Trabajo3

```
## Loading required package: gplots

## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess

## Loading required package: raster

## Loading required package: sp

##
## Attaching package: 'raster'

## The following object is masked from 'package:e1071':
##
##     interpolate

## The following objects are masked from 'package:MASS':
##
##     area, select

## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-5
```

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.

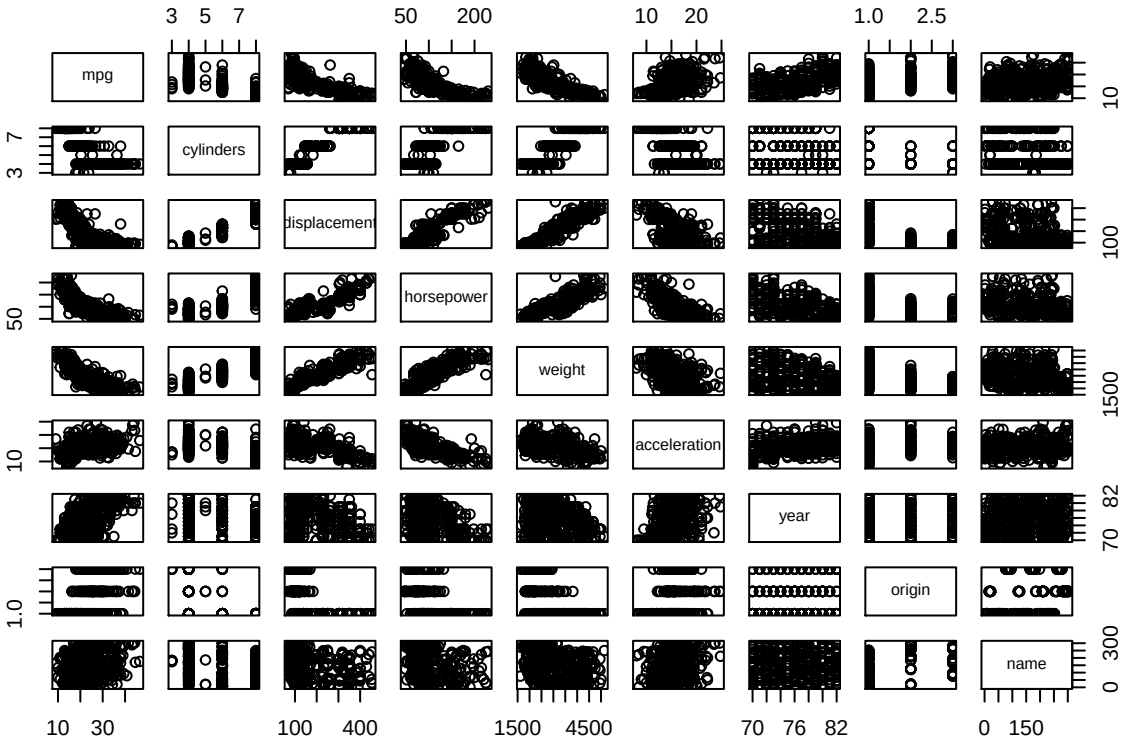
Lo convertimos a factor, asfactor

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

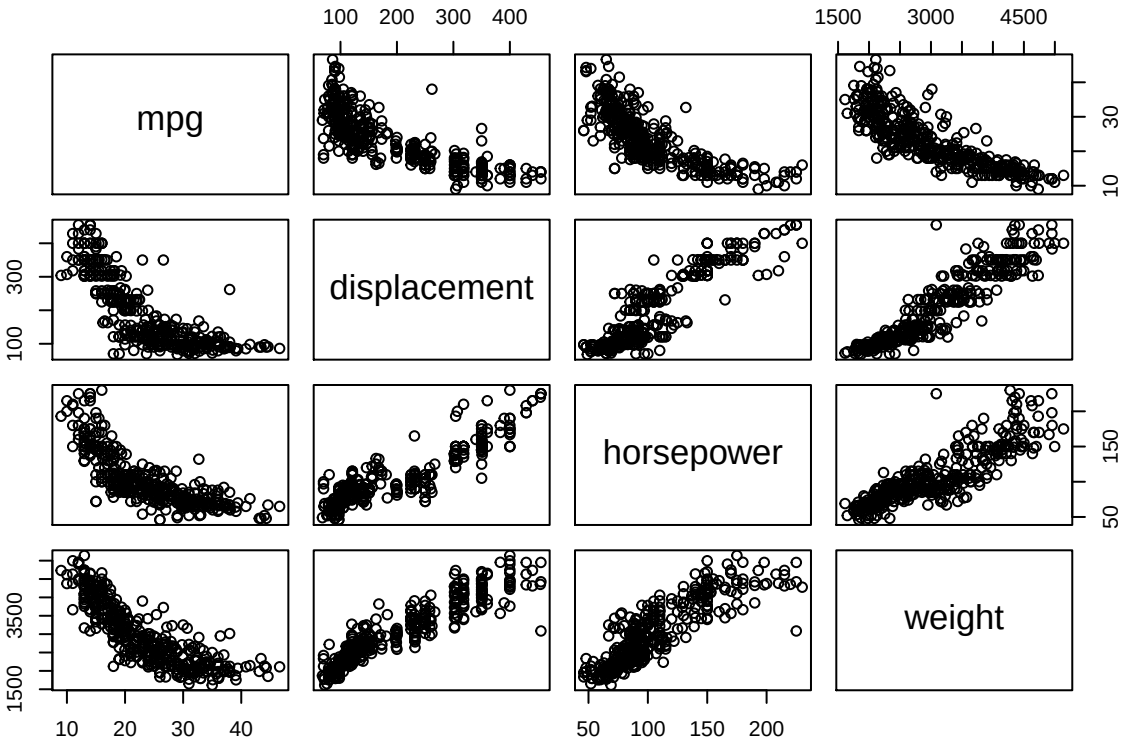
APARTADO 1

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)
```

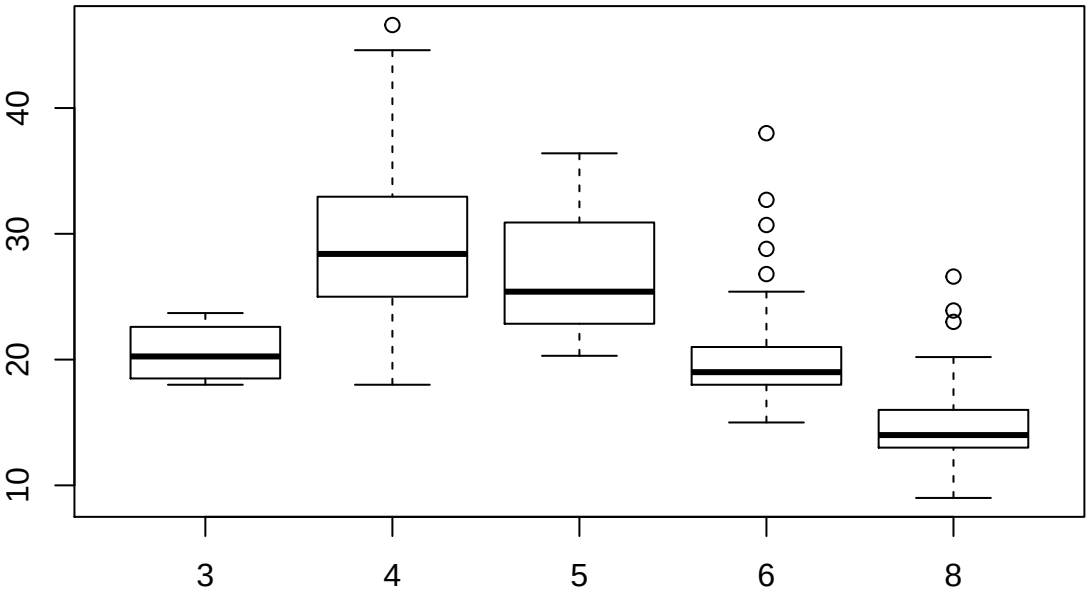



```
pairs(mpg ~ displacement + horsepower + weight)
```

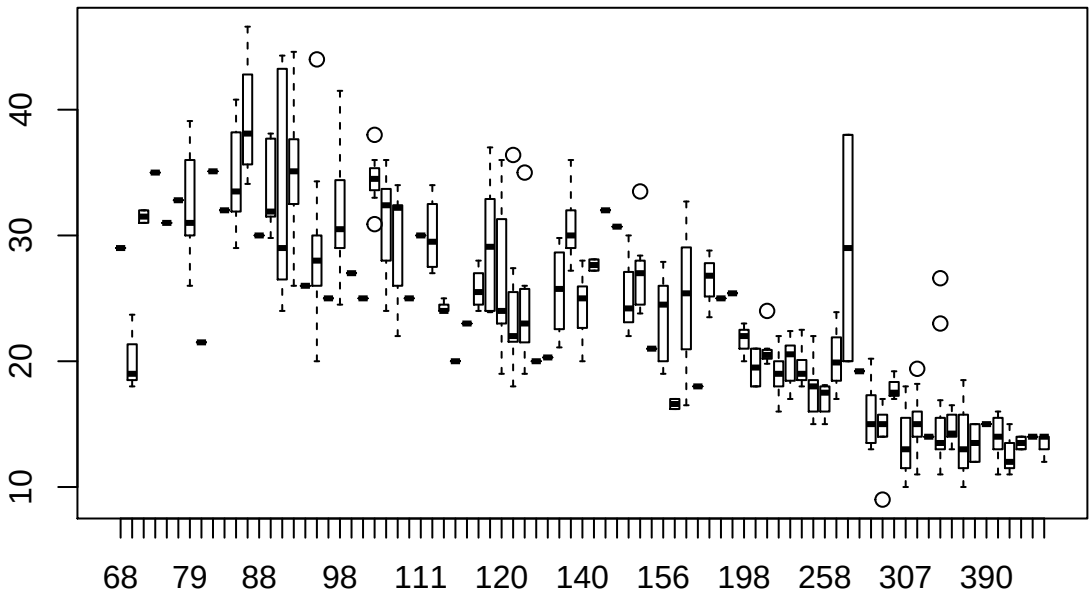


Como podemos ver las "gráficas de dependencias" de mpg con respecto a *displacement*, *horsepower* y *weight* son las gráficas que presentan un patrón más parecido entre ellas indicando que mpg tiene una relación fuerte con estas variables ya que se ajusta a ellas de un modo similar. Por ejemplo si vemos 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.

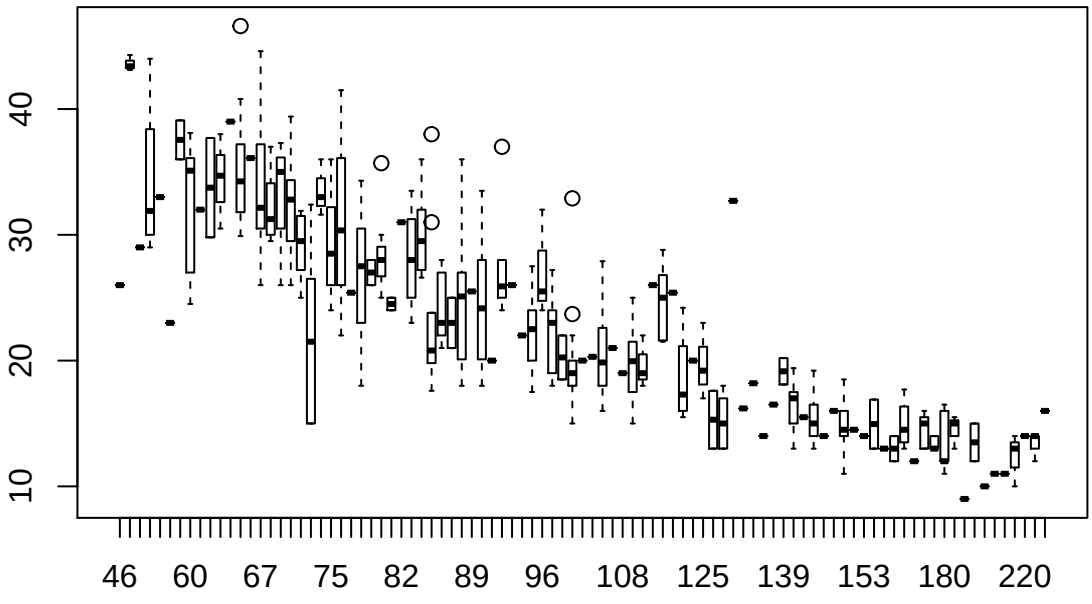
```
boxplot(mpg ~ cylinders)
```



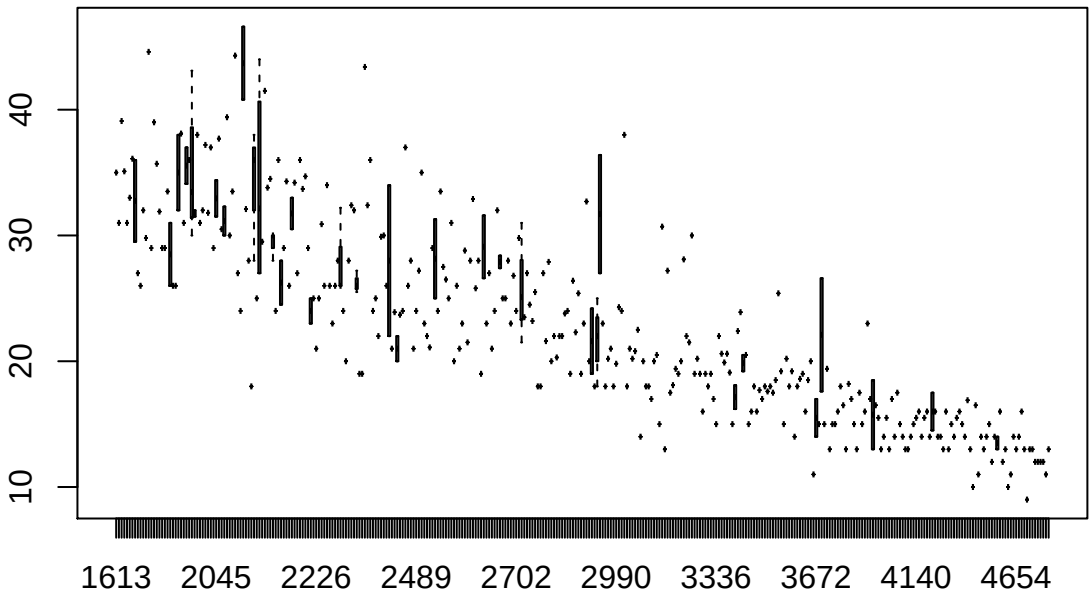

```
boxplot(mpg ~ displacement)
```



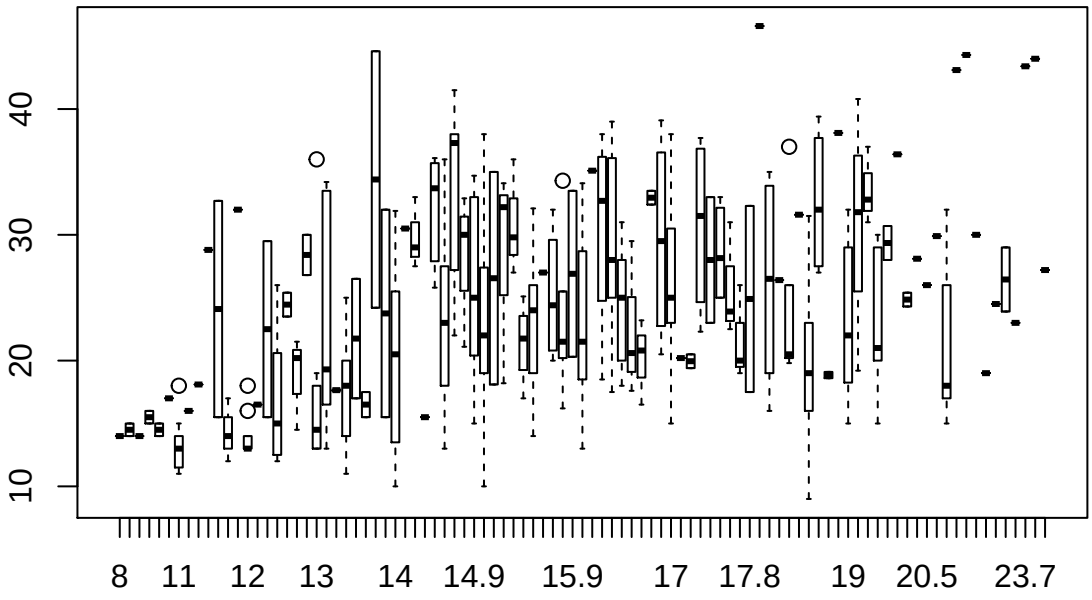

```
boxplot(mpg ~ horsepower)
```



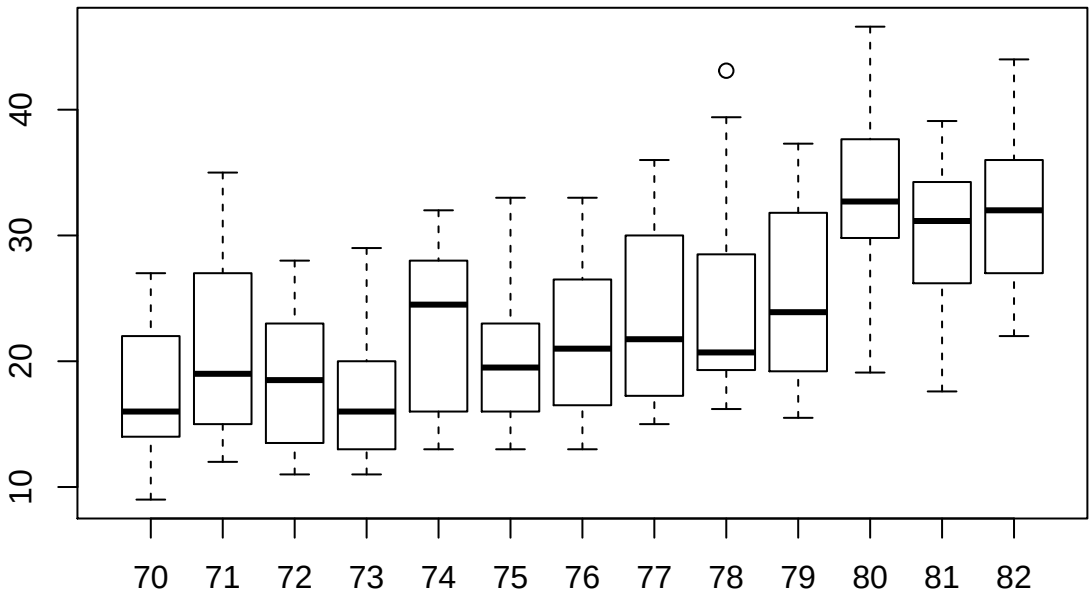

```
boxplot(mpg ~ weight)
```



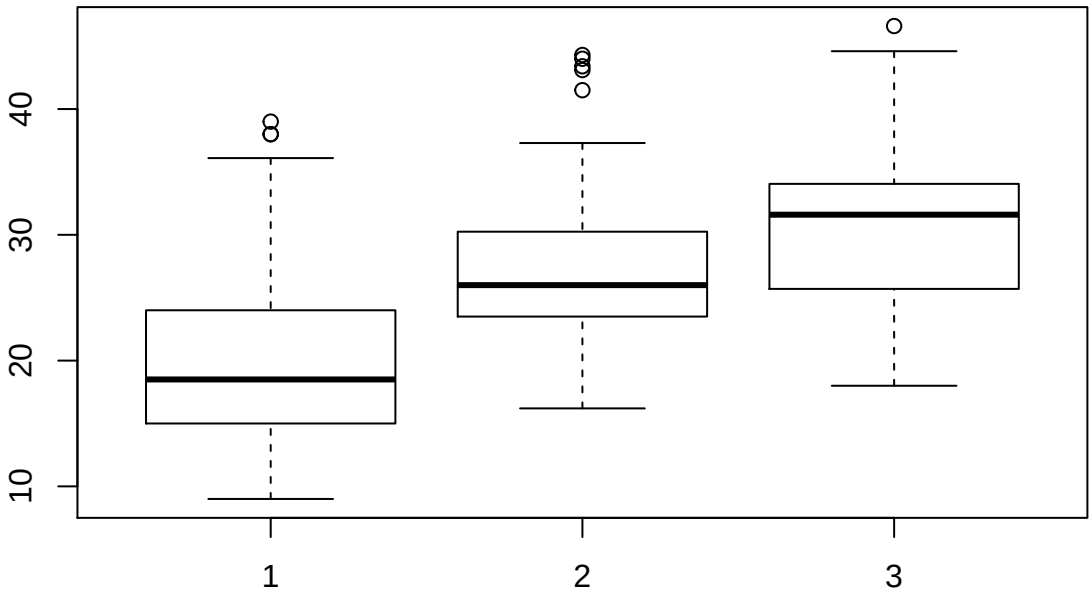

```
boxplot(mpg ~ acceleration)
```



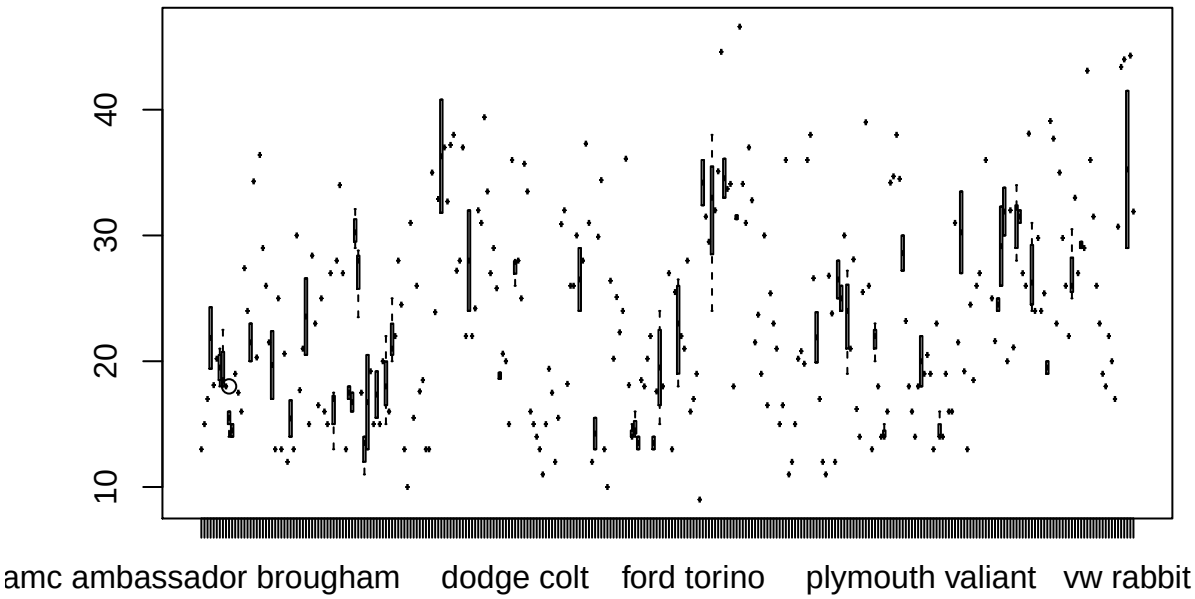

```
boxplot(mpg ~ year)
```




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




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



b) Seleccionar las variables predictoras que considere más relevantes.

Vamos a seleccionar aquellas que hemos visto en el apartado anterior que parecen seguir un patrón similar:

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

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

A priori había pensado en calcular en primer lugar la variable mpg1 del siguiente apartado para así poder realizar un particionamiento más homogéneo de las etiquetas positivas y negativas en los conjuntos de entrenamiento y test. El problema es que para hacer tal cosa tendría que calcular la mediana de todos los datos de Auto y lo que queremos es calcular la mediana, y por tanto el valor de mpg1 sólo en base a los datos de entrenamiento puesto que se dijo en teoría que para no contaminar el aprendizaje no podíamos usar los datos de test para calcular una mediana, sería como mirar los datos antes de aprender. Entonces he realizado simplemente un submuestreo de los datos aleatorio para dividirlos en entrenamiento y test.

```
n = nrow(datos)
idx_train = sample(seq(n), ceiling(0.8*n))

datos.train = datos[idx_train,]
datos.test = datos[-idx_train,]
```

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).

```
mediana = median(datos.train$mpg)
mpg1.train = sapply(datos.train$mpg, function(x) if (x < mediana) return(0) else return(1))
mpg1.test = sapply(datos.test$mpg, function(x) if (x < mediana) return(0) else return(1))
```

- 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.

```
trainRL = data.frame(mpg01 = mpg1.train, datos.train)
testRL = data.frame(mpg01 = mpg1.test, datos.test)

RL = glm(mpg01 ~ displacement+horsepower+weight, data = trainRL, family = binomial)

prediccion = predict(RL, newdata = testRL, type = "response")
RL.pred = rep(0, length(testRL$mpg01))
RL.pred[prediccion > .5] = 1

cat("El % de errores en test con RL es: ", sum(testRL$mpg01 != RL.pred)/length(testRL$mpg01)*100)
```

```
## El % de errores en test con RL es:  15.38462
```

- Ajustar un modelo K-NN a los datos de entrenamiento y predecir mpg01 usando solamente las variables seleccionadas en b). ¿Cuál es el error de test del modelo? ¿Cuál es el valor de K que mejor ajusta los datos?

```
getErrorKNN <- function(datos.train, datos.test, et.train, et.test ){
  #normalizamos los datos
  train.norm = scale(datos.train[,c("weight","displacement","horsepower")])
  medias = attr(train.norm, "scaled:center")
  escalados = attr(train.norm, "scaled:scale")
  test.norm = scale(datos.test[,c("weight","displacement","horsepower")], medias, escalados)

  datos.full = rbind(train.norm, test.norm)
  et.full = as.factor(c(et.train,et.test))

  set.seed(75570417)
  mknns = tune.knn(datos.full, et.full, k=1:20, tunecontrol = tune.control(sampling = "cross"), cross =

  mejor_k = mknns$best.model$k

  KNN = knn(train.norm, test.norm, et.train, k = mejor_k, prob = TRUE)
  cat("Se ha elegido como mejor k el: ", mejor_k, "\n")
  err.test = sum(et.test != KNN)/length(et.test)*100
  cat("El % de errores que obtenemos en el test es: ", err.test, "\n")

  KNN
}

KNN =  getErrorKNN(trainRL, testRL, mpg1.train, mpg1.test)
```

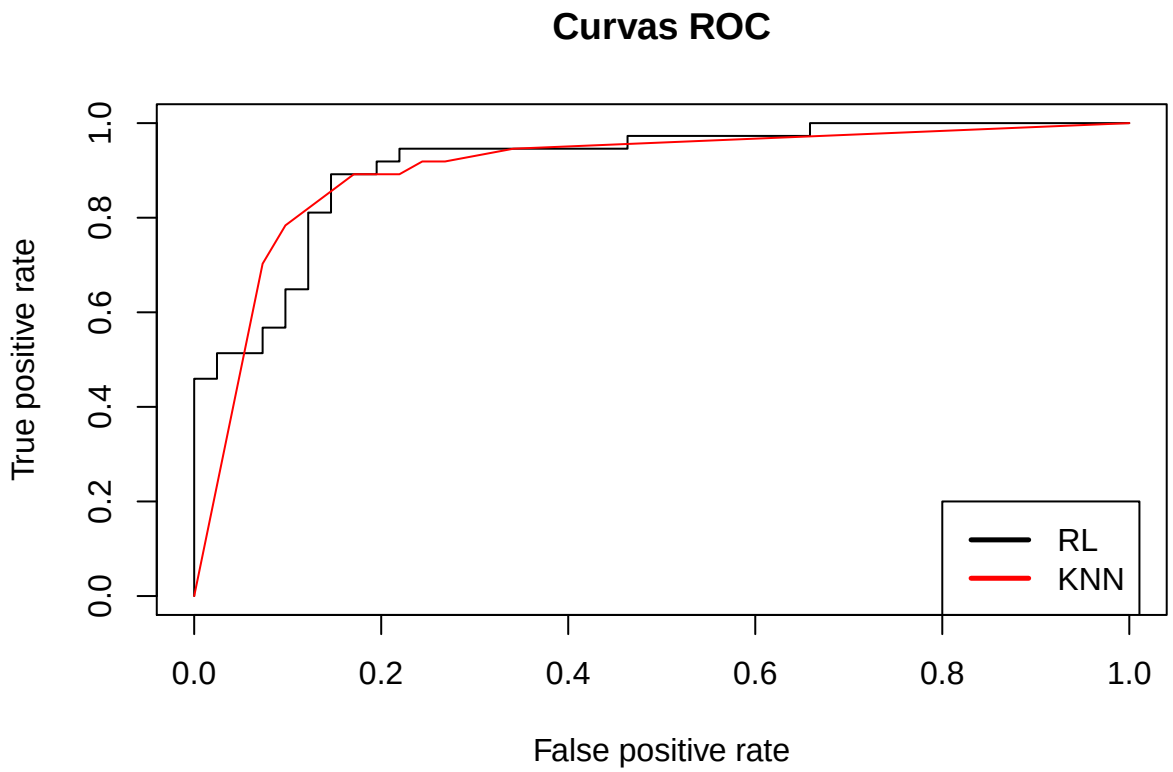
```
## Se ha elegido como mejor k el:  7
## El % de errores que obtenemos en el test es:  16.66667
```

- Pintar las curvas ROC (instalar paquete ROCR en R) y comparar y valorar los resultados obtenidos por ambos modelos.

```
pRL = prediction(prediccion, testRL$mpg01)
perf = performance(pRL, "tpr", "fpr")
plot(perf, main = "Curvas ROC")

prob = attr(KNN, "prob")
prob = ifelse(KNN == "0", 1-prob, prob)
pKNN = prediction(prob, testRL$mpg01)
perf = performance(pKNN, "tpr", "fpr")


plot(perf,  add = TRUE, col = "red")
legend(0.8,0.2,c("RL", "KNN"), lty=c(1,1),lwd=c(2.5,2.5),col=c("black","red"))
```



Al inicio RL parece mas estable puesto que pasa más tiempo teniendo un tasa nula de falso positivos, ahora bien cuando esta tasa aumenta vemos como antes la misma pérdida o el mismo error de clasificación es el KNN el que acierta un mayor número de veces, es decir, que aunque cometemos el mismo error al menos clasificamos bien más muestras.

Cuando la tasa de falsos positivos aumenta es nuevamente la RL la que mejor se comporta puesto que tiene una tasa de verdaderos positivos más elevada aunqeue no difiere en gran medidad de la del KNN. Con lo cual en mi opinión es el KNN el que tiene un mejor comportamiento puesto que es el que da mejores resultados ante un error igual.

e) Estimar el error de test de ambos modelos pero usando Validación Cruzada de 5-particiones. Comparar con los resulados obtenidos en el punto anterior.

```
fullRL = rbind(trainRL, testRL)
RL = glm(mpg01 ~ displacement+horsepower+weight, data = fullRL, family = binomial)
pp = cv.glm(fullRL, RL, K = 5)

cat("El error de test obtenido haciendo VC en RL ha sido: ", pp$delta[1]*100, "\n")
```

```
## El error de test obtenido haciendo VC en RL ha sido:  8.477098
```

```
'
folds = kfold(full.data, k = 5)
err = 0

for (i in seq(5)) {
  getErrorKNN(full.data[folds != i,], full.data[folds == i,], et.train, et.test)
  KNN = knn(full.data[folds != i,], full.data[folds == i,], full.labels[folds != i], k = mejor_k, prob =
  err = err + sum(full.labels[folds == i] != KNN)/length(full.labels[folds == i])*100
}

cat("El error de test obtenido haciendo VC en KNN ha sido: ", err/5)
'
```

```
## [1] "\nfolds = kfold(full.data, k = 5)\nerr = 0\n\nfor (i in seq(5)) {\n  getErrorKNN(full.data[folds
```

Ejercicio 2

Lo que vamos a hacer en primer lugar es emplear validación cruzada para obtener el mejor s para nuestros datos, este s lo que hace es regular el impacto de la restricción de "contracción", la cual lleva a coeficientes más pequeños cuanto mayor sea su impacto, es decir, el valor de s.

Con el objetivo de tener un buen s lo que vamos a hacer es obtenerlo mediante validación cruzada en la cual emplearemos todos los datos, ya vimos en teoria que con validacio

```
attach(Boston)

set.seed(41192)
cv.out = cv.glmnet(as.matrix(Boston[,-1]), Boston[,1],alpha=1)
bestlam = cv.out$lambda.min
```

```
out = glmnet(as.matrix(Boston[,-1]),Boston[,1], alpha = 1)
lasso.coef = predict(out, type="coefficients",s=bestlam)


umbral = 0.1
var_seleccionadas = which(abs(lasso.coef) > umbral)[-1]

#generamos un submuestreo aleatorio 80-20
train = sample(1:nrow(Boston), nrow(Boston)*0.8)
```

```
test = (-train)

ridge.mod = glmnet(as.matrix(Boston[train, var_seleccionadas]), Boston[train,1], alpha = 0)
ridge.pred = predict(ridge.mod, s=100,newx = as.matrix(Boston[test, var_seleccionadas]))

error = sqrt(mean((Boston[test,1] - ridge.pred)^2)/(nrow(Boston[test,])-2))
print(error)
```

```
## [1] 1.15581
```

Anteriormente ya hemos generado las muestras de entrenamiento y test, ahora vamos a tomar la mediana de los datos de entrenamiento (ya hemos dicho por qué esto se hace así) y luego a generar la variable que llamaremos, *crim1* a partir de ella:

```
mediana = median(Boston[train,1])
crim1.train = sapply(Boston[train,1], function(x) if (x < mediana) return(-1) else return(1))
crim1.test = sapply(Boston[test,1], function(x) if (x < mediana) return(-1) else return(1))

#generamos un dataframe con estos datos + la nueva var.
data.train = data.frame(Boston[train,-1], crim1 = as.factor(crim1.train))
data.test = data.frame(Boston[test, -1], crim1 = crim1.test)
```

A continuación vamos a aplicar un modelo de support vector machine a los datos de entrenamienot, a priori el kernel que vamos a usar va a ser lineal, luego veremos si es necesario aplicar algún kernel de una complejidad mayor o no.

```
svmfit = svm(crim1~., data=data.train[,-1], kernel = "linear")
```

Como vemos consideramos todas las variables excepto la variable crim ya que esta es la que queremos predecir con lo que estaríamos contaminando el aprendizaje. Indicar que como comentó la profesora no tenemos que usar solamente aquellas variables seleccionadas en apartados anteriores del ejercicio, sino que consideramos todas. Esto se debe a que según qué modelo de aprendizaje estemos empleando unas variables tendrán más importancia que otras; cada modelo tiene sus criterios y "preferencias", entonces a priori, sin un análisis más exhaustivo de los datos, no podemos descartar ninguna de las características.

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