EJERCICIO 1:

Cargamos el paquete ISLR para trabajar con Auto:

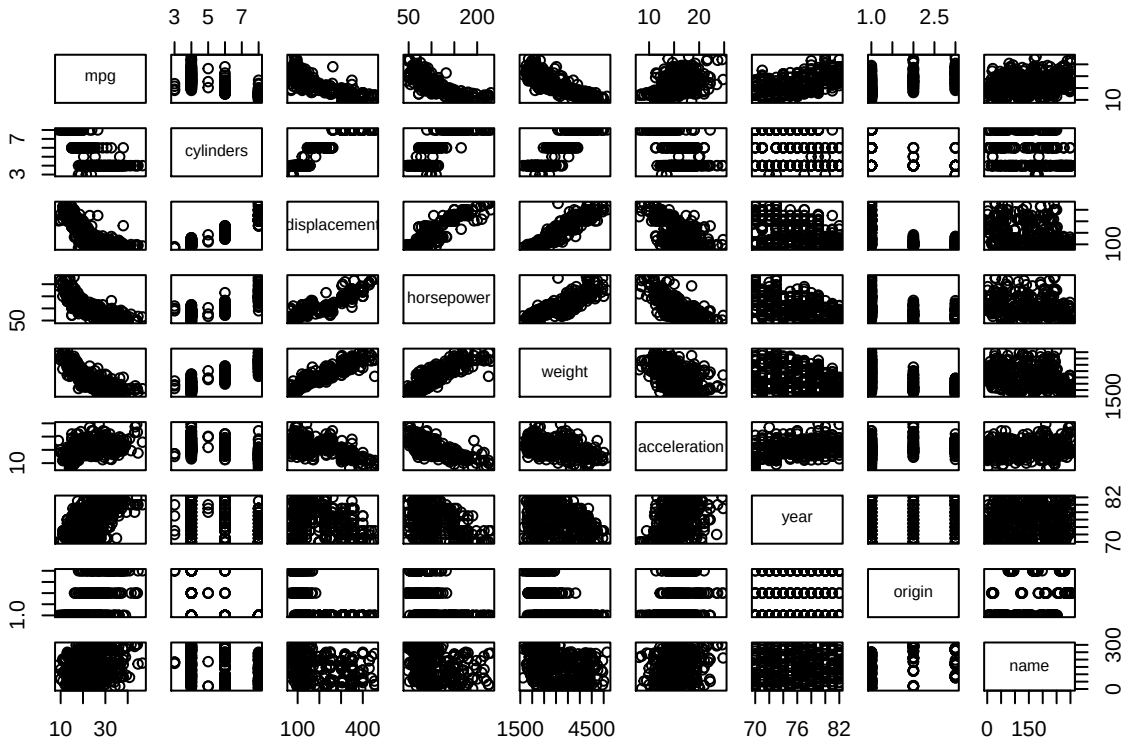
```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.2.5
```

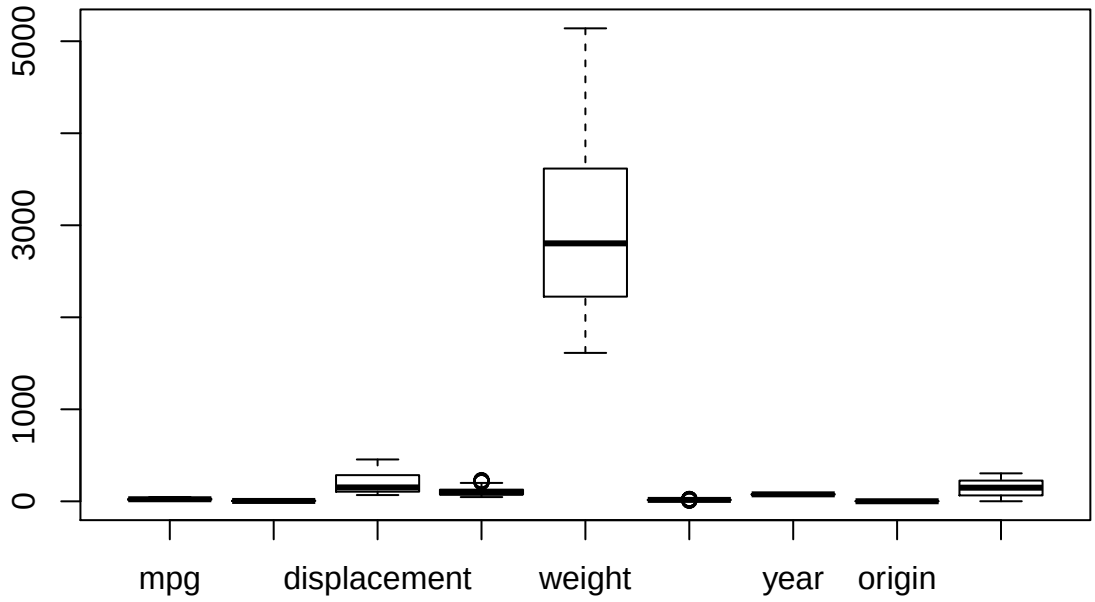
a) Usar las funciones de R pairs() y boxplot() para investigar la dependencia entre mpg y las otras características. ¿Cuáles de las otras características parece más útil para predecir mpg? Justificar la respuesta.

Vamos a visualizar la relación entre las distintas variables de la base Auto.

```
pairs(Auto)
```




```
boxplot(Auto)
```



Tras visualizar ambas gráficas, vemos que las variables origin y name no nos permiten predecir la variable mpg pues en name-mpg tenemos una nube de puntos dispersa en todo el intervalo y en origin-mpg no tenemos una distribución que nos permita establecer relaciones entre ambas. Las otras variables sí parecen útiles para predecir a mpg.

Analizando un poco más estas otras variables útiles, vemos que displacement, horsepowe y weight mantienen cierta dependencia lineal. De modo que podemos considerar sólamente una de ellas, en concreto, vamos a considerar la variable horsepowe ya que parece que nos va a permitir predecir mejor (mirando la tendencia de las gráficas).

Así pues, nos quedamos con las variables cylinders, horsepower, acceleration y year.

b) Seleccionar las variables predictoras que considere más relevantes.

Vamos a quedarnos con las variables cylinders, horsepower, acceleration y year, pues son las que hemos dicho anteriormente que nos interesan.

```
Predictoras = cbind(Auto$mpg, Auto$cylinders, Auto$horsepower, Auto$acceleration, Auto$year)
colnames(Predictoras) = c("mpg", "cylinders", "horsepower", "acceleration", "year")
```

Vemos las 5 primeras variables predictoras:

```
print(Predictoras[c(1,2,3,4,5),])
```

```
##      mpg cylinders horsepower acceleration year
## [1,]  18         8        130         12.0   70
## [2,]  15         8        165         11.5   70
## [3,]  18         8        150         11.0   70
## [4,]  16         8        150         12.0   70
## [5,]  17         8        140         10.5   70
```

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

Vamos a generar un 20% de los datos totales de forma aleatoria. Una vez tengamos los índices de los datos, vamos a quedarnos con los datos con dichos índices obteniendo las variables predictoras de conjunto test. Los datos con los índices que no hemos considerado para test, los consideramos para train y los almacenamos en Predictoras_train. Hemos tomado índices aleatorios pues no queremos que la muestra de test y de train se vean influenciadas por nuestra distinción de los conjuntos.

```
set.seed(2)
indices_test = sample(1:nrow(Predictoras),2*nrow(Predictoras)%/%10)
Predictoras_test = Predictoras[indices_test,]
Predictoras_train = Predictoras[-indices_test,]
```

Veamos cómo han quedado los 5 primeros datos para el conjunto test:

```
print(Predictoras_test[c(1,2,3,4,5),])
```

```
##       mpg cylinders horsepower acceleration year
## [1,] 13.0         8        130         14.0   72
## [2,] 21.6         4        115         15.7   78
## [3,] 17.5         6        110         16.4   77
## [4,] 17.0         8        150         11.5   72
## [5,] 29.0         4         84         16.0   82
```

Veamos cómo han quedado los 5 primeros datos para el conjunto train:

```
print(Predictoras_train[c(1,2,3,4,5),])
```

```
##      mpg cylinders horsepower acceleration year
## [1,]  18         8        130         12.0   70
## [2,]  15         8        165         11.5   70
## [3,]  17         8        140         10.5   70
## [4,]  14         8        220          9.0   70
## [5,]  14         8        215          8.5   70
```

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 de Auto).

Obtenemos la media de los valores mpg de Auto. Añadimos una columna con mpg01 a las variables Predictorias train y test según el signo que obtenemos al hacer la diferencia de la mediana con el valor mpg de cada dato.

```
mediana_mpg = median(Auto$mpg)
Predictoras_train_mpg01 = data.frame(Predictoras_train,
                                     mpg01=sign(Predictoras_train[,"mpg"] - mediana_mpg))
Predictoras_test_mpg01 = data.frame(Predictoras_test,
                                    mpg01=sign(Predictoras_test[,"mpg"] - mediana_mpg))
```

Veamos la media para ver que la nueva variable se obtiene correctamente.

```
print(mediana_mpg)
```

```
## [1] 22.75
```

Veamos cómo han quedado los 5 primeros datos para el conjunto test:

```
print(Predictoras_test_mpg01[c(1,2,3,4,5),])
```

```
##    mpg cylinders horsepower acceleration year mpg01
## 1 13.0         8        130         14.0   72    -1
## 2 21.6         4        115         15.7   78    -1
## 3 17.5         6        110         16.4   77    -1
## 4 17.0         8        150         11.5   72    -1
## 5 29.0         4         84         16.0   82     1
```

Veamos cómo han quedado los 5 primeros datos para el conjunto train:

```
print(Predictoras_train_mpg01[c(1,2,3,4,5),])
```

```
##   mpg cylinders horsepower acceleration year mpg01
## 1  18         8        130         12.0   70    -1
## 2  15         8        165         11.5   70    -1
## 3  17         8        140         10.5   70    -1
## 4  14         8        220          9.0   70    -1
## 5  14         8        215          8.5   70    -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.

Necesito que los valores de mpg01 estén comprendido entre 0 y 1. De modo que vamos a reevaluar los datos con variable mpg01 = -1 como mpg01= 0.

```
for(i in 1:nrow(Predictoras_test_mpg01)){
 if((Predictoras_test_mpg01[i,"mpg01"]) == -1)
     Predictoras_test_mpg01[i,"mpg01"] = 0
}
for(i in 1:nrow(Predictoras_train_mpg01)){
 if((Predictoras_train_mpg01[i,"mpg01"]) == -1)
     Predictoras_train_mpg01[i,"mpg01"] = 0
}
```

Hacemos la regresión logística con el método glm usando las variables cylinders, horsepowe, acceleration y year.

```
ajuste_rlog <- glm(mpg01 ~ cylinders + horsepower + acceleration + year,
                   data= Predictoras_test_mpg01, family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Ahora usamos el método predict para obtenerlas probabilidades y poder obtener la variable mpg01 que predice el modelo.

```
probabilidades = predict(ajuste_rlog, Predictoras_test_mpg01, type="response")
probabilidades01 = rep(1,dim(Predictoras_test_mpg01)[1])
probabilidades01[probabilidades<0.5] = 0
```

Obtenemos la matriz de confusión y obtenemos el error que viene dado como suma de las predicciones erroneas de 0 y 1 dividido por el numero total de datos.

```
matriz_confusion = table(probabilidades01, Predictoras_test_mpg01$mpg01)
print(matriz_confusion)
```

```
##
## probabilidades01  0  1
##                0 33  2
##                1  2 41
```

```
error = (matriz_confusion[1,2] + matriz_confusion[2,1])/sum(matriz_confusion)
print(error)
```

```
## [1] 0.05128205
```

Al tener un error tan bajo, concluimos que las variables que hemos seleccionado como predictoras son bastante buenas, así como el modelo.

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 en el modelo? ¿Cuál es el valor de K que mejor ajusta los datos? Justificar la respuesta. (Usar el paquete class de R) (1 punto)

Cargamos los paquetes class y e1071 de R.

```
library(class)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.2.5
```

```
set.seed(2)
```

Para usar el algormitmo K-NN tenemos que normalizar los datos. Vamos a ello: Normalizamos los datos iniciales y nos quedamos en test con los los datos normalizados de los indices que teníamos y en train con los restantes.

```
columnas=  dim(Predictoras_train_mpg01)[2]
datos_normalizados = scale(rbind(Predictoras_train_mpg01[,-columnas],
                                 Predictoras_test_mpg01[,-columnas]))
test_normalizada = datos_normalizados[indices_test,]
train_normalizada = datos_normalizados[-indices_test,]
```

Ahora nos quedamos con el vector mpg01 de los conjuntos train y test.

```
mpg01_train = Predictoras_train_mpg01[,columnas]
mpg01_test = Predictoras_test_mpg01[,columnas]
mpg01_ambos = c(mpg01_train, mpg01_test)
```

Veamos cuál es el mejor valor de k para ajustar los datos. Usaremos tune.knn:

```
posibles_k <- tune.knn(datos_normalizados, as.factor(mpg01_ambos), k=1:10,
                       tune_control = tune.control(sampling = "cross"))
summary(posibles_k)
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  k
##  3
##
## - best performance: 0.03064103
##
## - Detailed performance results:
##     k      error dispersion
## 1   1 0.05102564 0.03198455
## 2   2 0.03307692 0.02956326
## 3   3 0.03064103 0.02651089
## 4   4 0.03064103 0.02651089
## 5   5 0.03064103 0.02359501
## 6   6 0.03833333 0.02187600
## 7   7 0.03833333 0.02187600
## 8   8 0.04339744 0.02433657
## 9   9 0.04846154 0.02533327
## 10 10 0.04846154 0.02226369
```

//COMNETARRRRRRRRRRRRR. ¿FALLA ALGO EN KNN????ERROR PARECE MUY GRANDE.....

```
predicciones = knn(train_normalizada, test_normalizada, mpg01_train, k=3)
matriz_confusion_knn = table(predicciones, mpg01_test)
print(matriz_confusion_knn)
```

```
##             mpg01_test
## predicciones  0  1
##            0 19 20
##            1 16 23
```

```
errorknn = ((matriz_confusion_knn[1,2] + matriz_confusion_knn[2,1])/sum(matriz_confusion_knn))
print(errorknn)
```

```
## [1] 0.4615385
```

//FALTA ÚLTIMO PUNTO.

EJERCICIO4: Usar el conjunto de datos OJ que es parte del paquete ISLR.

1. Crear un conjunto de entrenamiento conteniendo una muestra aleatoria de 800 observaciones, y un conjunto de test conteniendo el resto de las observaciones. Ajustar un árbol a los datos de entrenamiento, con "Purchase" como la variable respuesta y las otras variables como predictores (paquete tree de R).

En primer lugar, para familiarizarnos con el conjunto de datos OJ, vamos a ver las variables de las que dispone. Hacemos uso de names.

```
library(ISLR)
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.2.5
```

```
names(OJ)
```

```
##  [1] "Purchase"       "WeekofPurchase" "StoreID"        "PriceCH"
##  [5] "PriceMM"        "DiscCH"         "DiscMM"         "SpecialCH"
##  [9] "SpecialMM"      "LoyalCH"        "SalePriceMM"    "SalePriceCH"
## [13] "PriceDiff"      "Store7"         "PctDiscMM"      "PctDiscCH"
## [17] "ListPriceDiff"  "STORE"
```

```
set.seed(1013)
```

Vamos a ajustar el arbol usando Purchase como variable respuesta. Tomo 800 indices de observaciones aleatorias y nos quedamos con estas observaciones para el conjunto train. El resto de observaciones lo introducimos en el conjunto test.

```
indices_train800 = sample(nrow(OJ),800)
trainOJ = OJ[indices_train800,]
testOJ = OJ[-indices_train800,]
treeOJ = tree(Purchase~. ,data = trainOJ)
```

En el siguiente apartado procedemos a analizar los resultados.

2. Usar la función summary() para generar un resumen estadístico acerca del árbol y describir los resultados obtenidos: tasa de error de "training", número de nodos del árbol, etc. (0.5 puntos)

Veamos el resumen estadístico del árbol:

```
summary(treeOJ)
```

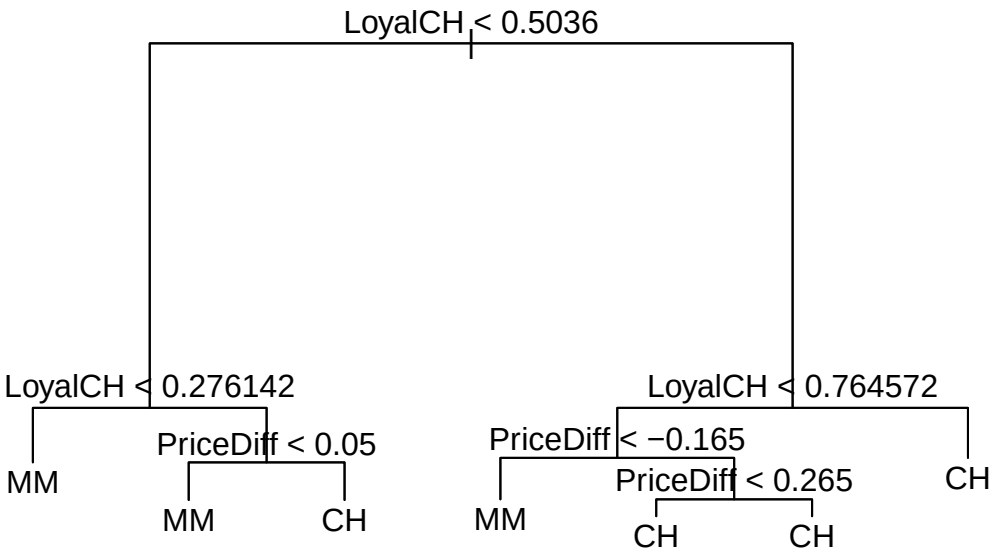
```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = trainOJ)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7517 = 596.1 / 793
## Misclassification error rate: 0.155 = 124 / 800
```

Vemos que para la construcción del árbol se han usado las variables "LoyalCH", "PriceDiff", "SpecialCH" y "ListPriceDiff". Se han generado 8 nodos terminales. La media de la desviación residual es 0.7517 y la tasa de error de traning es de 0.155.

3. Crear un dibujo del árbol e interpretar los resultados (0.5 puntos)

Veamos el dibujo del árbol:

```
plot(treeOJ)
text(treeOJ, pretty = 0)
```



Vemos que el árbol que comienza dividiendo en función de si el valor de LoyalCH(la fidelidad del usuario con respecto a el zumo de la empresa Citrus Hill) es mayor o menor que 0.5036. A continuación se vuelve a subdividir segun "LoyalCH": si la fidelidad inicial era menor que 0.5036 vuelve a dividir en función de si su fidelidad es menor o mayor que 0.276142, mientras que si su fidelidad era mayor que 0.5036, divide en función de si LoyalCH es menor o mayor que 0.764572. Volvemos a ver subdivisiones en el árbol que se analizan del mismo modo a como ya hemos visto.

Ahora vamos a ver las reglas de clasificación más relevantes. Serán aquellas que marcan una subdivisión entre MM(Minute Maid) y CH(Citrus Hill): LoyalCH <0.5036, LoyalCH <0.276142, LoyalCH <0.764572, PriceDiff <0.165 y PriceDiff <0.05.

Mientras que las menos relevantes son: PriceDiff <0.265.

4. Predecir la respuesta de los datos de test, y generar e interpretar la matriz de confusión de los datos de test. ¿Cuál es la tasa de error del test? ¿Cuál es la precisión del test? (1 punto)

```
probabilidades = predict(treeOJ, testOJ, type="class")
matriz_confusion = table(testOJ$Purchase, probabilidades)
print(matriz_confusion)
```

```
##     probabilidades
##       CH  MM
##   CH 152  19
```

```
##   MM  32  67
```

```
error = (matriz_confusion[1,2] + matriz_confusion[2,1])/sum(matriz_confusion)
print(error)
```

```
## [1] 0.1888889
```

Tenemos un error de test de 18.88% aprox., luego tendremos una precisión de test de 81.12% aprox..

5. Aplicar la función cv.tree() al conjunto de "training" y determinar el tamaño óptimo del árbol. ¿Qué hace cv.tree? (0.5 puntos)

```
cvtreeOJ = cv.tree(treeOJ, FUN= prune.tree)
cvtreeOJ
```

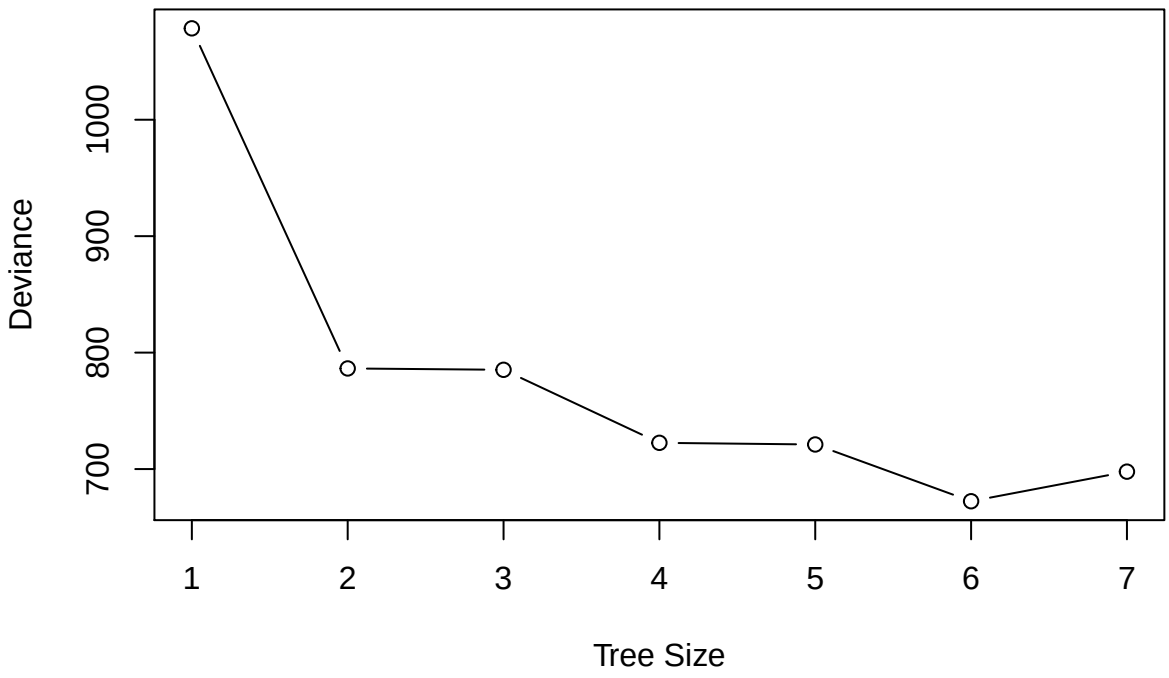
```
## $size
## [1] 7 6 5 4 3 2 1
##
## $dev
## [1]  697.7742  672.3676  721.1061  722.5296  785.2427  786.4040 1078.4624
##
## $k
## [1]      -Inf  14.33140  31.91234  35.17952  42.37864  46.23075 309.00727
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

La función cv.tree ejecuta un experimento de validación cruzada K veces para encontrar la desviación o el número de errores de clasificación.

El menor error de validación cruzada viene dado por 663, que se corresponde con el árbol que tiene 6 nodos terminales. Luego estos serán los tamaños óptimos del árbol.

Bonus-4 (1 punto). Generar un gráfico con el tamaño del árbol en el eje x (número de nodos) y la tasa de error de validación cruzada en el eje y. ¿Qué tamaño de árbol corresponde a la tasa más pequeña de error de clasificación por validación cruzada?

```
plot(cvtreeOJ$size, cvtreeOJ$dev, type = "b", xlab = "Tree Size", ylab = "Deviance")
```



Podemos ver que el tamaño 6 de árbol nos proporciona la tasa más pequeña de error de clasificación por validación cruzada.