

# 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
```

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 escribimos 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)`

Después 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.

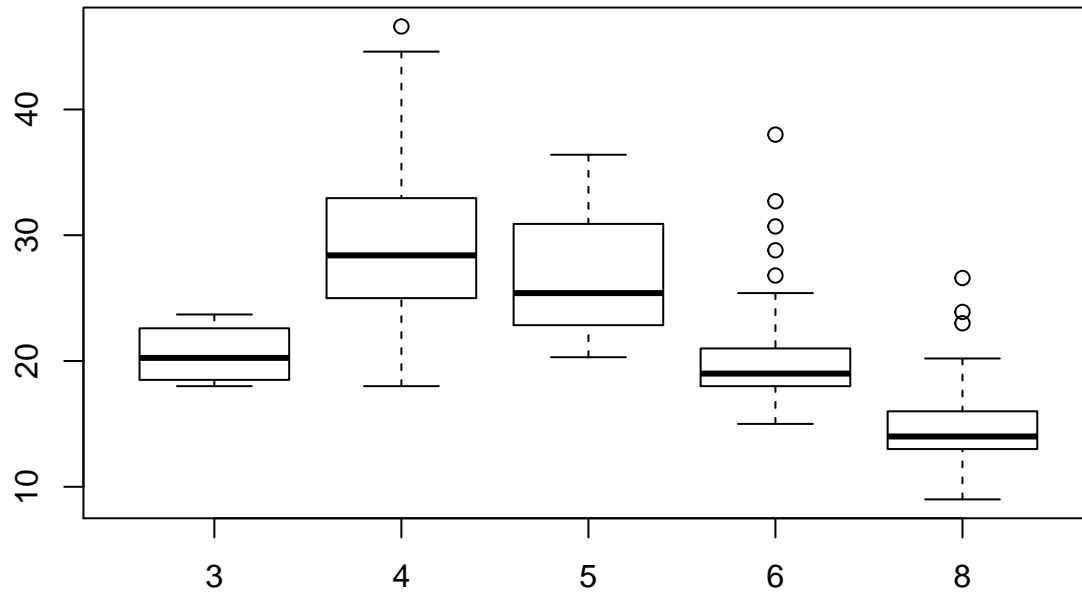
## APARTADO 1

**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.**

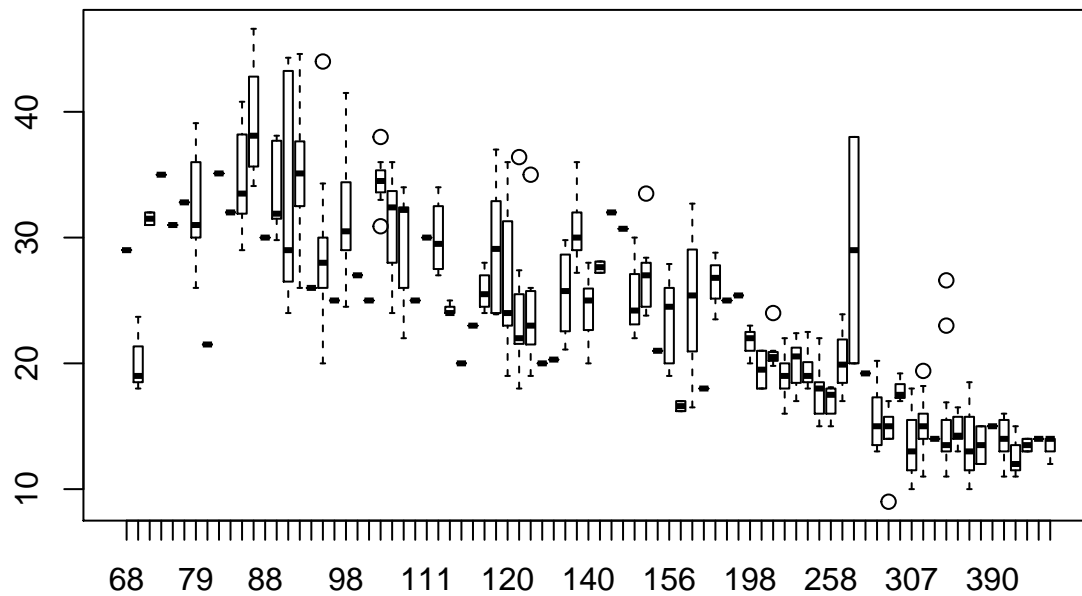
```
#pairs(Auto)
#pairs(mpg ~ displacement + horsepower + weight)
```

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 sí que tienen un aspecto de ser ajustables linealmente.

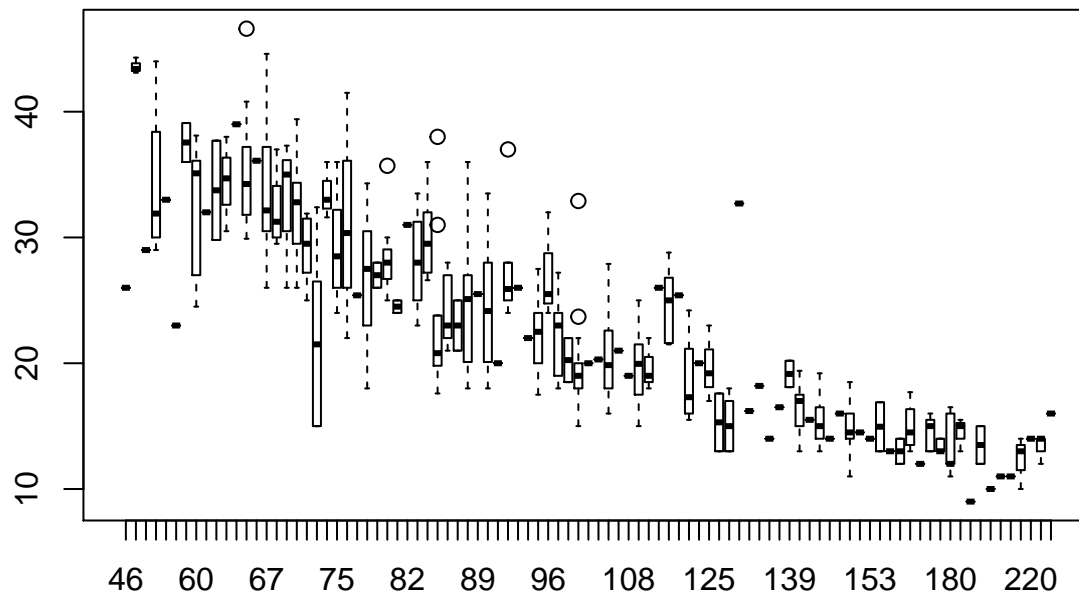
```
boxplot(mpg ~ cylinders)
```



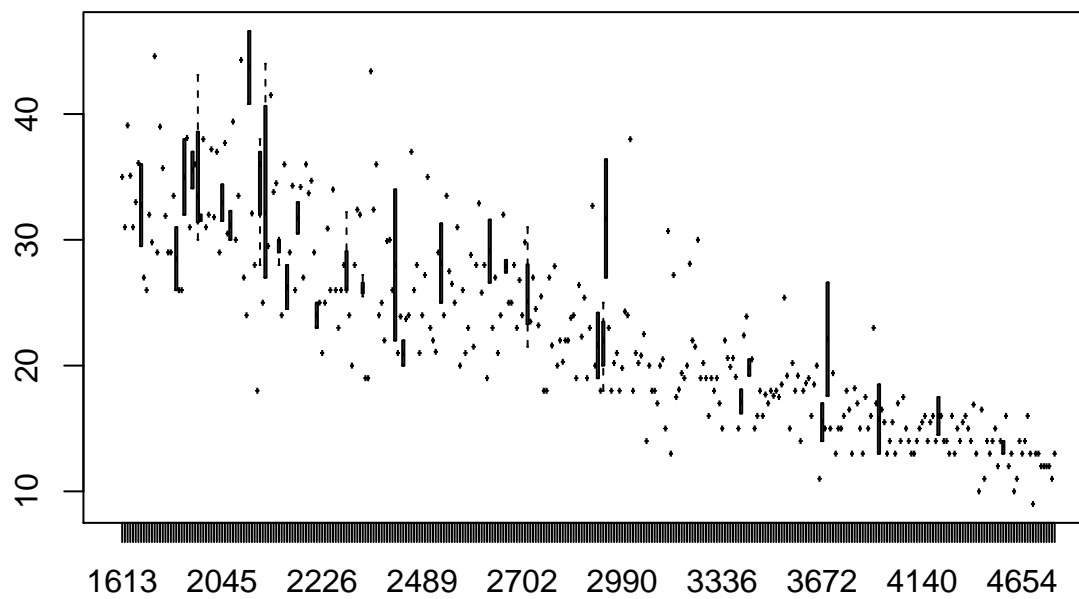
```
boxplot(mpg ~ displacement)
```



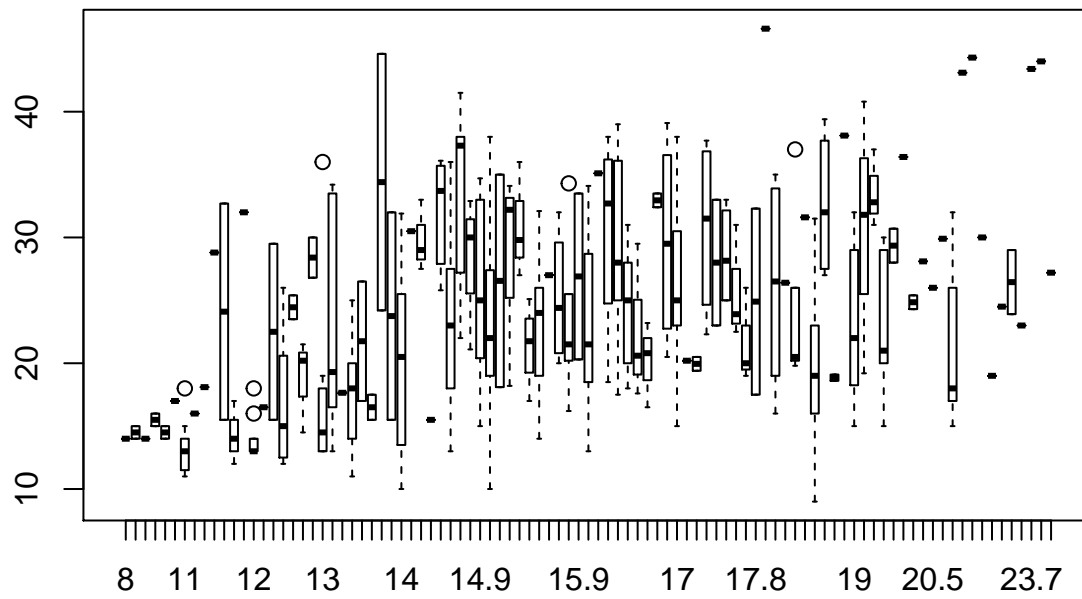
```
boxplot(mpg ~ horsepower)
```



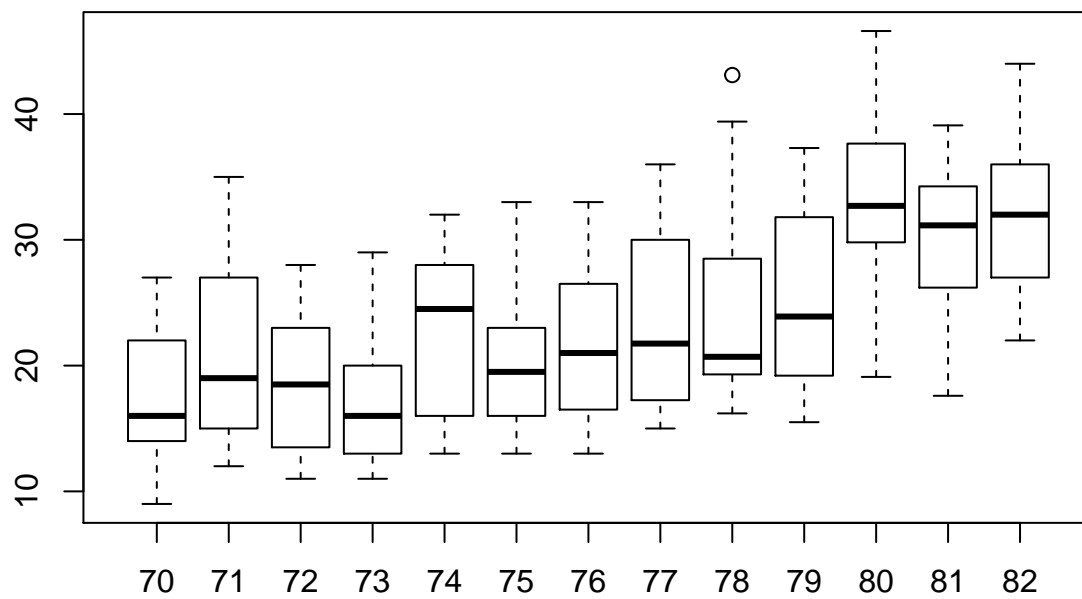
```
boxplot(mpg ~ weight)
```



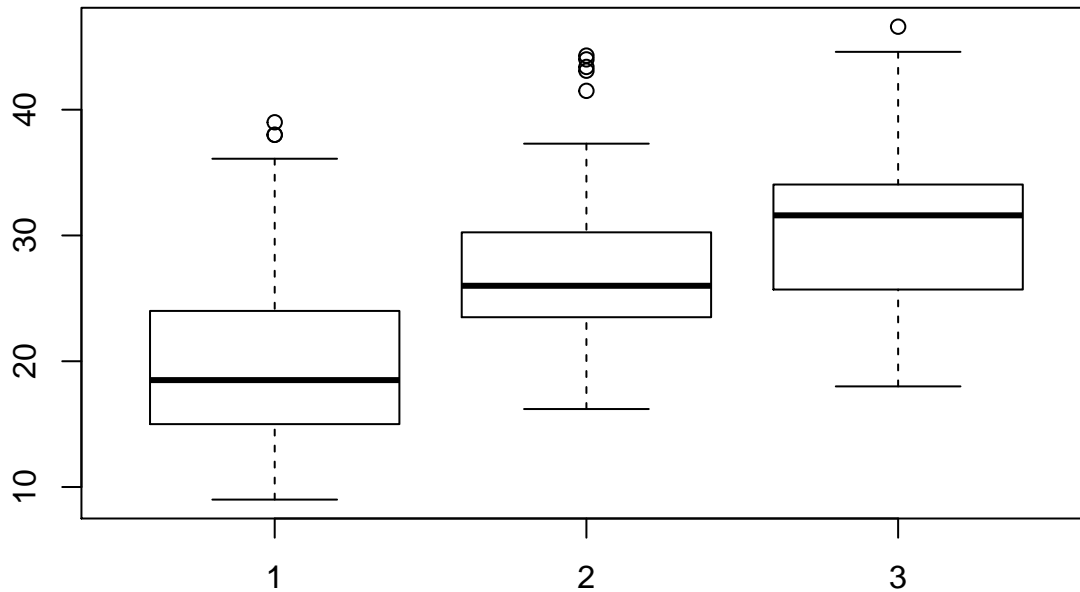
```
boxplot(mpg ~ acceleration)
```



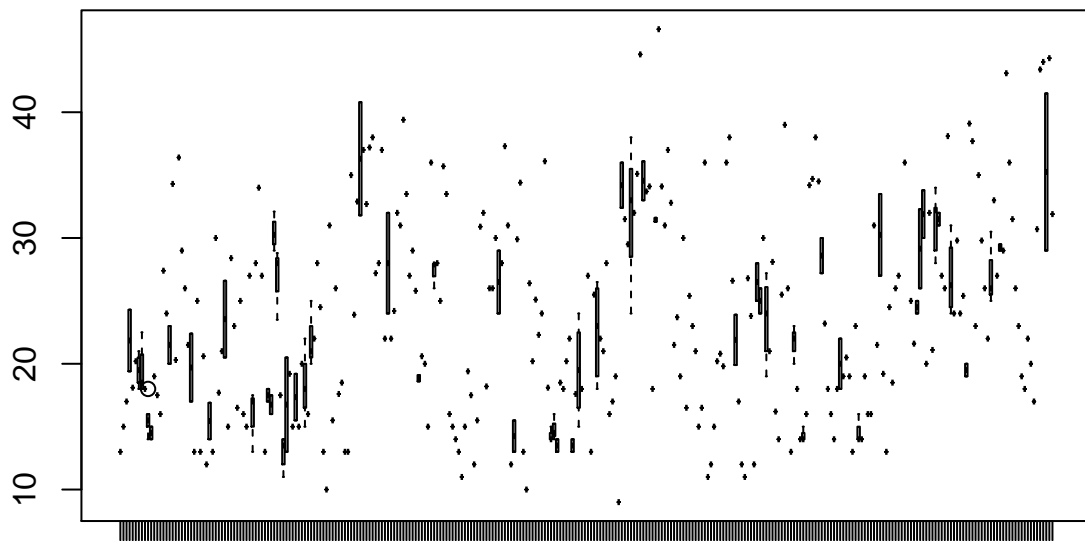
```
boxplot(mpg ~ year)
```



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



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



amc ambassador brougham    dodge colt    ford torino    plymouth valiant    vw rabbit

## 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.

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

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

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

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

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

full.data = rbind(train.norm, test.norm)
```

```
full.labels = as.factor(c(mpg1.train,mpg1.test))
set.seed(75570417)
mknns = tune.knn(full.data, full.labels, k=1:20, tunecontrol = tune.control(sampling = "cross"), cross = )

mejor_k = mknns$best.model$k

KNN = knn(train.norm, test.norm, train$mpg01, k = mejor_k, prob = TRUE)

cat("El % de errores que obtenemos en el test es: ", sum(test$mpg01 != KNN)/length(test$mpg01)*100, "\n")

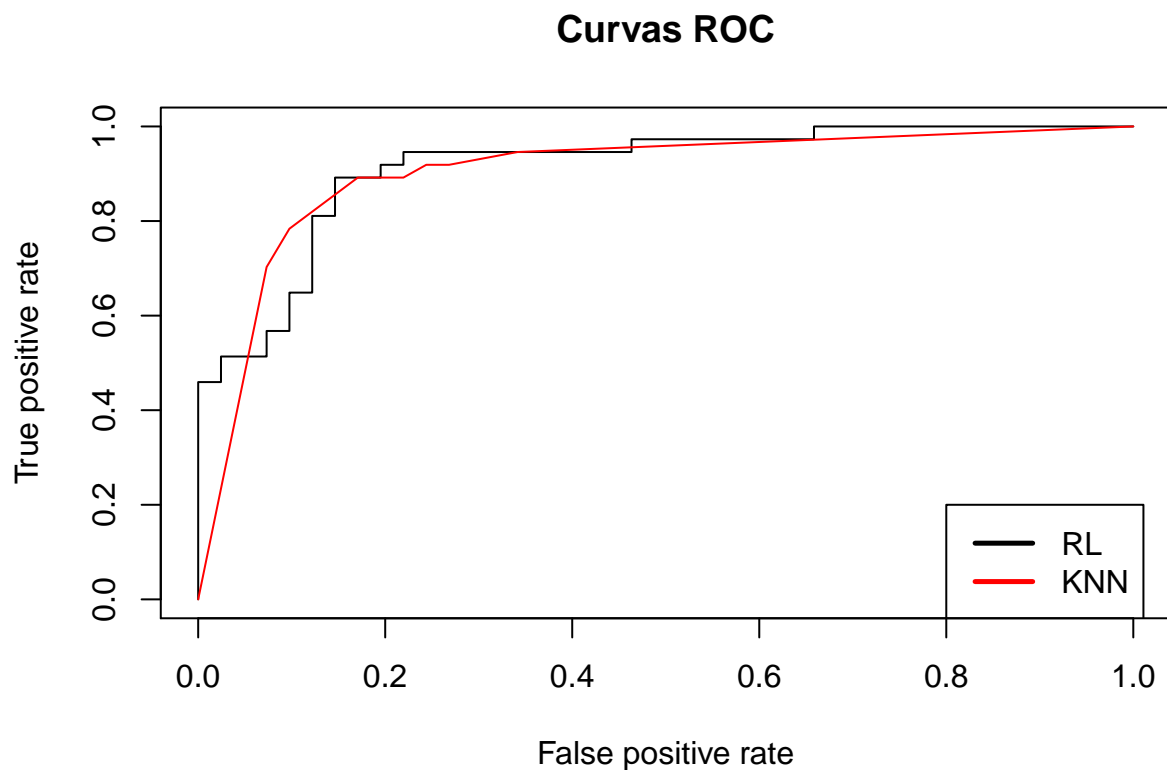
## El % de errores que obtenemos en el test es: 16.66667
```

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

```
pRL = prediction(prediccion, test$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, test$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 aunque no difiere en gran medida 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 resultados obtenidos en el punto anterior.**

```
full = data.frame(mpg01 = full.labels, full.data)
RL = glm(mpg01 ~ displacement+horsepower+weight, data = full, family = binomial)
pp = cv.glm(full, RL, K = 5)

cat("El error de test obtenido haciendo validacion cruzada ha sido: ", pp$delta[1], "\n")
```

```
## El error de test obtenido haciendo validacion cruzada ha sido: 0.08477098
```

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