EJERCICIO 1:

Cargamos el paquete ISLR para trabajar con Auto:

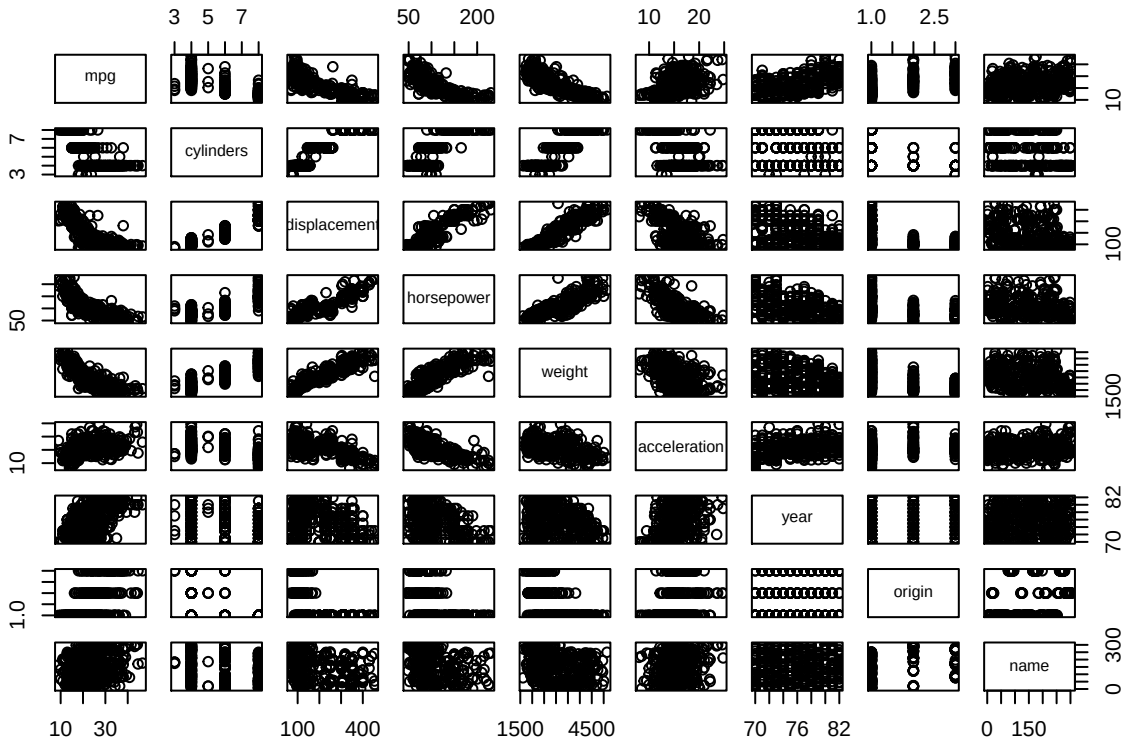
```
library(ISLR)
```

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

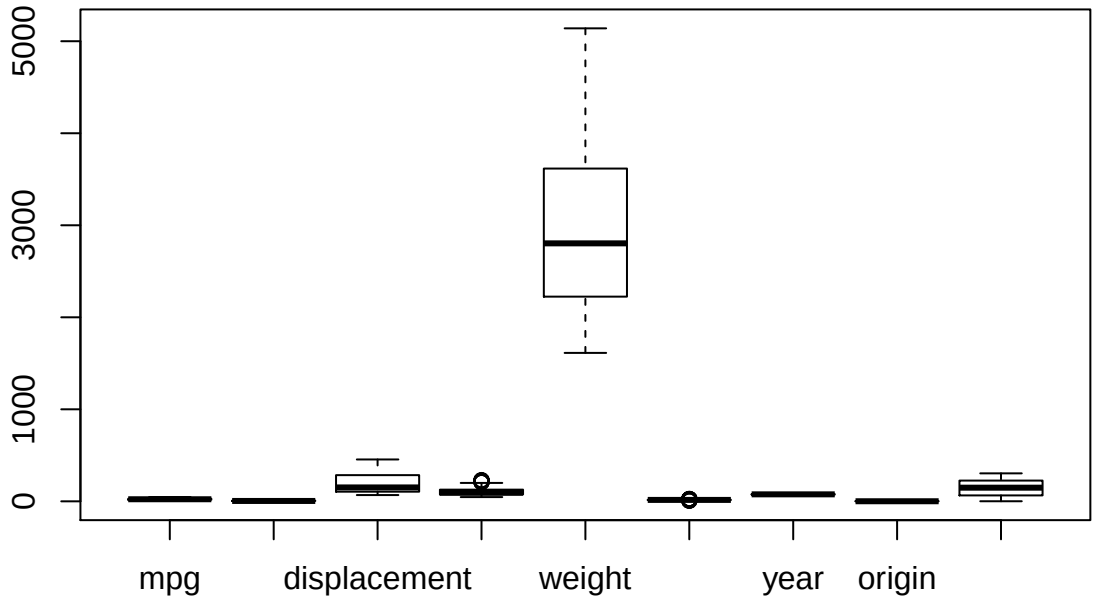
a) Usar las funciones de R pairs() y boxplot() para investigar la dependencia entre mpg y las otras características. ¿Cuáles de las otras características parece más útil para predecir mpg? Justificar la respuesta.

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$mpg, Auto$displacement, Auto$horsepower, Auto$weight, Auto$year)
colnames(Predictoras) = c("mpg", "displacement", "horsepower", "weight", "year")
```

Vemos las 5 primeras variables predictoras:

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

```
##      mpg displacement horsepower weight year
## [1,]  18          307        130   3504   70
## [2,]  15          350        165   3693   70
## [3,]  18          318        150   3436   70
## [4,]  16          304        150   3433   70
## [5,]  17          302        140   3449   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(nrow(Predictoras), 2*nrow(Predictoras)%/%10, replace=FALSE)
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 displacement horsepower weight year
## [1,] 13.0          307        130   4098   72
## [2,] 21.6          121        115   2795   78
## [3,] 17.5          250        110   3520   77
## [4,] 17.0          304        150   3672   72
## [5,] 29.0          135         84   2525   82
```

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

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

```
##      mpg displacement horsepower weight year
## [1,]  18          307        130   3504   70
## [2,]  15          350        165   3693   70
## [3,]  17          302        140   3449   70
## [4,]  14          454        220   4354   70
## [5,]  14          440        215   4312   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 displacement horsepower weight year mpg01
## 1 13.0          307        130   4098   72    -1
## 2 21.6          121        115   2795   78    -1
## 3 17.5          250        110   3520   77    -1
## 4 17.0          304        150   3672   72    -1
## 5 29.0          135         84   2525   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 displacement horsepower weight year mpg01
## 1  18          307        130   3504   70    -1
## 2  15          350        165   3693   70    -1
## 3  17          302        140   3449   70    -1
## 4  14          454        220   4354   70    -1
## 5  14          440        215   4312   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 ~ displacement + horsepower + weight + year,
                   data= Predictoras_train_mpg01, family=binomial)
```

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 30  1
##                1  5 42
```

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

```
## [1] 0.07692308
```

Al tener un error tan bajo (0.0769), 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_test_mpg01[,-columnas],
                                 Predictoras_train_mpg01[,-columnas]))
test_normalizada = datos_normalizados[1:dim(Predictoras_test_mpg01)[1],]
train_normalizada = datos_normalizados[(dim(Predictoras_test_mpg01)[1] +1):
                                       dim(datos_normalizados)[1],]
```

Ahora nos quedamos con el vector mpg01 de los conjuntos train y test. Los combinamos en mpg01_ambos.

```
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
##  4
##
## - best performance: 0.4132051
##
## - Detailed performance results:
##     k     error dispersion
## 1   1 0.5053205 0.08813063
## 2   2 0.4648077 0.09083708
## 3   3 0.4544231 0.12007880
## 4   4 0.4132051 0.11226521
## 5   5 0.4392308 0.10749448
## 6   6 0.4440385 0.12589008
## 7   7 0.4543590 0.09465103
## 8   8 0.4393590 0.10197131
## 9   9 0.4265385 0.09004031
## 10 10 0.4139744 0.10990212
```

Vemos que el valor de k que mejor ajusta los datos es k=4, de modo que usaremos este valor para aplicar KNN. Al igual que hacíamos antes, vemos la matriz de confunsión o obtenemos el error de test.

```
predicciones = knn(train_normalizada, test_normalizada, mpg01_train, k=4)
matriz_confusion_knn = table(predicciones, mpg01_test)
print(matriz_confusion_knn)
```

```
##             mpg01_test
## predicciones  0  1
##            0 33  0
##            1  2 43
```

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

```
## [1] 0.02564103
```

Obtenemos un error de 0.0256. Vemos que ha dismunuido con respecto al modelo que hacíamos en el apartado anterior.

//FALTAN BONUS.