Trabajo3

lm regresión logística por defecto aprende sobre todos, nos devuelve los pesos para cada una de las variables en el orden en el que están disponibles en el dataset, el w0 es el que llaman el Intercept.

Aquellos que tienen un umbral alto podríamos eliminarlos porque no los creemos, eliminar las columnas que no queremos. Pero aquí lo que podemos hacer de otra forma más cómoda introduciendo una fórmula.

modelo1 <- lm(ym, data=datos) modelo2 <- lm(ym ~ X1+X2+X3, data=datos) así construimos una solución que sólo tiene en cuenta las variables 1,2 y 3.

Una de las bases de datos con las que vamos a trabajar tiene "más pinta de cuadrática" entonces nos podemos plantear hacerlo del siguiente modo:

modelo3 <- lm(y ~ I(x2^2)+x2) tenemos que poner la palabra reservada I para que interprete correctamente la potencia.

poly() para hacer combinaciones polinómicas.

Puede ocurrir que haya sinergia entre atributos, es decir, que no sean independientes unos de otros, entonces vamos a ver cómo escrbimos una fórmula para que el modelo se ajuste como queremos a los datos que le pasamos.

modeloSinergico <- lm(y ~ x1*x2, data = datos) <=> lm(y ~ x1+x2+x1:x2, data = datos)

Despues de aprender un modelo podemos poner names(modelo) y nos dice los elementos que podemos consultar del modelo.

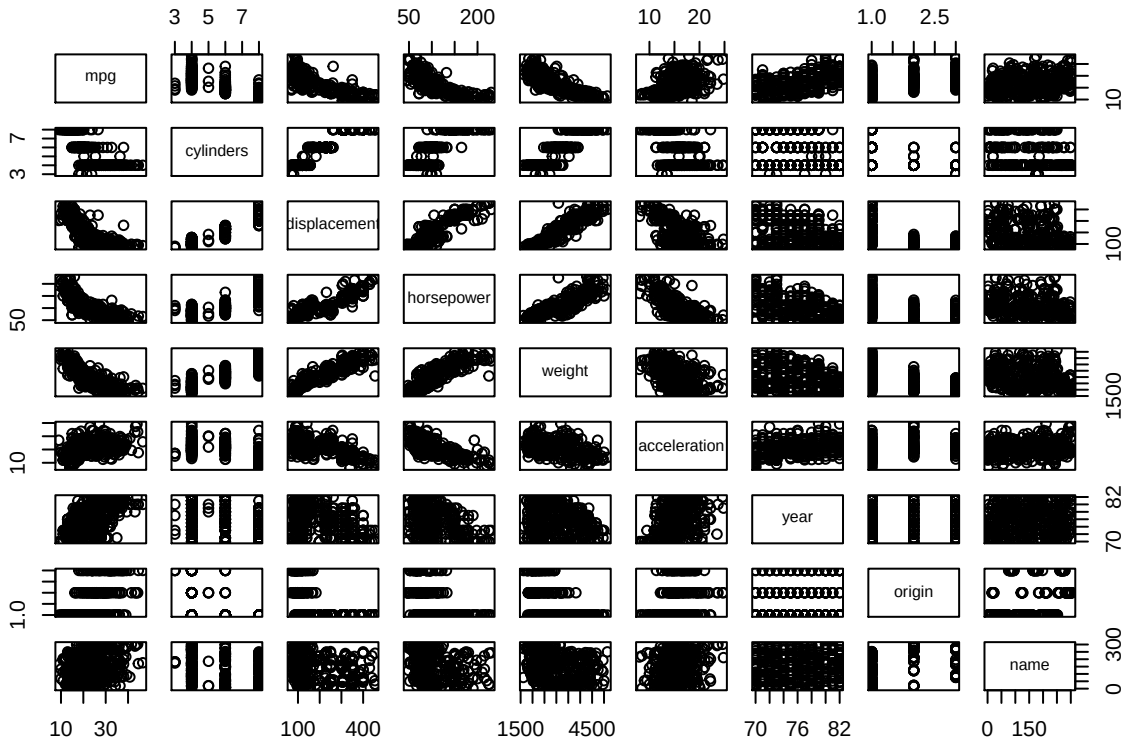
Lo convertimos a factor, asfactor

tune.knn va probando distintos parámetros y te devuelve el que mejor funciona.

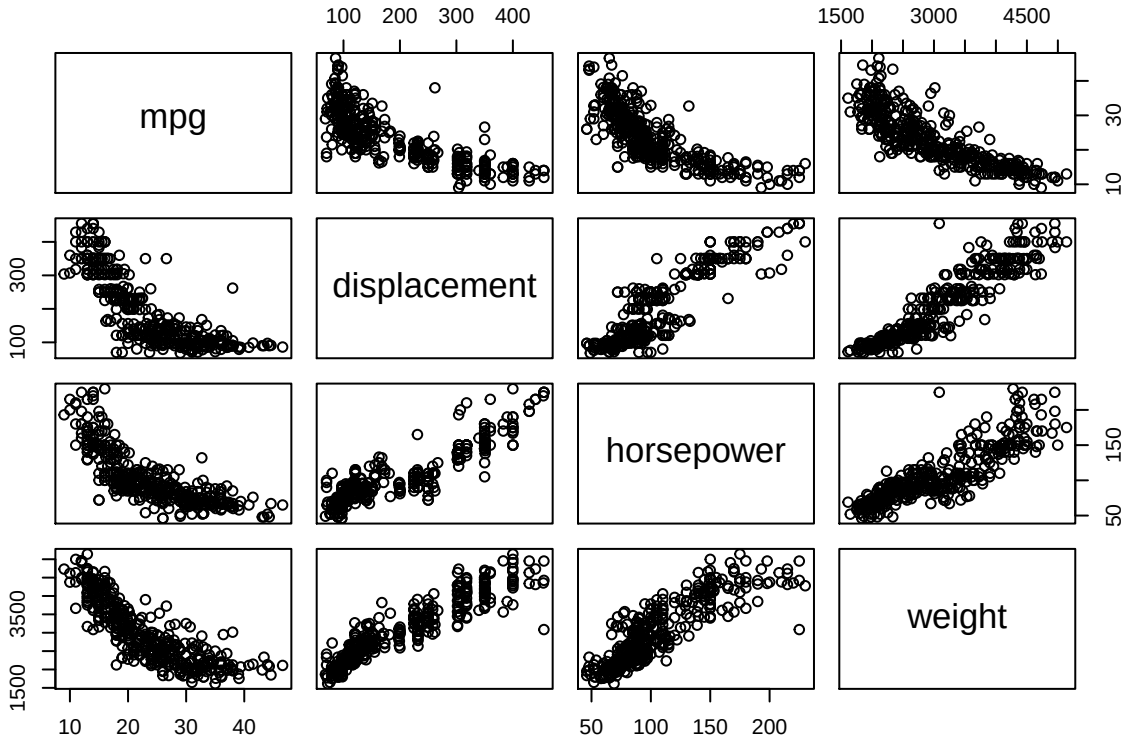
APARTADO 1

a) Usar las funciones de R pairs() y boxplot() para investigar la dependencia entre mpg y las otras características. ¿Cuáles de las otras características parece más útil para predecir mpg? Justificar la respuesta.

```
pairs(Auto)
```

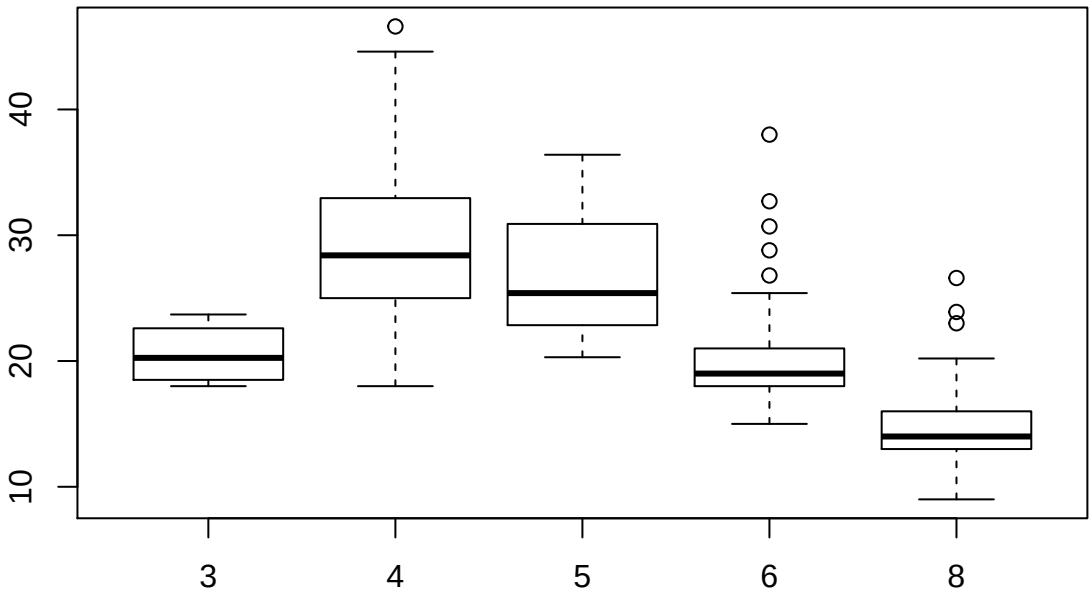



```
pairs(mpg ~ displacement + horsepower + weight)
```

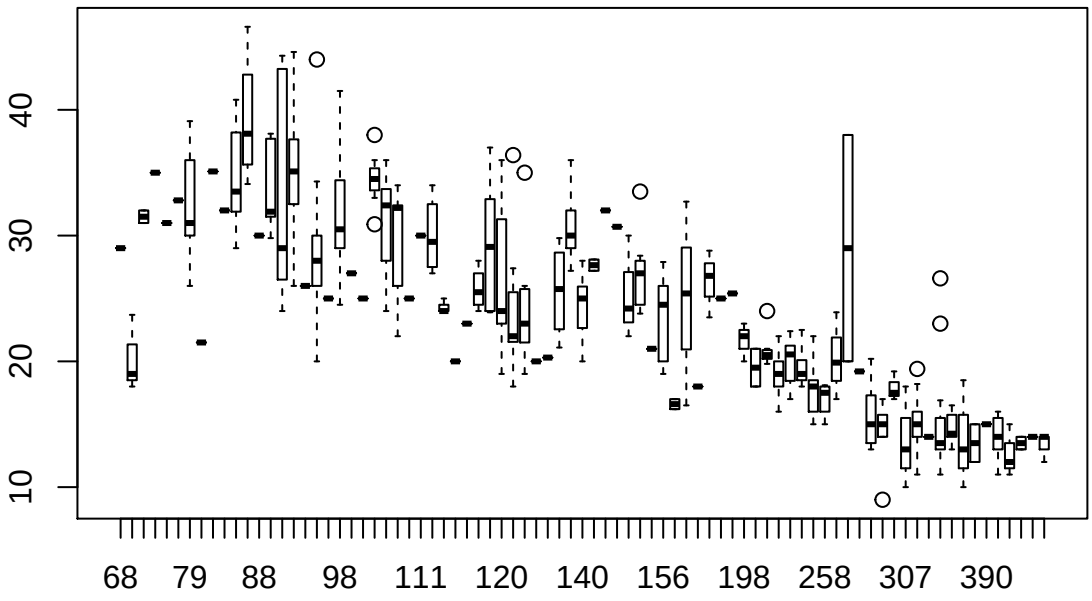


Como podemos ver las "gráficas de dependencias" de mpg con respecto a *displacement*, *horsepower* y *weight* son las gráficas que presentan un patrón más parecido entre ellas indicando que mpg tiene una relación fuerte con estas variables ya que se ajusta a ellas de un modo similar. Por ejemplo si vemos la gráfica con respecto a *acceleration* lo que tenemos es una nube de puntos mucho más dispersa. En cambio estas gráficas si que tienen un aspecto de ser ajustables linealmente.

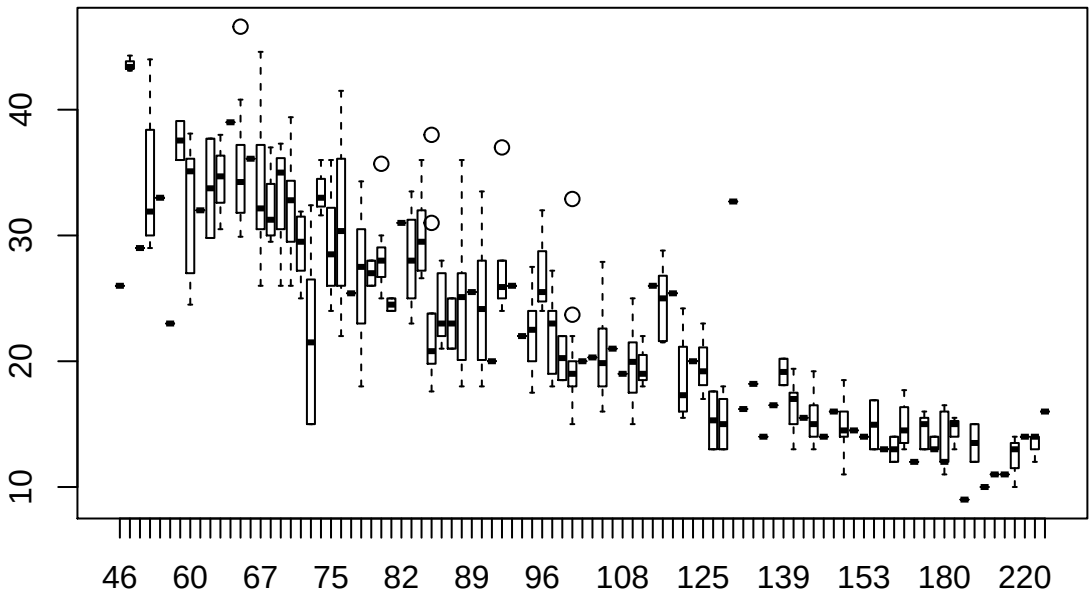
```
boxplot(mpg ~ cylinders)
```



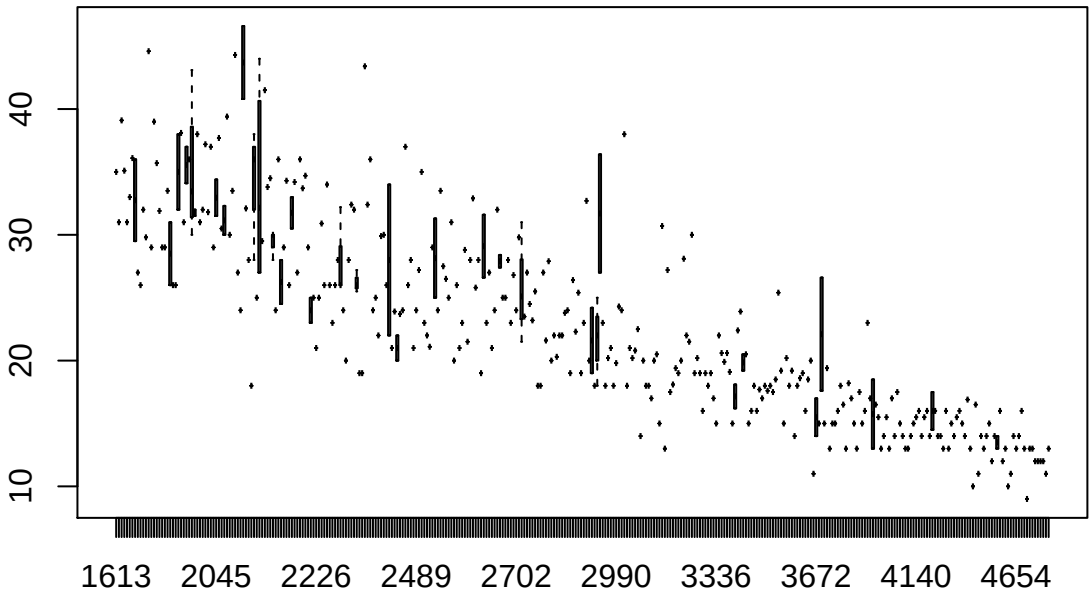

```
boxplot(mpg ~ displacement)
```



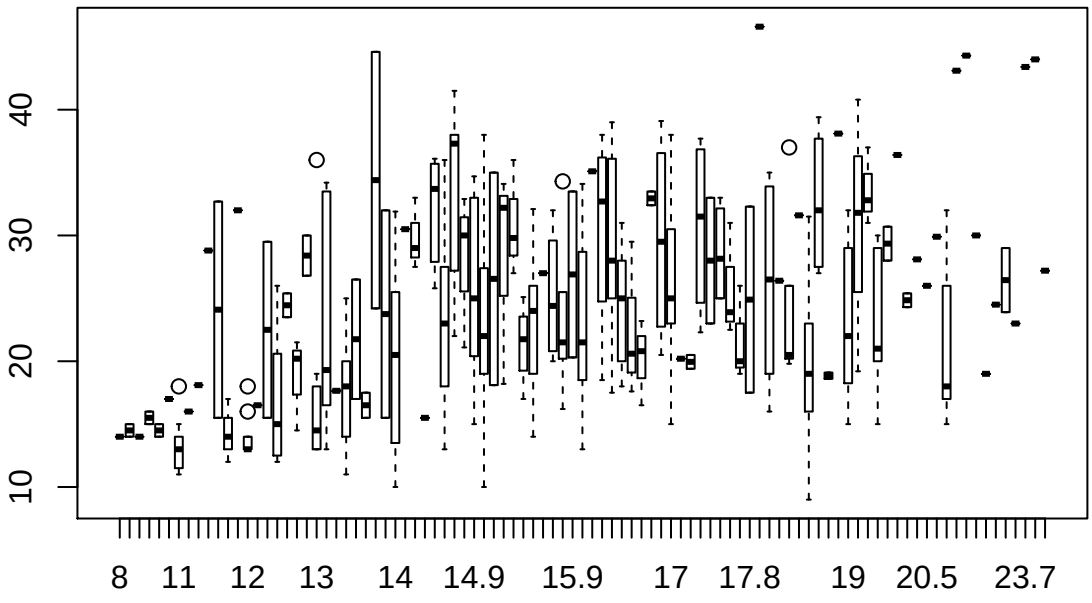

```
boxplot(mpg ~ horsepower)
```



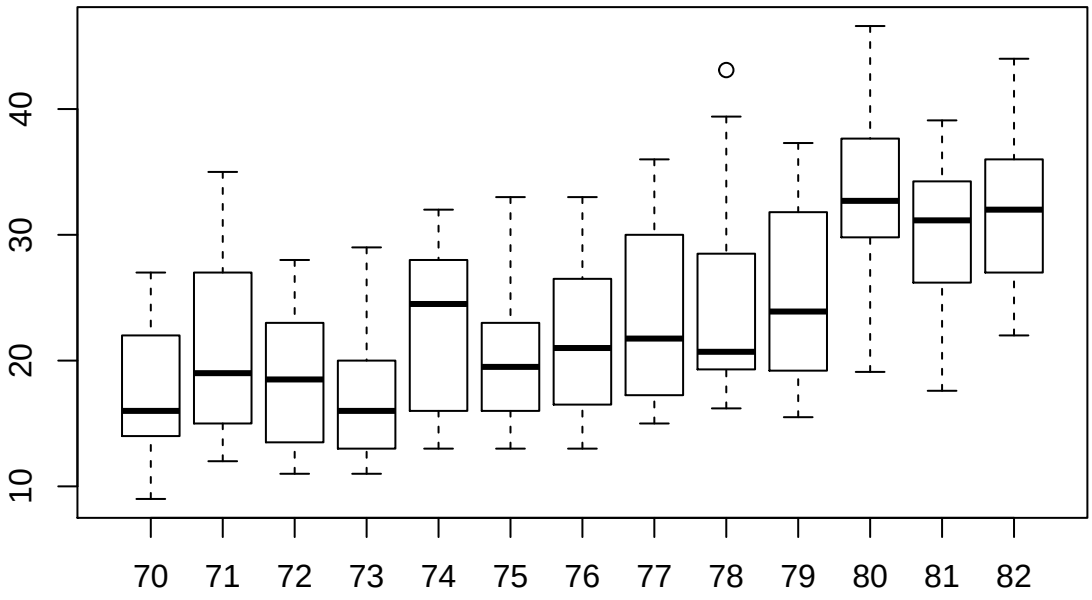

```
boxplot(mpg ~ weight)
```



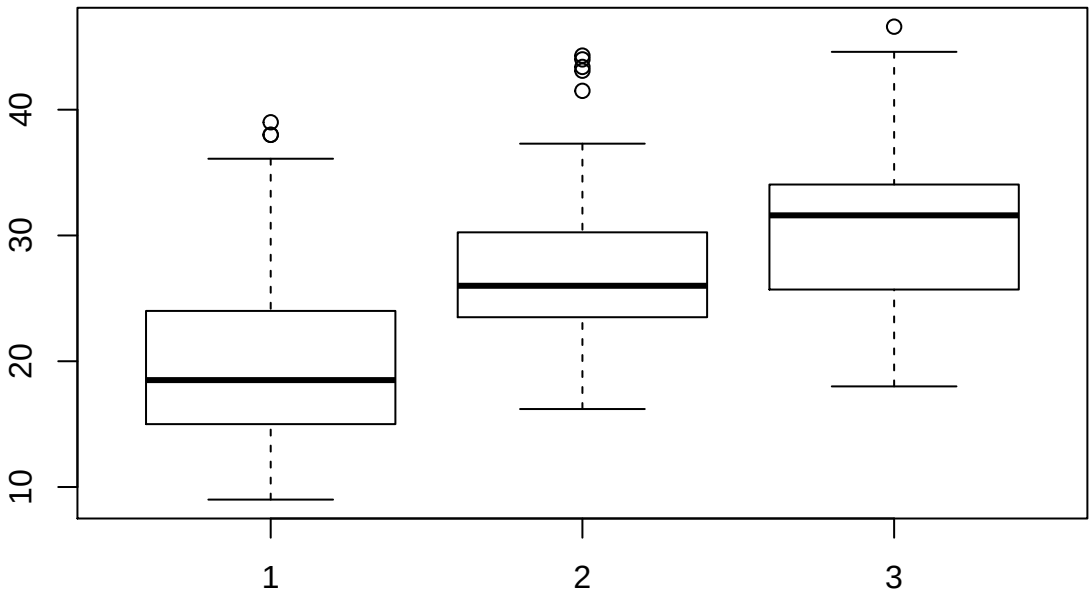

```
boxplot(mpg ~ acceleration)
```



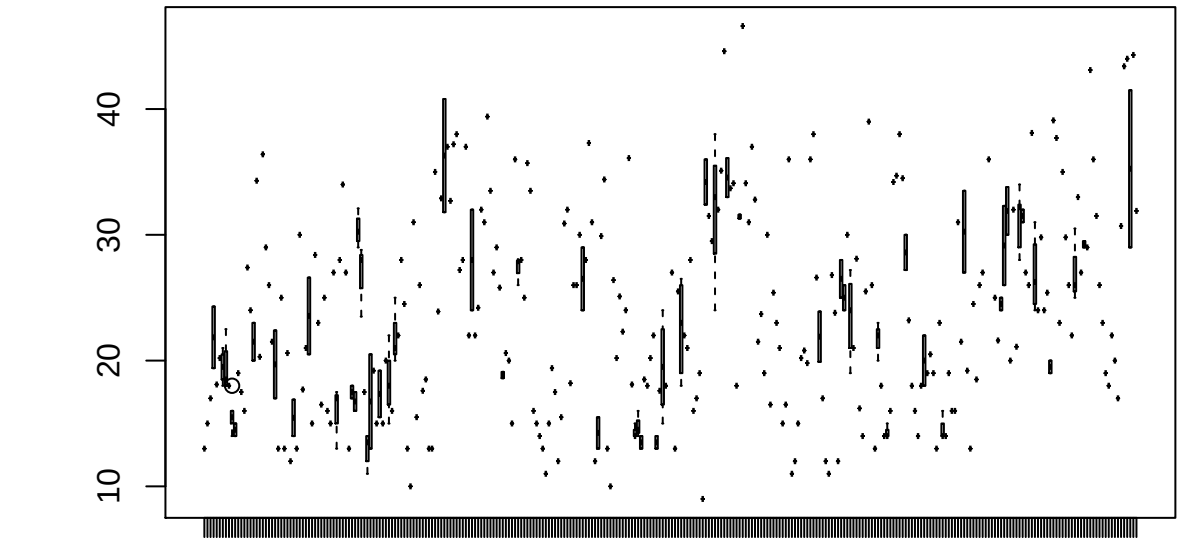

```
boxplot(mpg ~ year)
```




```
boxplot(mpg ~ origin)
```




```
boxplot(mpg ~ name)
```



b) Seleccionar las variables predictoras que considere más relevantes.

```
datos = Auto[,c("displacement", "horsepower", "weight")]
```

d) Crear una variable binaria, mpg01, que será igual 1 si la variable mpg contiene un valor por encima de la mediana, y -1 si mpg contiene un valor por debajo de la mediana. La mediana se puede calcular usando la función median(). (Nota: puede resultar útil usar la función data.frames() para unir en un mismo conjunto de datos la nueva variable mpg01 y las otras variables Auto).

```
mpg_median = median(mpg)
mpg1 = sapply(mpg, function(x) if (x < mpg_median) return(-1) else return(1))
```

c) Particionar el conjunto de datos en un conjunto de entrenamiento (80%) y otro de test (20%). Justificar el procedimiento usado.

Hemos realizado en primer lugar el apartado d ya que lo que buscamos es realizar un particionado de la muestra de modo que las etiquetas tengan una distribucion lo más uniforme posible en ambas particiones, así nuestro ajuste será mejor.

```
n_pos = sum(mpg1 == 1)
n_neg = sum(mpg1 == -1)

n_pos_train = ceiling(0.8 * n_pos)
n_neg_train = ceiling(0.8 * n_neg)

idx_pos = which(mpg1 == 1)
idx_neg = which(mpg1 == -1)

idx_pos_train = sample(idx_pos, n_pos_train)
idx_neg_train = sample(idx_neg, n_neg_train)

datos_train = datos[c(idx_neg_train, idx_pos_train),]
datos_test = datos[-c(idx_neg_train, idx_pos_train),]
```

- Ajustar un modelo de regresión logística a los datos de entrenamiento y predecir mpg01 usando las variables seleccionadas en b). ¿Cuál es el error de test del modelo? Justificar la respuesta.
- Ajustar un modelo K-NN a los datos de entrenamiento y predecir mpg01 usando solam

Fijar una semilla para que siempre salga lo mismo en el caso de tener que hacer desempate.

Para que el KNN funcione correctamente hemos de elimitar las columnas que no sean números y es conveniente normalizar los datos para que a la hora de calcular distancias pues normalizar los datos. Para ello tenemos la siguiente función: scale(Auto[columnas]).

Con tune.knn nos permite probar distintos valores para el k y ver cuál da mejores resultados. Para ello hay que usar el paquete **el071**.

Para obtener la curva rho lo que hacemos es: pred = prediction(modelo, test.label) perf = performance(pred, “tpr”, “fpr”) plot(perf)

El problema que tiene el KNN es que no devuelve un modelo, lo que hemos de hacer es en los argumentos del KNN pasarle prob = TRUE lo cual nos devuelve una probabilidad, pero que realmente es una serie de etiquetas: 0 1 0 1 1 0 1 0 vj = knn(,. . . . . ,prob=TRUE) attributes nos sirve para obtener estos vectores de etiquetas y probabilidades

a lo que sea cero le ponemos 1-probabilidad de que sea cero eso esta en el $prob para poder emplear esta probabilidad para usar performance y luego poder pintar la curva.

para el vaging mtray = ncol-1 de los datos que tenemos (es un random forest)