Trabajo 3

Alejandro García Montoro

2 de junio de 2016

Cómo se escribe nua fórmula para que el modelo devuelva los pesos que nos interesan, modelando lo que queremos:

- Modelo general: `m1 <- lm(Y~.,data=datos)`
- Modelo con columnas especificadas: `m2 <- lm(Y~X1+X2+X3,data=datos)`
- Modelos más complejos: `m1 <- lm(Y~I(X^2)+X2,data=datos)`
- Combinaciones polinómicas: `log gpoly()`
- Para predecir cosas dependientes (p.ej., dos atributos combinados): `lm(Y~X1+X2+X1:X2,data=datos)`, que es equivalente a escribir `lm(Y~X1*X2,data=datos)`.

Ejercicio 1

Carguemos primero los datos necesarios para realizar el ejercicio y echémosle un primer vistazo a la base de datos:

```
# Cargamos la librería necesaria para usar la base de datos Auto
# Para usarla, hay que instalar con la orden
# install.packages('ISLR')
library(ISLR)

# Usamos Auto por defecto, evitando así poner el prefijo Auto$
# siempre que queramos acceder a una característica de esa base de datos
attach(Auto)
```

Si ejecutamos las órdenes siguientes

```
class(Auto)
dim(Auto)
colnames(Auto)
```

podemos obtener información de la forma que tiene nuestra base de datos. Vemos así que tiene forma de data.frame, con 392 filas y 9 columnas, cuyos nombres son los siguientes: mpg, cylinders, displacement, horsepower, weight, acceleration, year, origin, name.

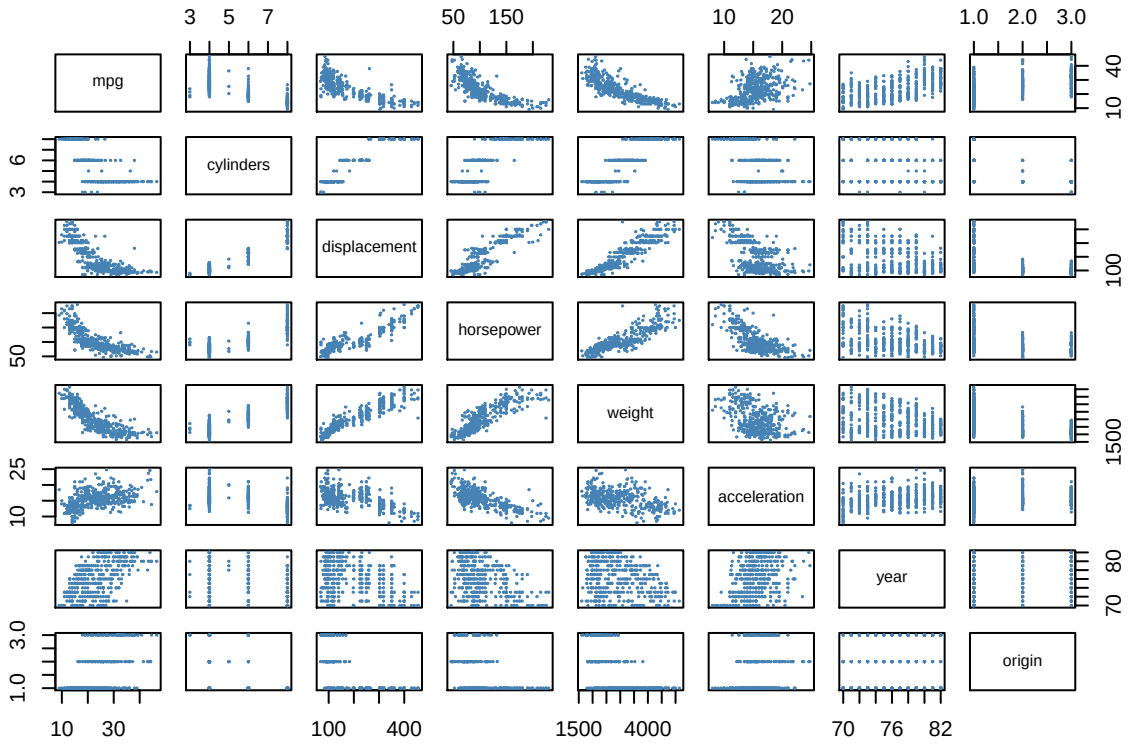
Podemos eliminar la última columna, que es el nombre del vehículo y la única variable no numérica, para poder trabajar con comodidad más adelante:

```
# Eliminamos la última columna
Auto <- Auto[,seq(ncol(Auto)-1)]
```

Apartado a

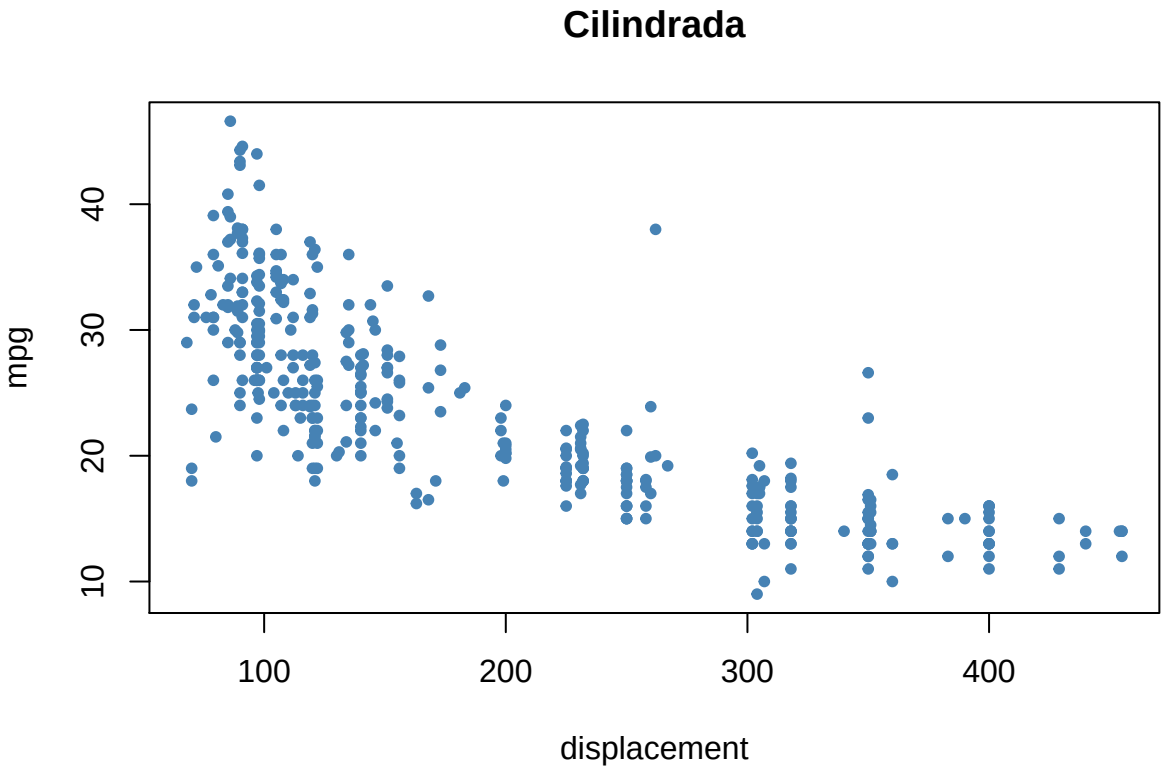
Para visualizar las dependencias entre `mpg` y las otras características podemos usar las funciones `pairs()` y `boxplot()`:

```
# Visualizamos la relación entre todos los pares de variables
pairs(Auto, pch=20, cex=0.2, col="steelblue")
```

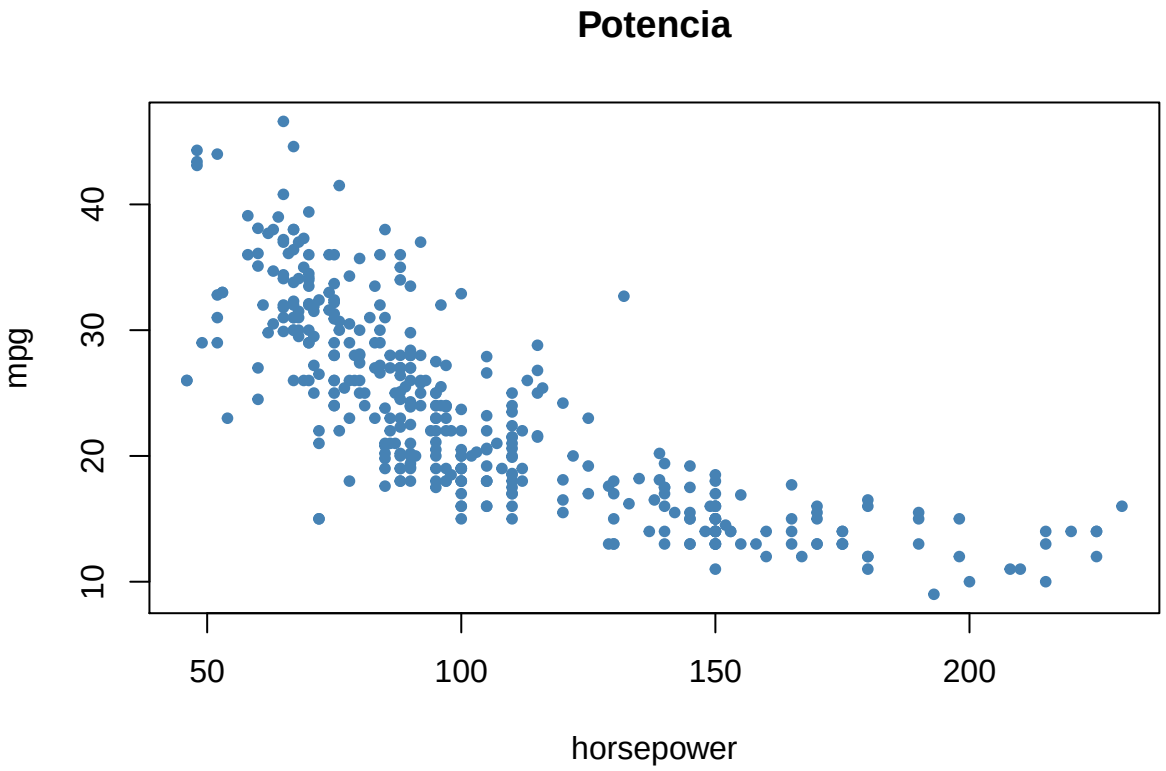


Podemos visualizar más de cerca las variables que parecen más relevantes para predecir la variable `mpg`:

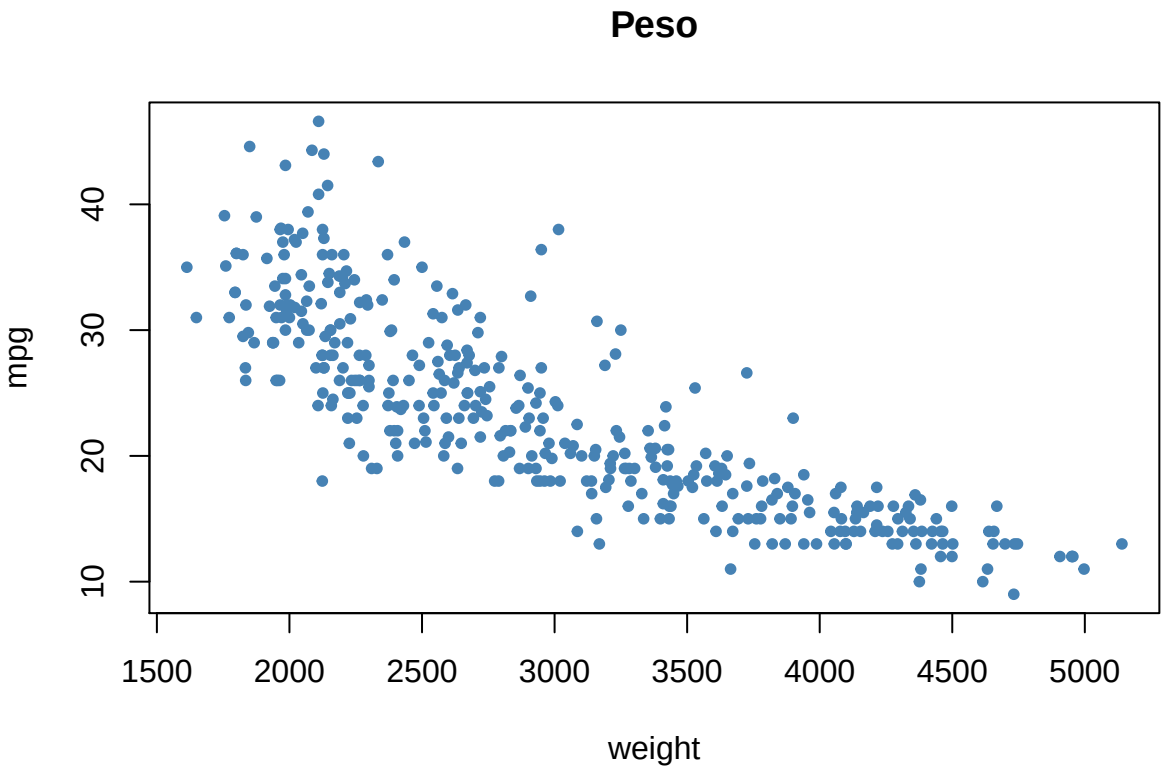
```
# Plot para mpg-displacement
plot(displacement, mpg, pch=20, col="steelblue",
     main="Cilindrada")
```



```r
# Plot para mpg-horsepower
plot(horsepower, mpg, pch=20, col="steelblue",
     main="Potencia")
```



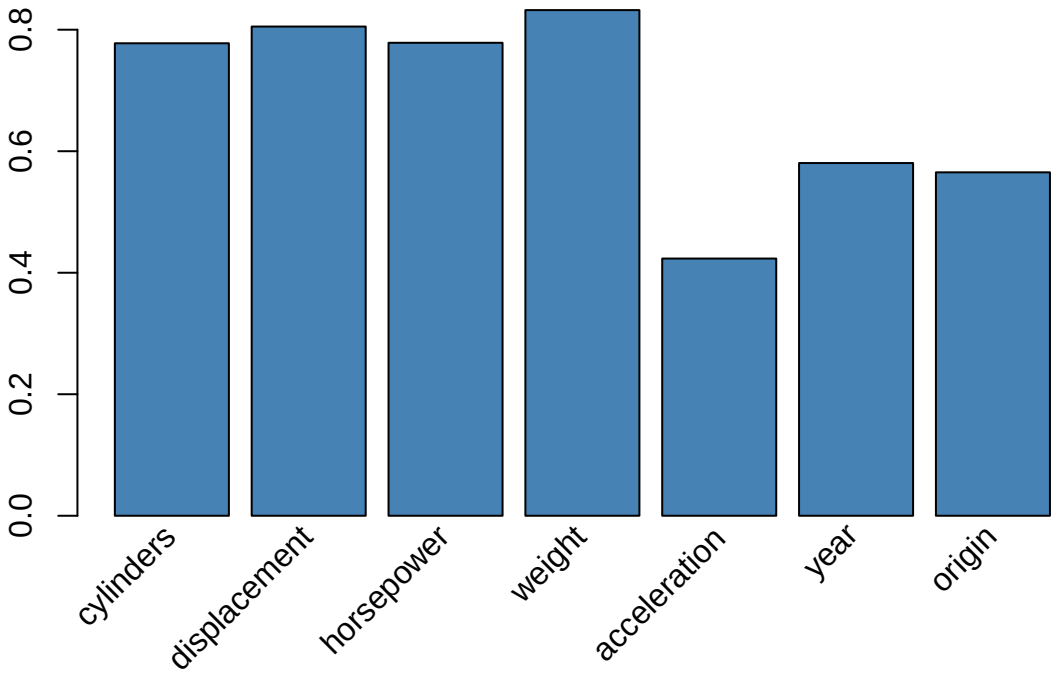

```r
# Plot para mpg-weight
plot(weight, mpg, pch=20, col="steelblue",
     main="Peso")
```



Apartado b

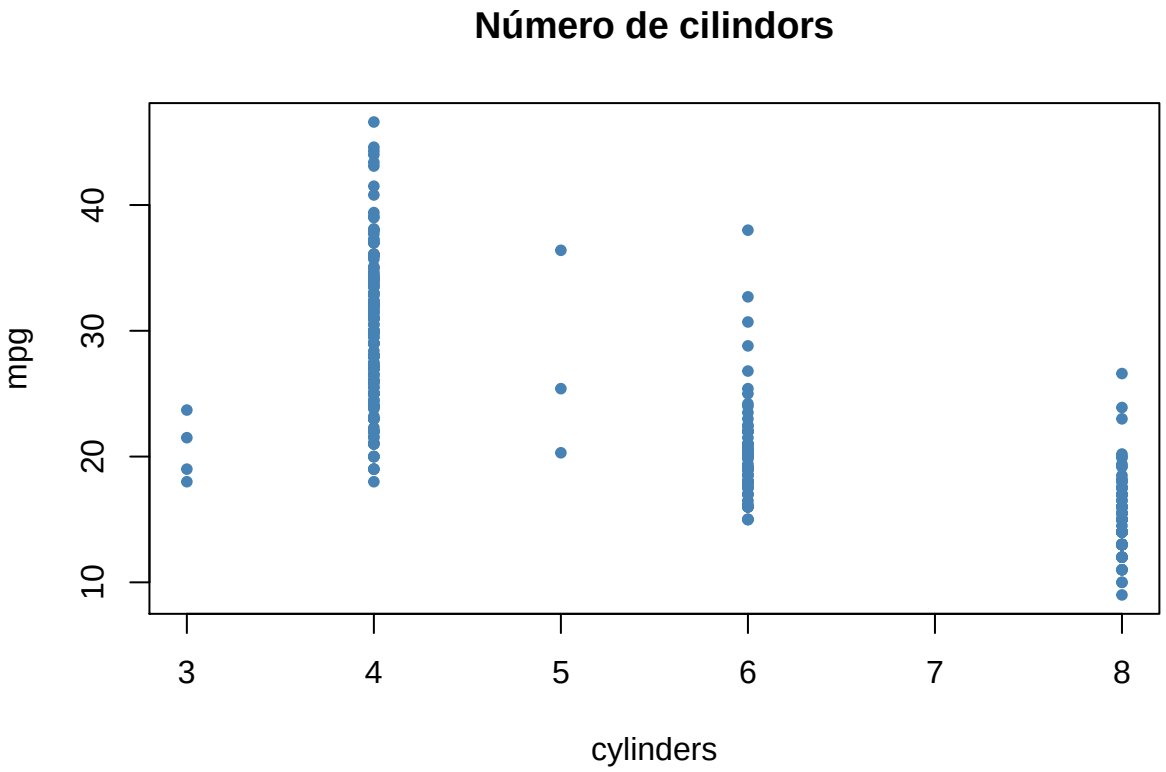
Podemos estudiar de forma numérica la correlación entre `mpg` y las demás variables:

```
# Tomamos los valores absolutos de la correlación entre mpg y todas las demás variables,
# sin incluirse a sí misma
corr <- abs(cor(Auto))["mpg",-1]

# Visualizamos el grado de correlación en un gráfico de barras.
# Creamos el gráfico.
bp <- barplot(corr, axes = FALSE, axisnames = FALSE, col = "steelblue",
              main="Correlación entre mpg y las demás variables")

# Añadimos el texto, girado 45 grados.
text(bp, par("usr")[3]-0.02, labels = colnames(Auto[-1]),
     srt = 45, adj = 1, xpd = TRUE)

# Dibujamos los ejes.
axis(2)
```



Como vemos son `cylinders`, `displacement`, `horsepower` y `weight` las variables que más correlación presentan con `mpg`. Sin embargo, vamos a estudiar las tres últimas; la correlación entre la primera y `mpg` es un dato arbitrario que tiene más que ver con la forma de la variable que con la relación entre ambas. Esto se ve claro si ampliamos el gráfico `mpg-cylinders`:

```
# Plot para mpg-cylinders
plot(cylinders, mpg, pch=20, col="steelblue",
     main="Número de cilindors")
```



Seleccionamos las variables escogidas:

```
selec <- c("displacement", "horsepower", "weight")
```

Apartado c

Vamos a crear dos vectores que usaremos para indexar las muestras de entrenamiento y de test:

```
# Vector de índices para la muestra de entrenamiento (80%)
trainIdx  <- sample(nrow(Auto), size=0.8*nrow(Auto))

# Vector de índices para la muestra de test
testIdx   <- setdiff(1:nrow(Auto), trainIdx)
```

Apartado d

Creamos la variable booleana `mpg01`. En vez de usar los valores simbólicos 1 y -1, usamos unos mucho más expresivos: `True` y `False`. Una vez añadida la variable, partimos los datos en las muestras de entrenamiento y test con los índices calculados anteriormente:

```
# Creamos una nueva variable booleana, mpg01, en función de la mediana,
# y la añadimos a la base de datos
mpg01 <- ifelse(mpg > median(mpg), T, F)
Auto <- data.frame(mpg01, Auto)

# Obtenemos las muestras de entrenamiento y de test
Auto.train <- Auto[trainIdx,]
Auto.test  <- Auto[testIdx,]
```

Regresión lineal

Para hacer la regresión lineal vamos a usar la función `lm`, que devuelve un modelo lineal `Y~X`, donde `Y` es la variable a predecir y `X` el conjunto de variables predictoras.

Este modelo lineal lo ajustaremos con la muestra de test:

```
# Ajustamos el modelo con los datos de entrenamiento
Auto.mod.lin = lm(mpg01~displacement+horsepower+weight, data=Auto.train)
```

Para ver la efectividad del modelo, intentamos predecir la variable `mpg01` con la función `predict` sobre la muestra de test:

```
# Usamos el modelo lineal ajustado para predecir con la muestra de test
Auto.mod.lin.pred <- predict(Auto.mod.lin, Auto.test)
```

Esto nos devuelve en la variable `Auto.mod.lin.pred` la probabilidad de acierto para cada punto de la muestra de test. Por tanto, vamos a asignar el valor `True` a aquellos puntos donde la probabilidad sea mayor al 50% y `False` a los demás:

```
# Si la predicción tiene probabilidad mayor que el 50%,
# asignamos el valor Verdad; en otro caso, asignamos el valor Falso
Auto.mod.lin.mpg01 <- ifelse(Auto.mod.lin.pred > 0.5, T, F)
```

Para calcular el error ya basta, tan sólo, contar el porcentaje de puntos mal clasificados en la muestra de test:

```
# Calculamos el error como el porcentaje de muestras mal clasificadas
Auto.mod.lin.E_test <- mean(Auto.test$mpg01 != Auto.mod.lin.mpg01)
```

De aquí obtenemos que el error producido por este modelo es del 8.8607595%. Podemos ver exactamente cuántos falsos positivos y falsos negativos tenemos; la siguiente tabla, cuyas columnas son los valores predecidos y cuyas filas los reales, resume la efectividad del método:

```
tab <- table(Real=Auto.test$mpg01, Predecido=Auto.mod.lin.mpg01)
kable(tab, caption="Regresión lineal")
```

Table 1: Regresión lineal

	FALSE	TRUE
FALSE	36	6
TRUE	1	36

Vecino más cercano

```
normalize <- function(x) {
    return((x - min(x))/(max(x) - min(x)))
}
```

```
library(class)
Auto.norm <- as.data.frame(lapply(Auto[,selec], normalize))
Auto.norm.train <-

knn.pred <- knn(Auto.train, Auto.test, Auto.train$mpg01, k=3)

mean(Auto.test$mpg01 != knn.pred)
```

```
## [1] 0.1139241
```

```
# install.packages("e1071")
library(e1071)
#Full Data set can be used for cross validation
knn.cross <- tune.knn(x = Auto[,selec], y = Auto$mpg01, k = 1:20, tunecontrol = tune.control(sampling =
#Summarize the resampling results set
plot(knn.cross)
```