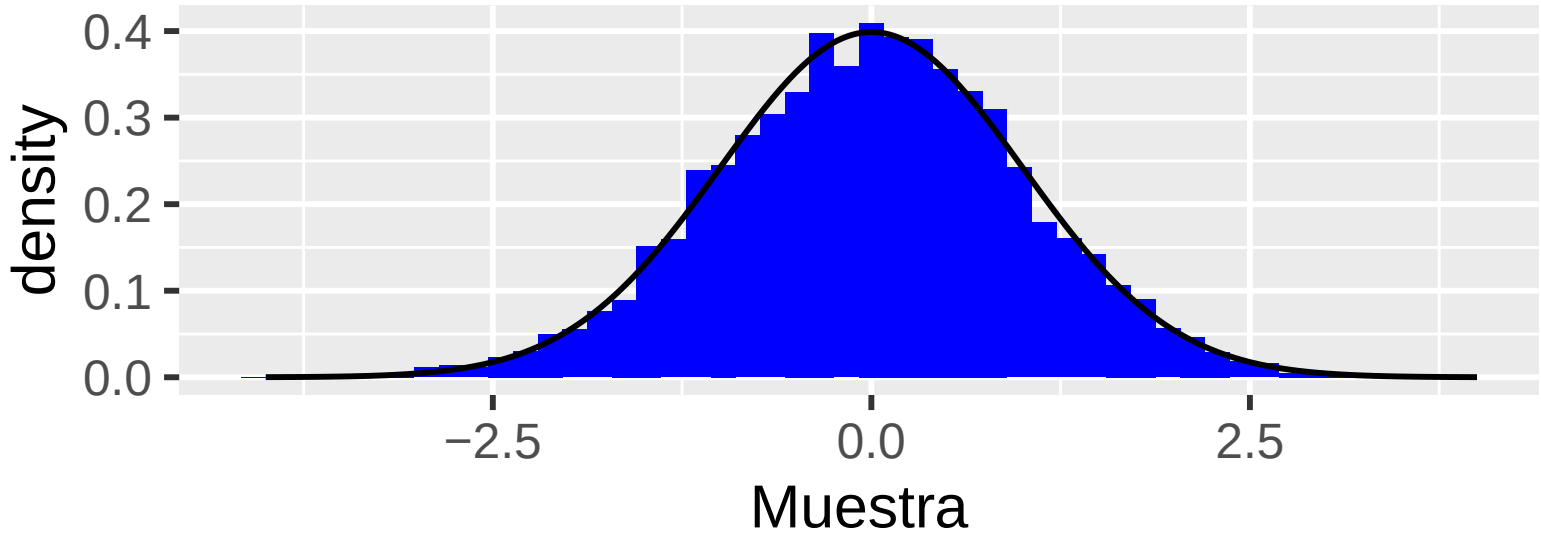
Distribución normal

Para cualquier distribución (en este caso la normal), el comando "r____(n)" (en este caso, "rnorm") genera n datos con distribución normal. Por ejemplo, para n=1000:

```
Normales=rnorm(5000);head(Normales,n=4)
```

```
## [1]  0.86072843 -0.74650181 -1.00773632  0.09813284
```

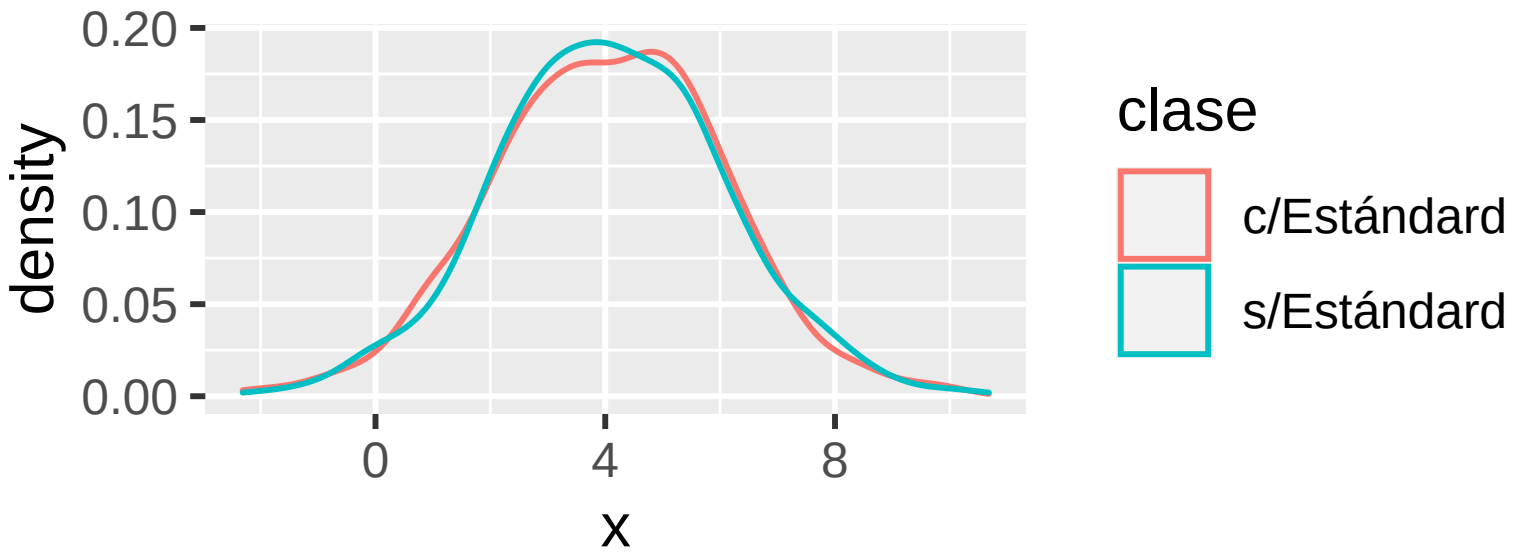
```
Datos=data.frame(Muestra=Normales)
ggplot(Datos,aes(x=Muestra))+
  geom_histogram(aes(y=..density..),fill="blue",bins=50)+
  geom_line(data=D,mapping=aes(x=x,y=f))
```



Distribución normal

La distribución normal tiene una propiedad. Si a una variable Z de distribución normal estándard (media 0 y desvío 1), se la multiplica por un desvío σ y se le suma una media μ la variable resultante X tiene distribución normal de media μ y desvío σ. Por lo tanto, podemos generar otras normales a partir de la normal estándard:

```
mu=4;sig=2;
Muestra1=rnorm(1000,mean = mu,sd= sig)
Muestra2=mu+sig*rnorm(1000)
DatNorm=data.frame(x=c(Muestra1,Muestra2),
                   clase=rep(c("s/Estándard","c/Estándard"),each=1000))
DatNorm %>%
  ggplot(aes(x=x,color=clase))+
  geom_density()
```



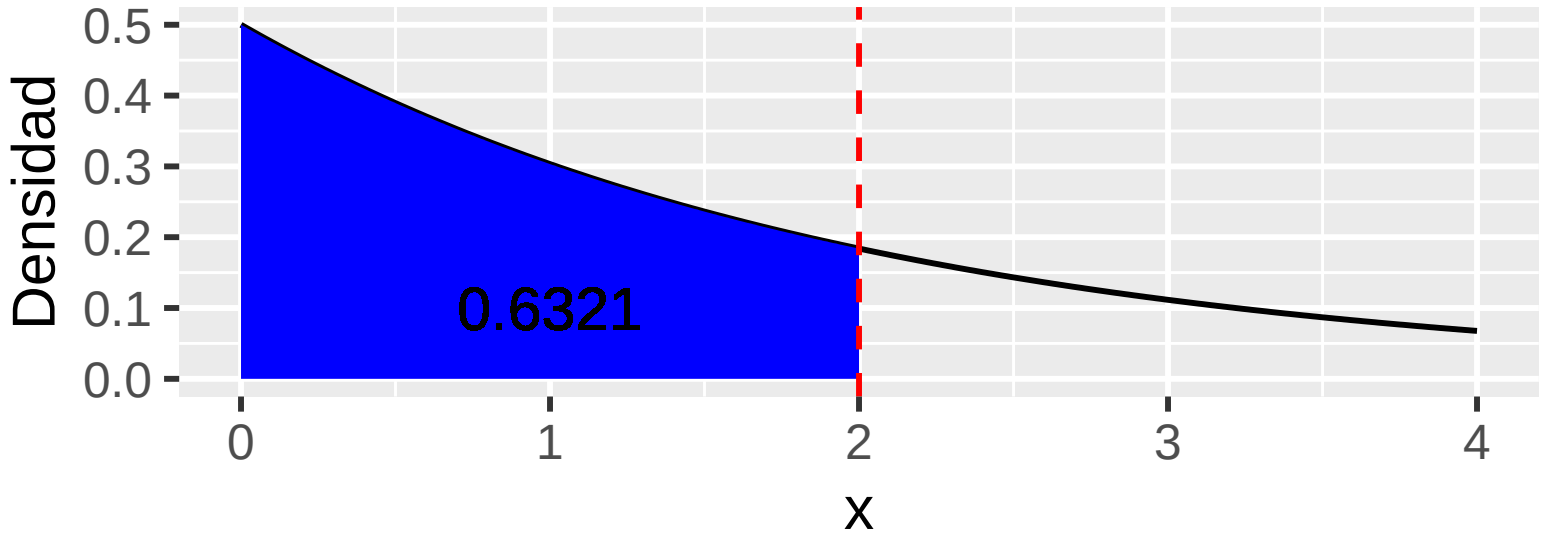
Otras distribuciones

Podemos extender varias de las ideas a otras distribuciones, por ejemplo, usando pexp para una exponencial de parámetro $\lambda = 0.5$:

```
P2=pexp(2,rate=0.5);P2
```

```
## [1] 0.6321206
```

```
D=data.frame(x=seq(0,4,by=0.01));D$f=dexp(D$x,rate = 0.5)
SubD=D %>% filter(x<=2)
D %>% ggplot(aes(x=x,y=f))+  geom_line()+
  geom_area(data=SubD,mapping=aes(x=x,y=f),fill="blue")+
  geom_vline(aes(xintercept=2),color="red",linetype="dashed")+
  geom_text(aes(x=1,y=0.1,label=signif(P2,4)))+ylab("Densidad")
```



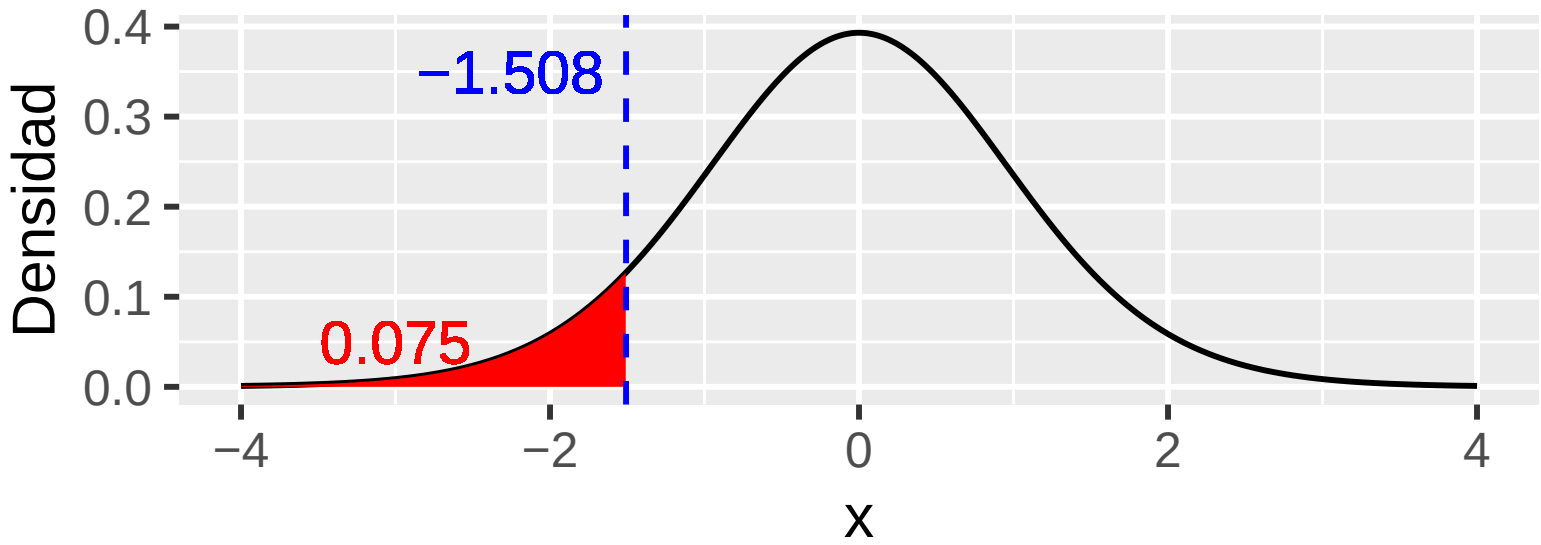
Otras distribuciones

Podemos extender varias de las ideas a otras distribuciones, por ejemplo, usando qt para una t de student con $df = 17$ grados de libertad:

```
pT7.5=qt(0.075,df = 17);pT7.5
```

```
## [1] -1.50766
```

```
D=data.frame(x=seq(-4,4,by=0.01));D$f=dt(D$x,df = 17)
SubD=D %>% filter(x<=pT7.5)
D %>% ggplot(aes(x=x,y=f))+  geom_line()+
  geom_area(data=SubD,mapping=aes(x=x,y=f),fill="red")+
  geom_vline(aes(xintercept=pT7.5),color="blue",linetype="dashed")+
  geom_text(aes(x=pT7.5,y=0.35,label=signif(pT7.5,4)),nudge_x = -0.75,color=
    geom_text(aes(x=-3,y=0.05,label="0.075"),color="red")
```



Otras distribuciones

Podemos extender varias de las ideas a otras distribuciones, por ejemplo, usando dpois calcular la probabilidad puntual de que una variable poisson de parámetro $\lambda = 5$ tome el valor 2:

```
dpois(2,lambda = 5)
```

```
## [1] 0.08422434
```

Notar que en el caso de que la variable sea discreta, la función d no devuelve la densidad, sino la probabilidad puntual. Recordar que para una variable X con distribución de poisson de parámetro 5, se cumple:

$$P(X = 2) = \frac{e^{-5} \cdot 5^2}{2!}$$

Veamos que se verifica esta ecuación:

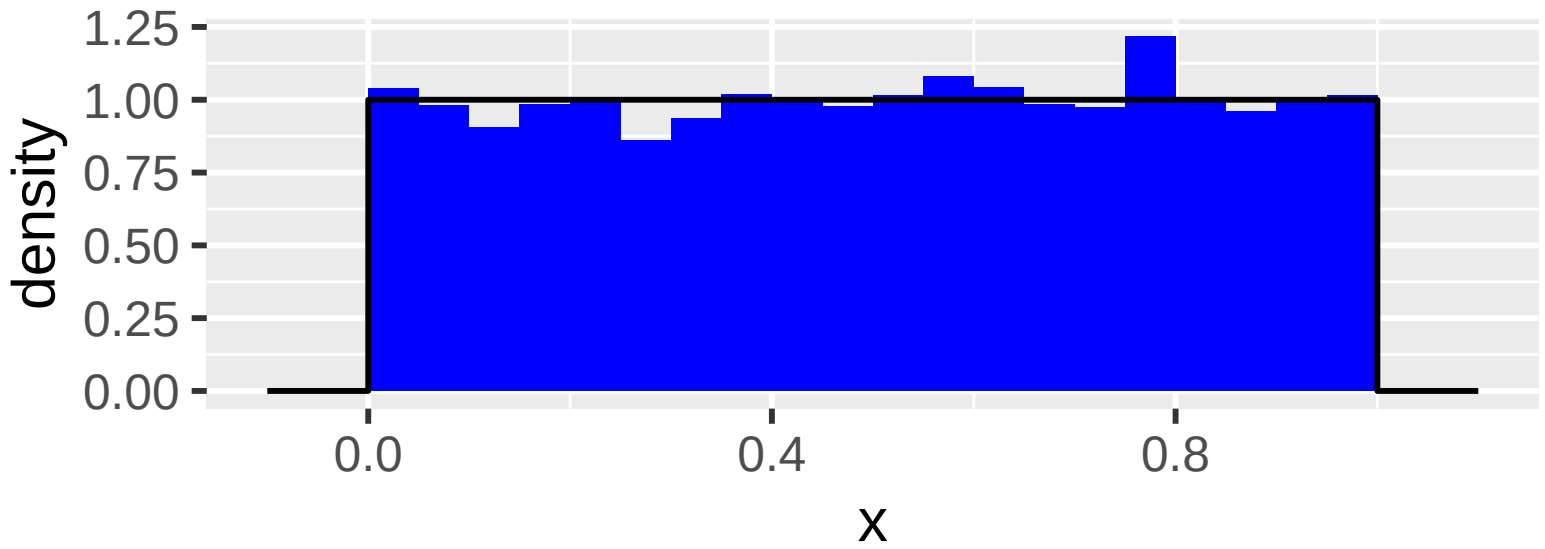
```
exp(-5)*5^2/factorial(2)
```

```
## [1] 0.08422434
```

Otras distribuciones

Podemos extender varias de las ideas a otras distribuciones, por ejemplo, usando runif para generar distribuciones uniformes entre 0 y 1:

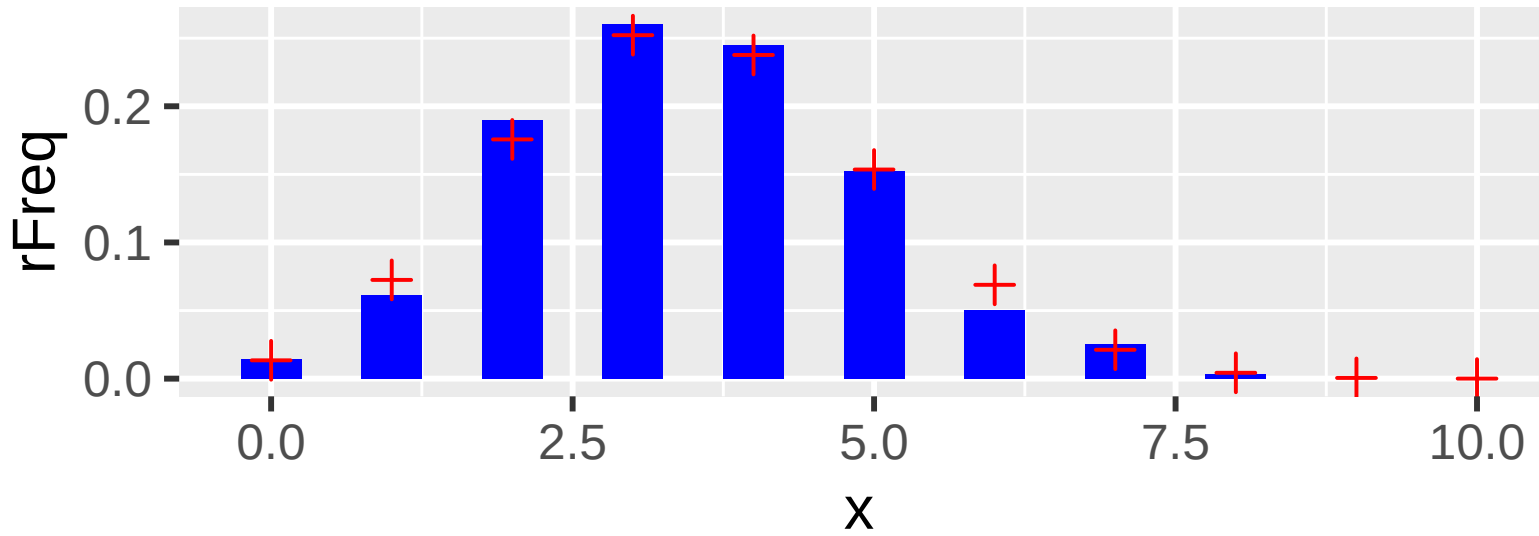
```
Muestra=runif(5000)
D=data.frame(x=Muestra);
Du=data.frame(x=c(-0.1,0,0,1,1,1.1),f=c(0,0,1,1,0,0))
D %>% ggplot(aes(x=x))+
  geom_histogram(aes(y=..density..),fill="blue",breaks=seq(0,1,by=0.05))+
  Du %>%
  geom_line(mapping=aes(x=x,y=f))
```



Otras distribuciones

Podemos extender varias de las ideas a otras distribuciones, por ejemplo, usando rbinom para generar distribuciones binomiales de parámetros $n = 10$ y $p = 0.35$:

```
Muestra=rbinom(1000,size = 10,prob = 0.35)
D=data.frame(x=Muestra);
Db=data.frame(x=0:10);Db$p=dbinom(Db$x,size = 10,prob = 0.35)
ResD=D %>% group_by(x) %>% summarise(rFreq=n()/1000)
ResD %>% ggplot(aes(x=x))+
  geom_bar(aes(y=rFreq),stat = "identity",width = 0.5,fill="blue")+
  Db %>%
  geom_point(mapping=aes(x=x,y=p),color="red",shape=3,size=2)
```



Regresión

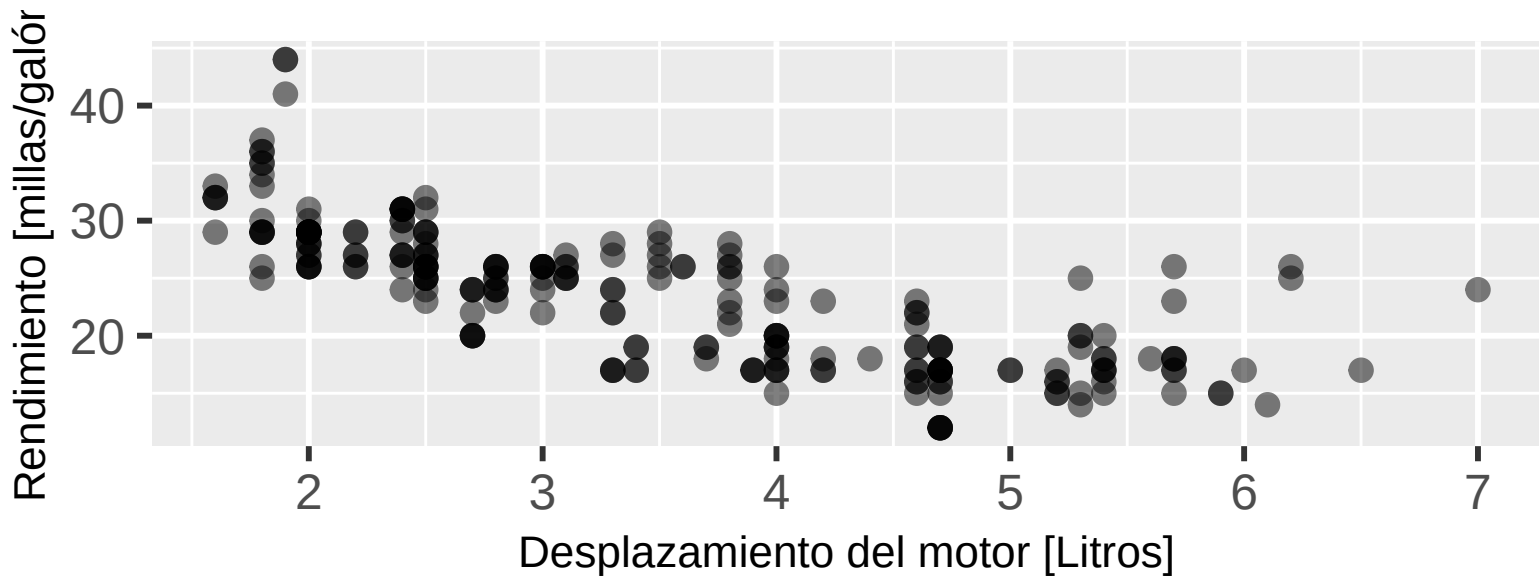
Generalidades

- Los modelos de regresión intentan cuantificar la relación entre dos o más variables (generalmente cuantitativas).
- Cada modelo propone una ecuación de regresión en la que se describe de qué forma la variable dependiente Y se vincula con las variables independientes $X_1, X_2, \cdots, X_p$.

Ejemplo

- Volviendo a la base mpg, recordemos el siguiente scatter plot que vincula el tamaño del motor con el rendimiento en autopista de los autos:

```
require(ggplot2)
GG.Scatter=mpg %>% ggplot(aes(x=displ,y=hwy)) + geom_point(alpha=0.5)+
  xlab("Desplazamiento del motor [Litros]")+
  ylab("Rendimiento [millas/galón]")+
  theme(axis.title = element_text(size=8))
GG.Scatter
```



- Por ejemplo, hay motivos para asumir que el tamaño del motor y el rendimiento en autopista están vinculados. Las preguntas son: ¿Cómo? y ¿Cuánto?

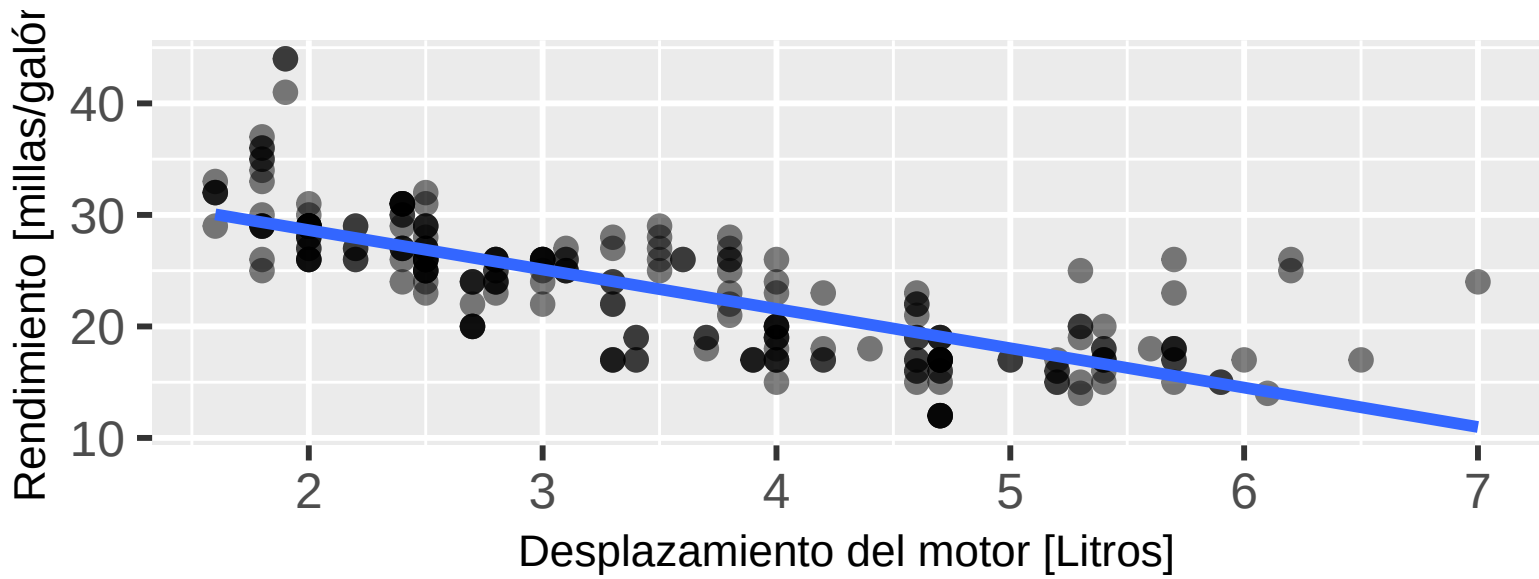
Regresión lineal simple

- El modelo más simple es la regresión lineal simple, en el que la variable hwy se comporta como una recta (con ordenada b y pendiente m, ambas desconocidas) que depende de una única variable displ, es decir:

$$\text{hwy}_i \approx b + m \cdot \text{displ}_i$$

  donde hwy_i y displ_i corresponden al rendimiento en autopista y el desplazamiento del motor de la i-ésima observación de la base mpg.
- La pregunta es, ¿cuál es la recta que "mejor" representa los datos? ggplot ya ofrece dicha recta con el comando geom_smooth(method="lm"):

```
GG.Lin=GG.Scatter+
  geom_smooth(method = "lm",se=F)
GG.Lin
```



Cálculo de parámetros

Sin embargo, no queda claro cuáles son los valores óptimos $\widehat{m}$ y $\widehat{b}$ que buscan representar los datos. Para calcular dichos valores, llamamos al comando lm (por linear model), que toma los siguientes argumentos:

- data: la base de datos a utilizar. (En este caso, mpg)
- formula: representación esquemática de qué variable depende de cuál otra. Se utiliza un símbolo ~, la variable a la izquierda del símbolo es la variable dependiente, mientras la que esté a la derecha es la variable independiente. (En este caso, la fórmula es hwy~displ)

```
LinMod=lm(data=mpg,formula = hwy~displ)
```

- Los coeficientes están dados extrayendo los coeficientes del modelo ajustado:

```
LinMod$coefficients
```

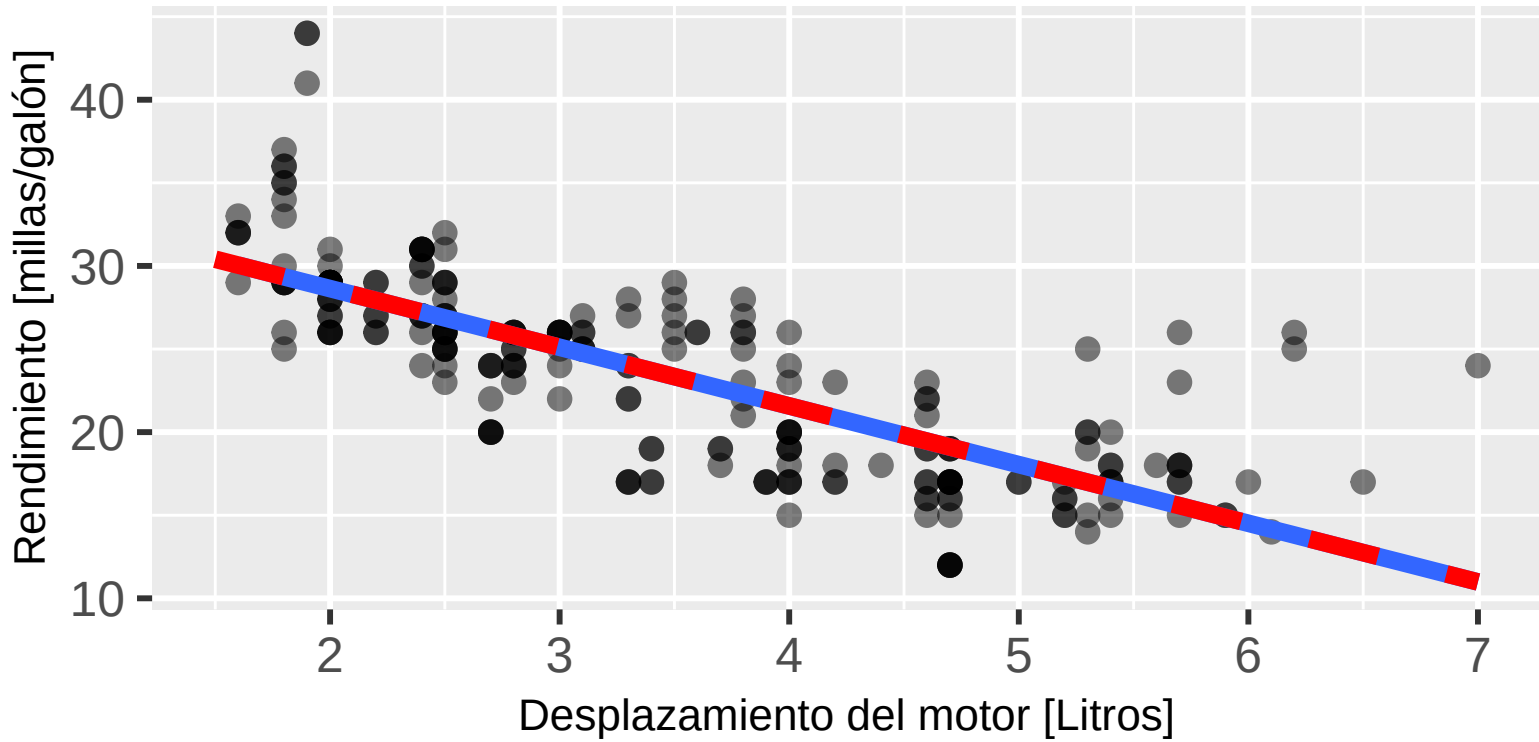
```
## (Intercept)       displ
##   35.697651   -3.530589
```

En este caso, las coordenadas correspondientes a (Intercept) y displ representan $\widehat{b}$ y $\widehat{m}$, respectivamente. Es decir, $\text{hwy}_i \approx 35.67 - 3.53 \cdot \text{displ}_i$.

Cálculo de parámetros

Podemos usar estos valores para verificar que son iguales a los dados por geom_smooth:

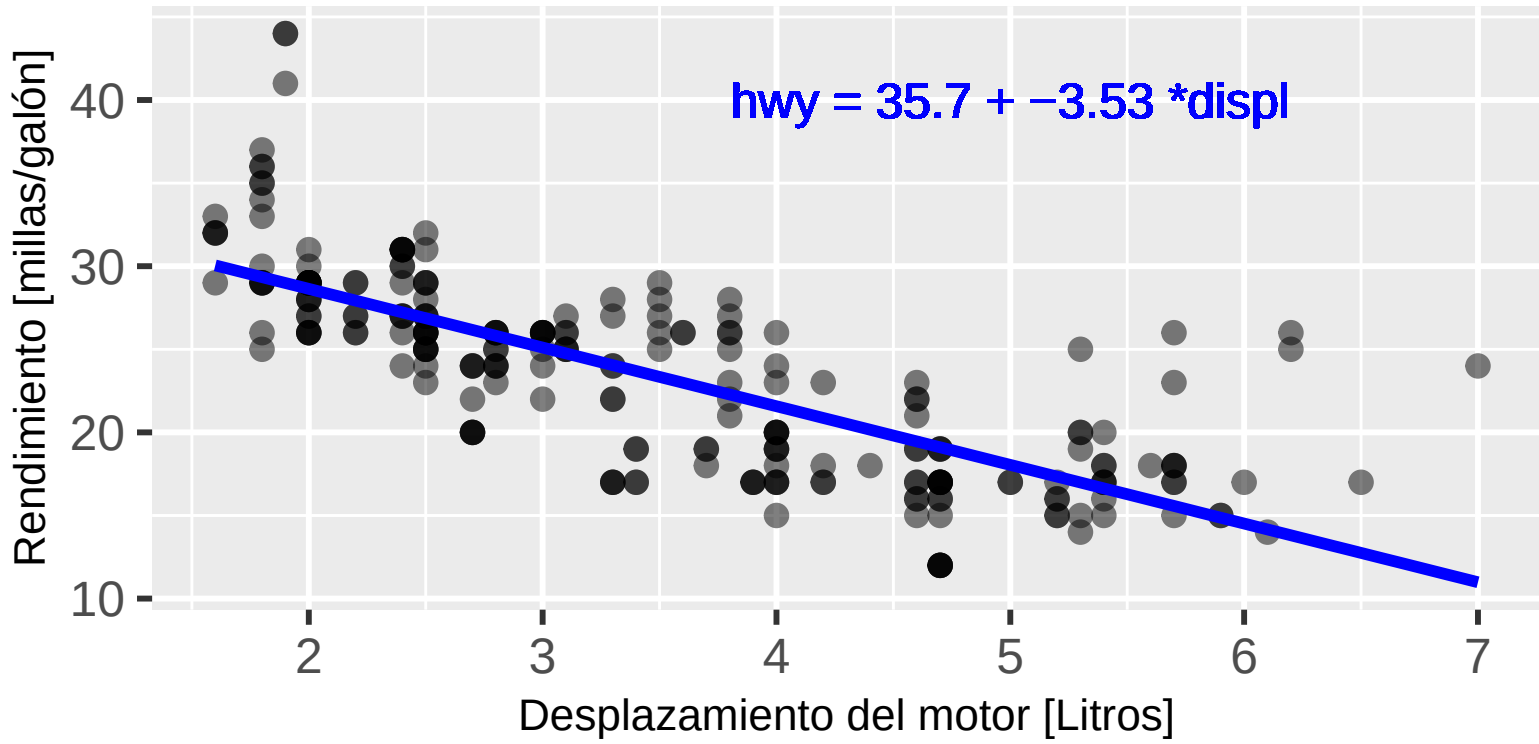
```
Coefs=LinMod$coefficients
dLM=data.frame(x=c(1.5,7));dLM$y=Coefs[1]+Coefs[2]*dLM$x
GG.Scatter+
  geom_smooth(method = "lm",se=F,size=1.5)+
  geom_line(data = dLM,mapping = aes(x=x,y=y),
            color="red", linetype="dashed",size=1.5)
```



Cálculo de parámetros

Más aún, nos permite agregar una leyenda con la recta obtenida:

```
GG.Scatter+
  geom_smooth(method = "lm",se=F,color="blue")+
  geom_text(mapping =
              aes(x=5,y=40,label=paste("hwy =",signif(Coefs[1],3),"+",signi
            color="blue",size=3)
```



Bondad de ajuste

Ya hemos calculado los parámetros que determinan la recta, pero eso no siempre dice que la recta represente bien a los datos. Para eso, se evalúa cuán bien el modelo lineal representa los datos (bondad de ajuste).

En los modelos lineales, la bondad de ajuste se calcula con una magnitud llamada R^2. Toma valores entre 0 y 1, mientras más cercano a 1 representa una mejor representación lineal de los datos. En R, se consigue su valor a través de la función summary:

```
summary(LinMod)$r.squared
```

```
## [1] 0.5867867
```

En las regresiones lineales simples, el R^2 coincide con el cuadrado del coeficiente de correlación:

```
rho=cor(mpg$displ,mpg$hwy);rho^2
```

```
## [1] 0.5867867
```

No da un valor tan alto, lo que lleva a pensar que las variables no se vinculan linealmente.

Modelo cuadrático

Podríamos pensar en otro modelo para el vínculo entre las variables, por ejemplo, que tengan una estructura cuadrática:

$$\text{hwy}_i \approx c + b \cdot \text{displ}_i + a \cdot \text{displ}_i^2$$

Para lograr esto, podríamos agregar una variable dependiente (displ^2) a la fórmula de lm:

```
CuadMod=lm(hwy~displ+displ^2,data=mpg)
CuadMod$coefficients
```

```
## (Intercept)       displ
##   35.697651   -3.530589
```

Comentario: uno podría preguntarse por qué se usa lm (linear model) para un modelo cuadrático. Esto se debe a que la ecuación de regresión es una combinación lineal de variables (1,displ y displ^2).

Notar que no se agregó un coeficiente para el cuadrado (de hecho, dan los mismos coeficientes que el ajuste lineal).

Modelo cuadrático

Esto se debe a que displ^2 no es una variable de la base mpg. Por lo tanto, podemos agregar una variable displSq que tenga los valores al cuadrado y luego ajustar el modelo:

```
mpg$displSq=mpg$displ^2
CuadMod=lm(hwy~displ+displSq,data=mpg)
CuadMod$coefficients
```

```
## (Intercept)       displ     displSq
##   49.245024  -11.760202    1.095424
```

Aquí si vemos los tres coeficientes. Otra forma de realizar lo mismo sin agregar una variable es agregando el comando I:

```
CuadMod=lm(hwy~displ+I(displ^2),data=mpg)
CuadMod$coefficients
```

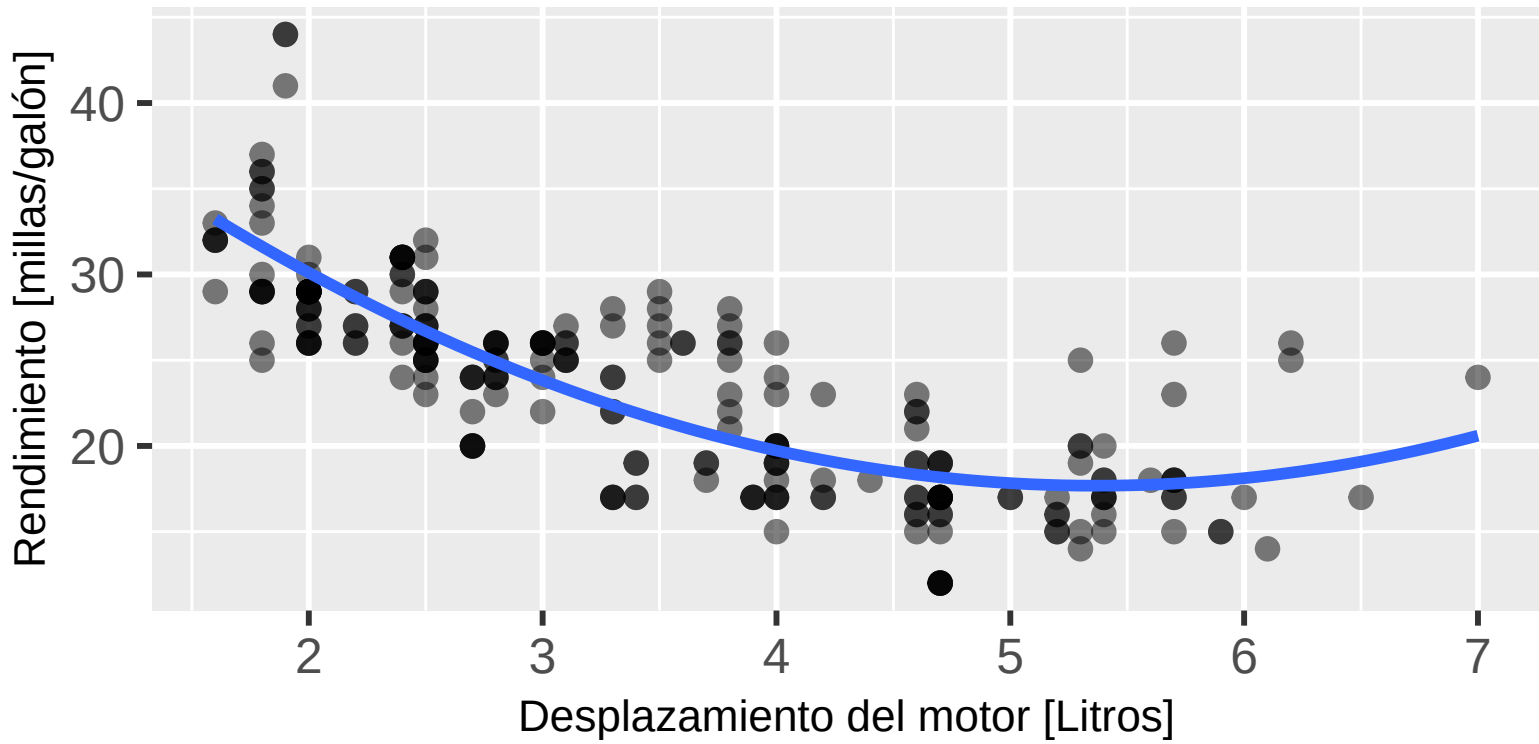
```
## (Intercept)       displ  I(displ^2)
##   49.245024  -11.760202    1.095424
```

Es decir, se considera el mejor modelo cuadrático como $\text{hwy}_i \approx 49.24 - 11.76 \cdot \text{displ}_i + 1.09 \cdot \text{displ}_i^2$.

Graficando la tendencia cuadrática

Para graficar la tendencia cuadrática, hay que agregar algo al comando geom_smooth, ya que si sólo se le pasa method="lm", considera la fórmula y~x (definidas por las variables estéticas, en este caso, hwy~displ):

```
GG.Scatter+
  geom_smooth(method = "lm",se=F,formula = y~x+I(x^2))
```



Modelo cuadrático

Parece ajustar mejor, sin embargo, vamos a verificarlo con el R^2:

```
summary(CuadMod)$r.squared
```

```
## [1] 0.6724572
```

Notemos que hay indicios de mejora en el ajuste respecto del ajuste lineal por dos motivos:

- El R^2 da un mayor valor, indicando que el modelo representa mejor a los datos.
- Agregar un término cuadrático cambia mucho los coeficientes de término lineal e independiente del modelo lineal inicial. Si no tuviera influencia, sería cercano a cero.

Modelos de mayor orden

Del mismo modo, podemos agregar un término cúbico y hasta cuártico:

```
CubMod=lm(hwy~displ+I(displ^2)+I(displ^3),data=mpg)
CubMod$coefficients
```

```
## (Intercept)       displ  I(displ^2)  I(displ^3)
##  40.2055719  -3.6012282  -1.1429219   0.1898886
```

```
summary(CubMod)$r.squared
```

```
## [1] 0.6778107
```

```
CuartMod=lm(hwy~displ+I(displ^2)+I(displ^3)+I(displ^4),data=mpg)
CuartMod$coefficients
```

```
##  (Intercept)       displ  I(displ^2)  I(displ^3)  I(displ^4)
##  49.63572807 -14.86906289  3.55197470 -0.62382268  0.04993659
```

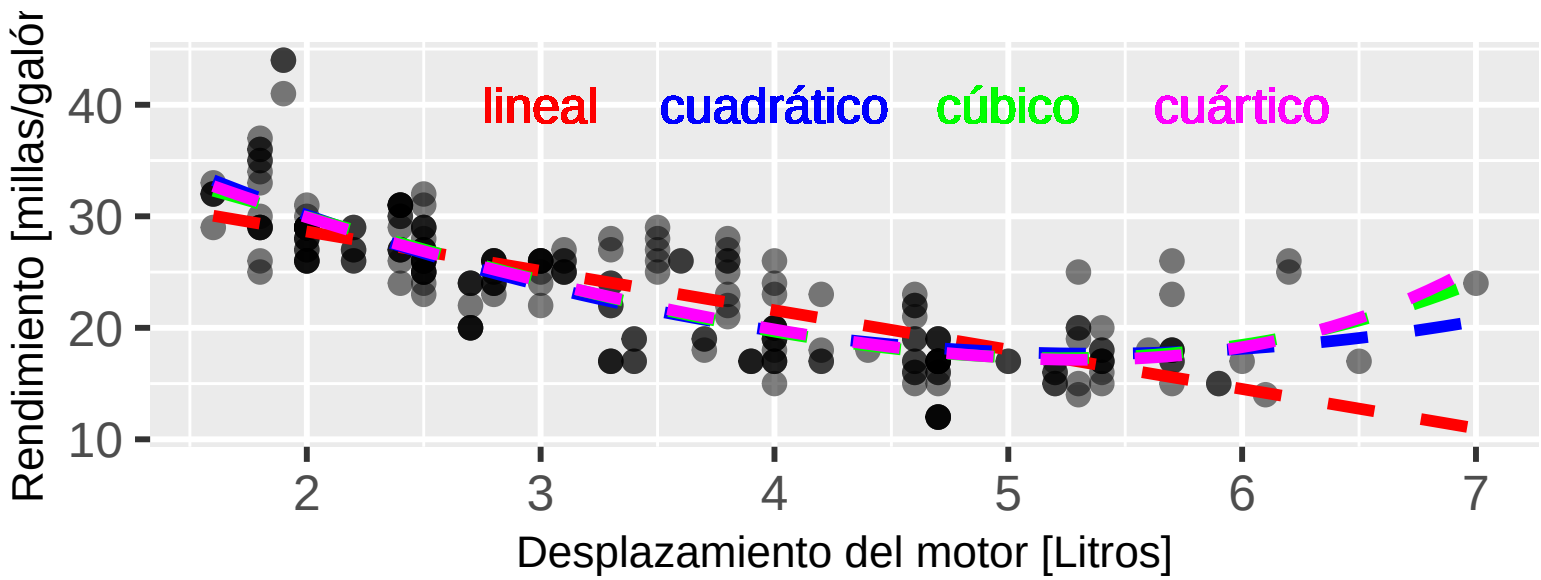
```
summary(CuartMod)$r.squared
```

```
## [1] 0.678651
```

Overfitting

Notar que a partir de un momento, el R^2 no mejora más al agregar más términos. Cuando se modela un vínculo entre variables, se busca un compromiso entre que tenga la menor cantidad de variables posibles, pero que siga explicando la relación entre las variables. Usar más variables de las necesarias se llama "overfitting".

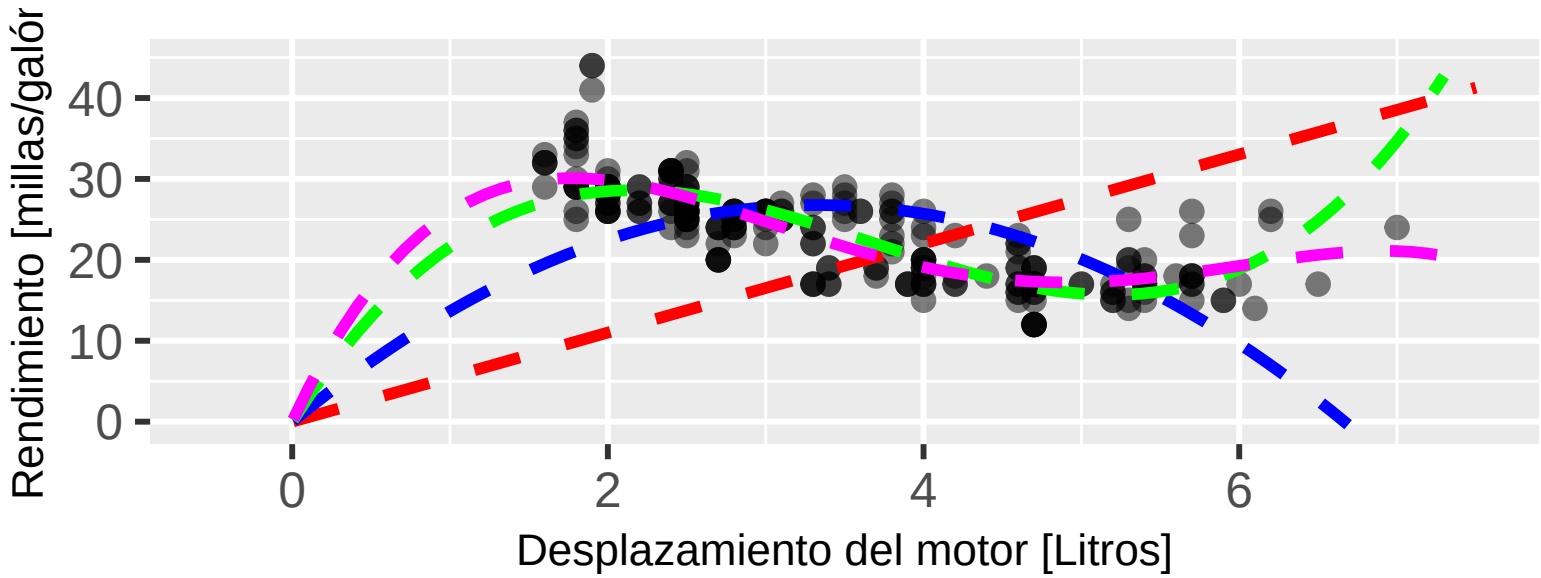
```
GG.Scatter+
  geom_smooth(method = "lm",se=F,color="red",alpha=0.5,linetype="dashed")+
  geom_smooth(method = "lm",se=F,formula = y~x+I(x^2),color="blue",alpha=0.5
  geom_smooth(method = "lm",se=F,formula = y~x+I(x^2)+I(x^3),color="green",a
  geom_smooth(method = "lm",se=F,formula = y~x+I(x^2)+I(x^3)+I(x^4),color="m
  geom_text(aes(x=3,y=40,label="lineal"),color="red",size=3)+  geom_text(aes
  geom_text(aes(x=5,y=40,label="cúbico"),color="green",size=3)+  geom_text(a
```



Ordenada

Comentario: Nunca agregamos un término para la ordenada en la fórmula, eso es porque R asume por default una ordenada al origen para todos los modelos. Si no queremos que tenga ordenada al origen, podemos agregar un -1 a la fórmula:

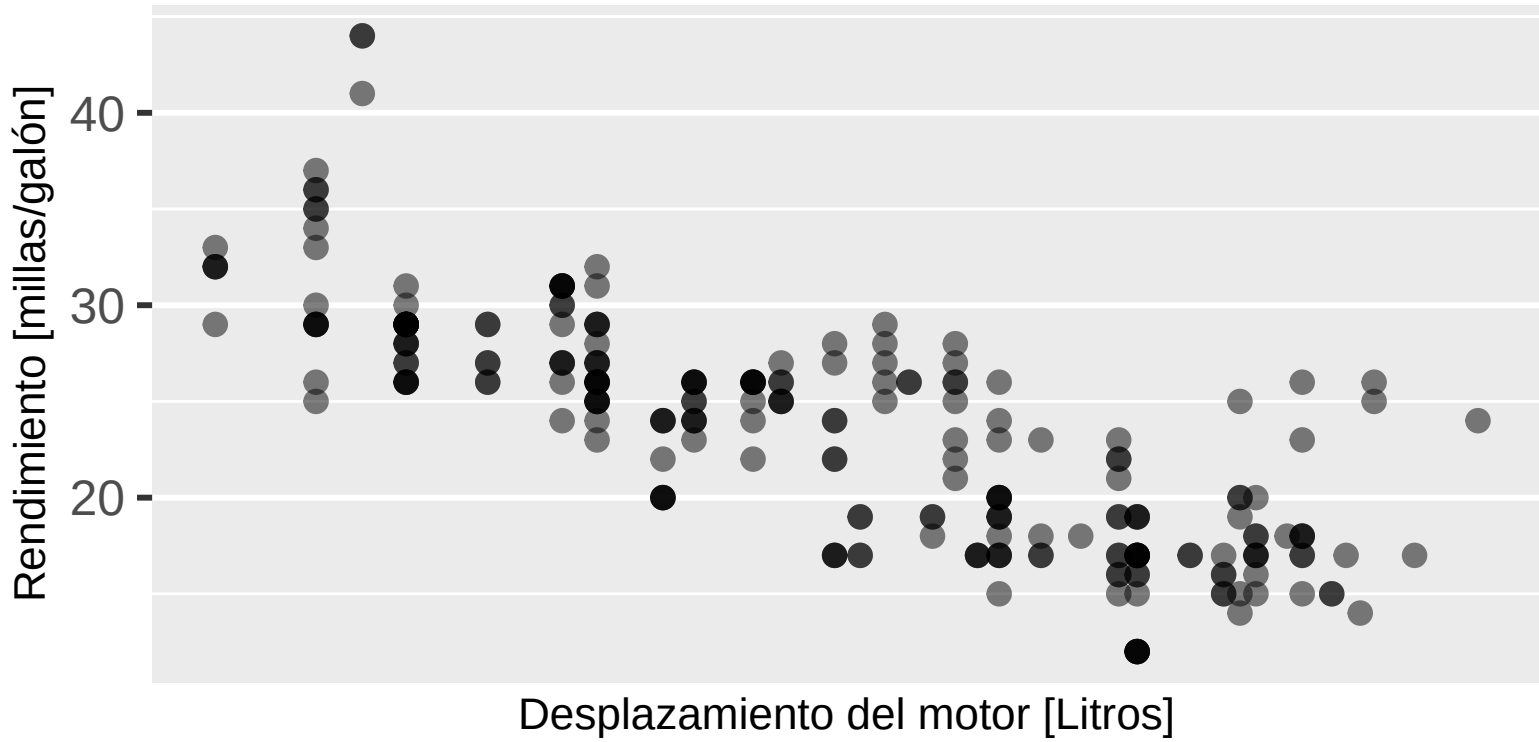
```
GG.Scatter+xlim(c(-0.5,7.5))+ylim(c(-0.5,45))+
  geom_smooth(method = "lm",se=F,formula = y~x-1,color="red",alpha=0.5,linet
  geom_smooth(method = "lm",se=F,formula = y~x-1+I(x^2),color="blue",alpha=0
  geom_smooth(method = "lm",se=F,formula = y~x-1+I(x^2)+I(x^3),color="green"
  geom_smooth(method = "lm",se=F,formula = y~x-1+I(x^2)+I(x^3)+I(x^4),color=
```



Transformaciones

Otra forma de mejorar el modelo, es realizando tranformaciones previas a alguna de las variables. Por ejemplo, cambiemos la escala x a logarítmica:

```
GG.Scatter+
  scale_x_log10()
```



Notemos que parece mejorar mucho la relación lineal entre las variables.

Transformaciones

Eso nos podría llevar a pensar en otro modelo:

$$\text{hwy}_i \approx c + b \cdot \log_{10}(\text{displ})_i$$

Podemos agregar una variable logDispl o agregar de nuevo el comando I:

```
logMod=lm(hwy~I(log10(displ)),data = mpg)
logMod$coefficients
```

```
##     (Intercept) I(log10(displ))
##        38.20802       -28.95491
```

```
summary(logMod)$r.squared
```

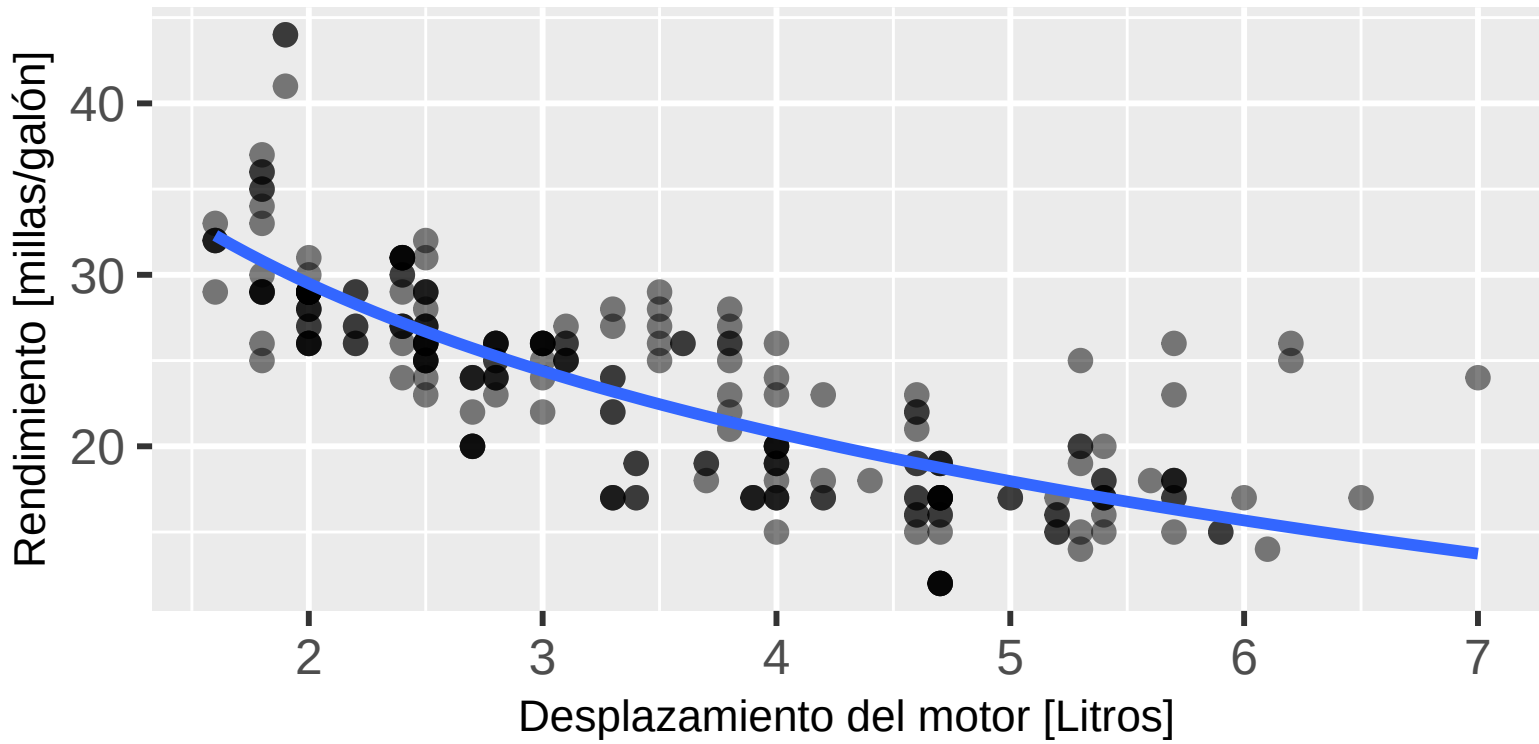
```
## [1] 0.6408921
```

Notar que da peor que el modelo cuadrático, pero al tener una única variable, mejora mucho respecto del valor de R^2 del primer modelo lineal.

Transformaciones

Grafiquemos el ajuste logarítmico:

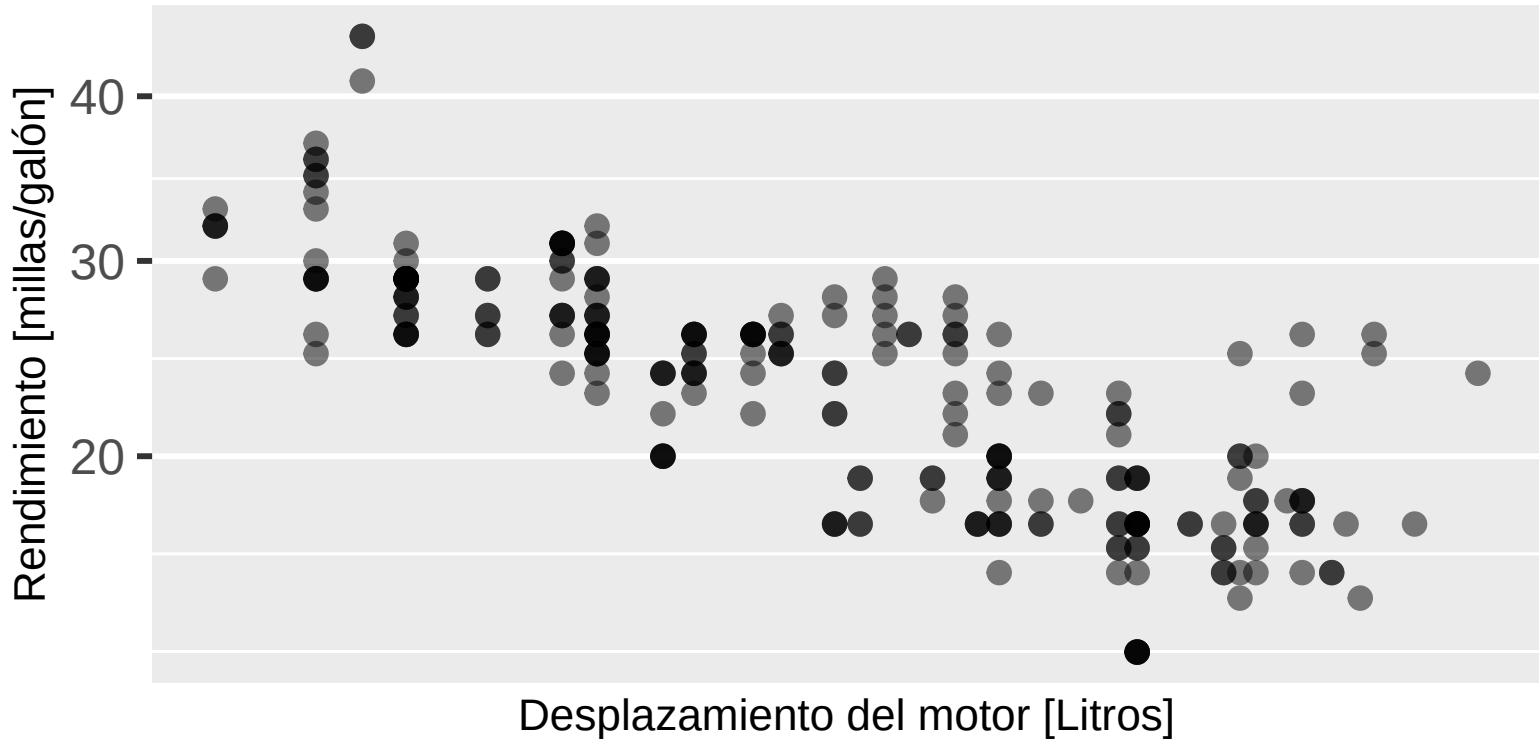
```
GG.Scatter+
  geom_smooth(method = "lm",formula = y~I(log10(x)),se=F)
```



Transformaciones

Más aún, si agregamos una escala radical al eje y:

```
GG.Scatter+
  scale_x_log10()+
  scale_y_sqrt()
```



Transformaciones

Eso nos podría llevar a pensar en otro modelo:

$$\sqrt{hwy}_i \approx c + b \cdot \log_{10}(\text{displ})_i$$

Podemos agregar una variable sqrtHwy o agregar de nuevo el comando I:

```
sqrtlogMod=lm(I(sqrt(hwy))~I(log10(displ)),data = mpg)
sqrtlogMod$coefficients
```

```
##      (Intercept) I(log10(displ))
##         6.333548       -3.002299
```

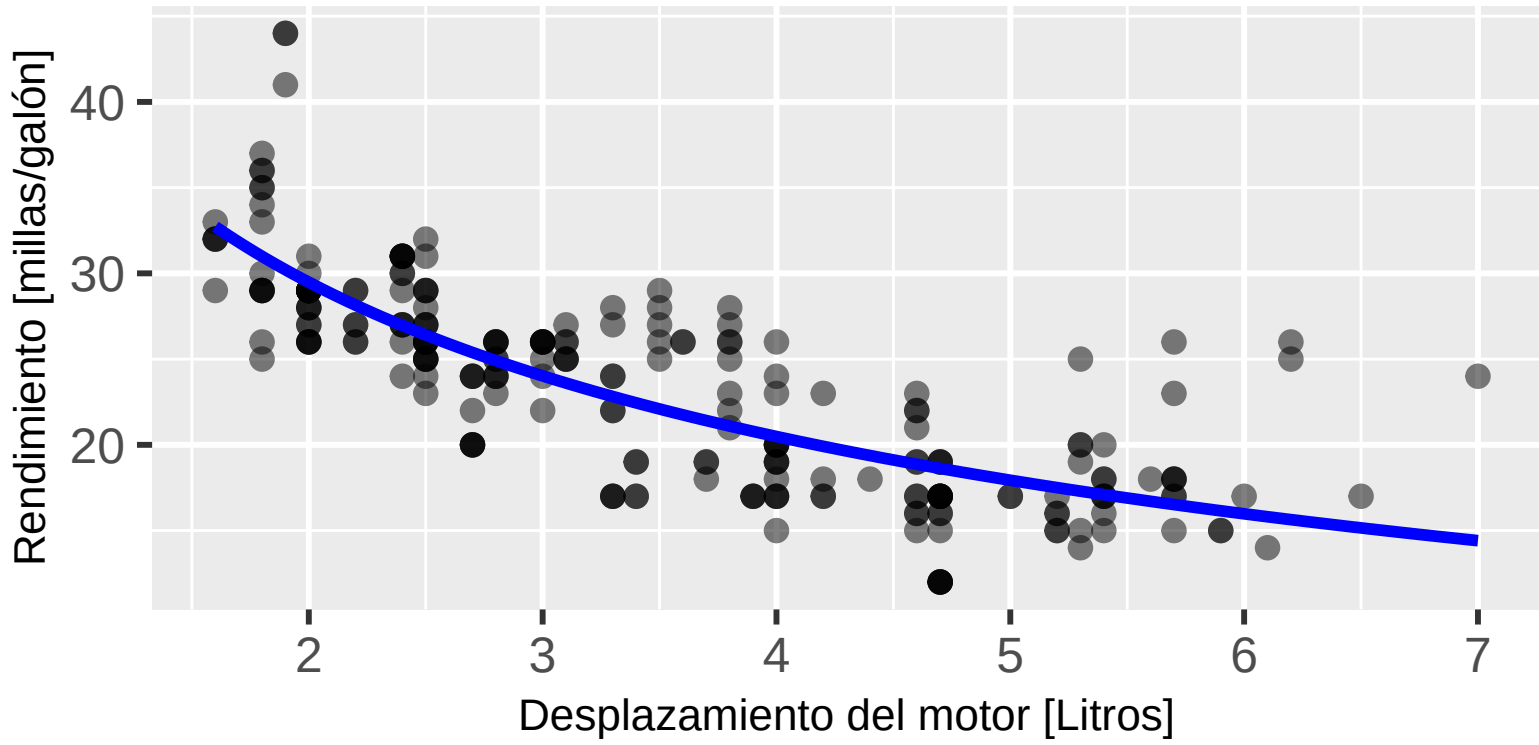
```
summary(sqrtlogMod)$r.squared
```

```
## [1] 0.6432422
```

Transformaciones

Grafiquemos este nuevo ajuste:

```
Dp=data.frame(displ=seq(1.6,7,by=0.01));Dp$Fit=predict(sqrtlogMod,Dp)^2
GG.Scatter+
  Dp %>% geom_line(mapping=aes(x=displ,y=Fit),color="blue",size=1)
```



Predicciones

Los modelos estadísticos sirven para explicar los datos, pero también para predecir el comportamiento de nuevas observaciones. Por ejemplo, una pregunta natural sería: Si un auto que no sea de la base tiene un desplazamiento del motor de 4 litros, ¿Cuánto rendiría en autopista?

Para eso, una vez calculados los parámetros de un modelo, podemos usar sus valores estimados para predecir una nueva observación. Por ejemplo, para el modelo lineal, habíamos obtenido:

$$\text{hwy}_i \approx 35.67 - 3.53 \cdot \text{displ}_i$$

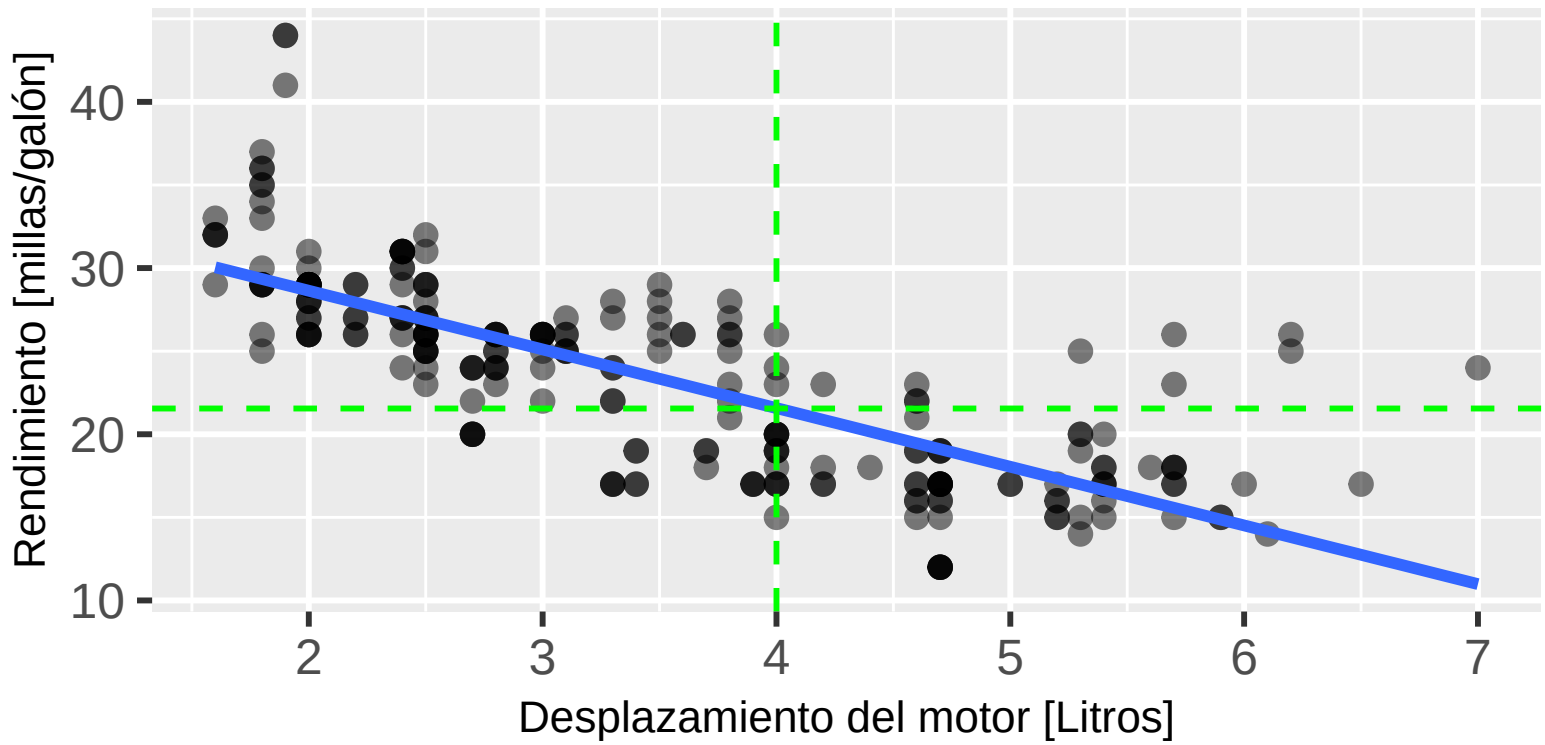
Por lo tanto, ante una observación con un valor de displ=4, el valor estimado de hwy sería:

$$\widehat{\text{hwy}} \approx 35.67 - 3.53 \cdot 4 = 21.55$$

Predicciones

De hecho, si sumamos al gráfico una recta vertical en x=4 y una horizontal en y=21.55, se intersecarán justo con la recta de estimación:

```
GG.Lin+
  geom_vline(aes(xintercept=4),color="green",linetype="dashed")+
  geom_hline(aes(yintercept=21.55),color="green",linetype="dashed")
```



Predicciones

Dado un modelo, para predecir el valor estimado de la variable dependiente a partir de una realización de las variables independientes (en un data.frame con los mismos nombres de las variables), se puede usar el comando predict:

```
NewObs=data.frame(displ=c(1.75,4,6.5))
PLin=predict(LinMod,NewObs);names(PLin)=c(1.75,4,6.5);PLin
```

```
##     1.75        4      6.5
## 29.51912 21.57530 12.74882
```

```
PCuad=predict(CuadMod,NewObs);names(PCuad)=c(1.75,4,6.5);PCuad
```

```
##     1.75        4      6.5
## 32.01940 19.73099 19.08536
```

```
PCub=predict(CubMod,NewObs);names(PCub)=c(1.75,4,6.5);PCub
```

```
##     1.75        4      6.5
## 31.42091 19.66678 20.65730
```

```
PCuart=predict(CuartMod,NewObs);names(PCuart)=c(1.75,4,6.5);PCuart
```

```
##     1.75        4      6.5
## 31.61784 19.85019 20.88038
```

Predicciones

En el caso de haber transformado la variable dependiente, hay que aplicar la transformación inversa:

```
predict(sqrtlogMod,NewObs)
```

```
##        1        2        3
## 5.603875 4.525984 3.892939
```

```
Psqlog=predict(sqrtlogMod,NewObs)^2;Psqlog
```

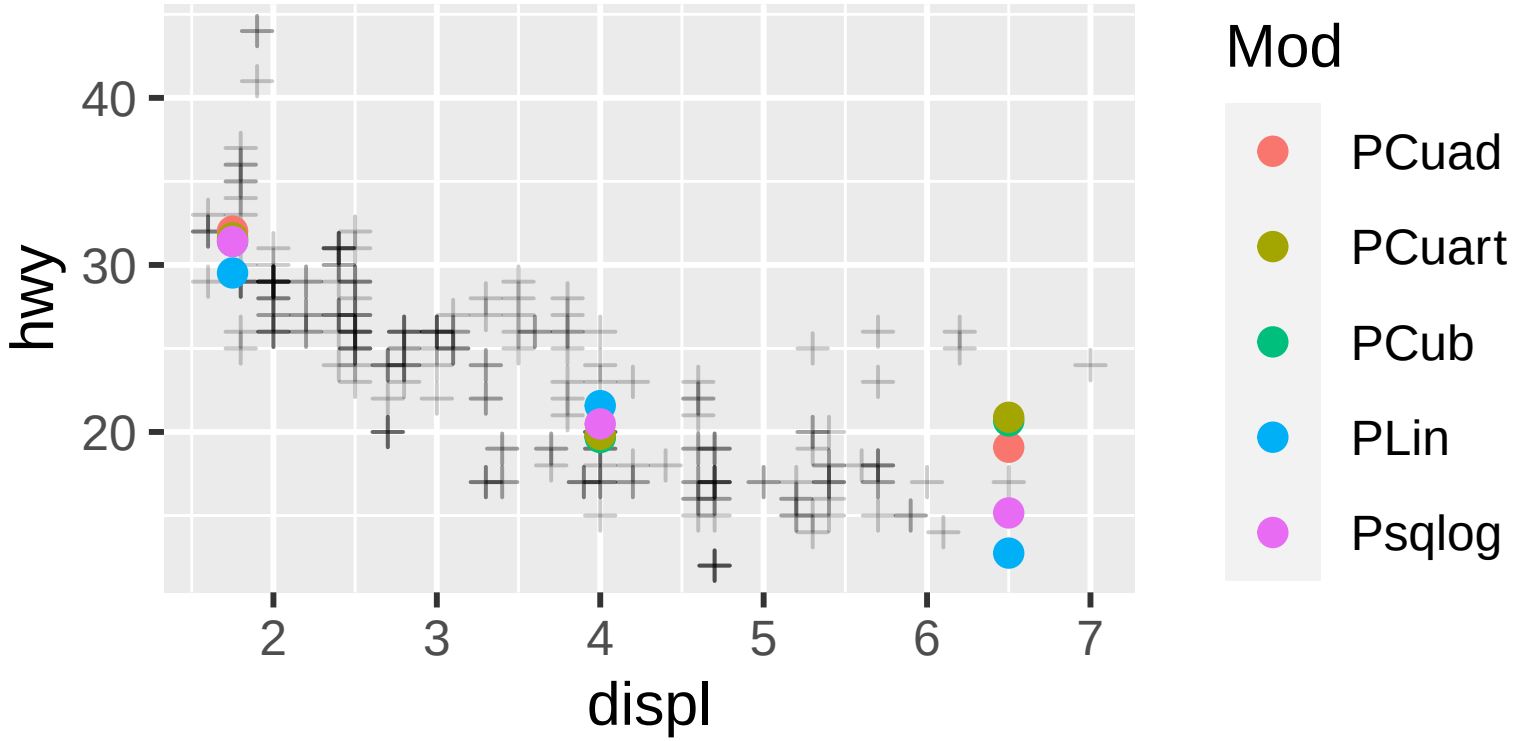
```
##        1        2        3
## 31.40342 20.48453 15.15497
```

Notar que en el primer caso, no da nada similar a lo que darían los resultados del rendimiento en autopista. Al elevarlos al cuadrado, dan resultados acordes.

Predicciones

Vamos a graficar los puntos de las predicciones según cada modelo:

```
require(tidyr);NewObs$PLin=PLin;NewObs$PCuad=PCuad;NewObs$PCub=PCub;NewObs$P
NewObs.L=NewObs %>% gather(key = "Mod",value = "Pred",-displ)
mpg %>% ggplot(aes(x=displ,y=hwy))+geom_point(alpha=0.2,shape=3)+
  geom_point(data=NewObs.L,mapping = aes(x=displ,y=Pred,color=Mod),shape=19,
```



Notar que las predicciones suelen ser similares para todos los modelos, pero se dispersan más para predecir rendimientos de motores de 6.5 litros.

Incertidumbre

Esto se debe a la poca cantidad de datos con tamaños de motores cercanos a 6.5, haciendo menos fidedigna la estimación en esa zona.

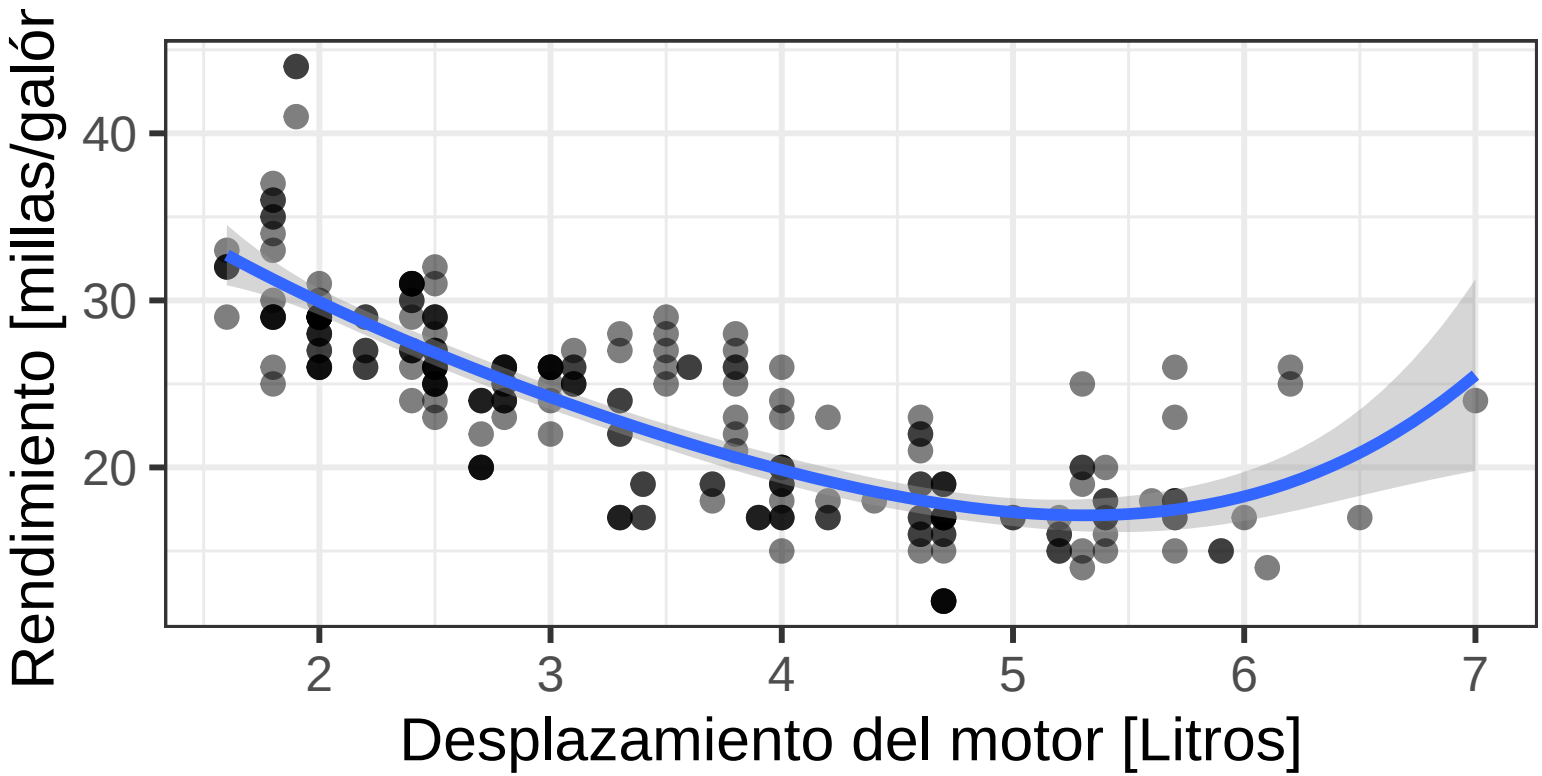
Esto acentúa el hecho de que los modelos fueron calculados con unos datos iniciales, y que extrapolar esa relación a datos muy distintos de los que usamos para modelar, no es tan recomendable.

Por lo tanto, generalmente se agrega a los gráficos una figura que muestre la incertidumbre en la estimación, que dependerá tanto de la cantidad de observaciones como de cuánto se dispersen en esa región.

Incertidumbre

Para agregar la incertidumbre de cada modelo, se le saca el argumento se=FALSE (se: standard error) a geom_smooth. Por ejemplo, aquí agregamos la incertidumbre al modelo cuártico:

```
GG.Scatter+theme_bw()+
  geom_smooth(method = "lm",formula = y~x+I(x^2)+I(x^3)+I(x^4))
```



Notar que hacia el final del gráfico (donde hay menos datos) la incertidumbre en la estimación aumenta.

Incertidumbre

Justamente este es uno de los problemas del overfitting. Al agregar muchas variables, quedan menos datos para estimar cada parámetro y agrega incertidumbre a la predicción.

Residuos

Se llama residuos a la diferencia entre los valores observados y aquellos predichos por el modelo. Si el modelo ajusta bien, son cercanos a cero.

Los residuos están incluidos en cada modelo ajustado:

```
RLin=LinMod$residuals; head(RLin,n=4)
```

```
##          1          2          3          4
## -0.3425912 -0.3425912  2.3635266  1.3635266
```

```
RCuad=CuadMod$residuals; head(RCuad,n=4)
```

```
##          1          2          3          4
## -2.6258323 -2.6258323  0.8936861 -0.1063139
```

```
RCub=CubMod$residuals; head(RCub,n=4)
```

```
##           1           2           3           4
## -2.12772445 -2.12772445  1.04946334  0.04946334
```

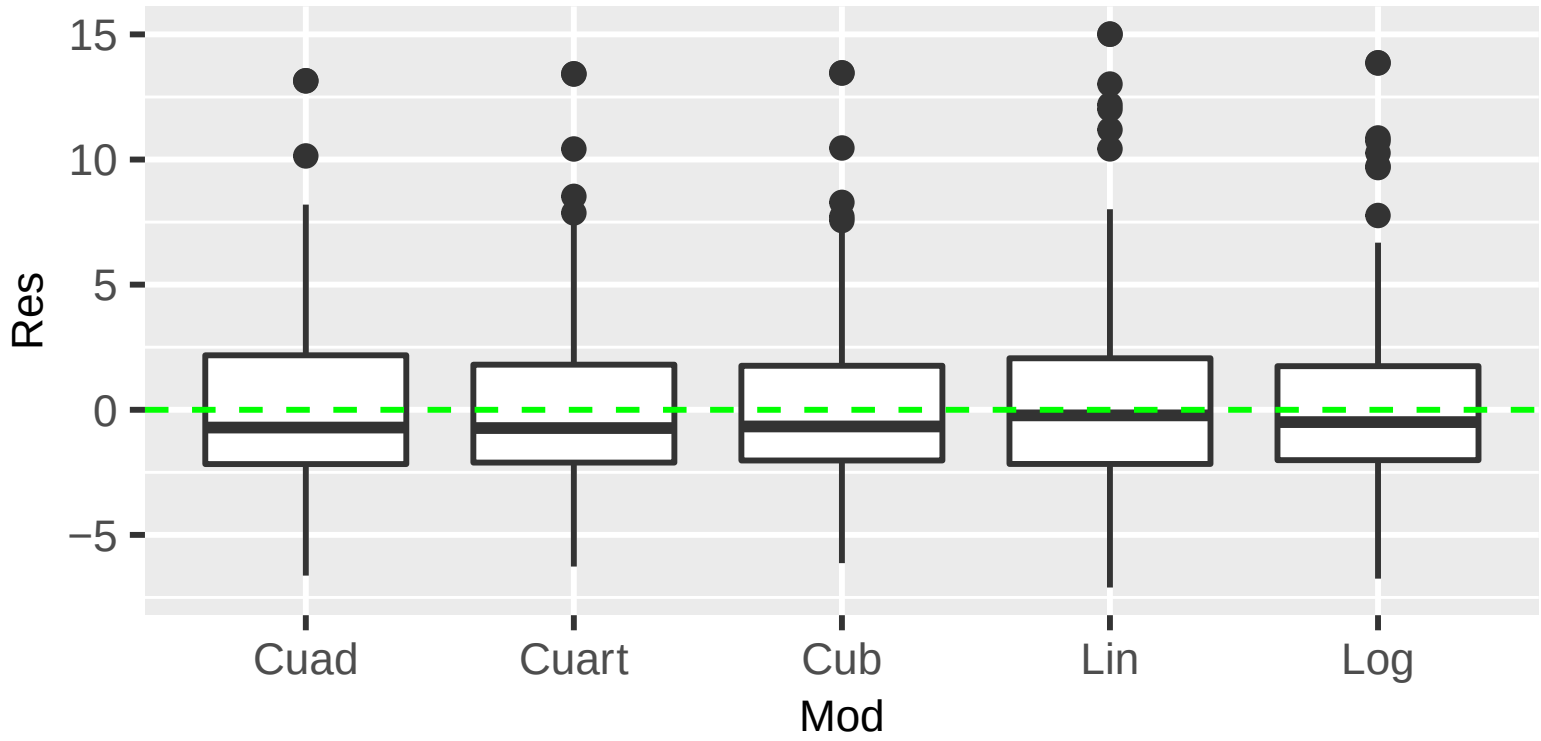
```
RCuart=CuartMod$residuals; head(RCuart,n=4)
```

```
##           1           2           3           4
## -2.26589332 -2.26589332  1.08609499  0.08609499
```

Residuos

El análisis de residuos es muy complejo, pero una cosa simple que se puede ver es cómo se distribuyen para distintos modelos:

```
Rlog=logMod$residuals;
rDat=data.frame(Res=c(RLin,RCuad,RCub,RCuart,Rlog),
                Mod=rep(c("Lin","Cuad","Cub", "Cuart","Log"), each=nrow(mpg)
rDat %>%
  ggplot(aes(x=Mod,y=Res))+geom_boxplot()+ geom_hline(aes(yintercept=0),colo
```



Variables categóricas

Vamos a intentar modelar los datos con una recta por cada nivel de drv. Es decir,

$$\text{hwy}_i \approx \begin{cases} a_1 + b_1 \cdot \text{displ}_i & \text{si drv=4} \\ a_2 + b_2 \cdot \text{displ}_i & \text{si drv=f} \\ a_3 + b_3 \cdot \text{displ}_i & \text{si drv=r} \end{cases}$$

Para tomar la variable drv como numérica, vamos a darles dos asignaciones:

```
mpg$drvNum1=(mpg$drv=="4")*0+(mpg$drv=="f")*1+(mpg$drv=="r")*2
mpg$drvNum2=(mpg$drv=="4")*16+(mpg$drv=="f")*(-10)+(mpg$drv=="r")*2
```

Variables categóricas

Veamos cómo quedaron las asignaciones

```
mpg %>% filter(drv=="4") %>% select(displ,hwy,drv,drvNum1,drvNum2) %>%    he
```

```
## # A tibble: 2 x 5
##   displ   hwy drv   drvNum1 drvNum2
##   <dbl> <int> <chr>   <dbl>   <dbl>
## 1   1.8    26 4           0      16
## 2   1.8    25 4           0      16
```

```
mpg %>% filter(drv=="f")%>% select(displ,hwy,drv,drvNum1,drvNum2) %>%   head
```

```
## # A tibble: 2 x 5
##   displ   hwy drv   drvNum1 drvNum2
##   <dbl> <int> <chr>   <dbl>   <dbl>
## 1   1.8    29 f           1     -10
## 2   1.8    29 f           1     -10
```

```
mpg %>% filter(drv=="r")%>% select(displ,hwy,drv,drvNum1,drvNum2) %>%   head
```

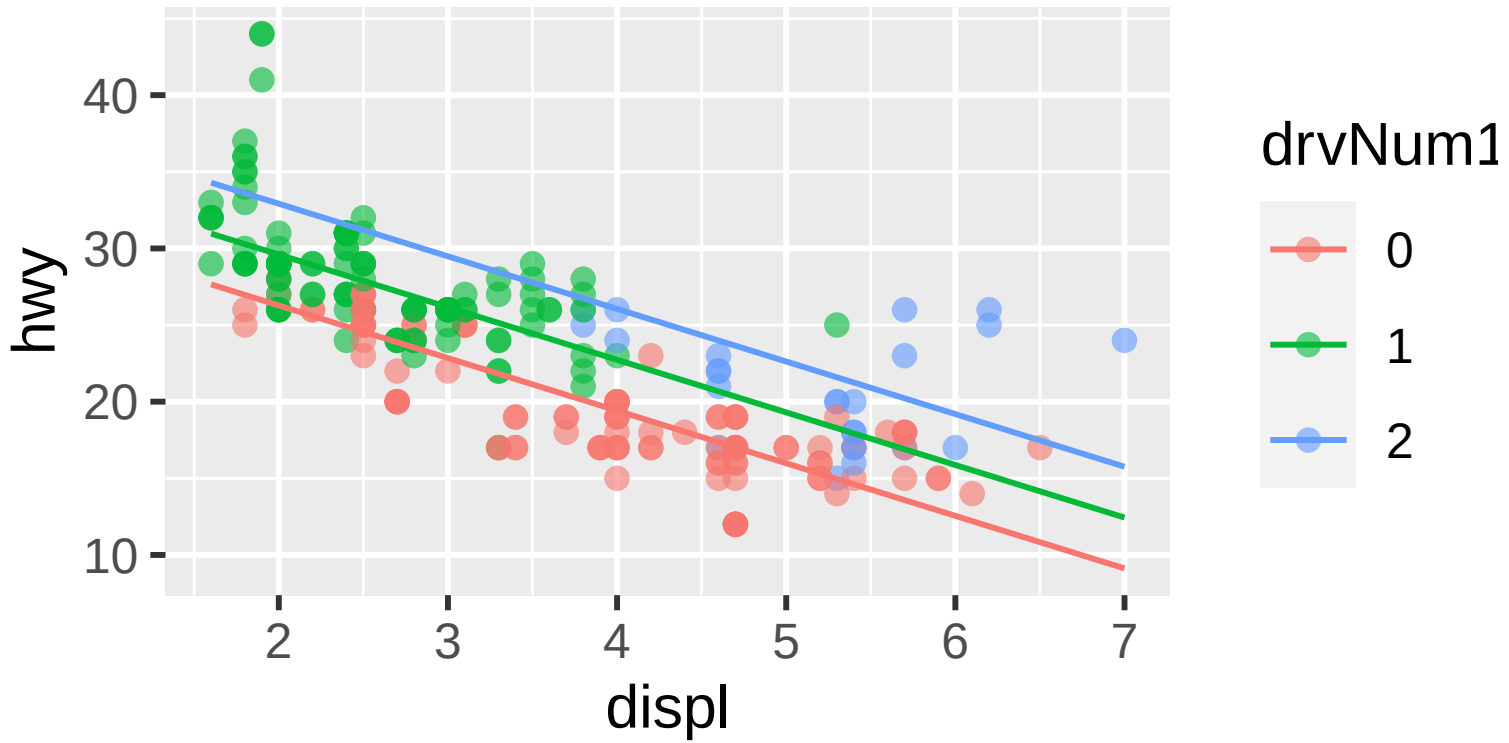
```
## # A tibble: 2 x 5
##   displ   hwy drv   drvNum1 drvNum2
##   <dbl> <int> <chr>   <dbl>   <dbl>
## 1   5.3    20 r           2       2
## 2   5.3    15 r           2       2
```

Variables categóricas

Vamos a intentar ajustar un modelo con drvNum1:

```
LMdrvN1=lm(hwy~displ+drvNum1,data = mpg)
```

```
mpg %>% mutate(drvNum1=factor(drvNum1))%>% ggplot(aes(x=displ,y=hwy))+
  geom_point(aes(color=drvNum1), alpha=0.6)+
geom_line(data = New,mapping=aes(x=displ,y=Pred,color=drvNum1))
```

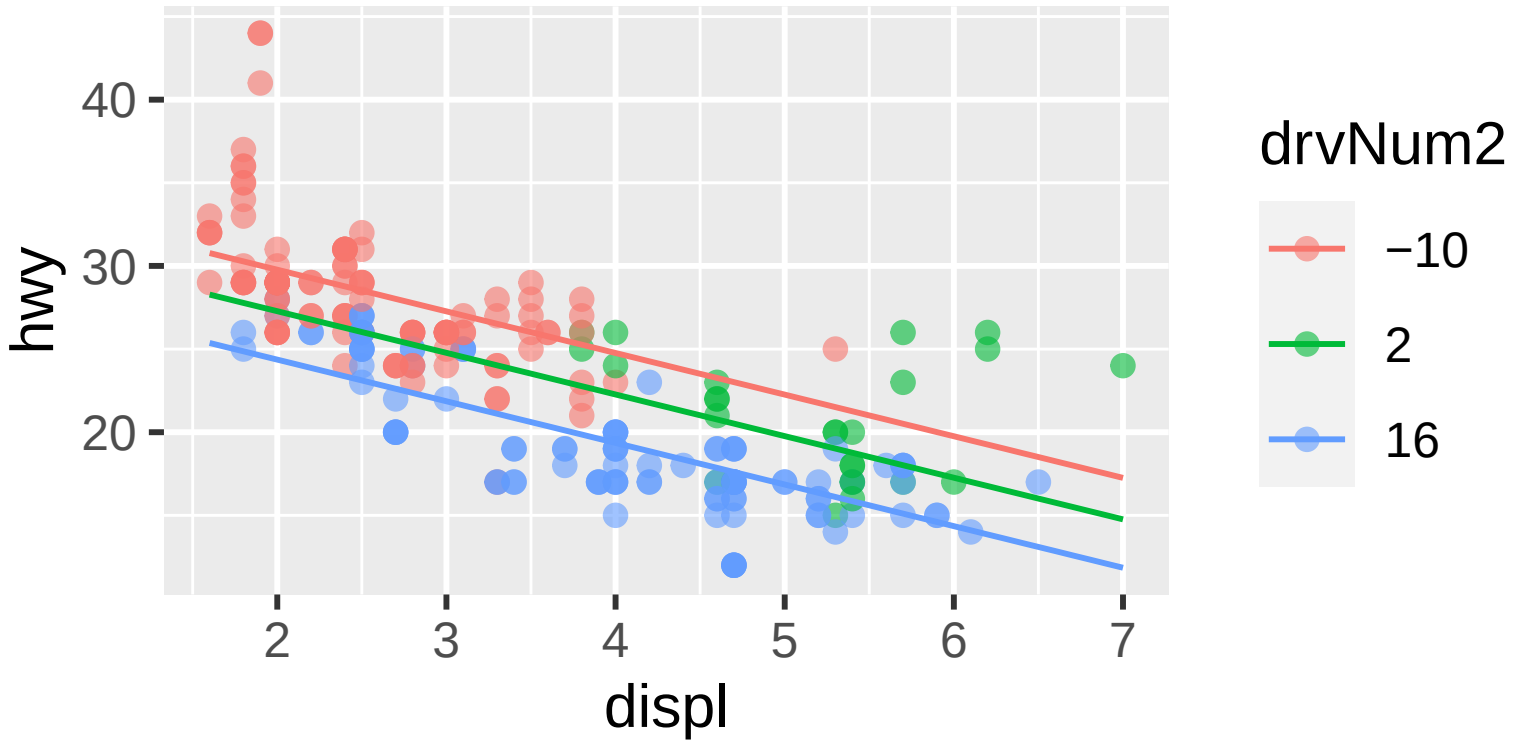


Variables categóricas

No parecen cambiar mucho las rectas según el factor (son casi paralelas), probemos con drvNum2:

```
LMdrvN2=lm(hwy~displ+drvNum2,data = mpg)
```

```
mpg %>% mutate(drvNum2=factor(drvNum2))%>% ggplot(aes(x=displ,y=hwy))+
  geom_point(aes(color=drvNum2), alpha=0.6)+
geom_line(data = New,mapping=aes(x=displ,y=Pred,color=drvNum2))
```



Variables categóricas

Nuevamente, parecen no cambiar mucho las pendientes por factor, pero notemos una cosa de los coeficientes estimados:

```
LMdrvN1$coefficients
```

```
## (Intercept)       displ     drvNum1
##   33.141380   -3.431597    3.318890
```

```
LMdrvN2$coefficients
```

```
## (Intercept)       displ     drvNum2
##  32.6994680  -2.5037044  -0.2079406
```

Notemos que cuando más raros los valores que le damos a la asignación numérica, menos vale el coeficiente correspondiente. En el primer modelo, los valores eran bajos y no afectaban mucho la pendiente correspondiente.

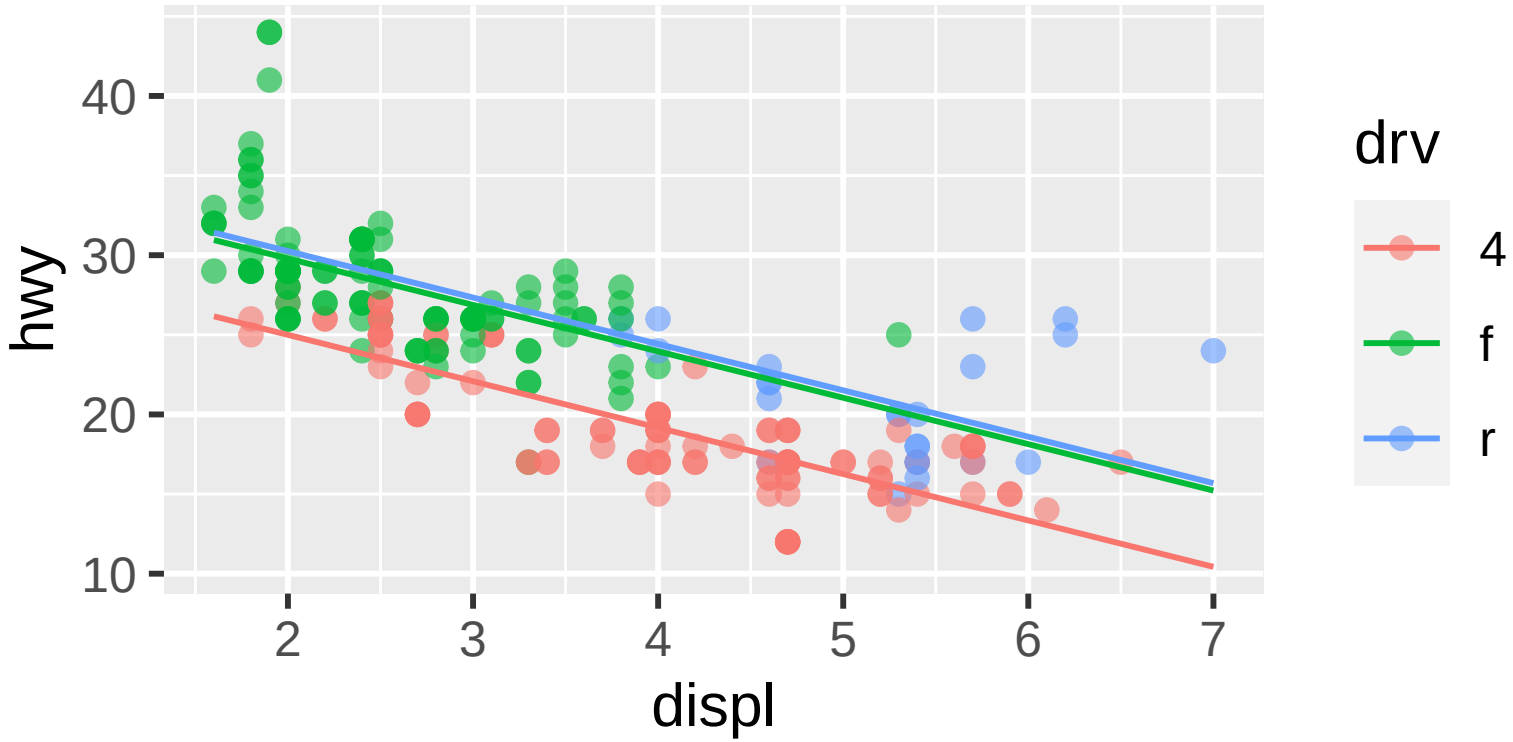
Hay otra cosa a notar. El modelo que queríamos tenía 6 parámetros (ordenada y pendiente para cada nivel de drv).

Variables categóricas

Estas asignaciones numéricas eran completamente a gusto del usuario. No debería depender de quién y cómo elige codificar las variables categóricas. R puede transformar las variables categóricas en numéricas de forma adecuada:

```
lmDRV=lm(hwy~displ+drv,data=mpg)
```

```
mpg %>% ggplot(aes(x=displ,y=hwy))+
  geom_point(aes(color=drv), alpha=0.6)+
geom_line(data = New,mapping=aes(x=displ,y=Pred,color=drv))
```



Variables categóricas

Nuevamente quedan paralelas las rectas. Veamos los coeficientes:

```
lmDRV$coefficients
```

```
## (Intercept)       displ        drvf        drvr
##   30.825437   -2.914085    4.790598    5.257865
```

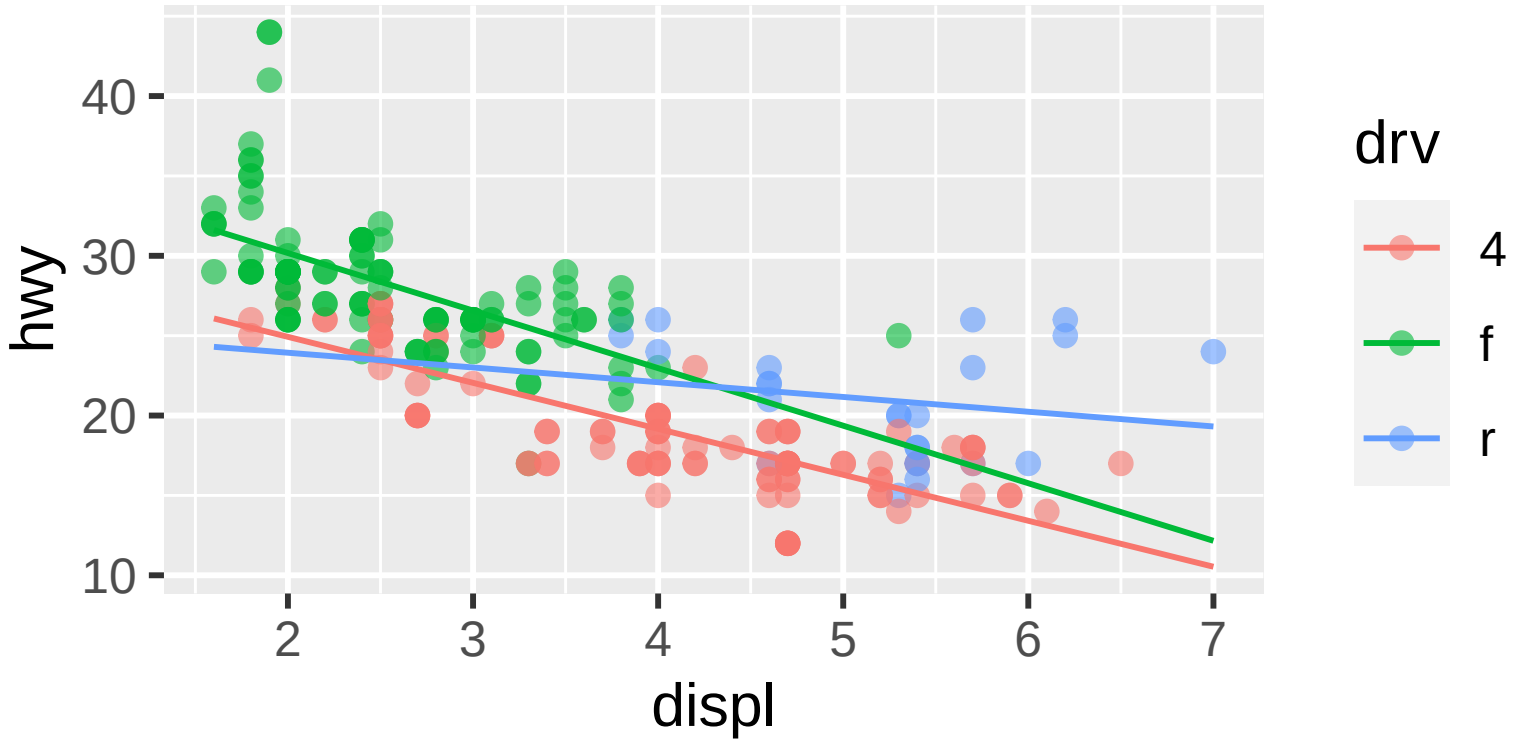
Notar que no obtuvimos los 6 parámetros deseados. Eso es porque no usamos el lenguaje correcto.

Variables categóricas

Para hacer una recta por nivel de drv, tenemos que usar el símbolo "*", que genera una interacción entre las variables independientes:

```
lmDRVint=lm(hwy~displ*drv,data = mpg)
```

```
mpg %>% ggplot(aes(x=displ,y=hwy))+
  geom_point(aes(color=drv), alpha=0.6)+
geom_line(data = New,mapping=aes(x=displ,y=Pred,color=drv))
```



Variables categóricas

Efectivamente, veamos que la cantidad de parámetros es correcta:

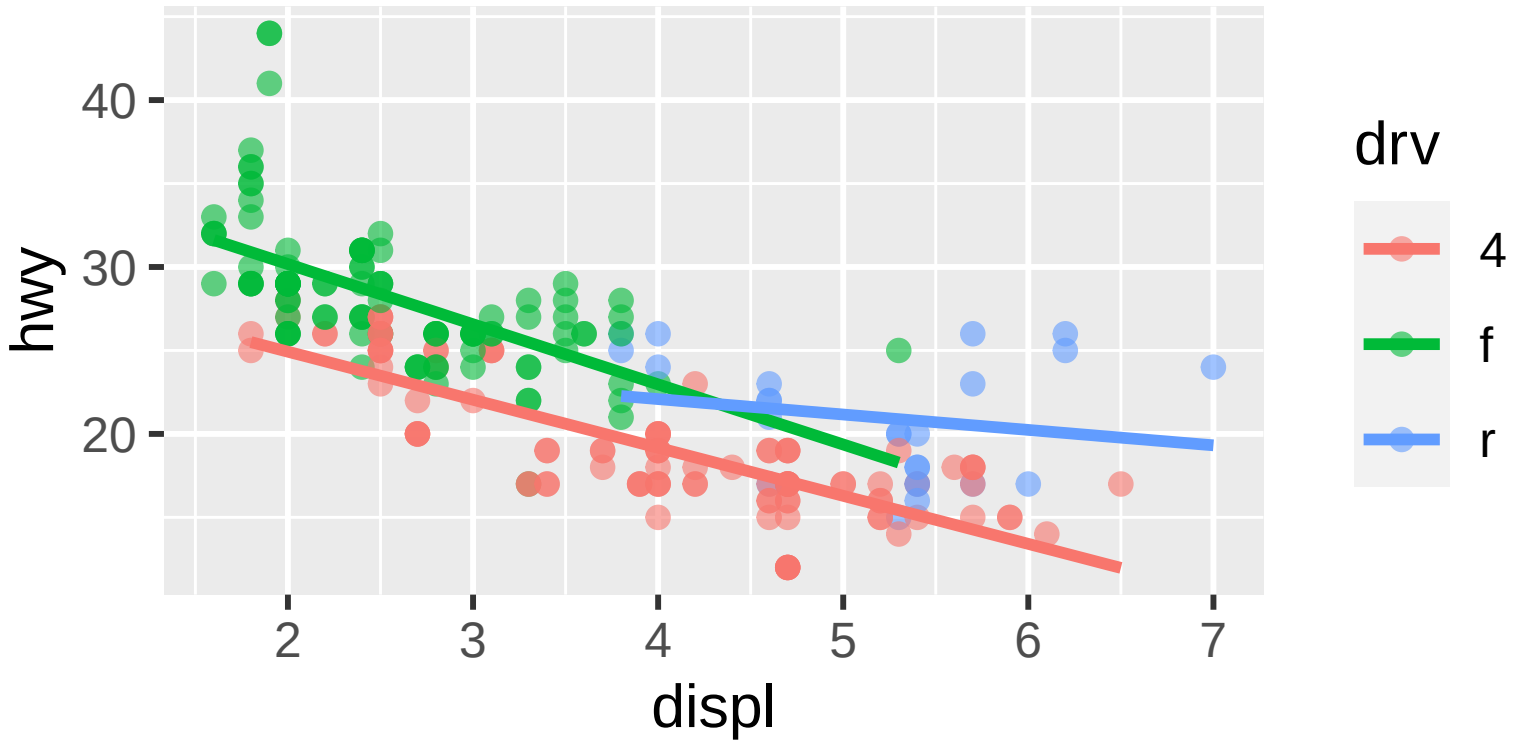
```
lmDRVint$coefficients
```

```
## (Intercept)       displ        drvf        drvr  displ:drvf  displ:drvr
##  30.6831131  -2.8784863   6.6949631  -4.9033952  -0.7243016   1.9550477
```

Variables categóricas

De hecho, el último gráfico lo hace automáticamente ggplot con el comando geom_smooth, siempre que haya una variable que agrupe (en este caso, color=drv):

```
mpg %>%
  ggplot(aes(x=displ,y=hwy,color=drv))+
  geom_point(alpha=0.6)+
  geom_smooth(method = "lm", se=F)
```



Variables categóricas

Más aún, comparemos los valores de R^2 de los modelos utilizados:

```
summary(LMdrvN1)$r.squared
```

```
## [1] 0.722319
```

```
summary(LMdrvN2)$r.squared
```

```
## [1] 0.7220466
```

```
summary(lmDRV)$r.squared
```

```
## [1] 0.7356267
```

```
summary(lmDRVint)$r.squared
```

```
## [1] 0.7460368
```

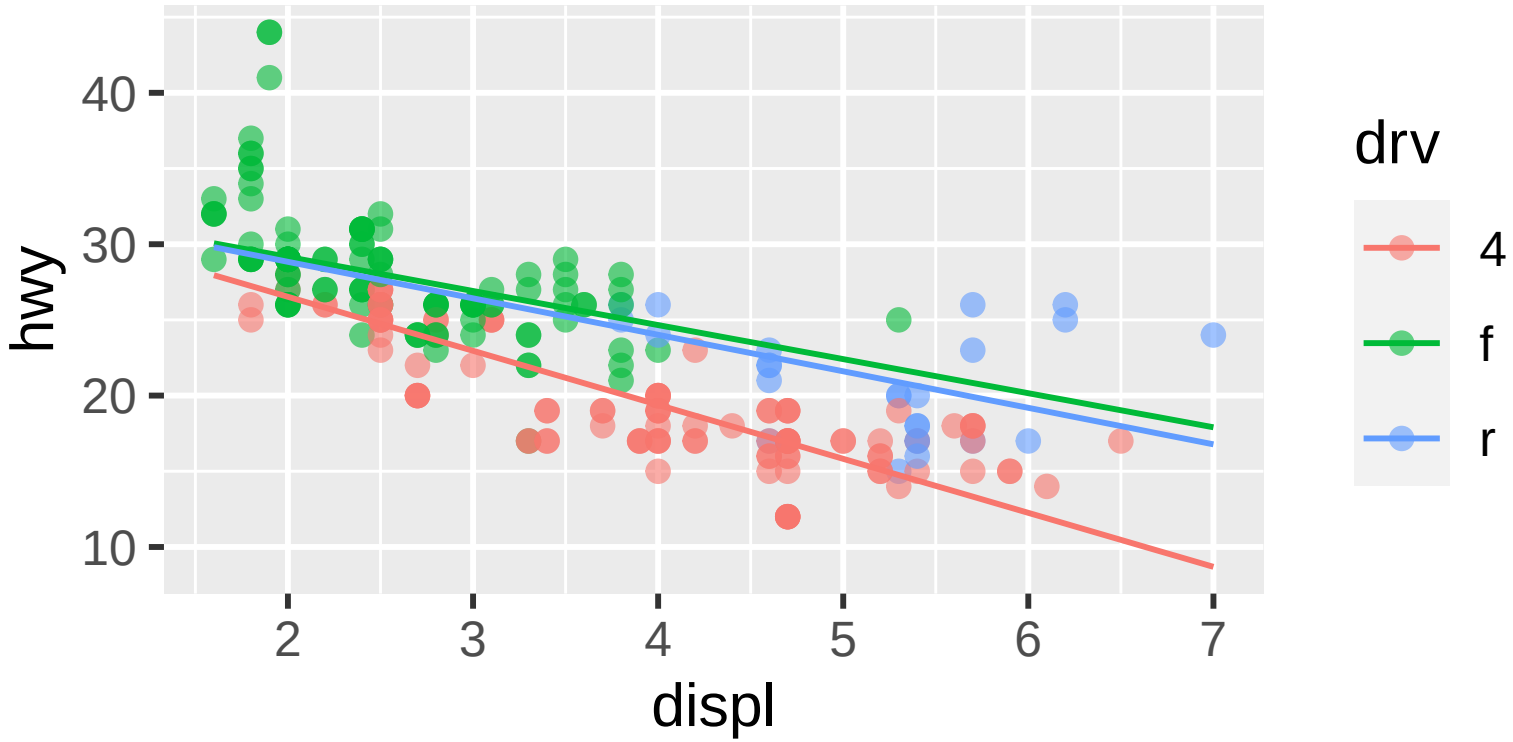
Moraleja del taller: A R le tirás un ladrillo (o sea, una variable categórica) y la para de taquito.

Variables categóricas

Si bien no era lo deseado al principio, hay otra opción para trabajar con variables categóricas, usando el símbolo ":", generando sólo una pendiente distinta por nivel de drv, pero con la misma ordenada:

```
lmDRVcol=lm(hwy~displ:drv,data = mpg)
```

```
mpg %>% ggplot(aes(x=displ,y=hwy))+
  geom_point(aes(color=drv), alpha=0.6)+
geom_line(data = New,mapping=aes(x=displ,y=Pred,color=drv))
```



Variables categóricas

Veamos los coeficientes. Son 4 ya que es una única ordenada y una pendiente por cada nivel de drv.

```
lmDRVcol$coefficients
```

```
## (Intercept)  displ:drv4  displ:drvf  displ:drvr
##   33.661529   -3.568049   -2.250700   -2.411788
```

En términos de modelos, tenemos las siguientes correspondencias:

- hwy~displ+drv (distinta ordenada, misma pendiente):

$$\text{hwy}_i \approx \begin{cases} a_1 + b \cdot \text{displ}_i & \text{si drv=4} \\ a_2 + b \cdot \text{displ}_i & \text{si drv=f} \\ a_3 + b \cdot \text{displ}_i & \text{si drv=r} \end{cases}$$

- hwy~displ:drv (misma ordenada, distinta pendiente):

$$\text{hwy}_i \approx \begin{cases} a + b_1 \cdot \text{displ}_i & \text{si drv=4} \\ a + b_2 \cdot \text{displ}_i & \text{si drv=f} \\ a + b_3 \cdot \text{displ}_i & \text{si drv=r} \end{cases}$$

- hwy~displ*drv (distinta ordenada y pendiente):

$$\text{hwy}_i \approx \begin{cases} a_1 + b_1 \cdot \text{displ}_i & \text{si drv=4} \\ a_2 + b_2 \cdot \text{displ}_i & \text{si drv=f} \\ a_3 + b_3 \cdot \text{displ}_i & \text{si drv=r} \end{cases}$$

Anscombe

Anscombe

Consideramos la base anscombe del paquete datasets:

```
require(datasets)
head(anscombe)
```

```
##   x1 x2 x3 x4   y1   y2    y3   y4
## 1 10 10 10  8 8.04 9.14  7.46 6.58
## 2  8  8  8  8 6.95 8.14  6.77 5.76
## 3 13 13 13  8 7.58 8.74 12.74 7.71
## 4  9  9  9  8 8.81 8.77  7.11 8.84
## 5 11 11 11  8 8.33 9.26  7.81 8.47
## 6 14 14 14  8 9.96 8.10  8.84 7.04
```

Son 4 combinaciones de vectores x e y que analizaremos a continuación.

Anscombe

Si calculamos las rectas que los representan, todas las combinaciones dan coeficientes casi idénticos:

```
LM1=lm(x1~y1,data = anscombe);LM2=lm(x2~y2,data = anscombe)
LM3=lm(x3~y3,data = anscombe);LM4=lm(x4~y4,data = anscombe)
LM1$coefficients
```

```
## (Intercept)          y1
##  -0.9975311   1.3328426
```

```
LM2$coefficients
```

```
## (Intercept)          y2
##  -0.9948419   1.3324841
```

```
LM3$coefficients
```

```
## (Intercept)          y3
##   -1.000315    1.333375
```

```
LM4$coefficients
```

```
## (Intercept)          y4
##   -1.003640    1.333657
```

Anscombe

Más aún, dan casi iguales los valores de R^2:

```
summary(LM1)$r.squared
```

```
## [1] 0.6665425
```

```
summary(LM2)$r.squared
```

```
## [1] 0.666242
```

```
summary(LM3)$r.squared
```

```
## [1] 0.666324
```

```
summary(LM4)$r.squared
```

```
## [1] 0.6667073
```

Anscombe

Sin embargo, notemos cómo quedan las representaciones de los gráficos de x e y. Primero lo pasamos a formato long, agregando los predichos y los residuos:

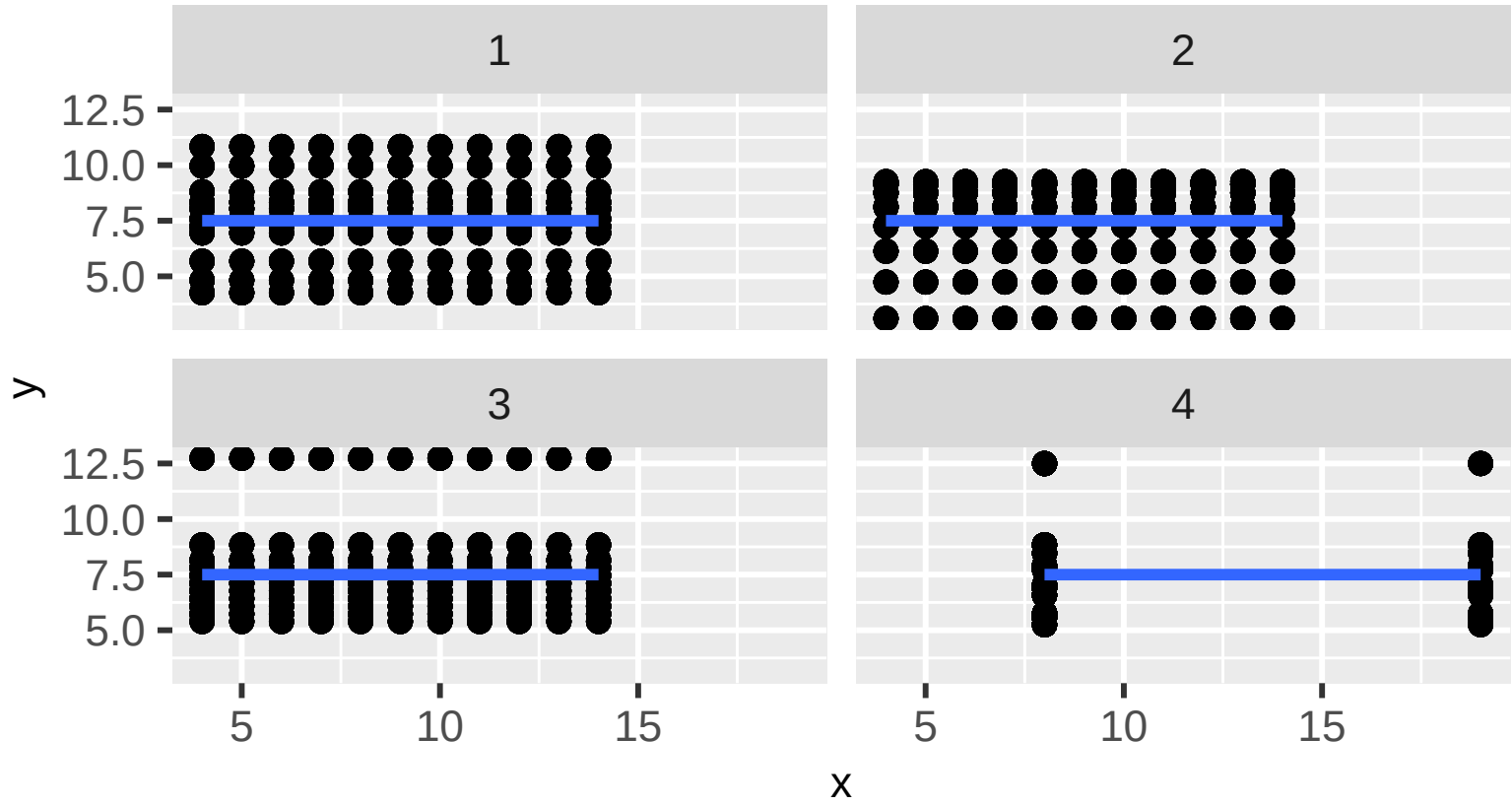
```
require(stringr);require(tidyr);
Dx=anscombe %>% select(x1,x2,x3,x4) %>%
  gather(key="Subject",value="x",x1,x2,x3,x4) %>%
  mutate(Subject=str_replace_all(Subject,"x",""))
Dy=anscombe %>% select(y1,y2,y3,y4) %>%
  gather(key="Subject",value="y",y1,y2,y3,y4)%>%
  mutate(Subject=str_replace_all(Subject,"y",""))
anscombe$p1=LM1$fitted.values;anscombe$p2=LM2$fitted.values;
anscombe$p3=LM3$fitted.values;anscombe$p4=LM4$fitted.values;
Dp=anscombe %>% select(p1,p2,p3,p4) %>%
  gather(key="Subject",value="fit",p1,p2,p3,p4) %>%
  mutate(Subject=str_replace_all(Subject,"p",""))
D=merge(Dx,Dy);D=merge(D,Dp);D$Res=D$y-D$fit;head(D)
```

```
##   Subject  x    y       fit        Res
## 1       1 10 8.04  9.718523 -1.6785233
## 2       1 10 8.04  8.265725 -0.2257249
## 3       1 10 8.04  9.105416 -1.0654157
## 4       1 10 8.04 10.744812 -2.7048121
## 5       1 10 8.04 10.105048 -2.0650477
## 6       1 10 8.04 12.277581 -4.2375811
```

Anscombe

Ahora pasamos a ver los scatterplots, notar que se verifica que las rectas son similares para cada combinación, a pesar de que los datos no se comportan de la misma forma:

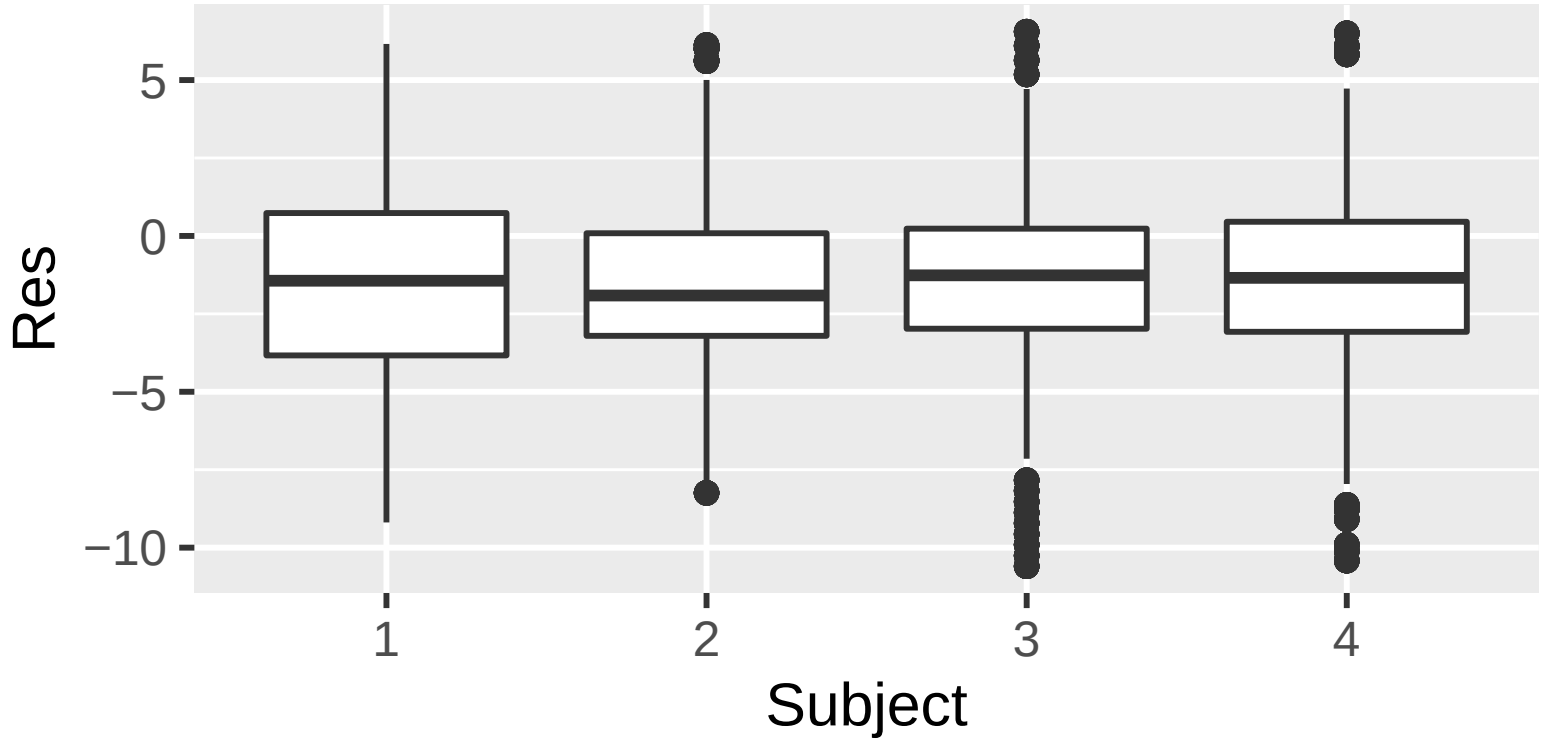
```
D %>%ggplot(aes(x=x,y=y))+geom_point()+
  facet_wrap(Subject~.)+geom_smooth(method="lm",se=F)+
  theme(axis.title = element_text(size=8),
        axis.text = element_text(size=8),
        strip.text = element_text(size=8))
```



Anscombe

Vamos a hacer una vista a la distribución de los residuos, para notar las diferencias:

```
D %>%
  ggplot(aes(x=Subject,y=Res))+
  geom_boxplot()
```



Anscombe

Conclusión: Muchas veces trabajando en estadística se buscan resumir los datos en unos pocos valores para no tener que mirarlos todos.

Sin embargo, muchas veces se corre el riesgo de resumir demasiado, perdiendo matices de la naturaleza de los datos.

Por eso, nunca nos olvidemos de pensar. En épocas de mucho poder computacional, tratamos a veces de automatizar demasiado, pero perdiendo de vista que tenemos la capacidad de ver cosas que la computadora no.

Vivimos en una gran época, en la que tanto nuestros cerebros como nuestras computadoras son herramientas para resolver problemas. Siempre usemos ambas y complementémoslas.