

# Trabajo Introducción a la ciencia de datos

*Miguel López Campos*

*25 de noviembre de 2017*

## Análisis de los datos

En esta sección se hará un análisis descriptivo de los datos usando para ello medidas que nos den pistas sobre la distribución de los datos así como otros recursos gráficos.

### Clasificación - Appendicitis dataset

En esta sección se hará un análisis de los datos que se usarán para el problema de clasificación. Nuestro conjunto de datos se llama appendicitis y muestra ciertas medidas médicas a pacientes y una clase binaria que indica si el paciente tiene appendicitis (1) o si no tiene (0).

En el siguiente código mostramos las primeras medidas descriptivas de nuestro conjunto de datos.

```
dim(appendicitis)
```

```
## [1] 106   8
appendicitis[!complete.cases(appendicitis),]
```

```
## [1] V1 V2 V3 V4 V5 V6 V7 V8
## <0 rows> (or 0-length row.names)
```

Como vemos en la ejecución del anterior trozo de código, no hay ningún caso no completo (con missing values) en todo el dataset, lo cual facilita nuestro análisis y nuestro posterior proceso de aprendizaje.

```
summary(appendicitis)
```

```
##      V1          V2          V3          V4
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2447   1st Qu.:0.5890   1st Qu.:0.2770   1st Qu.:0.0560
##  Median :0.4130   Median :0.7500   Median :0.4425   Median :0.1460
##  Mean   :0.3985   Mean   :0.6821   Mean   :0.4152   Mean   :0.2087
##  3rd Qu.:0.5058   3rd Qu.:0.8210   3rd Qu.:0.5455   3rd Qu.:0.3025
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##      V5          V6          V7          V8
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.04925   1st Qu.:0.5760   1st Qu.:0.2380   1st Qu.:0.0000
##  Median :0.11300   Median :0.7440   Median :0.3975   Median :0.0000
##  Mean   :0.16915   Mean   :0.6763   Mean   :0.3754   Mean   :0.1981
##  3rd Qu.:0.22675   3rd Qu.:0.8313   3rd Qu.:0.4868   3rd Qu.:0.0000
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

Como vemos, el conjunto de datos consta de 106 instancias con 7 variables. En nuestro ‘summary’ del conjunto de datos observamos que todos los datos están normalizados en [0,1]. Además se muestran las medidas de tendencias centrales (media y mediana), así como otras medidas de dispersión como el rango de los datos o los cuartiles. La última variable no es significativa ya que esta se trata de la clase, la cual convertiremos a factor y además cambiaremos el nombre de la columna. En el conjunto de datos no hay más información sobre qué es cada variable y en internet he encontrado poca cosa sobre el conjunto.

```
str(appendicitis)
```

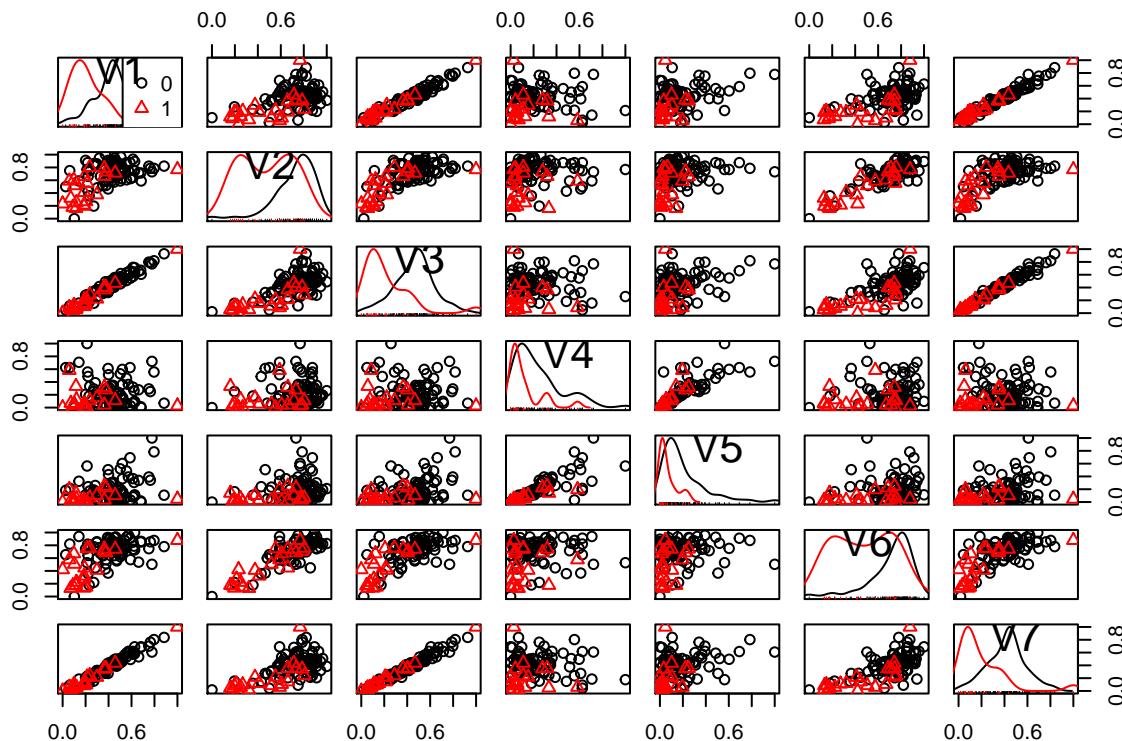
```
## 'data.frame': 106 obs. of 8 variables:  
## $ V1 : num 0.213 0.458 0.102 0.187 0.236 ...  
## $ V2 : num 0.554 0.714 0.518 0.196 0.804 ...  
## $ V3 : num 0.207 0.468 0.111 0.105 0.289 ...  
## $ V4 : num 0 0.111 0.056 0.056 0.111 ...  
## $ V5 : num 0 0.102 0.022 0.029 0.066 ...  
## $ V6 : num 0.749 0.741 0.506 0.133 0.756 ...  
## $ V7 : num 0.22 0.436 0.086 0.085 0.241 ...  
## $ class: Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

```
apply(appendicitis[,1:7], 2, sd)
```

```
##      V1      V2      V3      V4      V5      V6      V7  
## 0.1914517 0.2071341 0.2057989 0.1996495 0.1773287 0.2189092 0.1981319
```

Como vemos en la ejecución de la función 'str' tenemos un dataframe con 7 variables, todas numéricas, y la clase, la cual hemos convertido a factor. También, en la desviación típica de las variables, posteriormente calculadas, se ve que no tienen una desviación muy alta, por lo que más o menos los datos no tienen una gran dispersión en su distribución.

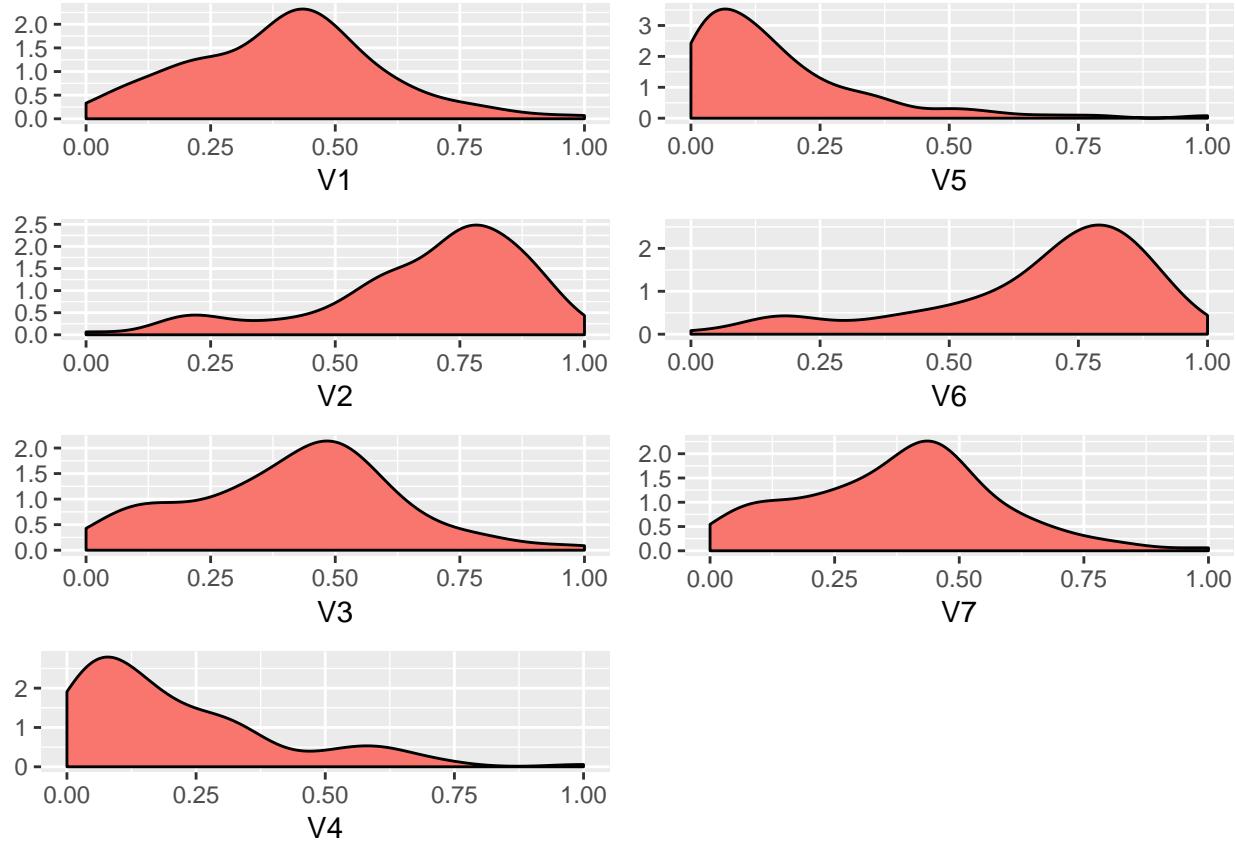
A continuación vamos a comprobar mediante scatter plots si se puede intuir gráficamente si existe correlación entre algunas de las 7 variables. Además, usando la función scatterplotMatrix podemos ver también histogramas de las variables para así poder ver gráficamente cómo se distribuyen los datos.

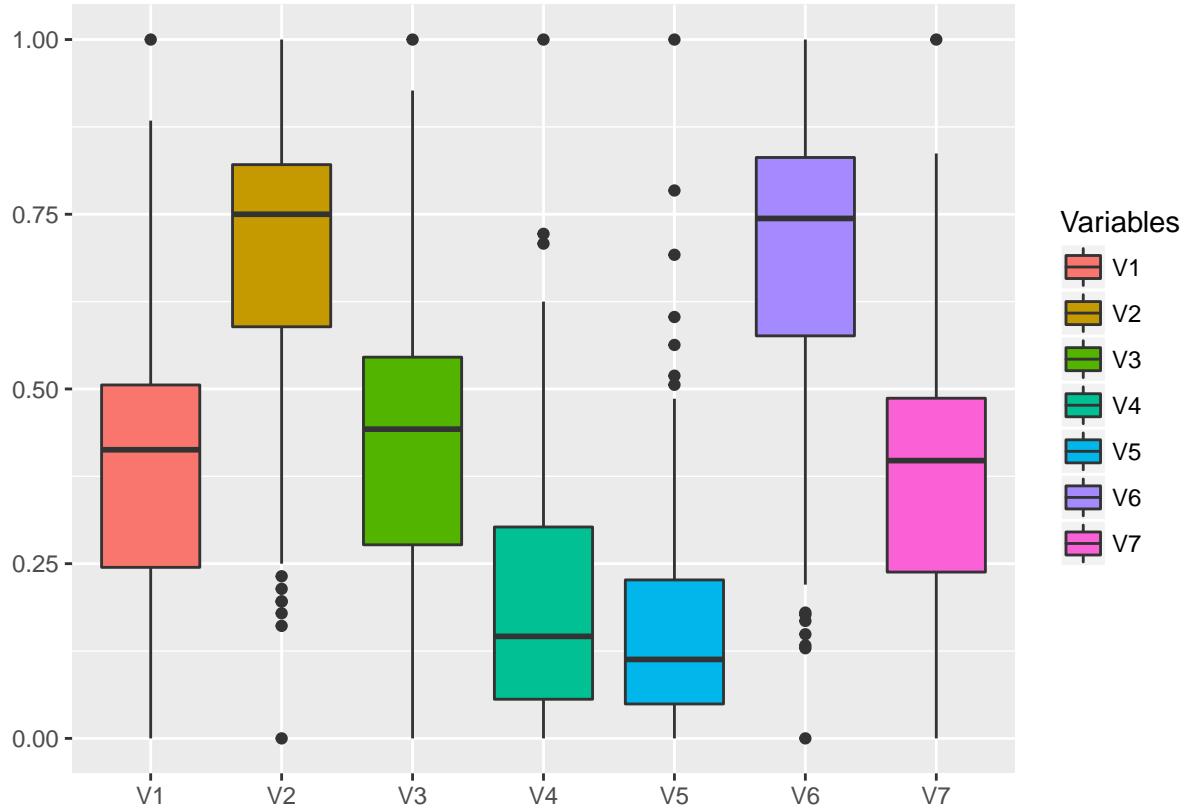


Según se muestra en la matriz de scatterPlots vemos cómo hay variables que tienen claramente cierta correlación. Por ejemplo, las variables V1 y V3 se aprecia cómo tienen una correlación muy alta. También hay una correlación muy alta entre la variable V3 y la V7. Con scatterPlotMatrix también se dibujan

estimaciones de funciones para la regresión. También hay otras posibles correlaciones pero en este caso no lineales, como entre V3 y V2 que podría ser una correlación logarítmica según la forma de la dispersión de los datos. También en el scatterMatrix hemos pintado las distribuciones de los datos según la clase (lo cual analizaremos posteriormente más en profundidad) y hemos pintado de distinto color los puntos según la clase. Vemos cómo por ejemplo para V3 casi todos los datos que tienen etiqueta 1 (tienen apendicitis) tienen un valor bajo. Igual pasa para V7 y V1, que son las que tienen correlación con V3. En cuanto a los demás scatters de 2 variables no existe una separación entre clases gráficamente muy clara.

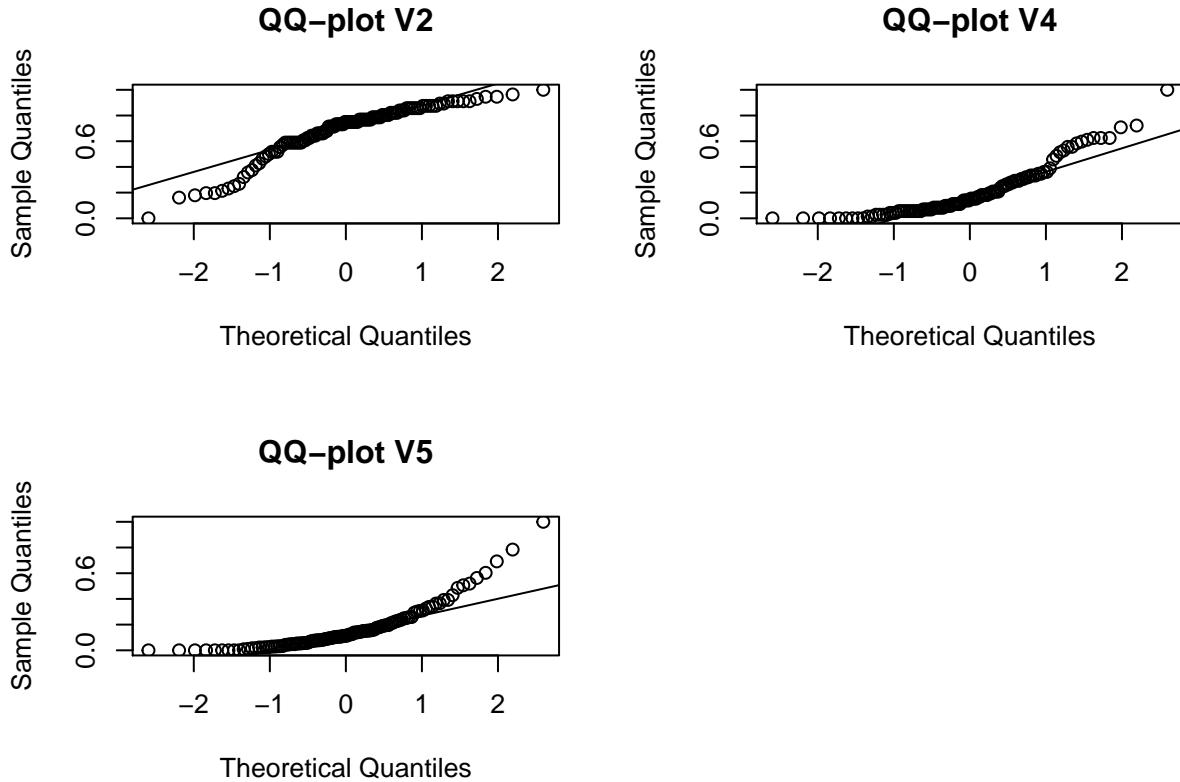
Ahora vamos a hacer un poco más de hincapié en ver la distribución de las distintas variables. Según se muestra en las siguientes figuras (histogramas suavizados con `geom_density` de `ggplot`), casi todas las variables muestran aparentemente una distribución que se puede asemejar a una distribución normal. Las variables v4 y v5 muestran lo que aparentemente sería una distribución sesgada a la derecha. V6, por el contrario, muestra lo que podría ser una distribución sesgada a la izquierda.





En los boxplots anteriores también se puede apreciar una idea de las distribuciones. Como se observa, para cada variable las cajas donde se encuentran el 50% de los datos coincide con el “pico” de los histogramas previamente comentados. Se aprecian también algunos outliers en casi todas las variables.

Podemos también comprobar la normalidad de los datos observando sus qqplots. Usaremos estos gráficos para las variables que gráficamente puedan mostrar más dudas de su distribución. Estas variables son V2, V4 y V5.

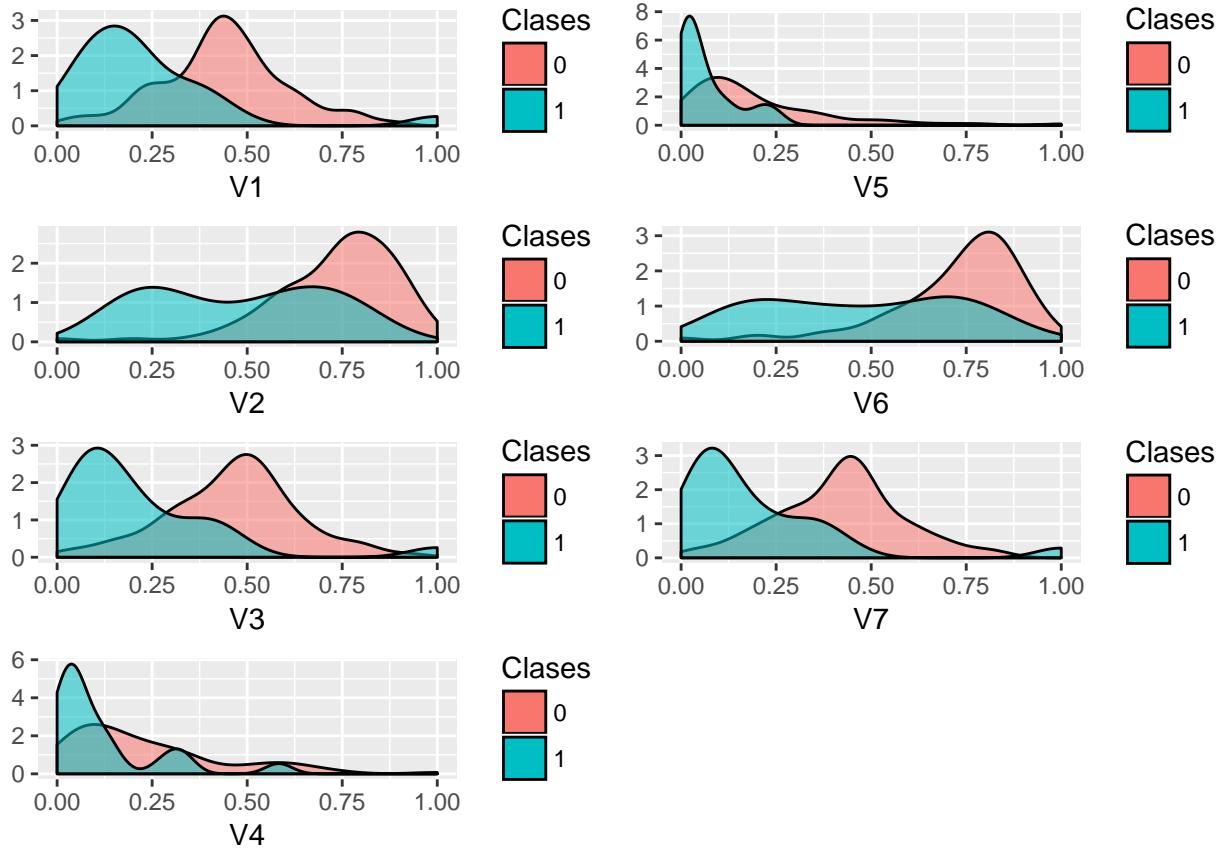


En los anteriores qqplots, se aprecia que la distribución de la variable V2 se puede considerar una distribución normal sesgada a la izquierda. En los qqplots de las variables V4 y V5 también se aprecia cómo se pueden considerar dos distribuciones normales sesgadas a la derecha.

Con un test shapiro ahora podemos también comprobar la normalidad de los datos. Si los test tienen un p-value menor a 0.1, entonces se rechaza la hipótesis nula (que los datos están normalmente distribuidos). En caso contrario, no se puede rechazar y por lo tanto tiene una distribución normal. En la siguiente tabla se muestra cómo solo se pueden admitir como normalmente distribuidas según el test lsa variables V1 y V3. V7, que está correlada con las anteriores, muestra sin embargo un p-value menor a 0.1, por lo que tampoco se acepta la hipótesis nula. Esta no normalidad en los datos puede influir en la calidad de nuestro modelo LDA, el cual considera apriori que todas las variables siguen una distribución normal para cada clase y que tienen varianzas similares.

	v1	v2	v3	v4	v5	v6	v7
p-value	0.183044	9e-07	0.106209	0	0	1e-07	0.0850259

Como último paso en nuestro análisis relacionado con la distribución de los datos, vamos a ver cómo es la distribución de estos según la clase a la que pertenecen y ver si existe una diferencia apreciable.



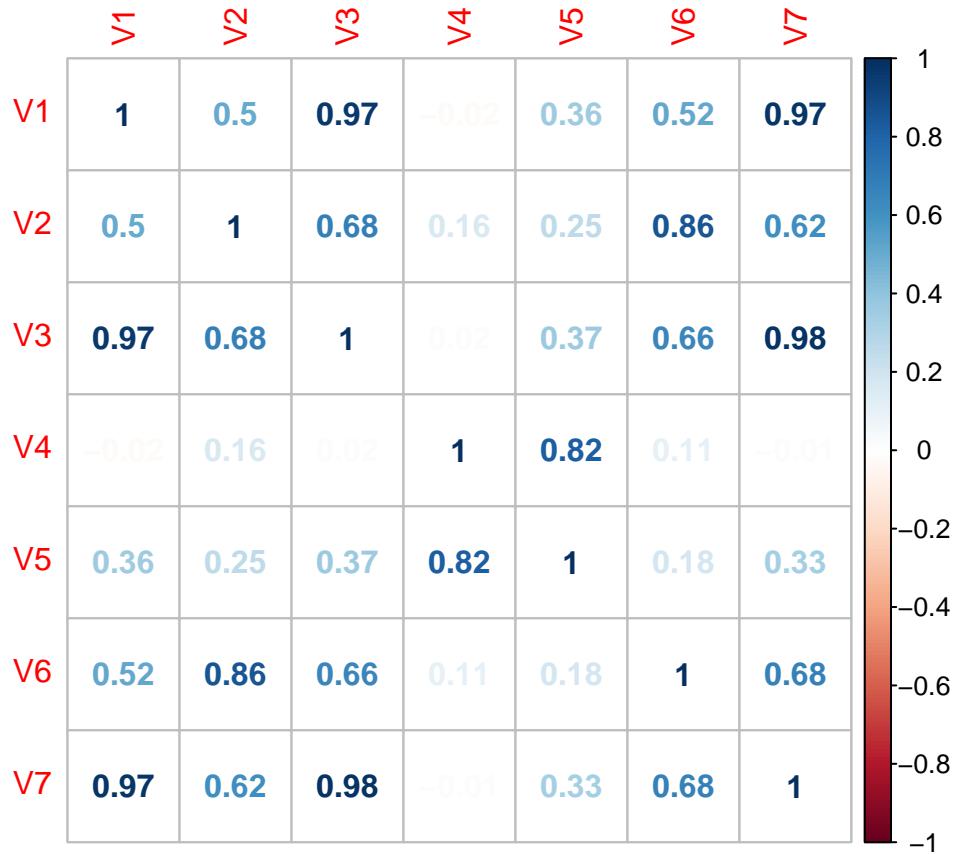
Se muestra en este último multiplot de la distribución según la clase cómo en las variables V1,V3 y V7 sí que se aprecia una diferencia clara en la distribución entre datos de distinta clase. Se ve cómo los pacientes con apendicitis tienden a tener un valor bajo de V1, V3 y V7 (que están correladas). En V6 vemos como los pacientes con apendicitis muestran una distribución aparentemente bastante uniforme. En V2 vemos que para los pacientes con apendicitis muestra una distribución normal multimodal.

A la hora de aplicar algoritmos como LDA y QDA es conveniente también ver la varianza de las variables para cada clase, ya que para LDA por ejemplo el algoritmo asume que las variables tienen una distribución normal todas las clases y también asume que la varianza es similar. Calcularemos entonces la varianza de las variables para cada clase. Como se muestra en la siguiente tabla, la varianza para cada clase no es similar. Esto puede significar que LDA no dé grandes resultados en nuestro problema.

	Var. clase=1	Var. clase=0
V1	0.0450774	0.0266832
V2	0.0522080	0.0279874
V3	0.0497010	0.0292926
V4	0.0216440	0.0419789
V5	0.0058272	0.0345499
V6	0.0634357	0.0315352
V7	0.0496050	0.0272686

A continuación, vamos a analizar la correlación entre las variables de forma más concreta que con la matriz de scatter plots. Para ello vamos a emplear un correlogram, que se trata de una matriz que representa visualmente y también numéricamente (dependiendo de los parámetros dados a la función) la correlación entre variables. La siguiente figura muestra este correlogram y podemos ver cómo existe una correlación muy

alta entre las variables V7 y V3, V3 y V1 y V1 y V7, lo que seguramente indique que las 3 variables están correladas entre sí. También hay una correlación alta, aunque menos que las anteriores, entre V6 y V2 y V5 y V4.



Otro aspecto a tener en cuenta para luego la hora del proceso de aprendizaje, aunque en esta asignatura no se estudia, es el balanceo de los datos (el balanceo entre clases).

```
length(appendicitis$class[appendicitis$class == 1])/nrow(appendicitis)
```

```
## [1] 0.1981132
```

```
length(appendicitis$class[appendicitis$class == 0])/nrow(appendicitis)
```

```
## [1] 0.8018868
```

Sí se aprecia un cierto desbalanceo, siendo el 80% de las instancias de clase 0 (no tiene appendicitis) y casi el 20% de clase 1 (tiene appendicitis). Este desbalanceo, aunque no es muy alto, sí que puede afectar a la calidad del modelo y habría que tomar medidas.

## Regresión - Abalone

El conjunto de datos para regresión se trata de Abalone. Este conjunto describe en cada instancia una especie de almeja con determinadas características. El problema que nos plantea este dataset es la predicción de la edad de la almeja, que se obtiene directamente del número de anillos de una de estas almejas. Por lo tanto, nuestra variable a predecir es el número de anillos a partir de las siguientes características:

- Sex. Variable categórica que indica el sexo de la almeja. Tiene 3 valores: Male, female o immature, que indica que aun es inmadura.

- Length. Indica la longitud.
- Diameter. Diámetro de la almeja
- Height. Altura de la almeja.
- Whole weight. Peso global.
- Shucked weight. Peso pelada.
- Viscera weight. Peso de la víscera.
- Shell weight. Peso de la almeja.

Todas las variables son numéricas excepto la variable Sex, la cual es categórica e indica el sexo (1 es Male, 2 Female y 3 es que es inmadura o “infantil”).

```
str(abalone)
```

```
## 'data.frame': 4177 obs. of 9 variables:
## $ Sex : Factor w/ 3 levels "1","2","3": 3 2 3 1 3 2 2 2 1 1 ...
## $ Length : num 0.4 0.635 0.37 0.68 0.375 0.58 0.525 0.63 0.565 0.605 ...
## $ Diameter : num 0.305 0.5 0.27 0.54 0.285 0.475 0.38 0.495 0.44 0.465 ...
## $ Height : num 0.1 0.15 0.09 0.155 0.09 0.155 0.14 0.19 0.125 0.165 ...
## $ Whole_weight : num 0.342 1.376 0.185 1.534 0.254 ...
## $ Shucked_weight: num 0.176 0.649 0.07 0.671 0.119 ...
## $ Viscera_weight: num 0.0625 0.361 0.0425 0.379 0.0595 ...
## $ Shell_weight : num 0.0865 0.31 0.065 0.384 0.0675 ...
## $ Rings : num 7 10 7 10 6 10 14 10 9 13 ...
```

```
summary(abalone)
```

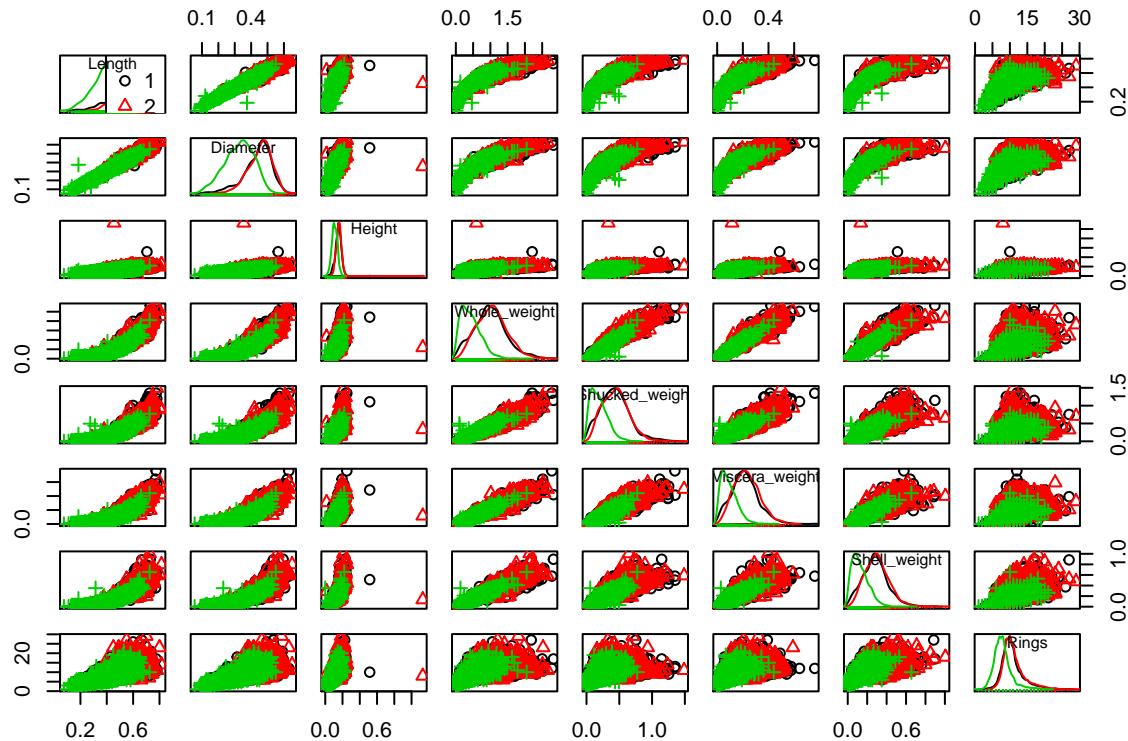
	Sex	Length	Diameter	Height	
##	1:1528	Min. :0.075	Min. :0.0550	Min. :0.0000	
##	2:1307	1st Qu.:0.450	1st Qu.:0.3500	1st Qu.:0.1150	
##	3:1342	Median :0.545	Median :0.4250	Median :0.1400	
##		Mean :0.524	Mean :0.4079	Mean :0.1395	
##		3rd Qu.:0.615	3rd Qu.:0.4800	3rd Qu.:0.1650	
##		Max. :0.815	Max. :0.6500	Max. :1.1300	
##		Whole_weight	Shucked_weight	Viscera_weight	Shell_weight
##		Min. :0.0020	Min. :0.0010	Min. :0.0005	Min. :0.0015
##		1st Qu.:0.4415	1st Qu.:0.1860	1st Qu.:0.0935	1st Qu.:0.1300
##		Median :0.7995	Median :0.3360	Median :0.1710	Median :0.2340
##		Mean :0.8287	Mean :0.3594	Mean :0.1806	Mean :0.2388
##		3rd Qu.:1.1530	3rd Qu.:0.5020	3rd Qu.:0.2530	3rd Qu.:0.3290
##		Max. :2.8255	Max. :1.4880	Max. :0.7600	Max. :1.0050
##		Rings			
##		Min. : 1.000			
##		1st Qu.: 8.000			
##		Median : 9.000			
##		Mean : 9.934			
##		3rd Qu.:11.000			
##		Max. :29.000			

```
apply(abalone[,2:9], 2, sd)
```

	Length	Diameter	Height	Whole_weight	Shucked_weight
##	0.12009291	0.09923987	0.04182706	0.49038902	0.22196295
##	Viscera_weight	Shell_weight	Rings		
##	0.10961425	0.13920267	3.22416903		

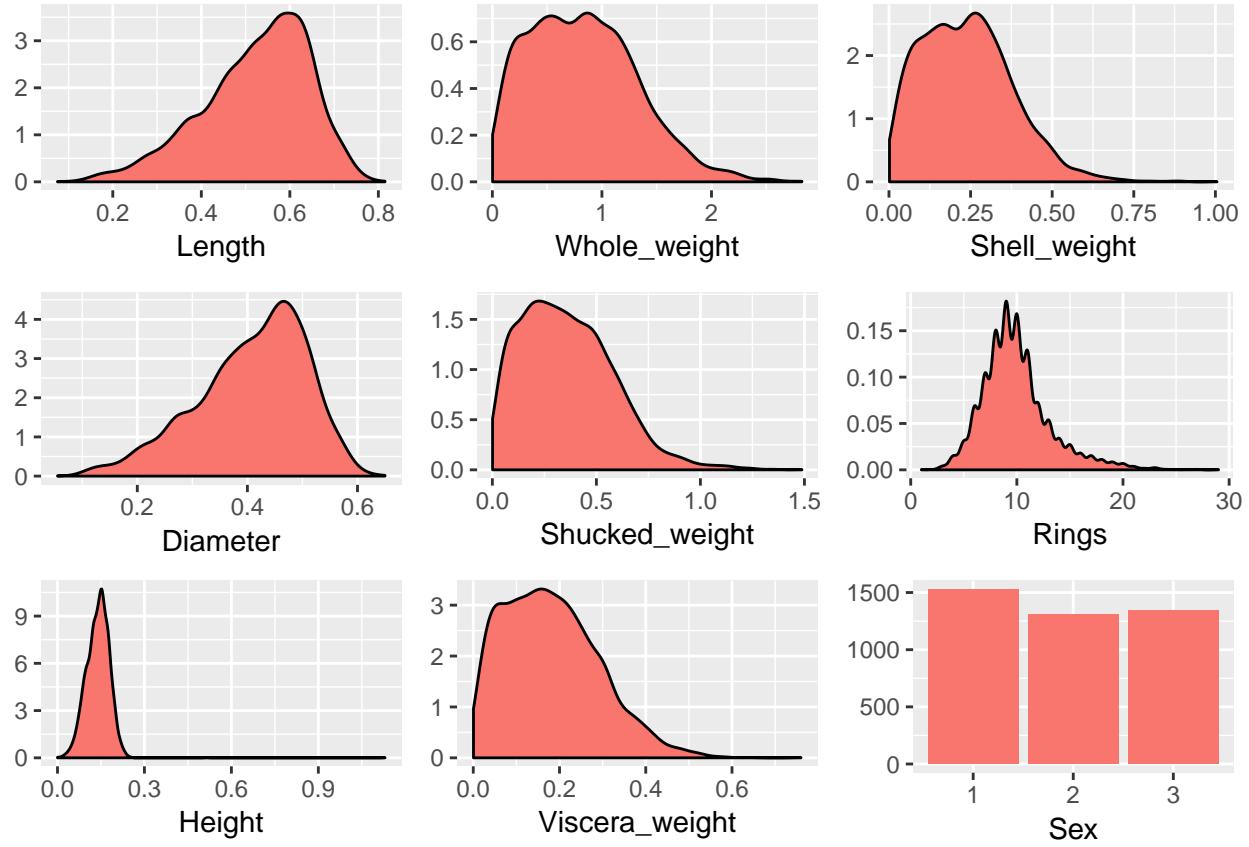
Se muestra como de media las instancias de las almejas tienen en torno a los 10 anillos. El máximo en nuestro dataset es 29, el cual se aleja bastante de la mediana y de la media. En cuanto a otras variables se muestran también algunas instancias bastante alejadas de la media. Por ejemplo en la altura de las almejas el máximo es 1.13, siendo la media de 0.14, pudiéndose tratar este máximo de una anomalía en los datos (una almeja fuera de lo normal).

Para visualizar los datos primero haré un scatterPlotMatrix como en el dataset de clasificación, dando así una visión general del conjunto de datos y así luego ir concretando. Además, diferenciaré los datos según el sexo para ver si existen diferencias en las distribuciones.



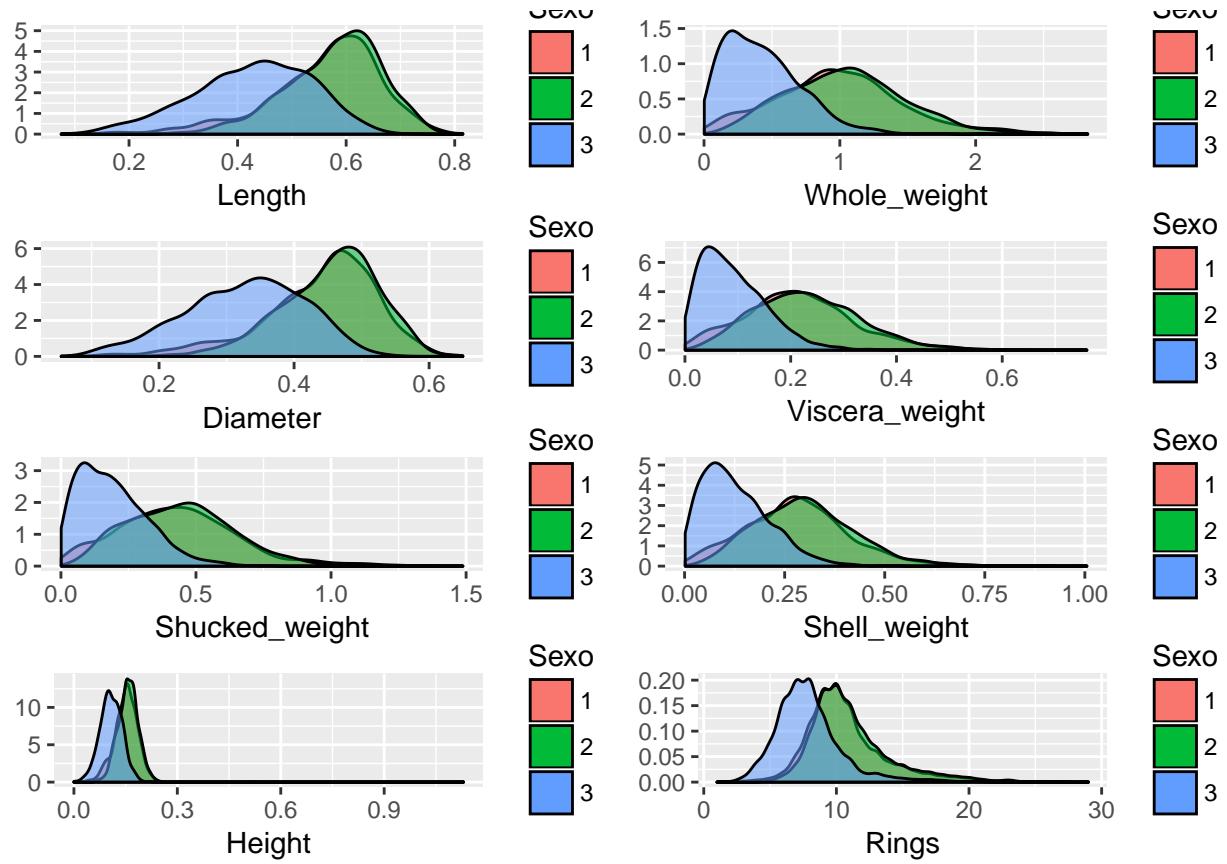
Como se muestra en el scatterPlotMatrix, hay muchísimas variables que muestran correlación entre sí. Por ejemplo, la variable Length muestra correlación con casi todas las demás variables (con unas más que con otras). Con Rings, nuestra variable de salida, las variables que más correlación muestran a simple vista podrían ser Length y Diameter (ambas muy correladas entre sí según su scatterPlot), por lo que estas variables, o alguna de ellas, podría ser importante en nuestro modelo de regresión. Otras curiosidades que se ven es que apenas hay diferencias en la distribución de los datos para sexo masculino y femenino, pero sin embargo, para la variable de sexo “immature” sí que tenemos distinta distribución, lo que es lógico ya que si la almeja es inmadura seguramente tenga menor diámetro u otras medidas de tamaño y masa que las ya maduras. Esta poca diferencia entre sexos puede significar que al realizar nuestro modelo solo tengamos que diferenciar entre maduras y no inmaduras.

Para ver mejor la distribución de los datos, como en el dataset de clasificación, haremos los histogramas de forma suavizada. También he añadido un barplot que muestra el conteo de instancias de cada sexo.



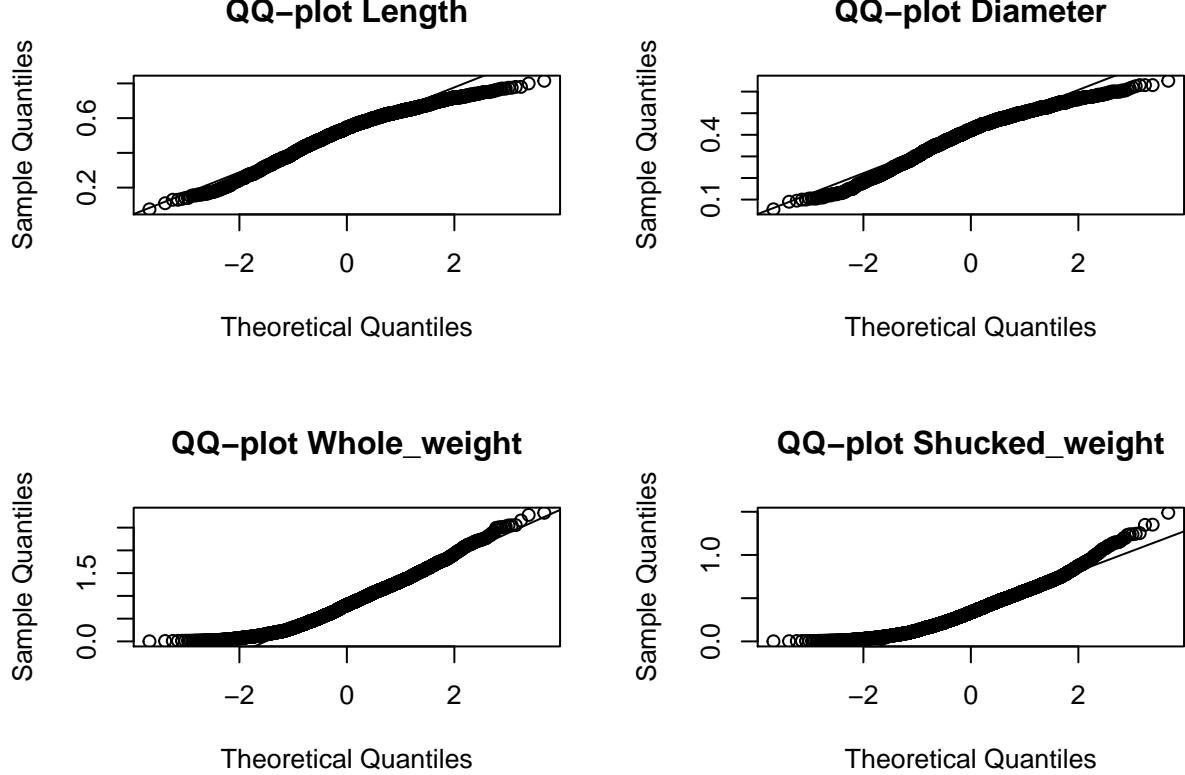
En los histogramas se ve como las variables muestran una leve normalidad en su distribución. Solo podríamos afirmar normalidad en Diameter, Length y Height. Para las demás variables habría que comprobar también empleando qqplots y Shapiro tests. En cuanto al sexo, no existe ningún desbalanceo en este, y como ya hemos comentado antes y veremos más en profundidad en el siguiente plot tampoco muestra una distribución muy distinta, a excepción de las almejas inmaduras frente a las maduras (ya sean hembras o macho). También es importante ver la distribución del número de anillos, que como se aprecia, donde más instancias tenemos es en torno a los 9-10 anillos.

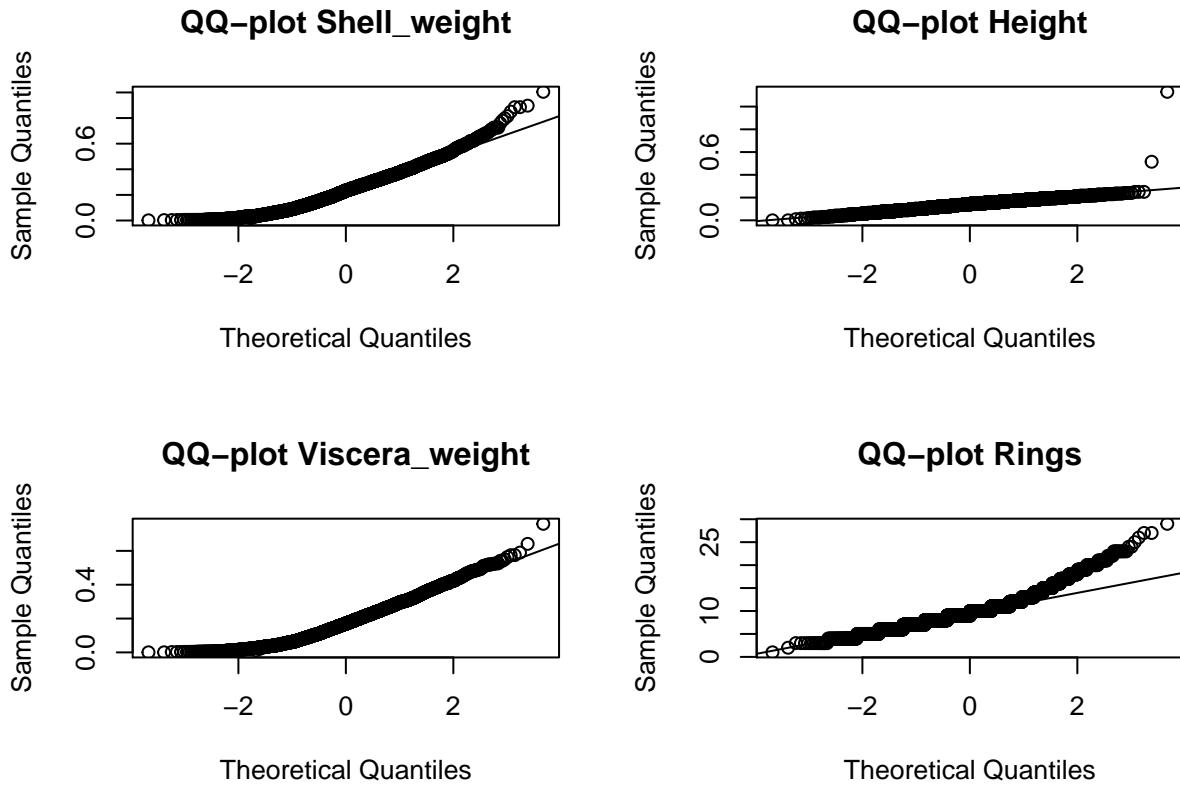
En el siguiente plot vemos una representación de las distribuciones (histogramas) según el distinto sexo.



Como ya se comentó previamente en el ScatterPlotMatrix, se ve cómo entre Male y Female no hay una gran diferencia en las distribuciones, de hecho en casi todos los histogramas se solapan sus gráficos. Sin embargo, el sexo 3, que es Immature, sí muestra una diferencia con respecto a los otros 2, lo que es normal ya que se tratan de almejas “jóvenes” por lo que sus medidas serán siempre o casi siempre inferiores.

Para contrastar mejor la normalidad de nuestros datos, haremos tanto los qqplots correspondientes como tests shapiro. Primero haremos los qqplots.





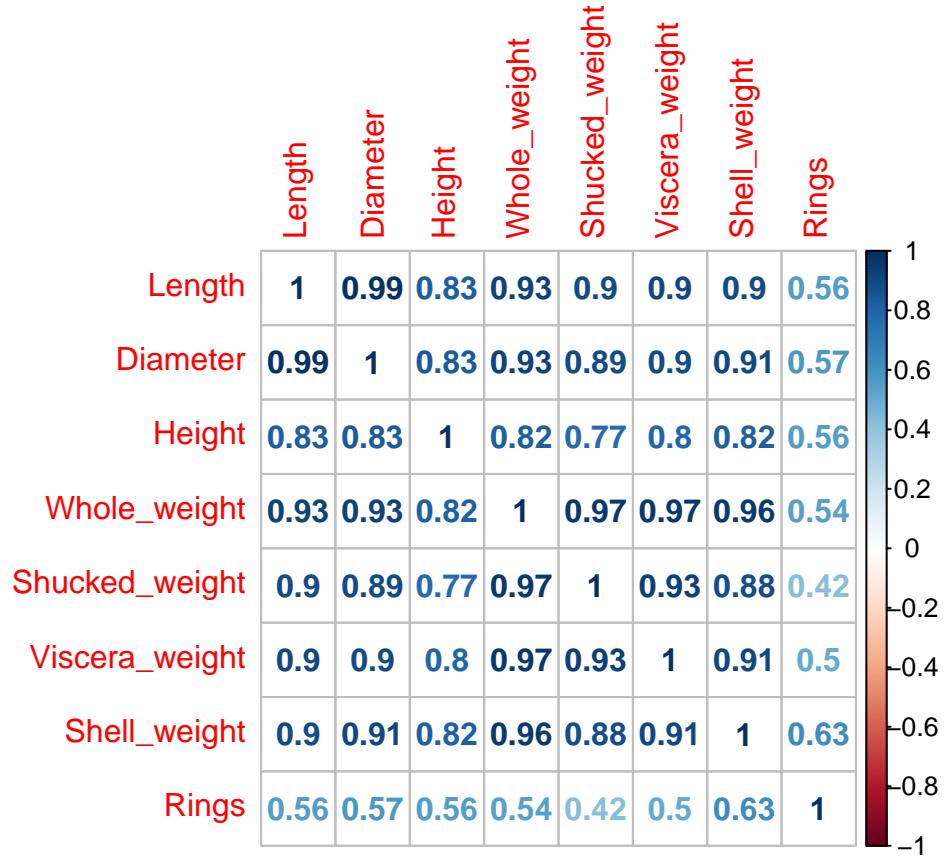
Como se observa en los qqplots, casi todas las distribuciones se acercan a una normal. Por ejemplo para Height es donde mejor se ve. También tenemos distribuciones sesgadas a la derecha como Shell\_weight o Shucked\_weight y otras sesgadas a la izquierda como Length o Diameter.

Como paso final para estudiar la distribución, haremos el test shapiro igual que en el dataset previo.

	p-value
Length	0
Height	0
Shell_weight	0
Shucked_weight	0
Viscera_weight	0
Whole_weight	0
Diameter	0
Rings	0

Observamos cómo el test shapiro nos da para todas las variables un p-value de 0, lo que significa que rechazamos la hipótesis nula y no admitimos nuestros datos como normalmente distribuidos.

Como paso final en nuestro análisis de los datos, vamos a comprobar la correlación entre las variables predictoras y también con la variable a predecir. Para ello, como previamente hicimos, vamos a hacer un corplot.



Se aprecia en el corrplot que hay muchas variables correladas entre sí. Es algo obvio que, por ejemplo, el diámetro se correle con por ejemplo la longitud o que se correle con todas las medidas relacionadas con el peso. También todas las medidas relacionadas con el peso de la almeja se relacionarán bastante, ya que por ejemplo el peso de la viscera sea alto es muy probable que esto signifique que también el peso de la almeja pelada sea alto. Por otra parte, la variable que mayor correlación muestra frente a nuestra variable de salida (Rings) es Shell\_weight, la que aun así tampoco muestra una relación demasiado alta.

El hecho de que haya tantos predictores correlados entre sí, nos puede hacer dudar de la calidad de estos predictores ya que esta correlación significa que en conjunto quizás nos ofrezcan muy poca información para nuestro problema, lo que hace aun más complejo el problema.

## Clasificación

Como ya he descrito en la parte de EDA, mi conjunto de datos es “appendicitis” y consta de 7 variables numéricas ya normalizadas que son 7 medidas médicas y la clase indica si el paciente no tiene apendicitis (0) o si sí la tiene (1). En esta sección se realizarán todas las tareas de clasificación, es decir, el entrenamiento de modelos k-nn, LDA y QDA y la comparación entre estos.

Como hemos visto en regresión, la existencia de variables predictoras correladas entre sí puede afectar negativamente en nuestro proceso de búsqueda de la función  $f$  que, en caso de clasificación, nos realiza las divisiones en el espacio para la asignación de las distintas etiquetas. Por ello, para cada modelo añadiremos una pequeña ampliación que será la supresión de estos predictores correlados para ver si el modelo original mejora.

## K-nn

En esta sección vamos a entrenar distintos modelos de k-nn, viendo qué ‘k’ es la mejor para nuestro problema usando una 10-Fold-CrossValidation y también probaremos variaciones en el modelo, por ejemplo eliminando variables correladas entre sí.

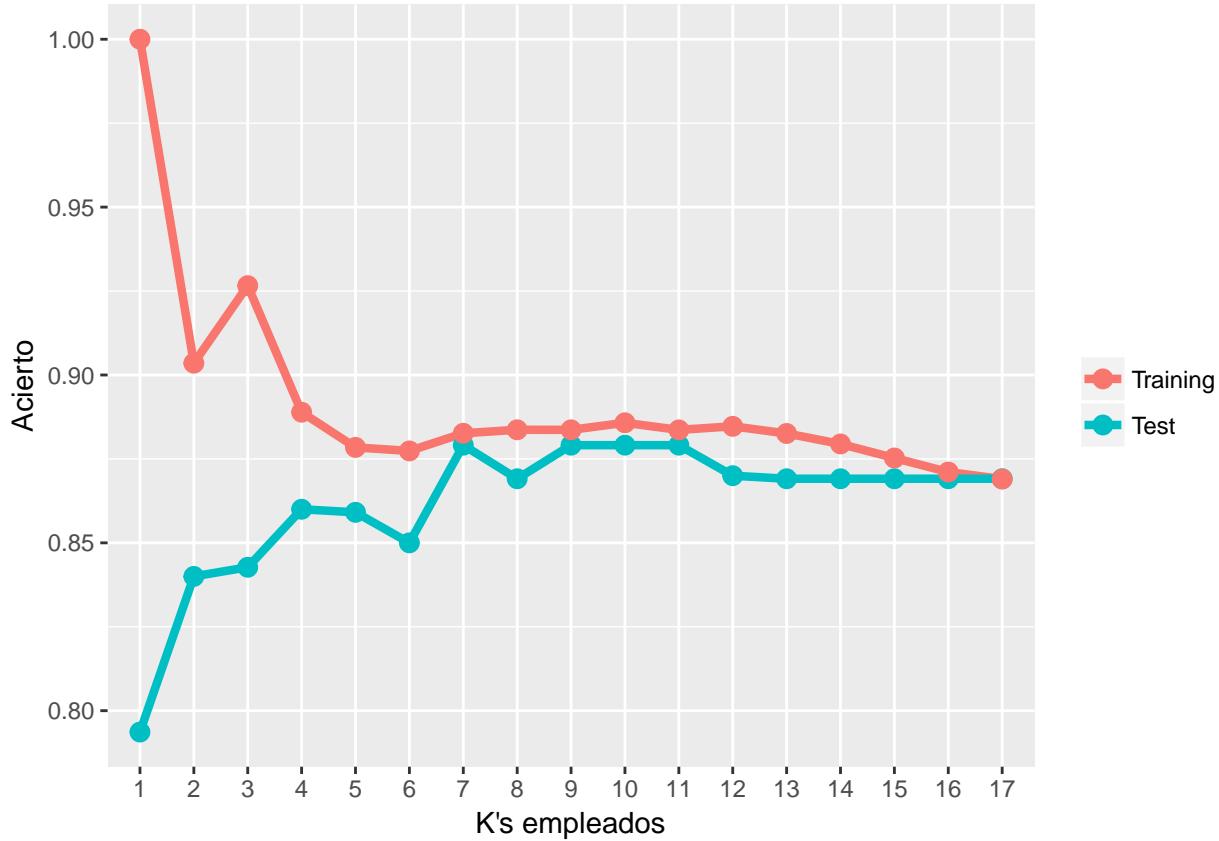
### Búsqueda de un k óptimo

Se empleará una 10-foldCrossValidation para la búsqueda de este ‘k’ y nos fijaremos en el error medio en los conjuntos de tests para elegirlo. Lo primero de todo es leer las particiones. En el siguiente ‘chunk’ de código hago una función para realizar la cross-validation. Para realizar esta validación emplearé la función knn de la librería ‘class’ para realizar la predicción.

```
library(class)
nombre<- "appendicitis"
run_knn_fold<-function(i, x, k, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <- "Y"
  names(x_tst)[In+1] <- "Y"
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  pred <- knn(x_tra[,1:In], k=k, test[,1:In], x_tra$Y)
  #Devolvemos el acierto
  length(pred[pred==test[,ncol(test)]])/length(pred)
}
```

Para comparar el ‘k’ vamos a ejecutar la validación cruzada para todos los k’s hasta 7. Veremos en una gráfica cómo varía la tasa de acierto según el ‘k’ tanto en los aciertos de test como en los de training:

```
compareK <- function(k, num_partitions, name, tt = "test")
{
  mean(sapply(1:num_partitions, run_knn_fold, name, k, tt))
}
tst.results <- sapply(1:17, compareK, 10, "appendicitis", "test")
tra.results <- sapply(1:17, compareK, 10, "appendicitis", "train")
```



Como se observa en el gráfico, con un ‘k’ más pequeño obtenemos un mejor acierto en Training (con k=1 es algo obvio) y conforme vamos subiéndolo esta tasa de acierto se va reduciendo. Sin embargo en los errores de test vemos cómo con el aumento del ‘k’ vamos mejorando el acierto hasta k=7 o k=9, donde ya se mantiene un poco más constante esta tasa de acierto, lo que puede indicar que 7 ó 9 pueden ser los k’s óptimos para nuestro conjunto de datos.

	tst.results	tra.results	ks
7	0.8790909	0.8826316	7
9	0.8790909	0.8836623	9

Los resultados como observamos tanto para k=7 como para 9, son idénticos. Por ello, para contrastar nuestra hipótesis podemos usar el knn implementado en la librería ‘caret’, el cual nos recomienda un ‘k’ realizando también una validación cruzada 10-fold-Cross-Validation, pero en este caso con otras particiones distintas a las usadas en la anterior. Como vemos se muestra que el mejor k para estas pruebas es k=7, por lo que será el ‘k’ que elegiremos como el mejor.

```

require(caret)
#Configuro el entrenamiento para que sea una CV con 10-fold
ctrl <- trainControl(method="repeatedcv", repeats=10)
knnmodel <- train(x=appendicitis[,1:7], y=appendicitis[,8], method="knn", trControl = ctrl)
knnmodel$bestTune

##    k
## 2 7

```

## Ampliaciones de k-nn

Como ya se comentó anteriormente, en esta subsección vamos a probar nuestro modelo suprimiendo predictores correlados. En el caso de knn es probable que esto no afecte para bien al ser un modelo basado en distancias. Como se ha visto en la parte de EDA, hay algunas variables altamente correladas. Por ejemplo, V3, V7 y V1 están muy correladas entre sí, por lo tanto probaremos nuestro modelo dejando solo una de estas variables, en este caso probaremos con dejar solo V1. Lo que haremos será modificar el código de la función run\_knn\_fold anteriormente implementada para que suprima estas 2 variables a los datasets.

Otra ampliación que vamos a proponer es el entrenamiento del modelo sin V6, la cual también muestra cierta correlación con V2.

Para ambas ampliaciones se realizará el mismo procedimiento que con el modelo base, es decir, buscaremos también su mejor 'k' y también mostraremos el resultado para los mejores ks del modelo original (7 y 9). Además mostraremos el mejor acierto y la media de acierto para todos los k's explorados.

	Mejor accuracy	Mejor k	Media accuracy	k=7	k=9
Original	0.8790909	7	0.8615508	0.8790909	0.8790909
Ampliación 1	0.8790909	8	0.8581818	0.8700000	0.8700000
Ampliación 2	0.8972727	14	0.8694652	0.8781818	0.8790909

Como se muestra, la tasa de acierto más alta de los dos primeros modelos (el resultado del mejor 'k') son similares y la media de acierto para todos los k's es mayor en la última ampliación. Sin embargo, el mejor 'k' de este último modelo es quizás demasiado alto, lo que, al existir desbalanceo, puede dar cierta ventaja en la predicción a la clase mayoritaria (0) que, por probabilidad, será más frecuente entre los 14 vecinos más cercanos. Por lo tanto, yo seguiría eligiendo el modelo base con k=7.

## LDA

En esta sección trabajaremos con modelos basados en LDA (Linear Discriminant Analysis). LDA determina una función lineal de manera que intenta separar los datos de distinta clase. Este método tiene como desventaja que asume que las distintas clases siguen una distribución normal con una varianza similar. En nuestro caso, como se ha mostrado en la sección de EDA, las varianzas no son similares y no en todas las variables hay una distribución normal fácilmente apreciable, véase como ejemplo los histogramas para las distintas clases en la variable V6 donde para la clase 1 aparenta tener una distribución demasiado uniforme. Estos últimos aspectos se pueden tener en cuenta para ampliar nuestro modelo, así como la correlación entre variables.

###Modelo base En esta sección probaremos nuestro primer modelo que será el básico teniendo en cuenta todas las variables por igual. Programaremos una función para la validación cruzada similar a la de K-nn.

```
require(MASS)
run_LDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <- "Y"
  names(x_tst)[In+1] <- "Y"
  if (tt== "train") {
    test <-x_tra
  }
  else {
```

```

    test <-x_tst
}
x_tra$Y <- as.factor(x_tra$Y)
test$Y <- as.factor(test$Y)
model <- lda(Y~, data=x_tra)
pred <- predict(model, test)
#Devolvemos el acierto
length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

```

## Ampliaciones

A continuación probaremos a crear modificaciones sobre nuestro modelo base (el anterior con todos los predictores incluídos en nuestro modelo). Las ampliaciones que haremos serán las siguientes:

- Descarte de variables correladas entre sí, igual que hicimos en k-nn, descartando 2 variables que se correlan con V1 (V3 y V7).
- Descarte de V6 por estar correlada con V2.
- Normalización z-score sobre las variables con el fin de igualar las varianzas. Este tipo de normalización transforma la distribución en una  $N(0,1)$ , si los datos siguen una distribución normal.
- Híbrido entre ampliación 1 y ampliación 2.
- Híbrido entre las 3 ampliaciones.

	Accuracy Train CV	Accuracy Test CV
Modelo base	0.8815461	0.8690909
Ampliación 1	0.8920395	0.8881818
Ampliación 2	0.8836513	0.8700000
Ampliación 3	0.8815461	0.8690909
Ampliación 1+2	0.8941447	0.8972727
Ampliación 1+2+3	0.8941447	0.8972727

Se muestra en la tabla como las 2 primeras ampliaciones introducen cierta mejora al modelo base, es decir, el suprimir variables correladas mejora nuestros resultados. El híbrido entre estas 2 mejoras también afectan para bien, obteniendo casi un 90% de porcentaje de acierto. Sin embargo, las ampliaciones empleando una normalización z-score no afecta a los resultados, ni sobre el modelo base ni sobre el híbrido entre las ampliaciones 1 y 2.

## QDA

En este apartado emplearemos un modelo QDA, sobre el que, como en modelos anteriores, hemos hecho algunas ampliaciones o modificaciones sobre un modelo base. QDA (Quadratic Discriminant Analysis) funciona de forma similar a LDA pero estima varianzas distintas para las clases. QDA suele funcionar mejor que LDA en el caso de que las varianzas entre clases sean muy distintas.

### Modelo base

Basándonos en las funciones de validación cruzada ya implementadas para los algoritmos anteriores, vamos a hacer un modelo base empleando todas las variables predictoras. Posteriormente en la sección de ampliaciones

veremos el resultado y lo compararemos con el de las ampliaciones.

```
require(klaR)
run_QDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <- "Y"
  names(x_tst)[In+1] <- "Y"
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  model <- qda(Y~, data=x_tra)
  pred <- predict(model, test)
  #Devolvemos el acierto
  length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}
```

## Ampliaciones

Las ampliaciones que se realizarán para QDA serán similares a los algoritmos previos. Suprimiremos variables correladas entre sí, que además en QDA afecta bastante. Las modificaciones, por lo tanto, son las siguientes:

- Supresión en la predicción de V3 y V7.
- Supresión en la predicción de V6.
- Híbrido entre las 2 anteriores.

	Accuracy Train CV	Accuracy Test CV
Modelo base	0.8690241	0.8109091
Ampliación 1	0.8616776	0.8318182
Ampliación 2	0.8784759	0.8209091
Ampliación 1+2	0.8648355	0.8490909

Vemos que con QDA obtenemos resultados muy pobres en comparación con K-nn y LDA. Puede deberse a que la función sea mucho más simple de lo que QDA modela. Sin embargo, aunque sigan siendo resultados malos comparados con los otros algoritmos, vemos cómo las modificaciones mejoran al modelo base y esto puede deberse a que QDA es sensible a la existencia de predictores correlados. El mejor modelo es el que suprime del modelo las variables 3,7 y 6 por tener correlación estas variables con otras.

## Comparación algoritmos

En este apartado se realizarán los tests estadísticos con el fin de comparar entre sí los distintos algoritmos empleados. Para ello partimos de un dataset con los resultados de los 3 algoritmos en varios datasets y sustituiremos el resultado en nuestro dataset por el que hemos conseguido nosotros (los modelos base).

Tenemos tanto los resultados para training como para test. A continuación se muestran las tablas con los resultados.

X	out_train_knn	out_train_lda	out_train_qda
appendicitis	0.8826316	0.8815461	0.8690241
australian	0.7277419	0.8605475	0.8072464
balance	0.9072122	0.8791122	0.9167999
bupa	0.7405521	0.7024224	0.6447628
contraceptive	0.6168944	0.5236485	0.5314180
haberman	0.7795116	0.7519934	0.7567115
hayes-roth	0.6475524	0.5604167	0.7361111
heart	0.7342975	0.8576132	0.8777778
iris	0.9791045	0.9800000	0.9814815
led7digit	0.7636971	0.7635556	0.7680556
mammographic	0.8160856	0.8274465	0.8196843
monk-2	0.9793684	0.7821826	0.9303010
newthyroid	0.9158409	0.9183457	0.9700283
pima	0.7791914	0.7792266	0.7633125
tae	0.5263460	0.5584858	0.5688072
titanic	0.7892319	0.7760111	0.7732851
vehicle	0.7213300	0.7989229	0.9123989
vowel	0.8378652	0.6457912	0.9701459
wine	0.7745126	1.0000000	0.9956250
wisconsin	0.9739304	0.9614471	0.9588436

X	out_test_knn	out_test_lda	out_test_qda
appendicitis	0.8790909	0.8690909	0.8109091
australian	0.6838235	0.8579710	0.8028986
balance	0.9024546	0.8624101	0.9167905
bupa	0.6865775	0.6837924	0.5991759
contraceptive	0.5448653	0.5091561	0.5173102
haberman	0.7462069	0.7481720	0.7512903
hayes-roth	0.5666667	0.5500000	0.5875000
heart	0.6692308	0.8481481	0.8296296
iris	0.9642857	0.9800000	0.9733333
led7digit	0.7510204	0.7420000	0.6975000
mammographic	0.7977698	0.8241269	0.8194042
monk-2	0.9743632	0.7703433	0.9235535
newthyroid	0.9071429	0.9164502	0.9629870
pima	0.7348861	0.7709930	0.7412403
tae	0.3838095	0.5245833	0.5425000
titanic	0.7850353	0.7760304	0.7733032
vehicle	0.6291452	0.7813305	0.8522409
vowel	0.6428571	0.6030303	0.9191919
wine	0.6959559	0.9944444	0.9888889
wisconsin	0.9735023	0.9592185	0.9519476

Primero, para la comparativa, realizaremos comparativas por pares. Para ello usaremos el test de Wilcoxon, que compara un algoritmo con otro. Compararemos así knn con lda, lda con qda y finalmente knn con qda. En el caso de clasificación, no tenemos que normalizar nuestra medida (en este caso acierto).

En el siguiente chunk de código hago todos los tests

```
LDAvsKNNtrain <- wilcox.test(clasif_train_alumnos$out_train_knn,
                               clasif_train_alumnos$out_train_lda, alternative="two.sided", paired=T)
LDAvsKNNtest <- wilcox.test(clasif_test_alumnos$out_test_knn,
                             clasif_test_alumnos$out_test_lda, alternative="two.sided", paired=T)

QDAvsKNNtrain <- wilcox.test(clasif_train_alumnos$out_train_knn,
                               clasif_train_alumnos$out_train_qda, alternative="two.sided", paired=T)
QDAvsKNNtest <- wilcox.test(clasif_test_alumnos$out_test_knn, clasif_test_alumnos$out_test_qda, alternative=T)

QDAvsLDAtest <- wilcox.test(clasif_train_alumnos$out_train_lda,
                             clasif_train_alumnos$out_train_qda, alternative="two.sided", paired=T)
QDAvsLDAtest <- wilcox.test(clasif_test_alumnos$out_test_lda, clasif_test_alumnos$out_test_qda, alternative=T)
```

Como se muestra en la siguiente tabla, no hay ningún par de algoritmos que según el test muestre un p-value muy bajo y que , por lo tanto, muestren diferencias en su calidad. El más bajo se produce en QDA vs LDA en Training y en QDA vs KNN en test, habiendo en torno a un 83% de probabilidad de que haya diferencias entre los algoritmos.

	p-Value Train	p-Value Test
LDA vs KNN	0.6476555	0.5458755
QDA vs KNN	0.2942524	0.1768532
QDA vs LDA	0.1768532	0.8408222

A continuación realizaré comparación múltiple mediante post Holm test.

```
clasif_train_alumnos <- clasif_train_alumnos[,-1]
clasif_test_alumnos <- clasif_test_alumnos[,-1]
tam <- dim(clasif_train_alumnos)
groups <- rep(1:dim(clasif_train_alumnos), each=tam[1])
pairwise.wilcox.test(x=as.matrix(clasif_train_alumnos), groups, p.adjust="holm", paired=T)

##
##  Pairwise comparisons using Wilcoxon signed rank test
##
## data:  as.matrix(clasif_train_alumnos) and groups
##
##    1     2
## 2 0.65 -
## 3 0.59 0.53
##
## P value adjustment method: holm
tam <- dim(clasif_test_alumnos)
groups <- rep(1:dim(clasif_test_alumnos), each=tam[1])
pairwise.wilcox.test(x=as.matrix(clasif_test_alumnos), groups, p.adjust="holm", paired=T)

##
##  Pairwise comparisons using Wilcoxon signed rank test
##
## data:  as.matrix(clasif_test_alumnos) and groups
##
##    1     2
## 2 1.00 -
```

```

## 3 0.53 1.00
##
## P value adjustment method: holm

```

Como muestran los resultados, en ningún 1vs1 de las matrices hay un p-value muy bajo, por lo que podemos considerar, según el test postHolm, que ninguno de los algoritmos muestran diferencias significativas con respecto a los otros.

## Regresión

En esta sección trabajaremos sobre el dataset “abalone”, ya descrito previamente, realizando distintos modelos de regresión.

### R1. Regresión simple

En este apartado probaremos regresión simple, es decir, empleando solo una variable predictora. Nos fijaremos en el corrplot realizado en el EDA para elegir los predictores con mayor correlación. Además, para los modelos simples, no tendremos en cuenta la variable Sex ya que no tiene sentido realizar una regresión simple con una variable categórica o, en su defecto, con una variable dummy que solo toma valores 1 ó 0.

Como se muestra en el corrplot, las 5 variables más correladas con la variable de salida Rings son Shell\_weight, Diameter, Height y Whole\_weight, por lo que haremos los 5 modelos simples con estas variables. En la siguiente tabla se muestra el R ajustado para cada modelo simple.

Se ve como los  $R^2$  ajustados son bastante malos, siendo el menos malo el de Shell\_weight, que coincide con la variable con más correlación con Rings. Viendo los resultados, el mejor modelo simple por lo tanto es el de Shell\_weight. Sin embargo, para contrastar nuestra hipótesis, realizaremos validación cruzada para ver qué modelo es el mejor. Devolveremos la raíz del medium squared error.

A continuación la descripción de la función para la validación cruzada.

```

run_lm_fold<-function(i,formula,x, tt= "test") {
  file <-paste(x, "-5-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@")
  file <-paste(x, "-5-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@")
  names(x_tst) <- c("Sex", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
  names(x_tra) <- c("Sex", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")

  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  fitMulti=lm(formula,x_tra)
  yprime=predict(fitMulti,test)
  sqrt(sum(abs(test$Rings-yprime)^2)/length(yprime)) ##RMSE
}

```

	CV Training RMSE	CV Test RMSE
Shell_weight	2.509980	2.511924

	CV Training RMSE	CV Test RMSE
Diameter	2.638424	2.638607
Height	2.672885	2.695442
Length	2.678096	2.678399
Whole_weights	2.712608	2.714080

Como se muestra en la tabla anterior, Shell\_weight, como su  $R^2$  ajustado nos hacía pensar, es el mejor modelo simple.

## R2. Regresión múltiple

En este apartado realizaremos distintos modelos de regresión múltiple, además de buscar distintas interacciones entre variables. Primero realizaremos un modelo de regresión múltiple sin interacciones y lineal y posteriormente buscaremos interacciones o no linealidad. Empezamos realizando un modelo genérico con todas las variables y seguiremos un procedimiento hacia atrás para ir quitando las variables con menos p-value.

Además, antes de realizar el modelo, voy a tener en cuenta la variable Sex, pero para ello la haré variable dummy solo teniendo en cuenta que la almeja sea immature o no, es decir, como vimos en el EDA, no se aprecian diferencias en las distribuciones de los datos diferenciando entre Male y Female, pero sí diferenciando Immature de los 2 anteriores.

```
#Hago la variable dummie. 1 será immature y 0 male o female
Sex.inmature <- ifelse(abalone$Sex==1 | abalone$Sex==2, 0, 1)
abalone.dummy <- cbind(Sex.inmature, abalone[,2:ncol(abalone)])
model.base <- lm(Rings~., data=abalone.dummy)
summary(model.base)

##
## Call:
## lm(formula = Rings ~ ., data = abalone.dummy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -10.4919  -1.3027  -0.3398   0.8618  13.9141 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3.93961   0.28423 13.861 < 2e-16 ***
## Sex.inmature -0.85922   0.08958 -9.592 < 2e-16 ***
## Length      -0.44876   1.80896 -0.248   0.804    
## Diameter     11.03325   2.22632  4.956 7.49e-07 ***
## Height      10.73722   1.53571  6.992 3.15e-12 ***
## Whole_weight  8.97230   0.72534 12.370 < 2e-16 ***
## Shucked_weight -19.75356  0.81588 -24.211 < 2e-16 ***
## Viscera_weight -10.61793  1.29262 -8.214 2.82e-16 ***
## Shell_weight   8.74090   1.12466  7.772 9.66e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.194 on 4168 degrees of freedom
## Multiple R-squared:  0.5378, Adjusted R-squared:  0.5369 
## F-statistic: 606.3 on 8 and 4168 DF,  p-value: < 2.2e-16
```

En el summary de nuestro modelo, vemos que Length tiene un p-value muy alto, por lo que será la primera variable que eliminaremos de nuestro modelo. Además vemos un R squared ajustado de 0.54, que mejora por bastante al mejor modelo simple.

```
model.base.nolength <- lm(Rings~.-Length, abalone.dummy)
summary(model.base.nolength)

##
## Call:
## lm(formula = Rings ~ . - Length, data = abalone.dummy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -10.4765 -1.3056 -0.3400  0.8635 13.9285 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3.91573   0.26740 14.644 < 2e-16 ***
## Sex.inmature -0.86066  0.08938 -9.629 < 2e-16 ***
## Diameter     10.53830  0.98778 10.669 < 2e-16 ***
## Height       10.72512  1.53476  6.988 3.23e-12 ***
## Whole_weight  8.97433  0.72522 12.375 < 2e-16 ***
## Shucked_weight -19.76904 0.81340 -24.304 < 2e-16 ***
## Viscera_weight -10.64815 1.28672 -8.275 < 2e-16 ***
## Shell_weight   8.74968  1.12398  7.785 8.76e-15 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.194 on 4169 degrees of freedom
## Multiple R-squared:  0.5378, Adjusted R-squared:  0.537 
## F-statistic: 693.1 on 7 and 4169 DF,  p-value: < 2.2e-16
```

En este caso obtenemos un model con un R squared de 0.54 también, un poco más alto mirando las milésimas, pero una mejora casi insignificante. En los p-values de este modelo vemos que son todos cercanos a 0. El más alto es el de Height, por lo que probaremos otro modelo suprimiendo este predictor.

```
model.base.noheight <- lm(Rings~.-Length-Height, abalone.dummy)
summary(model.base.noheight)

##
## Call:
## lm(formula = Rings ~ . - Length - Height, data = abalone.dummy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -8.1597 -1.3126 -0.3417  0.8697 13.7453 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 4.27930   0.26379 16.222 < 2e-16 ***
## Sex.inmature -0.90566  0.08966 -10.101 < 2e-16 ***
## Diameter     12.64787  0.94590 13.371 < 2e-16 ***
## Whole_weight  9.06125  0.72926 12.425 < 2e-16 ***
## Shucked_weight -19.93199 0.81772 -24.375 < 2e-16 ***
## Viscera_weight -10.22104 1.29262 -7.907 3.34e-15 *** 
## Shell_weight   9.57105  1.12421  8.514 < 2e-16 ***
```

```

## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.206 on 4170 degrees of freedom
## Multiple R-squared: 0.5324, Adjusted R-squared: 0.5317
## F-statistic: 791.3 on 6 and 4170 DF, p-value: < 2.2e-16

```

Vemos cómo quitando la variable Height obtenemos un R squared menor, por lo que consideraremos que nuestro mejor modelo lineal múltiple es el que tiene todos los predictores menos Length, que además, es una variable muy correlada con Diameter.

Ahora probaremos interacciones y no linealidad para nuestro modelo. Para hacerlo partiremos del modelo anterior el cual iremos extendiendo según las formas de las gráficas del ScatterPlotMatrix del EDA. Los pasos dados para buscar un modelo no lineal y con interacciones han sido los siguientes (también más o menos descritos en el código).

- Pruebo Height de forma cuadrática debido al rápido crecimiento de Rings con respecto a esta variable. Podría haber probado exponencial pero finalmente me decidí por cuadrática.
- Pruebo la interacción de Height con otras variables. En este caso con Shell\_weight. Mejoramos el R squared.
- Pruebo añadir logaritmos al modelo debido a la forma de las gráficas de los weights con respecto a Rings, pareciendo esta última que crece de forma logarítmica. Añado log(Shell\_weight) entre los predictores.
- Veo que la interacción de Height con Shell\_weight tiene un p-value muy alto por lo que la elimino y mejoramos el modelo.
- Pruebo a añadir más logaritmos para todas las variables 'weight'. Seguimos mejorando el R.
- Probamos a añadir log(Diameter) que también tiene su gráfica una forma algo logarítmica. Mejoramos un poco pero aparecen predictores con p-value alto.
- Eliminamos predictores con p-value alto, que en este caso era log(Viscera\_weight). Volvemos a mejorar el modelo.
- Ahora vamos a buscar interacción de Sex.inmature con otras variables ya que es importante que la almeja sea inmadura o no para ver la edad. Interaccionamos Sex.inmature con Shell\_weight.
- Pruebo a interaccionar Sex.inmature con Diameter. El R squared se mantiene igual pero la nueva interacción tiene un p-value muy alto.
- Elimino la última interacción y pruebo a quitar algunos log de weights. Empeoramos el R squared.
- Pruebo otras interacciones. En este caso pruebo Diameter con Whole\_weight. Empeoramos el R squared.
- Probamos interacción entre Shell\_weight y Whole\_weight y mejoramos el R squared.
- Añadimos en la interacción previa log(Whole\_weight) y mejoramos el R squared.
- La última prueba, a partir del anterior modelo, quitamos log(Shell\_weight) que tiene un p-value alto. Igualamos el R squared anterior pero este modelo es más simple.

En los siguientes trozos de código no muestro los summary en el pdf para evitar llenar el pdf.

```

#Mejoramos el primer modelo múltiple añadiendo Height^2 ya que en su
#gráfica se ve un crecimiento bastante rápido.
#Podríamos también probar con una exponencial pero finalmente decidí cuadrática.
model.quadratic.height <- lm(Rings~.+I(Height^2)-Length, abalone.dummy)

#Probamos interacciones. Mejoramos el R-Squared
model.quadratic.height.2 <- lm(Rings~.+I(Height^2*Shell_weight)

```

```

+I(Height^2)+I(Height*Shell_weight)-Length, abalone.dummy)

#Otrs funciones no lineales fijandonos en las formas de las gráficas
model.with.log <- lm(Rings~.+I(Height^2*Shell_weight)+I(Height^2)+I(Height*Shell_weight)
+log(Shell_weight)-Length, abalone.dummy)

#Probando más interacciones. Eliminamos elementos con p-value alto
model.with.log.2 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+log(Shell_weight)
-Length, abalone.dummy)

#Más logaritmos. Mejoramos R^2
model.with.log.3 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Viscera_weight))
-Length, abalone.dummy)

#Probamos logaritmos sobre Diameter también, que la forma de su gráfica es parecida.
#Mejoramos un poco. Probemos a eliminar
#algunos logs que tienen p-value alto como el de viscera_weight
model.with.log.4 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Viscera_weight))
+I(log(Diameter))
-Length, abalone.dummy)

#Volvemos a mejorar eliminando viscera_weight
model.with.log.5 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
-Length, abalone.dummy)

#Probamos a interactuar con Sex.inmature con variables como las de los pesos.
#Mejoramos y llegamos a 0.5608
model.Sex.1 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
I(Sex.inmature*Shell_weight)-Length, abalone.dummy)

#Probemos a probar otras interacciones más con Sex. En este caso empeoramos
#y como se ve en el p.value de este producto es alto.
model.Sex.2 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
+I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)
-Length, abalone.dummy)

#Elimino algunos pesos. Pero empeoramos
model.Sex.3 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
+I(log(Whole_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length, abalone.dummy)

#Veamos otras interacciones. Diameter con whole_weight, conseguimos menos
#R squared que con model.sex.1
model.Sex.4 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
+I(Sex.inmature*Shell_weight)
-Length+I(Diameter*Whole_weight), abalone.dummy)

```

```

#Otras Interacciones. Ahora pruebo interacción entre los pesos Shell y Whole.
#Mejoramos el R^2.
model.Sex.5 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
                    I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
                    I(Sex.inmature*Shell_weight)
                    -Length+I(Shell_weight*Whole_weight), abalone.dummy)

#Probamos no linealidad en las últimas interacciones.
#Mejoramos añadiendo logaritmo a Whole_weight
model.Sex.6 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
                    +I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
                    +I(Sex.inmature*Shell_weight)
                    -Length+I(Shell_weight*log(Whole_weight)), abalone.dummy)

#Quitamos log(shell_weight) que tiene un p-value alto. Obtenemos el mismo
#Rsquared pero hace algo más sencillo el modelo por lo que mejor cogemos este.
model.Sex.7 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Whole_weight))+
                    +I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
                    -Length+I(Shell_weight*log(Whole_weight)), abalone.dummy)

```

Aquí la descripción de la función para la validación cruzada:

```

#Función para la validación cruzada.
run_lm_fold<-function(i,formula,x, tt= "test") {
  file <-paste(x, "-5-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@")
  file <-paste(x, "-5-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@")
  names(x_tst) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
  names(x_tra) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
  x_tst$Sex.inmature <- ifelse(x_tst$Sex.inmature == 1 | x_tst$Sex.inmature == 2, 0, 1)
  x_tra$Sex.inmature <- ifelse(x_tra$Sex.inmature == 1 | x_tra$Sex.inmature == 2, 0, 1)

  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  fitMulti=lm(formula,x_tra)
  yprime=predict(fitMulti,test)
  sqrt(sum(abs(test$Rings-yprime)^2)/length(yprime)) ##RMSE
}

```

	Rsquared	Training.RMSE	Test.RMSE
model.base	0.5369441	2.189745	2.215680
model.base.nolenght	0.5370484	2.189830	2.215034
model.base.noheight	0.5317378	2.189830	2.208859
model.quadratic.height	0.5439350	2.173894	2.291793
model.quadratic.height.2	0.5516654	2.155316	2.202499
model.with.log	0.5542020	2.148810	2.399484
model.with.log.2	0.5543090	2.149123	2.168667

	Rsquared	Training.RMSE	Test.RMSE
model.with.log.3	0.5592474	2.136130	2.145640
model.with.log.4	0.5593066	2.135625	2.150325
model.with.log.5	0.5593936	2.135760	2.150421
model.Sex.1	0.5608062	2.132081	2.143332
model.Sex.2	0.5607201	2.132026	2.142478
model.Sex.3	0.5572224	2.141205	2.155722
model.Sex.4	0.5607136	2.131866	2.158218
model.Sex.5	0.5610541	2.131166	2.144786
model.Sex.6	0.5612633	2.130479	2.165705
model.Sex.7	0.5613312	2.130583	2.169666

Se muestra en la tabla anterior para cada modelo propuesto los resultados del  $R^2$  adjusted (con el dataset completo) y los Root Medium Squared Errors tanto en Training como en Test en la validación cruzada. Los 3 primeros modelos son los 3 modelos de regresión múltiple pero lineales que se han probado y los otros son todos los no lineales propuestos. El mejor  $R^2$  ajustado nos lo da el modelo model.Sex.7, pero sin embargo, el que mejor generaliza según nuestra CV es model.Sex.2, que consigue un RMSE de 2.1425. Este modelo es el definido por la función  $Rings = Height^2 + Height * Shell\_weight + log(Shell\_weight) + log(Whole\_weight) + log(Shucked\_weight) + log(Diameter) + Sex.Inmature * Shell\_weight + Sex.Inmature * Diameter + Height + Shell\_weight + Shucked\_weight + Whole\_weight + Shucked\_weight + Diameter + Sex.Inmature$ . Como vemos, se trata de una función muy compleja por lo que no sería del todo descabellado coger algún modelo más simple que este aunque sacrificáramos algo de calidad de nuestro modelo.

En cuanto al  $R^2$  se ve que todos los modelos no lineales mejoran a los lineales, lo que indica que la función de nuestro problema es más compleja que una simple función lineal. En cuanto al RMSE, casi todos los modelos no lineales en Test nos ofrecen mejor resultado (aunque sea por décimas) que la función lineal a excepción de unos pocos.

Es posible que con otro tipo de interacciones y de no linealidades nuestro modelo fuese mejor, pero he decidido realizar estos modelos porque le encontré lógica al ver sus gráficas.

## KNN

En este apartado probaremos KNN en problema de regresión. Knn es un algoritmo basado en similitud, es decir, se fija en los datos más parecidos (más cercanos en el espacio en nuestro caso) para determinar la clase o, en el caso de regresión, el valor de la variable a predecir. En el siguiente chunk de código se define la función para la validación cruzada y una ejecución con el modelo básico (con todos los predictores). Como 'k' usaré k=7 que es el que viene por defecto en la función.

```
require(kknn)

run_kknn_fold<-function(i,formula, k=7, x, tt= "test") {
  file <-paste(x, "-5-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@")
  file <-paste(x, "-5-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@")

  names(x_tst) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight","Viscera_weight","Shell_weight","Rings")
  names(x_tra) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight","Viscera_weight","Shell_weight","Rings")
  x_tst$Sex.inmature <- ifelse(x_tst$Sex.inmature == 1 | x_tst$Sex.inmature == 2, 0, 1)
  x_tra$Sex.inmature <- ifelse(x_tra$Sex.inmature == 1 | x_tra$Sex.inmature == 2, 0, 1)
```

```

if (tt == "train") {
  test <- x_tra
}
else {
  test <- x_tst
}
fitMulti=kknn(formula,x_tra,test,k=k)
yprime <- fitMulti$fitted.values
sqrt(sum(abs(test$Rings-yprime)^2)/length(yprime)) ##RMSE
}

#Modelo general con todos los predictores
mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "test"))

## [1] 2.324244
mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "train"))

## [1] 1.478211

```

En KNN también podemos probar interacciones entre variables. Además también podemos realizar el mismo procedimiento que en regresión múltiple de ver posibles interacciones y no linealidad. Probaré algunos de los modelos anteriormente probados.

```

Training.RMSE <- c(mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "train")))
Test.RMSE <- c(mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "test")))

#Eliminando Length del modelo
Test.RMSE <- c(Test.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.-Length, 7, "abalone",
                                         "test")))
Training.RMSE <- c(Training.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.-Length, 7,
                                              "abalone", "train")))

#Probando mejor modelo anteriormente calculado
Test.RMSE <- c(Test.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)+
                                    I(Height*Shell_weight)+I(log(Shell_weight))+
                                    I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
                                    +I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)-Length, 7, "abalone", "test")))

Training.RMSE <- c(Training.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)+
                                             +I(Height*Shell_weight)+I(log(Shell_weight))+
                                             +I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
                                             +I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)-Length, 7, "abalone", "train")))

#Probando el modelo con menor R^2 anteriormente calculado
Test.RMSE <- c(Test.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)+
                                    +I(Height*Shell_weight)+I(log(Whole_weight))+
                                    +I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)-Length+I(Shell_weight*log(Whole_weight)), 7, "abalone", "test")))

Training.RMSE <- c(Training.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)+
                                             +I(Height*Shell_weight)+I(log(Whole_weight))+
                                             +I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)))

```

```

-Length+I(Shell_weight*log(Whole_weight)), 7, "abalone", "train"))

tabla <- data.frame(Test.RMSE, Training.RMSE)
row.names(tabla) <- c("Modelo base", "Sin Length", "Sex.inmature.2", "Sex.inmature.7")
knitr::kable(tabla)

```

	Test.RMSE	Training.RMSE
Modelo base	2.324244	1.478211
Sin Length	2.307777	1.492830
Sex.inmature.2	2.307927	1.421344
Sex.inmature.7	2.314307	1.427055

En los resultados se muestra cómo el modelo base es peor que el modelo base con regresión lineal (modelo múltiple lineal). Además, con las interacciones, tampoco se consigue mejorar el resultado básico pero sí se mejora un poco el modelo base en el propio KNN. Aun así, es posible que las interacciones, hablando en términos generales, no mejoren mucho para KNN. Podemos concluir por lo tanto, que lm es mejor en problema de regresión que KNN para nuestro problema.

## Comparación algoritmos

En este apartado realizaremos mediante tests estadísticos una comparación entre los distintos algoritmos de regresión empleados (lm y knn) y también se añade el M5. Las tablas de los algoritmos se muestran a continuación, primero para training y después para test.

out_train_lm	out_train_kknn	out_train_m5p
4.820000e+00	2.220000e+00	4.250000e+00
1.700000e-01	6.300000e-03	5.900000e-03
1.129000e+01	3.530000e+00	6.870000e+00
1.079000e+01	3.550000e+00	6.600000e+00
4.481590e+05	2.020880e+05	3.925890e+05
4.826190e+09	1.560869e+09	2.558518e+09
1.070000e+02	2.870000e+01	3.000000e+01
1.618800e-01	7.611000e-02	1.620100e-01
0.000000e+00	0.000000e+00	0.000000e+00
2.100000e-06	1.000000e-06	2.000000e-06
3.945000e+03	2.206000e+03	3.980000e+03
7.230000e+00	1.420000e+00	4.360000e+00
2.061567e+09	5.259870e+08	9.384299e+08
1.354300e-02	8.827000e-03	1.101500e-02
5.350000e+00	1.800000e-01	5.900000e-01
5.520300e-02	1.599800e-02	4.040400e-02
2.430000e+00	2.740000e+00	1.510000e+00
1.565000e+00	2.538000e+00	1.358000e+00

out_test_lm	out_test_kknn	out_test_m5p
4.950000e+00	5.400000e+00	4.680000e+00
1.700000e-01	1.200000e-02	7.000000e-03
1.162000e+01	7.740000e+00	8.240000e+00
1.140000e+01	8.110000e+00	8.350000e+00

out_test_lm	out_test_kknn	out_test_m5p
5.366760e+05	5.661130e+05	5.464640e+05
4.844366e+09	3.845914e+09	3.158145e+09
1.090000e+02	6.835600e+01	3.800000e+01
1.705200e-01	1.732600e-01	1.699600e-01
0.000000e+00	0.000000e+00	0.000000e+00
2.100000e-06	2.400000e-06	2.000000e-06
4.060940e+03	5.841000e+03	4.071040e+03
7.298700e+00	3.196100e+00	5.349100e+00
2.072908e+09	1.425915e+09	1.305419e+09
1.484100e-02	3.003600e-02	1.448300e-02
5.510000e+00	4.500000e-01	1.000000e+00
6.082100e-02	4.743900e-02	8.124800e-02
2.490000e+00	6.790000e+00	1.650000e+00
1.605000e+00	6.060000e+00	1.449000e+00

Primero hago una comparación de 1 vs 1 mediante un test de Wilcoxon. Al contrario que en clasificación, las medidas de calidad de los algoritmos (el MSE) se encuentran en distintas escalas por lo que es necesario normalizar. Para normalizar seguiré las recomendaciones de las diapositivas. Haré primero LM vs KNN tomando KNN como referencia y después KNN vs LM tomando LM como referencia.

```

regr_test_alumnos$out_test_lm[[1]] <- 2.215680^2
regr_train_alumnos$out_train_lm[[1]] <- 2.189745^2
regr_test_alumnos$out_test_kknn[[1]] <- 2.324244^2
regr_train_alumnos$out_train_kknn[[1]] <- 1.478211^2

##lm (other) vs knn(ref)
# + 0.1 porque wilcoxR falla para valores==0 en la tabla
difs<-(regr_train_alumnos$out_train_lm - regr_train_alumnos$out_train_kknn) / regr_train_alumnos$out_train_kknn
wilc_1_2 <-cbind(ifelse(difs<0, abs(difs)+0.1, 0+0.1), ifelse(difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) <-c(colnames(regr_train_alumnos)[1], colnames(regr_train_alumnos)[2])

LMvsKNNtra<-wilcox.test( wilc_1_2[,1], wilc_1_2[,2], alternative = "two.sided", paired=TRUE)

difs<-(regr_test_alumnos$out_test_lm - regr_test_alumnos$out_test_kknn) / regr_test_alumnos$out_test_kknn
wilc_1_2 <-cbind(ifelse(difs<0, abs(difs)+0.1, 0+0.1), ifelse(difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) <-c(colnames(regr_test_alumnos)[1], colnames(regr_test_alumnos)[2])

LMvsKNNtst<-wilcox.test(wilc_1_2[,1], wilc_1_2[,2], alternative = "two.sided", paired=TRUE)

```

Como se muestra en la siguiente tabla, en los resultados en Training hay una alta probabilidad de que ambos algoritmos sean distintos. Sin embargo, en los resultados en test, no se puede asegurar que muestren diferencias significativas (solo un 23.4% de probabilidad aproximadamente). Esto quiere decir que Knn, el algoritmo que proponemos como referente, aparentemente realiza sobreaprendizaje.

	Training.p.value	Test.p.value
LM vs KNN	0.0003281	0.7660294

Ahora realizamos un test de Friedman para la comparación múltiple:

```

test_friedman.train<-friedman.test(as.matrix(regr_train_alumnos))
test_friedman.test<-friedman.test(as.matrix(regr_test_alumnos))

```

p.value.Training	p.value.Test
3.84e-05	0.014666

Vemos efectivamente cómo en Train obtenemos un p-value muy bajo, por lo que hay una muy alta probabilidad de que al menos 2 algoritmos presenten diferencias entre sí. También, en el p-value para nuestra comparación de los resultados para test, es bastante bajo, por lo que hay en torno a un 90% de probabilidad de que al menos 2 de los algoritmos presenten diferencias entre sí.

Con el test post-Holm vemos una comparación múltiple 1vs1.

```
tam <-dim(regr_train_alumnos)
groups <-rep(1:tam[2], each=tam[1])
pairwise.wilcox.test(as.matrix(regr_train_alumnos), groups, p.adjust= "holm", paired = TRUE)

##
##  Pairwise comparisons using Wilcoxon signed rank test
##
##  data:  as.matrix(regr_train_alumnos) and groups
##
##    1      2
## 2 0.0031 -
## 3 0.0032 0.0032
##
## P value adjustment method: holm

tam <-dim(regr_test_alumnos)
groups <-rep(1:tam[2], each=tam[1])
pairwise.wilcox.test(as.matrix(regr_test_alumnos), groups, p.adjust= "holm", paired = TRUE)

##
##  Pairwise comparisons using Wilcoxon signed rank test
##
##  data:  as.matrix(regr_test_alumnos) and groups
##
##    1      2
## 2 0.580 -
## 3 0.081 0.108
##
## P value adjustment method: holm
```

Como vemos, para train tenemos un p-value muy bajo para todas las combinaciones, lo que indica que todos los algoritmos se comportan de forma distinta en training. En test, sin embargo, los algoritmos 1 y 2 (Knn y LM) muestran un p-value alto, por lo que no se le puede considerar que muestren diferencia. Sin embargo, el algoritmo 3 (M5) sí que muestra un p-value muy bajo con respecto a ambos anteriores algoritmos, por lo que tiene unos resultados bastante distintos.

## Anexos - Código

A continuación pongo todo el código empleado para realizar el proyecto. Lo he hecho con Rmarkdown por lo que la opción de exportar a .R exporta de forma estructurada.

## EDA

```
#' ---
#' title: "Trabajo Introducción a la ciencia de datos"
#' author: "Miguel López Campos"
#' date: "25 de noviembre de 2017"
#' output:
#'   pdf_document: default
#'   html_document:
#'     df_print: paged
#' ---
#'
## ----setup, include=FALSE-----
knitr::opts_chunk$set(echo = TRUE)

#'
## ----echo=FALSE, message=FALSE, warning=FALSE-----
library(ggplot2)
library(dplyr)
library(tidyr)
library(corrplot)
set.seed(12345)
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                    ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout)))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
```

```

        layout.pos.col = matchidx$col))
    }
}
}

#'
#' #Análisis de los datos
#' En esta sección se hará un análisis descriptivo de los datos usando para ello medidas que nos den pistas
#'
#' ##Clasificación - Appendicitis dataset
#' En esta sección se hará un análisis de los datos que se usarán para el problema de clasificación. Nuestro objetivo es predecir si una persona tiene o no apendicitis.
#'
## ----echo=FALSE-----
appendicitis <- read.csv("D:/Documentos/Master/ICD/Trabajo/Datasets Clasificacion/Datasets Clasificacion/appendicitis.csv")

#'
#' En el siguiente código mostramos las primeras medidas descriptivas de nuestro conjunto de datos.
## -----
dim(appendicitis)

## -----
appendicitis[!complete.cases(appendicitis),]

#' Como vemos en la ejecución del anterior trozo de código, no hay ningún caso no completo (con missing values).
#'
## -----
summary(appendicitis)

#' Como vemos, el conjunto de datos consta de 106 instancias con 7 variables. En nuestro 'summary' del conjunto de datos vemos que la variable 'class' es de tipo factor.
#'
## ----echo=FALSE-----
names(appendicitis)[8] <- "class"
appendicitis$class <- as.factor(appendicitis$class)

#'
## -----
str(appendicitis)
apply(appendicitis[,1:7], 2, sd)

#' Como vemos en la ejecución de la función 'str' tenemos un dataframe con 7 variables, todas numéricas.
#'
#' A continuación vamos a comprobar mediante scatter plots si se puede intuir gráficamente si existe alguna relación entre las variables.
## ----message=FALSE, echo=FALSE-----
library(car)
scatterplotMatrix(~V1+V2+V3+V4+V5+V6+V7|class, data=appendicitis,
                 smoother="", reg.line="")

#' Según se muestra en la matriz de scatterPlots vemos cómo hay variables que tienen claramente cierta tendencia.
#'
#' Ahora vamos a hacer un poco más de hincapié en ver la distribución de las distintas variables. Seguiremos con el mismo dataset.
## ----echo=FALSE-----

```

```

p1 <- ggplot(appendicitis, aes(x=V1))+geom_density(aes(fill="red"))+
  theme(legend.position = 'none', axis.title.y = element_blank())
p2 <- ggplot(appendicitis, aes(x=V2))+geom_density(aes(fill="red"))+
  theme(legend.position = 'none', axis.title.y = element_blank())
p3 <- ggplot(appendicitis, aes(x=V3))+geom_density(aes(fill="red"))+
  theme(legend.position = 'none', axis.title.y = element_blank())
p4 <- ggplot(appendicitis, aes(x=V4))+geom_density(aes(fill="red"))+
  theme(legend.position = 'none', axis.title.y = element_blank())
p5 <- ggplot(appendicitis, aes(x=V5))+geom_density(aes(fill="red"))+
  theme(legend.position = 'none', axis.title.y = element_blank())
p6 <- ggplot(appendicitis, aes(x=V6))+geom_density(aes(fill="red"))+
  theme(legend.position = 'none', axis.title.y = element_blank())
p7 <- ggplot(appendicitis, aes(x=V7))+geom_density(aes(fill="red"))+
  theme(legend.position = 'none', axis.title.y = element_blank())

multiplot(p1,p2,p3,p4,p5,p6,p7,cols=2)

aux <- gather(appendicitis, var, value, -class)
ggplot(aux, aes(y=value, x=factor(var)))+geom_boxplot(aes(fill=factor(var)))
+guides(fill=guide_legend(title="Variables"))+labs(x="",y="")
#ggplot(aux, aes(x=value))+geom_density(aes(fill=factor(var), alpha=0.05))

#' En los boxplots anteriores también se puede apreciar una idea de las distribuciones. Como se observa
#'
#' Podemos también comprobar la normalidad de los datos observando sus qqplots. Usaremos estos gráficos
#'
## ----echo=FALSE-----
par(mfrow=c(2,2))
qnorm(appendicitis$V2, main="QQ-plot V2")
qqline(appendicitis$V2)
qnorm(appendicitis$V4, main="QQ-plot V4")
qqline(appendicitis$V4)
qnorm(appendicitis$V5, main="QQ-plot V5")
qqline(appendicitis$V5)

#
#' En los anteriores qqplots, se aprecia que la distribución de la variable V2 se puede considerar una
#'
#' Con un test shapiro ahora podemos también comprobar la normalidad de los datos. Si los test tienen un
## ----echo=F-----
#Hacemos el test shapiro
aux <- apply(appendicitis[1:7], 2, shapiro.test)
v1 <- aux$V1$p.value
v2 <- aux$V2$p.value
v3 <- aux$V3$p.value
v4 <- aux$V4$p.value
v5 <- aux$V5$p.value
v6 <- aux$V6$p.value
v7 <- aux$V7$p.value
table <- data.frame(v1,v2,v3,v4,v5,v6,v7)

```

```

row.names(table) <- c("p-value")
knitr::kable(table)

#'
#'
##' Como último paso en nuestro análisis relacionado con la distribución de los datos, vamos a ver cómo
## ----echo=F-----
p1 <- ggplot(appendicitis, aes(x=V1))+geom_density(aes(fill=factor(class), alpha=0.05))+guides(fill=guide)
+theme(axis.title.y = element_blank())
p2 <- ggplot(appendicitis, aes(x=V2))+geom_density(aes(fill=factor(class), alpha=0.05))+guides(fill=guide)
+theme(axis.title.y = element_blank())
p3 <- ggplot(appendicitis, aes(x=V3))+geom_density(aes(fill=factor(class), alpha=0.05))+guides(fill=guide)
+theme(axis.title.y = element_blank())
p4 <- ggplot(appendicitis, aes(x=V4))+geom_density(aes(fill=factor(class), alpha=0.05))+guides(fill=guide)
+theme(axis.title.y = element_blank())
p5 <- ggplot(appendicitis, aes(x=V5))+geom_density(aes(fill=factor(class), alpha=0.05))+guides(fill=guide)
+theme(axis.title.y = element_blank())
p6 <- ggplot(appendicitis, aes(x=V6))+geom_density(aes(fill=factor(class), alpha=0.05))+guides(fill=guide)
+theme(axis.title.y = element_blank())
p7 <- ggplot(appendicitis, aes(x=V7))+geom_density(aes(fill=factor(class), alpha=0.05))+guides(fill=guide)
+theme(axis.title.y = element_blank())

multiplot(p1,p2,p3,p4,p5,p6,p7,cols=2)

#'
##' Se muestra en este último multiplot de la distribución según la clase cómo en las variables V1,V3 y
#'
##' A la hora de aplicar algoritmos como LDA y QDA es conveniente también ver la varianza de las variables
## ----echo=F-----
var.class1 <- apply(appendicitis[appendicitis$class==1,1:7], 2, var)
var.class2 <- apply(appendicitis[appendicitis$class==0,1:7], 2, var)
tabla <- data.frame(var.class1, var.class2)
names(tabla) <- c("Var. clase=1", "Var. clase=0")
knitr::kable(tabla)

#'
##' A continuación, vamos a analizar la correlación entre las variables de forma más concreta que con la
#'
## ----echo=FALSE-----
y <- cor(appendicitis[,1:7])
corrplot(y, method="number")

#'
##' Otro aspecto a tener en cuenta para luego la hora del proceso de aprendizaje, aunque en esta asignatura
## -----
length(appendicitis$class[appendicitis$class == 1])/nrow(appendicitis)
length(appendicitis$class[appendicitis$class == 0])/nrow(appendicitis)

##' Sí se aprecia un cierto desbalanceo, siendo el 80% de las instancias de clase 0 (no tiene apendicitis)
#'
##' ##Regresión - Abalone
##' El conjunto de datos para regresión se trata de Abalone. Este conjunto describe en cada instancia un

```

```

#'
#'
## ----echo=F-----
abalone <- read.csv("./abalone.dat", header=FALSE, comment.char="@")
names(abalone) <- c("Sex", "Length", "Diameter", "Height", "Whole_weight", "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
abalone$Sex <- as.factor(abalone$Sex)
abalone$Rings <- as.numeric(abalone$Rings)

#' Todas las variables son numéricas excepto la variable Sex, la cual es categórica e indica el sexo (1=femenino, 0=macho)
## -----
str(abalone)

summary(abalone)

apply(abalone[,2:9], 2, sd)

#' Se muestra como de media las instancias de las almejas tienen en torno a los 10 anillos. El máximo es de 20 y el mínimo de 0.
#'
#' Para visualizar los datos primero haré un scatterPlotMatrix como en el dataset de clasificación, dando una visión general de las variables.
#'
## ----echo=F-----
scatterplotMatrix(~Length+Diameter+Height+Whole_weight+Shucked_weight+Viscera_weight+Shell_weight+Rings,
                  data=abalone, smoother="", reg.line="")

#'
#' Como se muestra en el scatterPlotMatrix, hay muchísimas variables que muestran correlación entre sí.
#'
#' Para ver mejor la distribución de los datos, como en el dataset de clasificación, haremos los histogramas.
#'
## ----echo = F-----
p1 <- ggplot(abalone, aes(x=Length))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p2 <- ggplot(abalone, aes(x=Diameter))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p3 <- ggplot(abalone, aes(x=Height))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p4 <- ggplot(abalone, aes(x=Whole_weight))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p5 <- ggplot(abalone, aes(x=Shucked_weight))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p6 <- ggplot(abalone, aes(x=Viscera_weight))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p7 <- ggplot(abalone, aes(x=Shell_weight))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p8 <- ggplot(abalone, aes(x=Sex))+geom_bar(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
p9 <- ggplot(abalone, aes(x=Rings))+geom_density(aes(fill="red"))
+theme(legend.position = 'none', axis.title.y = element_blank())
multiplot(p1,p2,p3,p4,p5,p6,p7,p9,p8,cols=3)

##Prueba
#aux <- gather(abalone, var, value, -Sex, -Rings)
#ggplot(aux, aes(y=Rings, x=factor(Sex)))+geom_boxplot(aes(fill=factor(Sex)))+guides(fill=guide_legend(

```

```

#'
#' En los histogramas se ve como las variables muestran una leve normalidad en su distribución. Solo por
#'
#' En el siguiente plot vemos una representación de las distribuciones (histogramas) según el distinto .
#'
## ----echo=FALSE-----
p1 <- ggplot(abalone, aes(x=Length))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=guide_
  theme(axis.title.y = element_blank())
p2 <- ggplot(abalone, aes(x=Diameter))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=guide_
  theme(axis.title.y = element_blank())
p3 <- ggplot(abalone, aes(x=Shucked_weight))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=
  theme(axis.title.y = element_blank())
p4 <- ggplot(abalone, aes(x=Height))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=guide_
  theme(axis.title.y = element_blank())
p5 <- ggplot(abalone, aes(x=Whole_weight))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=
  theme(axis.title.y = element_blank())
p6 <- ggplot(abalone, aes(x=Viscera_weight))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=
  theme(axis.title.y = element_blank())
p7 <- ggplot(abalone, aes(x=Shell_weight))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=
  theme(axis.title.y = element_blank())
p8 <- ggplot(abalone, aes(x=Rings))+geom_density(aes(fill=factor(Sex), alpha=0.05))+guides(fill=guide_
  theme(axis.title.y = element_blank())
multiplot(p1,p2,p3,p4,p5,p6,p7,p8,cols=2)

#' Como ya se comentó previamente en el ScatterPlotMatrix, se ve cómo entre Male y Female no hay una gr
#'
#' Para contrastar mejor la normalidad de nuestros datos, haremos tanto los qqplots correspondientes co
#'
## ----echo=F-----
par(mfrow=c(2,2))
qnorm(abalone$Length, main="QQ-plot Length")
qqline(abalone$Length)
qnorm(abalone$Diameter, main="QQ-plot Diameter")
qqline(abalone$Diameter)
qnorm(abalone$Whole_weight, main="QQ-plot Whole_weight")
qqline(abalone$Whole_weight)
qnorm(abalone$Shucked_weight, main="QQ-plot Shucked_weight")
qqline(abalone$Shucked_weight)

#'
## ----echo=F-----
par(mfrow=c(2,2))
qnorm(abalone$Shell_weight, main="QQ-plot Shell_weight")
qqline(abalone$Shell_weight)
qnorm(abalone$Height, main="QQ-plot Height")
qqline(abalone$Height)
qnorm(abalone$Viscera_weight, main="QQ-plot Viscera_weight")
qqline(abalone$Viscera_weight)
qnorm(abalone$Rings, main="QQ-plot Rings")
qqline(abalone$Rings)

#'
#' Como se observa en los qqplots, casi todas las distribuciones se acercan a una normal. Por ejemplo p

```

```

#'
#' Como paso final para estudiar la distribución, haremos el test shapiro igual que en el dataset previo
#'
## ----echo=F-----
aux <- apply(abalone[,2:9], 2, shapiro.test)
Length <- aux$Length$p.value
Height<- aux$Height$p.value
Shell_weight <- aux$Shell_weight$p.value
Viscera_weight <- aux$Viscera_weight$p.value
Shucked_weight <- aux$Shucked_weight$p.value
Whole_weight <- aux$Whole_weight$p.value
Diameter <- aux$Diameter$p.value
Rings <- aux$Rings$p.value
table <- data.frame(Length, Height, Shell_weight, Shucked_weight, Viscera_weight, Whole_weight, Diameter)

row.names(table) <- c("p-value")
table <- t(table)
knitr::kable(table)

#'
#' Observamos cómo el test shapiro nos da para todas las variables un p-value de 0, lo que significa que
#'
#' Como paso final en nuestro análisis de los datos, vamos a comprobar la correlación entre las variables
#'
## ----echo = F-----
y <- cor(abalone[,2:9])
corrplot(y, method="number")

#'
#' Se aprecia en el corrplot que hay muchas variables correladas entre sí. Es algo obvio que, por ejemplo,
#'
#' El hecho de que haya tantos predictores correlados entre sí, nos puede hacer dudar de la calidad de
#'

```

## Clasificación

```

#' #Clasificación
#' Como ya he descrito en la parte de EDA, mi conjunto de datos es "appendicitis" y consta de 7 variables
#'
#' Como hemos visto en regresión, la existencia de variables predictoras correladas entre sí puede afectar
#'
#' ##K-nn
#' En esta sección vamos a entrenar distintos modelos de k-nn, viendo qué 'k' es la mejor para nuestro problema
#'
#' ###Búsqueda de un k óptimo
#' Se empleará una 10-foldCrossValidation para la búsqueda de este 'k' y nos fijaremos en el error medio cuadrático
#'
## ----echo=F-----

library(class)
nombre<-"appendicitis"
run_knn_fold<-function(i, x, k, tt= "test") {

```

```

file <-paste(x, "-10-", i, "tra.dat", sep="")
x_tra<-read.csv(file, comment.char="@", header=F)
file <-paste(x, "-10-", i, "tst.dat", sep="")
x_tst<-read.csv(file, comment.char="@", header=F)
In <-length(names(x_tra)) -1
names(x_tra)[In+1] <-"Y"
names(x_tst)[In+1] <-"Y"
if (tt=="train") {
  test <-x_tra
}
else {
  test <-x_tst
}
x_tra$Y <- as.factor(x_tra$Y)
test$Y <- as.factor(test$Y)
pred <- knn(x_tra[,1:In], k=k, test[,1:In], x_tra$Y)
#Devolvemos el acierto
length(pred[pred==test[,ncol(test)]]))/length(pred)
}

#' Para comparar el 'k' vamos a ejecutar la validación cruzada para todos los k's hasta 7. Veremos en u
## -----
compareK <- function(k, num_partitions, name, tt = "test")
{
  mean(sapply(1:num_partitions, run_knn_fold, name, k, tt))
}
tst.results <- sapply(1:17, compareK, 10, "appendicitis", "test")
tra.results <- sapply(1:17, compareK, 10, "appendicitis", "train")

#'
## ----echo=F-----
ks <- as.factor(1:17)
errors <- data.frame(tst.results, tra.results, ks)
errors_2 <- gather(errors, var, value, -ks)
ggplot(errors_2, aes(x=ks, y=value, group=var, col=var)) + geom_line(size=1.5) +
  geom_point(size=3, fill="white") + scale_shape_discrete(name="", labels=c("Training", "Test")) + scale_
#'
#' Como se observa en el gráfico, con un 'k' más pequeño obtenemos un mejor acierto en Training (con k=
## ----echo=FALSE-----
knitr::kable(errors[c(7,9),])

#'
#' Los resultados como observamos tanto para k=7 como para 9, son idénticos. Por ello, para contrastar
#'
## ----warning=FALSE, message=FALSE-----
require(caret)
#Configuro el entrenamiento para que sea una CV con 10-fold
ctrl <- trainControl(method="repeatedcv", repeats=10)
knnmodel <- train(x=appendicitis[,1:7], y=appendicitis[,8], method="knn", trControl = ctrl)
knnmodel$bestTune

```

```

#'
#'
## Ampliaciones de k-nn
## Como ya se comentó anteriormente, en esta subsección vamos a probar nuestro modelo suprimiendo predicciones
## ----echo=FALSE-----
#NOTA: Las funciones podría habérmelas ahorrado y dejar una para cada modelo sin tener que hacer una para cada uno
#pero fui consciente tarde (no caí en el uso del argumento formula de las funciones) y ya lo he dejado así
run_knn_fold<-function(i, x, k, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  x_tra <- x_tra[,c(-3,-7)]
  x_tst <- x_tst[,c(-3,-7)]
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <-"Y"
  names(x_tst)[In+1] <-"Y"
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  pred <- knn(x_tra[,1:In], k=k, test[,1:In], x_tra$Y)
  #Devolvemos el acierto
  length(pred[pred==test[,ncol(test)]])/length(pred)
}

#'
## Otra ampliación que vamos a proponer es el entrenamiento del modelo sin V6, la cual también muestra mejoras
## Para ambas ampliaciones se realizará el mismo procedimiento que con el modelo base, es decir, buscar los mejores parámetros
## ----echo=FALSE-----
tst.results2 <- sapply(1:17, compareK, 10, "appendicitis", "test")

#'
#'
## ----echo=F-----
run_knn_fold<-function(i, x, k, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  x_tra <- x_tra[,-6]
  x_tst <- x_tst[,-6]
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <-"Y"
  names(x_tst)[In+1] <-"Y"
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  pred <- knn(x_tra[,1:In], k=k, test[,1:In], x_tra$Y)
  #Devolvemos el acierto
  length(pred[pred==test[,ncol(test)]])/length(pred)
}

```

```

    test <-x_tra
}
else {
  test <-x_tst
}
x_tra$Y <- as.factor(x_tra$Y)
test$Y <- as.factor(test$Y)
pred <- knn(x_tra[,1:In], k=k, test[,1:In], x_tra$Y)
#Devolvemos el acierto
length(pred[pred==test[,ncol(test)]])/length(pred)
}

tst.results3 <- sapply(1:17, compareK, 10, "appendicitis", "test")

mejores.ks <- c(which.max(tst.results), which.max(tst.results2), which.max(tst.results3))
mejor.acierto <- c(max(tst.results), max(tst.results2), max(tst.results3))
nombres <- c("Original", "Ampliación 1","Ampliación 2")
k7 <- c(tst.results[7], tst.results2[7], tst.results3[7])
k9 <- c(tst.results[9], tst.results2[9], tst.results3[9])
medias <- c(mean(tst.results), mean(tst.results2), mean(tst.results3))
mods <- data.frame(mejor.acierto, mejores.ks, medias, k7, k9)
row.names(mods) <- nombres
names(mods) <- c("Mejor accuracy", "Mejor k", "Media accuracy","k=7", "k=9")
knitr::kable(mods)

#
#' Como se muestra, la tasa de acierto más alta de los dos primeros modelos (el resultado del mejor 'k').
#
#' ##LDA
#' En esta sección trabajaremos con modelos basados en LDA (Linear Discriminant Analysis). LDA determina
#' ####Modelo base
#' En esta sección probaremos nuestro primer modelo que será el básico teniendo en cuenta todas las var
## ----warning=F, message=F-----
require(MASS)
run_LDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <- "Y"
  names(x_tst)[In+1] <- "Y"
  if (tt=="train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  model <- lda(Y~, data=x_tra)
  pred <- predict(model, test)
}

```

```

#Devolvemos el acierto
length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

#'
## ----echo=F-----
results <- c()
results.training <- c()
results <- c(results, mean(sapply(1:10, run_LDA_fold, "appendicitis", "test")))
results.training <- c(results.training, mean(sapply(1:10, run_LDA_fold, "appendicitis", "train")))

#'
#' ####Ampliaciones
#' A continuación probaremos a crear modificaciones sobre nuestro modelo base (el anterior con todos los
#' \begin{itemize}
#'   \item Descarte de variables correladas entre sí, igual que hicimos en k-nn, descartando 2 variables.
#'   \item Descarte de V6 por estar correlada con V2.
#'   \item Normalización z-score sobre las variables con el fin de igualar las varianzas. Este tipo de
#'   \item Híbrido entre ampliación 1 y ampliación 2.
#'   \item Híbrido entre las 3 ampliaciones.
#' \end{itemize}
#' 
## ----echo=F-----
run_LDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <- "Y"
  names(x_tst)[In+1] <- "Y"
  x_tra <- x_tra[,c(-3,-7)]
  x_tst <- x_tst[,c(-3,-7)]
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  model <- lda(Y~, data=x_tra)
  pred <- predict(model, test)
  #Devolvemos el acierto
  length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results<- c(results,mean(sapply(1:10, run_LDA_fold, "appendicitis", "test")))
results.training <- c(results.training, mean(sapply(1:10, run_LDA_fold, "appendicitis", "train")))

#'
## ----echo=F-----

```

```

run_LDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <-"Y"
  names(x_tst)[In+1] <-"Y"
  x_tra <- x_tra[,c(-6)]
  x_tst <- x_tst[,c(-6)]
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  model <- lda(Y~, data=x_tra)
  pred <- predict(model, test)
  #Devolvemos el acierto
  length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results <- c(results,mean(sapply(1:10, run_LDA_fold, "appendicitis", "test")))
results.training <- c(results.training, mean(sapply(1:10, run_LDA_fold, "appendicitis", "train")))

#'
## ----echo=F-----
#Normalizando con z-score
run_LDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  #Junto todos los datos para normalizarlos juntos (test y training)
  aux <- rbind(x_tra,x_tst)
  aux <- apply(aux[,1:In], 2, scale, center=TRUE, scale=TRUE)
  x_tst <- cbind(aux[(nrow(x_tra)+1):nrow(aux),],x_tst[,ncol(x_tst)])
  x_tra <- cbind(aux[1:nrow(x_tra),], x_tra[,ncol(x_tra)])
  x_tst <- as.data.frame(x_tst)
  x_tra <- as.data.frame(x_tra)
  names(x_tra)[In+1] <-"Y"
  names(x_tst)[In+1] <-"Y"
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
}

```

```

model <- lda(Y~, data=x_tra)
pred <- predict(model, test)
#Devolvemos el acierto
length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results <- c(results,mean(sapply(1:10, run_LDA_fold, "appendicitis", "test")))
results.training <- c(results.training, mean(sapply(1:10, run_LDA_fold, "appendicitis", "train")))

#'
## ----echo=F-----
run_LDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <-"Y"
  names(x_tst)[In+1] <-"Y"
  x_tra <- x_tra[,c(-6,-3,-7)]
  x_tst <- x_tst[,c(-6, -3, -7)]
  if (tt=="train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  model <- lda(Y~, data=x_tra)
  pred <- predict(model, test)
  #Devolvemos el acierto
  length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results <- c(results,mean(sapply(1:10, run_LDA_fold, "appendicitis", "test")))
results.training <- c(results.training, mean(sapply(1:10, run_LDA_fold, "appendicitis", "train")))

#'
## ----echo=F-----
#Normalizando con z-score
run_LDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  #Junto todos los datos para normalizarlos juntos (test y training)
  aux <- rbind(x_tra,x_tst)
  aux <- apply(aux[,1:In], 2, scale, center=TRUE, scale=TRUE)
  x_tst <- cbind(aux[(nrow(x_tra)+1):nrow(aux),],x_tst[,ncol(x_tst)])
  x_tra <- cbind(aux[1:nrow(x_tra),], x_tra[,ncol(x_tra)])
}

```

```

x_tst <- as.data.frame(x_tst)
x_tra <- as.data.frame(x_tra)
names(x_tra)[In+1] <- "Y"
names(x_tst)[In+1] <- "Y"
x_tra <- x_tra[,c(-6,-3,-7)]
x_tst <- x_tst[,c(-6, -3, -7)]
if (tt=="train") {
  test <-x_tra
}
else {
  test <-x_tst
}
x_tra$Y <- as.factor(x_tra$Y)
test$Y <- as.factor(test$Y)
model <- lda(Y~., data=x_tra)
pred <- predict(model, test)
#Devolvemos el acierto
length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results <- c(results,mean(sapply(1:10, run_LDA_fold, "appendicitis", "test")))
results.training <- c(results.training, mean(sapply(1:10, run_LDA_fold, "appendicitis", "train")))
results <- data.frame(results.training, results)
row.names(results) <- c("Modelo base", "Ampliación 1", "Ampliación 2", "Ampliación 3", "Ampliación 1+2", "Accuracy Train CV", "Accuracy Test CV")
names(results) <- c("Accuracy Train CV", "Accuracy Test CV")
knitr::kable(results)

#'
#' Se muestra en la tabla como las 2 primeras ampliaciones introducen cierta mejora al modelo base, es
#'
#' ##QDA
#' En este apartado emplearemos un modelo QDA, sobre el que, como en modelos anteriores, hemos hecho al
#'
#' ####Modelo base
#' Basándonos en las funciones de validación cruzada ya implementadas para los algoritmos anteriores, v
## ----warning=F, message=F-----
require(klaR)
run_QDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <- "Y"
  names(x_tst)[In+1] <- "Y"
  if (tt=="train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
}

```

```

test$Y <- as.factor(test$Y)
model <- qda(Y~, data=x_tra)
pred <- predict(model, test)
#Devolvemos el acierto
length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

#'
## ----echo=F-----
results <- c()
results.training <- c()
results <- c(results,mean(sapply(1:10, run_QDA_fold, "appendicitis", "test")))
results.training <- c(results.training,mean(sapply(1:10, run_QDA_fold, "appendicitis", "train")))

#'
#' ####Ampliaciones
#' Las ampliaciones que se realizarán para QDA serán similares a los algoritmos previos. Suprimiremos v
#' \begin{itemize}
#'   \item Supresión en la predicción de V3 y V7.
#'   \item Supresión en la predicción de V6.
#'   \item Híbrido entre las 2 anteriores.
#' \end{itemize}
#' \end{itemize}
## ----echo=F-----

run_QDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <-"Y"
  names(x_tst)[In+1] <-"Y"
  x_tra <- x_tra[,c(-3,-7)]
  x_tst <- x_tst[,c(-3,-7)]
  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  model <- qda(Y~, data=x_tra)
  pred <- predict(model, test)
  #Devolvemos el acierto
  length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results <- c(results,mean(sapply(1:10, run_QDA_fold, "appendicitis", "test")))
results.training <- c(results.training,mean(sapply(1:10, run_QDA_fold, "appendicitis", "train")))

#'
## ----echo = F-----
run_QDA_fold<-function(i, x, tt= "test") {

```

```

file <-paste(x, "-10-", i, "tra.dat", sep="")
x_tra<-read.csv(file, comment.char="@", header=F)
file <-paste(x, "-10-", i, "tst.dat", sep="")
x_tst<-read.csv(file, comment.char="@", header=F)
In <-length(names(x_tra)) -1
names(x_tra)[In+1] <-"Y"
names(x_tst)[In+1] <-"Y"
x_tra <- x_tra[,c(-6)]
x_tst <- x_tst[,c(-6)]
if (tt=="train") {
  test <-x_tra
}
else {
  test <-x_tst
}
x_tra$Y <- as.factor(x_tra$Y)
test$Y <- as.factor(test$Y)
model <- qda(Y~, data=x_tra)
pred <- predict(model, test)
#Devolvemos el acierto
length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results <- c(results,mean(sapply(1:10, run_QDA_fold, "appendicitis", "test")))
results.training <- c(results.training,mean(sapply(1:10, run_QDA_fold, "appendicitis", "train")))

#'
## ----echo=F-----
run_QDA_fold<-function(i, x, tt= "test") {
  file <-paste(x, "-10-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@", header=F)
  file <-paste(x, "-10-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@", header=F)
  In <-length(names(x_tra)) -1
  names(x_tra)[In+1] <-"Y"
  names(x_tst)[In+1] <-"Y"
  x_tra <- x_tra[,c(-3,-7, -6)]
  x_tst <- x_tst[,c(-3,-7, -6)]
  if (tt=="train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  x_tra$Y <- as.factor(x_tra$Y)
  test$Y <- as.factor(test$Y)
  model <- qda(Y~, data=x_tra)
  pred <- predict(model, test)
  #Devolvemos el acierto
  length(pred[pred$class==test[,ncol(test)]])/length(pred$class)
}

results <- c(results,mean(sapply(1:10, run_QDA_fold, "appendicitis", "test")))

```

```

results.training <- c(results.training, mean(sapply(1:10, run_QDA_fold, "appendicitis", "train")))
results <- data.frame(results.training, results)
row.names(results) <- c("Modelo base", "Ampliación 1", "Ampliación 2", "Ampliación 1+2")
names(results) <- c("Accuracy Train CV", "Accuracy Test CV")
knitr::kable(results)

#'
#' Vemos que con QDA obtenemos resultados muy pobres en comparación con K-nn y LDA. Puede deberse a que
#'
## ##Comparación algoritmos
## En este apartado se realizarán los tests estadísticos con el fin de comparar entre sí los distintos
## ----echo=F-----
clasif_train_alumnos <- read.csv("./clasif_train_alumnos.csv")
clasif_train_alumnos$out_train_knn[[1]] <- 0.8826316
clasif_train_alumnos$out_train_lda[[1]] <- 0.8815461
clasif_train_alumnos$out_train_qda[[1]] <- 0.8690241
clasif_test_alumnos <- read.csv("./clasif_test_alumnos.csv")
clasif_test_alumnos$out_test_knn[[1]] <- 0.8790909
clasif_test_alumnos$out_test_lda[[1]] <- 0.8690909
clasif_test_alumnos$out_test_qda[[1]] <- 0.8109091

knitr::kable(clasif_train_alumnos)
knitr::kable(clasif_test_alumnos)

#'
#' Primero, para la comparativa, realizaremos comparativas por pares. Para ello usaremos el test de Wilcoxon
#'
## En el siguiente chunk de código hago todos los tests
## -----
LDAvsKNNtrain <- wilcox.test(clasif_train_alumnos$out_train_knn,
                               clasif_train_alumnos$out_train_lda, alternative="two.sided", paired=T)
LDAvsKNNtest <- wilcox.test(clasif_test_alumnos$out_test_knn,
                             clasif_test_alumnos$out_test_lda, alternative="two.sided", paired=T)

QDAvsKNNtrain <- wilcox.test(clasif_train_alumnos$out_train_knn,
                               clasif_train_alumnos$out_train_qda, alternative="two.sided", paired=T)
QDAvsKNNtest <- wilcox.test(clasif_test_alumnos$out_test_knn, clasif_test_alumnos$out_test_qda, alternative="two.sided", paired=T)

QDAvsLDAtrain <- wilcox.test(clasif_train_alumnos$out_train_lda,
                               clasif_train_alumnos$out_train_qda, alternative="two.sided", paired=T)
QDAvsLDAtest <- wilcox.test(clasif_test_alumnos$out_test_lda, clasif_test_alumnos$out_test_qda, alternative="two.sided", paired=T)

#'
#'
## Como se muestra en la siguiente tabla, no hay ningún par de algoritmos que según el test muestre un
## ----echo = F-----
Test.tests <- c(LDAvsKNNtest$p.value, QDAvsKNNtest$p.value, QDAvsLDAtest$p.value)
Train.tests <- c(LDAvsKNNtrain$p.value, QDAvsKNNtrain$p.value, QDAvsLDAtrain$p.value)
tabla <- data.frame(Train.tests, Test.tests)
names(tabla) <- c("p-Value Train", "p-Value Test")
row.names(tabla) <- c("LDA vs KNN", "QDA vs KNN", "QDA vs LDA")
knitr::kable(tabla)

```

```

#'
#' A continuación realizaré comparación múltiple mediante post Holm test.
#'
## -----
clasif_train_alumnos <- clasif_train_alumnos[,-1]
clasif_test_alumnos <- clasif_test_alumnos[,-1]
tam <- dim(clasif_train_alumnos)
groups <- rep(1:tim[2],each=tam[1])
pairwise.wilcox.test(x=as.matrix(clasif_train_alumnos),groups,p.adjust="holm",paired=T)

tam <- dim(clasif_test_alumnos)
groups <- rep(1:tim[2],each=tam[1])
pairwise.wilcox.test(x=as.matrix(clasif_test_alumnos),groups,p.adjust="holm",paired=T)

#'
#' Como muestran los resultados, en ningún 1vs1 de las matrices hay un p-value muy bajo, por lo que pod
#'

```

## Regresión

```

#' #Regresión
#' En esta sección trabajaremos sobre el dataset "abalone", ya descrito previamente, realizando distintos tipos de regresión.
#'
## ##R1. Regresión simple
#' En este apartado probaremos regresión simple, es decir, empleando solo una variable predictora. Nos centraremos en la variable "Rings".
#'
## ----echo=F, warning=F, message=F-----
#require(dummies)
#abalone.dummies <- dummy.data.frame(abalone, sep=".")

#' Como se muestra en el corrplot, las 5 variables más correladas con la variable de salida Rings son Shell_weight, Diameter, Height, Length y Whole_weight.
#'
## ----results=F, echo=F-----
radjusted <- c()
model.shell.weight <- lm(Rings~Shell_weight,data=abalone)
radjusted <- c(radjusted,summary(model.shell.weight)$adj.r.squared)
model.diameter <- lm (Rings~Diameter, data=abalone)
radjusted <- c(radjusted, summary(model.diameter)$adj.r.squared)
model.height <- lm (Rings~Diameter, data=abalone)
radjusted <- c(radjusted, summary(model.height)$adj.r.squared)
model.length <- lm(Rings~Length, data=abalone)
radjusted <- c(radjusted, summary(model.length)$adj.r.squared)
model.whole.weight <- lm(Rings~Whole_weight, data=abalone)
radjusted <- c(radjusted, summary(model.whole.weight)$adj.r.squared)
tabla <- data.frame(radjusted)
row.names(tabla) <- c("Shell_weight", "Diameter", "Height", "Length", "Whole_weight")
knitr::kable(tabla)

#'
#' Se ve como los $R^2$ ajustados son bastante malos, siendo el menos malo el de Shell_weight, que coincide con el resultado del corrplot.
#'

```

```

#' A continuación la descripción de la función para la validación cruzada.
## -----
run_lm_fold<-function(i,formula,x, tt= "test") {
  file <-paste(x, "-5-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@")
  file <-paste(x, "-5-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@")
  names(x_tst) <- c("Sex", "Length", "Diameter", "Height", "Whole_weight",
    "Shucked_weight","Viscera_weight","Shell_weight","Rings")
  names(x_tra) <- c("Sex", "Length", "Diameter", "Height", "Whole_weight",
    "Shucked_weight","Viscera_weight","Shell_weight","Rings")

  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  fitMulti=lm(formula,x_tra)
  yprime=predict(fitMulti,test)
  sqrt(sum(abs(test$Rings-yprime)^2)/length(yprime)) ##RMSE
}

#'
#'
## ----echo=F-----
MSE.Test <- c()
MSE.Training <- c()

MSE.Training <-c(MSE.Training,mean(sapply(1:5,run_lm_fold, Rings~Shell_weight,"abalone","train")))
MSE.Test <- c(MSE.Test,mean(sapply(1:5,run_lm_fold, Rings~Shell_weight,"abalone","test")))
MSE.Training <-c(MSE.Training,mean(sapply(1:5,run_lm_fold, Rings~Diameter,"abalone","train")))
MSE.Test <- c(MSE.Test,mean(sapply(1:5,run_lm_fold, Rings~Diameter,"abalone","test")))
MSE.Training <-c(MSE.Training,mean(sapply(1:5,run_lm_fold, Rings~Height,"abalone","train")))
MSE.Test <- c(MSE.Test,mean(sapply(1:5,run_lm_fold, Rings~Height,"abalone","test")))
MSE.Training <-c(MSE.Training,mean(sapply(1:5,run_lm_fold, Rings~Length,"abalone","train")))
MSE.Test <- c(MSE.Test,mean(sapply(1:5,run_lm_fold, Rings~Length,"abalone","test")))
MSE.Training <-c(MSE.Training,mean(sapply(1:5,run_lm_fold, Rings~Whole_weight,"abalone","train")))
MSE.Test <- c(MSE.Test,mean(sapply(1:5,run_lm_fold, Rings~Whole_weight,"abalone","test")))

tabla <- data.frame(MSE.Training, MSE.Test)
row.names(tabla) <- c("Shell_weight", "Diameter", "Height", "Length", "Whole_weights")
names(tabla) <- c("CV Training RMSE", "CV Test RMSE")
knitr::kable(tabla)

#'
#' Como se muestra en la tabla anterior, Shell_weight, como su  $R^2$  ajustado nos hacía pensar, es el m...
#'
#' ##R2. Regresión múltiple
#' En este apartado realizaremos distintos modelos de regresión múltiple, además de buscar distintas in...
#'
#' Además, antes de realizar el modelo, voy a tener en cuenta la variable Sex, pero para ello la haré v...
#'

```

```

## -----
#Hago la variable dummie. 1 será immature y 0 male o female
Sex.inmature <- ifelse(abalone$Sex==1 | abalone$Sex==2, 0, 1)
abalone.dummy <- cbind(Sex.inmature, abalone[,2:ncol(abalone)])
model.base <- lm(Rings~., data=abalone.dummy)
summary(model.base)

#' En el summary de nuestro modelo, vemos que Length tiene un p-value muy alto, por lo que será la primera variable a quitar
## -----
## -----
model.base.nolength <- lm(Rings~.-Length, abalone.dummy)
summary(model.base.nolength)

#' En este caso obtenemos un model con un R squared de 0.54 también, un poco más alto mirando las milésimas
## -----
model.base.noheight <- lm(Rings~.-Length-Height, abalone.dummy)
summary(model.base.noheight)

#
#' Vemos cómo quitando la variable Height obtenemos un R squared menor, por lo que consideraremos que no es importante
#
#' Ahora probaremos interacciones y no linealidad para nuestro modelo. Para hacerlo partiremos del modelo anterior
#
#' \begin{itemize}
#'   \item Pruebo Height de forma cuadrática debido al rápido crecimiento de Rings con respecto a esta variable.
#'   \item Pruebo la interacción de Height con otras variables. En este caso con Shell\_weight. Mejoramos el R squared.
#'   \item Pruebo añadir logaritmos al modelo debido a la forma de las gráficas de los weights con respecto a Height.
#'   \item Veo que la interacción de Height con Shell\_weight tiene un p-value muy alto por lo que la eliminamos.
#'   \item Pruebo a añadir más logaritmos para todas las variables 'weight'. Seguimos mejorando el R squared.
#'   \item Probamos a añadir log(Diameter) que también tiene su gráfica una forma algo logarítmica. Mejoramos el R squared.
#'   \item Eliminamos predictores con p-value alto, que en este caso era log(Viscera\_weight). Volvemos a calcular el R squared.
#'   \item Ahora vamos a buscar interacción de Sex.inmature con otras variables ya que es importante
#'   \item Pruebo a interaccionar Sex.inmature con Diameter. El R squared se mantiene igual pero la nube de puntos es más compacta.
#'   \item Elimino la última interacción y pruebo a quitar algunos log de weights. Empeoramos el R squared.
#'   \item Pruebo otras interacciones. En este caso pruebo Diameter con Whole\_weight. Empeoramos el R squared.
#'   \item Probamos interacción entre Shell\_weight y Whole\_weight y mejoramos el R squared.
#'   \item Añadimos en la interacción previa log(Whole\_weight) y mejoramos el R squared.
#'   \item La última prueba, a partir del anterior modelo, quitamos log(Shell\_weight) que tiene un p-value muy bajo.
#' \end{itemize}
#
#' En los siguientes trozos de código no muestro los summary en el pdf para evitar llenar el pdf.
## -----results=F-----
#Mejoramos el primer modelo múltiple añadiendo Height^2 ya que en su gráfica se ve un crecimiento bastante rápido.
#Podríamos también probar con una exponencial pero finalmente decidimos cuadrática.
model.quadratic.height <- lm(Rings~.+I(Height^2)-Length, abalone.dummy)

#Probamos interacciones. Mejoramos el R-Squared
model.quadratic.height.2 <- lm(Rings~.+I(Height^2*Shell_weight)
                                +I(Height^2)+I(Height*Shell_weight)-Length, abalone.dummy)

#Otras funciones no lineales fijandonos en las formas de las gráficas
model.with.log <- lm(Rings~.+I(Height^2*Shell_weight)+I(Height^2)+I(Height*Shell_weight))

```

```

+log(Shell_weight)-Length, abalone.dummy)

#Probando más interacciones. Eliminamos elementos con p-value alto
model.with.log.2 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+log(Shell_weight)
-Length, abalone.dummy)

#Más logaritmos. Mejoramos R^2
model.with.log.3 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Viscera_weight))
-Length, abalone.dummy)

#Probamos logaritmos sobre Diameter también, que la forma de su gráfica es parecida.
#Mejoramos un poco. Probemos a eliminar
#algunos logs que tienen p-value alto como el de viscera_weight
model.with.log.4 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Viscera_weight))
+I(log(Diameter))
-Length, abalone.dummy)

#Volvemos a mejorar eliminando viscera_weight
model.with.log.5 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
-Length, abalone.dummy)

#Probamos a interactuar con Sex.inmature con variables como las de los pesos.
#Mejoramos y llegamos a 0.5608
model.Sex.1 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
I(Sex.inmature*Shell_weight)-Length, abalone.dummy)

#Probemos a probar otras interacciones más con Sex. En este caso empeoramos
#y como se ve en el p.value de este producto es alto.
model.Sex.2 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
+I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)
-Length, abalone.dummy)

#Elimino algunos pesos. Pero empeoramos
model.Sex.3 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length, abalone.dummy)

#Veamos otras interacciones. Diameter con whole_weight, conseguimos menos
#R squared que con model.sex.1
model.Sex.4 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
+I(Sex.inmature*Shell_weight)
-Length+I(Diameter*Whole_weight), abalone.dummy)

#Otras Interacciones. Ahora pruebo interacción entre los pesos Shell y Whole.
#Mejoramos el R^2.
model.Sex.5 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))+
```

```

I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*Whole_weight), abalone.dummy)

#Probamos no linealidad en las últimas interacciones.
#Mejoramos añadiendo logaritmo a Whole_weight
model.Sex.6 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), abalone.dummy)

#Quitamos log(shell_weight) que tiene un p-value alto. Obtenemos el mismo
#Rsquared pero hace algo más sencillo el modelo por lo que mejor cogemos este.
model.Sex.7 <- lm(Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Whole_weight))
+I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), abalone.dummy)

#'
#' Aquí la descripción de la función para la validación cruzada:
#'
## -----
#Función para la validación cruzada.
run_lm_fold<-function(i,formula,x, tt= "test") {
  file <-paste(x, "-5-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@")
  file <-paste(x, "-5-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@")
  names(x_tst) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
    "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
  names(x_tra) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
    "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
  x_tst$Sex.inmature <- ifelse(x_tst$Sex.inmature == 1 | x_tst$Sex.inmature == 2, 0, 1)
  x_tra$Sex.inmature <- ifelse(x_tra$Sex.inmature == 1 | x_tra$Sex.inmature == 2, 0, 1)

  if (tt=="train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  fitMulti=lm(formula,x_tra)
  yprime=predict(fitMulti,test)
  sqrt(sum(abs(test$Rings-yprime)^2)/length(yprime)) ##RMSE
}

#'
## ----echo=F-----
#Se realizan las pruebas de los modelos (CV) para compararlos
Rsquared <- c()
Training.RMSE <- c()
Test.RMSE <- c()
Rsquared <- c(Rsquared, summary(model.base)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.,"abalone","train")))

```

```

Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.base.nolength)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.-Length, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.base.noheight)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.-Length-Height, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.quadratic.height)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)-Length, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.quadratic.height.2)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2*Shell_weight)
+I(Height^2)+I(Height*Shell_weight)-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2*Shell_weight)
+I(Height^2)+I(Height*Shell_weight)-Length, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.with.log)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2*Shell_weight)+I(Height^2*log(Shell_weight)-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2*Shell_weight)+I(Height^2)+I(Height^2*log(Shell_weight)-Length, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.with.log.2)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)
-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+log(Shell_weight)-Length, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.with.log.3)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Viscera_weight))
-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Whole_weight))
+I(log(Shucked_weight))+I(log(Viscera_weight))
-Length, "abalone", "test")))

Rsquared <- c(Rsquared, summary(model.with.log.4)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Viscera_weight))
+I(log(Diameter))
-Length, "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(Whole_weight))
+I(log(Shucked_weight))+I(log(Viscera_weight))
+I(log(Diameter))
-Length, "abalone", "test"))

Rsquared <- c(Rsquared, summary(model.with.log.5)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter)))

```

```

-Length,"abalone","train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
-Length,"abalone","test")))

Rsquared <- c(Rsquared, summary(model.Sex.1)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)-
I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
I(Sex.inmature*Shell_weight)-Length,"abalone","train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
I(Sex.inmature*Shell_weight)-Length,"abalone","test")))

Rsquared <- c(Rsquared, summary(model.Sex.2)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)-
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
+I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)
-Length,"abalone","train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
+I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)
-Length,"abalone","test")))

Rsquared <- c(Rsquared, summary(model.Sex.3)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)-
+I(log(Whole_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length,"abalone","train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
+I(log(Whole_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length,"abalone","test")))

Rsquared <- c(Rsquared, summary(model.Sex.4)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)-
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
+I(Sex.inmature*Shell_weight)
-Length+I(Diameter*Whole_weight),"abalone","train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
+I(Sex.inmature*Shell_weight)
-Length+I(Diameter*Whole_weight),"abalone","test")))

Rsquared <- c(Rsquared, summary(model.Sex.5)$adj.r.squared)
Training.RMSE <-c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)-
I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*Whole_weight),"abalone","train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))+
I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*Whole_weight),"abalone","test")))

Rsquared <- c(Rsquared, summary(model.Sex.6)$adj.r.squared)

```

```

Training.RMSE <- c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), "abalone", "test"))))

Rsquared <- c(Rsquared, summary(model.Sex.7)$adj.r.squared)
Training.RMSE <- c(Training.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)
+I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), "abalone", "train")))
Test.RMSE <- c(Test.RMSE,mean(sapply(1:5,run_lm_fold, Rings~.+I(Height^2)+I(Height*Shell_weight)+I(log(
+I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), "abalone", "test"))))

tabla <- data.frame(Rsquared, Training.RMSE, Test.RMSE)
row.names(tabla) <- c("model.base", "model.base.nolenght", "model.base.noheight" , "model.quadratic.height")
knitr::kable(tabla)

#'
#' Se muestra en la tabla anterior para cada modelo propuesto los resultados del $R^2$ adjusted (con el
#'
#' En cuanto al $R^2$ se ve que todos los modelos no lineales mejoran a los lineales, lo que indica que
#'
#' Es posible que con otro tipo de interacciones y de no linealidades nuestro modelo fuese mejor, pero
#'
#' ##KNN
#' En este apartado probaremos KNN en problema de regresión. Knn es un algoritmo basado en similitud, e
## ----message=F, warning=F-----
require(kknn)

run_kknn_fold<-function(i,formula, k=7, x, tt= "test") {
  file <-paste(x, "-5-", i, "tra.dat", sep="")
  x_tra<-read.csv(file, comment.char="@")
  file <-paste(x, "-5-", i, "tst.dat", sep="")
  x_tst<-read.csv(file, comment.char="@")

  names(x_tst) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
  names(x_tra) <- c("Sex.inmature", "Length", "Diameter", "Height", "Whole_weight",
                     "Shucked_weight", "Viscera_weight", "Shell_weight", "Rings")
  x_tst$Sex.inmature <- ifelse(x_tst$Sex.inmature == 1 | x_tst$Sex.inmature == 2, 0, 1)
  x_tra$Sex.inmature <- ifelse(x_tra$Sex.inmature == 1 | x_tra$Sex.inmature == 2, 0, 1)

  if (tt== "train") {
    test <-x_tra
  }
  else {
    test <-x_tst
  }
  fitMulti=kknn(formula,x_tra,test,k=k)
}

```

```

yprime <- fitMulti$fitted.values
sqrt(sum(abs(test$Rings-yprime)^2)/length(yprime)) ##RMSE
}

#Modelo general con todos los predictores
mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "test"))
mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "train"))

#
#' En KNN también podemos probar interacciones entre variables. Además también podemos realizar el mismo
#
## -----
Training.RMSE <- c(mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "train")))
Test.RMSE <- c(mean(sapply(1:5,run_kknn_fold,Rings~., 7, "abalone", "test")))

#Eliminando Length del modelo
Test.RMSE <- c(Test.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.-Length, 7, "abalone",
                                         "test")))
Training.RMSE <- c(Training.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.-Length, 7,
                                              "abalone", "train")))

#Probando mejor modelo anteriormente calculado
Test.RMSE <- c(Test.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)+
                                    I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
+I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)
-Length, 7, "abalone", "test")))

Training.RMSE <- c(Training.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)
+I(Height*Shell_weight)+I(log(Shell_weight))
+I(log(Whole_weight))+I(log(Shucked_weight))+I(log(Diameter))
+I(Sex.inmature*Shell_weight)+I(Sex.inmature*Diameter)
-Length, 7, "abalone", "train")))

#Probando el modelo con menor R^2 anteriormente calculado
Test.RMSE <- c(Test.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)
+I(Height*Shell_weight)+I(log(Whole_weight))
+I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), 7, "abalone", "test")))

Training.RMSE <- c(Training.RMSE, mean(sapply(1:5,run_kknn_fold,Rings~.+I(Height^2)
+I(Height*Shell_weight)+I(log(Whole_weight))
+I(log(Shucked_weight))+I(log(Diameter))+I(Sex.inmature*Shell_weight)
-Length+I(Shell_weight*log(Whole_weight)), 7, "abalone", "train")))

tabla <- data.frame(Test.RMSE, Training.RMSE)
row.names(tabla) <- c("Modelo base", "Sin Length", "Sex.inmature.2", "Sex.inmature.7")
knitr::kable(tabla)

#
#' En los resultados se muestra cómo el modelo base es peor que el modelo base con regresión lineal (mo
#

```

```

#' ##Comparación algoritmos
#' En este apartado realizaremos mediante tests estadísticos una comparación entre los distintos algoritmos
## ----echo=F-----
regr_train_alumnos <- read.csv("./regr_train_alumnos.csv")
regr_test_alumnos <- read.csv("./regr_test_alumnos.csv")
regr_train_alumnos <- regr_train_alumnos[,-1]
regr_test_alumnos <- regr_test_alumnos[,-1]

knitr::kable(regr_train_alumnos)
knitr::kable(regr_test_alumnos)

#'
#' Primero hago una comparación de 1 vs 1 mediante un test de Wilcoxon. Al contrario que en clasificación
#'
## -----
regr_test_alumnos$out_test_lm[[1]] <- 2.215680^2
regr_train_alumnos$out_train_lm[[1]] <- 2.189745^2
regr_test_alumnos$out_test_kknn[[1]] <- 2.324244^2
regr_train_alumnos$out_train_kknn[[1]] <- 1.478211^2

##lm (other) vs knn(ref)
# + 0.1 porque wilcox.R falla para valores == 0 en la tabla
difs<-(regr_train_alumnos$out_train_lm - regr_train_alumnos$out_train_kknn) / regr_train_alumnos$out_train_kknn
wilc_1_2 <- cbind(ifelse(difs<0, abs(difs)+0.1, 0+0.1), ifelse(difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) <- c(colnames(regr_train_alumnos)[1], colnames(regr_train_alumnos)[2])

LMvsKNNtra<-wilcox.test( wilc_1_2[,1], wilc_1_2[,2],alternative = "two.sided", paired=TRUE)

difs<-(regr_test_alumnos$out_test_lm - regr_test_alumnos$out_test_kknn) / regr_test_alumnos$out_test_kknn
wilc_1_2 <- cbind(ifelse(difs<0, abs(difs)+0.1, 0+0.1), ifelse(difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) <- c(colnames(regr_test_alumnos)[1], colnames(regr_test_alumnos)[2])

LMvsKNNtst<-wilcox.test(wilc_1_2[,1], wilc_1_2[,2], alternative = "two.sided", paired=TRUE)

#'
#' Como se muestra en la siguiente tabla, en los resultados en Training hay una alta probabilidad de que
#'
## ----echo = F-----
Training.p.value <- c(LMvsKNNtra$p.value)
Test.p.value <- c(LMvsKNNtst$p.value)
tabla <- data.frame(Training.p.value, Test.p.value)
row.names(tabla) <- c("LM vs KNN")
knitr::kable(tabla)

#'
#' Ahora realizamos un test de Friedman para la comparación múltiple:
#'
## -----
test_friedman.train<-friedman.test(as.matrix(regr_train_alumnos))
test_friedman.test<-friedman.test(as.matrix(regr_test_alumnos))

#'
## ----echo=F-----

```

```

p.value.Training <- c(test_friedman.train$p.value)
p.value.Test <- c(test_friedman.test$p.value)
tabla <- data.frame(p.value.Training, p.value.Test)
knitr::kable(tabla)

#
#' Vemos efectivamente cómo en Train obtenemos un p-value muy bajo, por lo que hay una muy alta probabi
#
#' Con el test post-Holm vemos una comparación múltiple 1vs1.
## -----
tam <-dim(regr_train_alumnos)
groups <-rep(1:tim[2], each=tam[1])
pairwise.wilcox.test(as.matrix(regr_train_alumnos), groups, p.adjust= "holm", paired = TRUE)

tam <-dim(regr_test_alumnos)
groups <-rep(1:tim[2], each=tam[1])
pairwise.wilcox.test(as.matrix(regr_test_alumnos), groups, p.adjust= "holm", paired = TRUE)

#
#' Como vemos, para train tenemos un p-value muy bajo para todas las combinaciones, lo que indica que t

```