Práctica 2: Limpieza y validación de datos

Raúl Sánchez

7/12/2018

Contents

1 Descripción del data set y objetivos

1.1 Objetivo de análisis

A partir de datos disponibles sobre nutrientes que componen los alimentos y su calificación nutricional se plantea una tarea de aprendizaje supervisado de clasificación, se trata de responder a partir de los nutrientes si un alimento es saludable o no según su valor nutricional etiquetándolos de la **A** a la **E**. La pregunta a responder será por tanto, ¿A que categoría pertenece este alimento? Para ello se utiliza un algoritmo de clasificación basado en redes neuronales. Existe en la página openfoodfacts.org una fórmula que es la qué se usa para calificar los alimentos, la importacia de este problema planteado viene de la posibilidad de crear un modelo de clasificación en el cual ante la falta de datos en determinados nutrientes se pueda igualmente clasificar los alimentos de manera precisa.

1.2 Descripción de los datos

Disponemos del conjunto de datos obtenido en la práctica 1 de la asignatura, el conjunto consiste en los datos obtenidos de la página web openfoodfacts.org, la web contiene una gran información relativa a productos alimenticios que se distribuyen alrededor del mundo. Entre los datos de la web se encuentran los datos de composición de los diferentes nutrientes que forman un producto alimenticio, una imagen del producto, un código de barras, fecha inclusión en la base de datos, marca, donde se fabrica, una imagen de la composición detallada en el proucto, su categoría, a parte de muchos otros y finalmente una calificación nutricional del mismo en función de sus características que varía de A a E. La web se nutre de las aportaciones que hacen sus usuarios en los diferentes países del mundo. Es por ello que se observa que muchos campos de diferentes productos se encuentran vacíos o con imágenes que no han sido cargadas o incluso productos que contienen únicamente el nombre comercial de este.

Nuestro conjunto consta de un archivo de texto .csv, nombre de archivo **es_food_nutrients.csv** que contiene los valores separados por coma, almacenados en formato tabla, las variables representadas en las columnas y las filas identifican las observaciones. En concreto contiene los datos relativos a 8215 observaciones, productos alimenticios, y 16 variables, características del producto. Y el conjunto de imágenes de imágenes de los productos almacenados en formato jpg en carpeta adjunta es_images.

Variables archivo **es_food_nutrients.csv**:

- **Name**: Nombre del producto.
- **Energy**: Valor energético del alimento, contiene dos valores en diferentes medidas, kilocalorias (kcal) y kilojulios (kj).
- **Fat**: Contenido en gramos de grasas.
- **Saturated fat**: Contenido en gramos de grasas saturadas.
- **Carbohidrate**: Contenido en gramos de carbohidratos.
- **Sugars**: Contenido en gramos de azúcares.
- **Fiber**: Contenido en gramos de fibra.
- **Proteins**: Contenido en gramos de proteinas.
- **Salt**: Contenido en gramos de sales.
- **Sodium**: Contenido en gramos de sodio.
- **Alcohol**: Contenido en % vol. de alcohol.
- **Zinc**: Contenido en miligramos de zinc
- **Magnesium**: Contenido en miligramos de magnesio.
- **Omega3**: Contenido en gramos de grasas Omega 3.
- **Score**: Clasificación nutricional del producto.
- **Img**: Nombre de la imagen descargada relacionada con el producto.

Los valores de las diferentes variables muestra el contenido por cada 100 gr./ml. de producto.

2 Integración y selección de datos a analizar

Se cargan los datos del archivo **es_food_nutrients.csv** mediante el comando de R y se almacena en el objeto **data**.

```
data <- read.csv("es_food_nutrients.csv", na.strings=c("Na"))

head(data,3)
```

```
##                                                                    Name
## 1                        Coca Cola original taste \342\200\224 330 ml
## 2 Passata di pomodoro Rustica \342\200\224 Italiamo \342\200\224 700 g
## 3                 Nesquik \342\200\224 Nestl\303\251 \342\200\224 1 kg
##            Energy   Fat Saturated_fat Carbohydrate Sugars Fiber Proteins
## 1 176 kj(42 kcal)   0 g           0 g       10,6 g 10,6 g   0 g      0 g
## 2 176 kj(42 kcal) 0,2 g           0 g        6,6 g    3 g  <NA>    2,1 g
## 3 293 kj(70 kcal) 1,8 g           1 g        9,8 g  9,5 g 0,4 g    3,6 g
##     Salt  Sodium Alcohol    Zinc Magnesium Omega3 Score
## 1    0 g     0 g    <NA>    <NA>      <NA>   <NA>     E
## 2 0,04 g 0,016 g    <NA>    <NA>      <NA>   <NA>     A
## 3 0,14 g 0,055 g    <NA> 0,86 mg      <NA>   <NA>  <NA>
##                                    Img
## 1 _544_900_000_0996_front_es.211.200.jpg
## 2           _20164034_front_fr.9.200.jpg
## 3  _303_371_006_5967_front_fr.80.200.jpg
```

```
str(data)
```

```
## 'data.frame':    8215 obs. of  16 variables:
##  $ Name         : Factor w/ 8022 levels "  Pr\303\251sident \342\200\224 Rico en em calcio \342\200\2
##  $ Energy       : Factor w/ 2030 levels "0 kj(0 kcal)",..: 626 626 1317 1128 892 65 101 1742 1665 169
##  $ Fat          : Factor w/ 685 levels "0 g","0,001 g",..: 1 16 81 58 46 459 141 522 660 656 ...
##  $ Saturated_fat: Factor w/ 454 levels "0 g","0,001 g",..: 1 1 94 24 39 58 242 58 100 354 ...
##  $ Carbohydrate : Factor w/ 915 levels "0 g","0,001 g",..: 58 613 892 79 893 317 230 223 399 616 ...
##  $ Sugars       : Factor w/ 648 levels "0 g","0,001 g",..: 103 298 625 465 565 288 572 461 385 469 .
##  $ Fiber        : Factor w/ 304 levels "0 g","0,002 g",..: 1 NA 21 23 46 152 NA NA NA NA ...
##  $ Proteins     : Factor w/ 498 levels "0 g","0,001 g",..: 1 218 338 16 69 90 87 366 436 428 ...
##  $ Salt         : Factor w/ 563 levels "0 g","0,001 g",..: 1 31 79 100 64 387 236 285 269 250 ...
##  $ Sodium       : Factor w/ 488 levels "0 g","0,001 g",..: 1 17 51 68 38 485 185 231 215 198 ...
##  $ Alcohol      : Factor w/ 42 levels "0 % vol","1 % vol",..: NA NA NA NA 1 NA NA NA NA NA ...
##  $ Zinc         : Factor w/ 15 levels "0,04 mg","0,14 mg",..: NA NA 3 NA NA NA NA NA NA NA ...
##  $ Magnesium    : Factor w/ 76 levels "0,41 mg","0,42 mg",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ Omega3       : Factor w/ 25 levels "0 g","0,1 g",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ Score        : Factor w/ 5 levels "A","B","C","D",..: 5 1 NA 2 2 4 3 3 3 3 ...
##  $ Img          : Factor w/ 7489 levels "_00009_front_es.3.200.jpg",..: 1944 390 1265 2288 2057 6693
```

Tras un análisis preliminar de la estructura de los datos importados, como se ha dicho, el conjunto consta de 8215 observaciones y 16 variables.Son todas las variables de tipo categórico, contienen valores junto con la unidad de medidad, ya sea g, mg, % vol, ... que habrá que eliminar. Se observa la existencia de NA's en diferentes variables.

3 Limpieza de los datos

3.1 Conversión de tipos.

Para poder trabajar con los datos se deben convertir a tipos numéricos las variables ya que se han importado todas de tipo categórico, en las siguientes transformaciones se convertirán las variables, excepto **Name**, **Img** y **Score**, a variables de tipo cuantitativas continuas y se eliminarán las unidades de medida g y mg. La variable **Energy** se puede observar que contiene la medida en dos unidades kilojulios y kilocalorías, se prescindirá de la medida kcal debido a que esta medida está cayendo en desuso en ámbitos de la nutrición donde se usa más la medida de kilojulios. Se sustituirá el separador decimal "," por ".". Se eleminarán tambien caracteres "~", ">" y "<" de los valores.

```
# Valor Energy
data$Energy <- gsub(" kj([()0-9 kcal]+)","",data$Energy)

# Elimina caracteres
cols <- c(1, 15, 16)
data[,-cols] <- as.data.frame(apply(data[,-cols], 2, function(y) gsub(" g", "", y)))
data[,-cols] <- as.data.frame(apply(data[,-cols], 2, function(y) gsub(" mg", "", y)))
data[,-cols] <- as.data.frame(apply(data[,-cols], 2, function(y) gsub(" % vol", "", y)))
data[,-cols] <- as.data.frame(apply(data[,-cols], 2, function(y) gsub("~ ", "", y)))
data[,-cols] <- as.data.frame(apply(data[,-cols], 2, function(y) gsub("< ", "", y)))
data[,-cols] <- as.data.frame(apply(data[,-cols], 2, function(y) gsub("> ", "", y)))

data[, -cols] <- as.data.frame(apply(data[, -cols], 2,
                                     function(y) as.numeric(gsub(",", ".", y))))
```

```
## Warning in FUN(newX[, i], ...): NAs introduced by coercion
```

```
summary(data[, -cols])
```

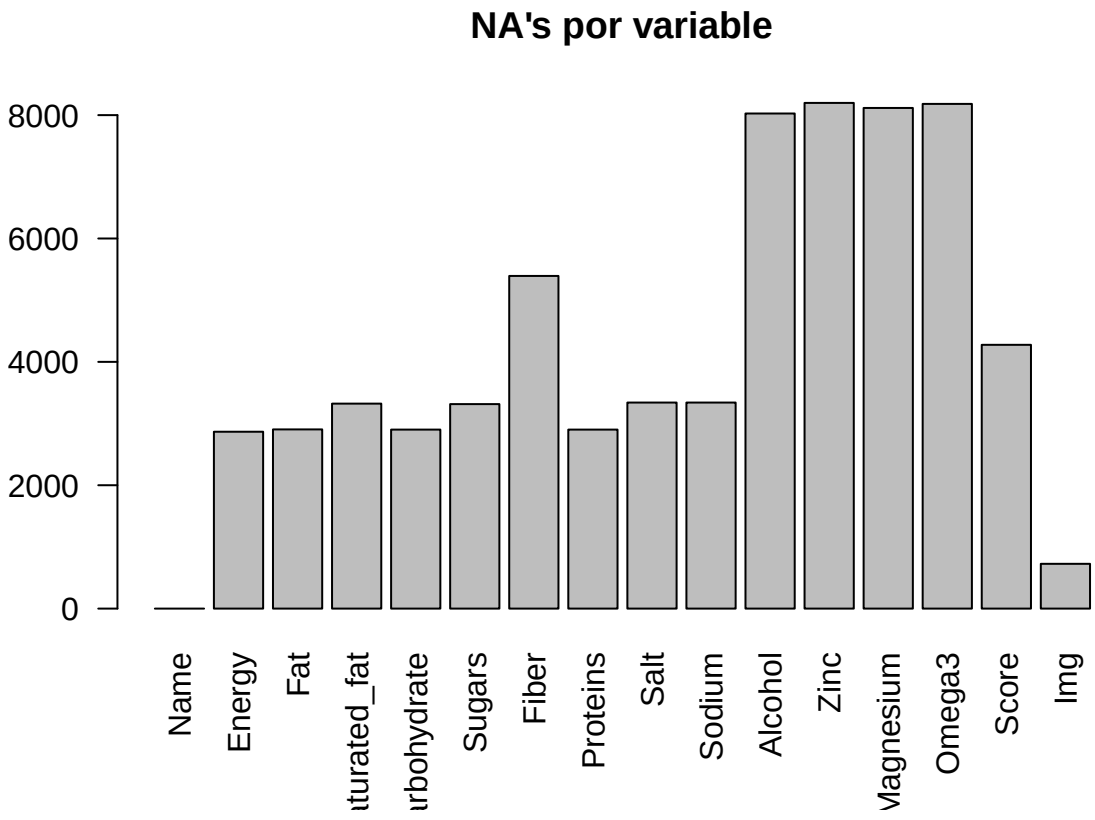
```
##      Energy          Fat          Saturated_fat     Carbohydrate
##  Min.   :   0   Min.   :  0.00   Min.   : 0.000   Min.   :  0.00
##  1st Qu.: 264   1st Qu.:  0.60   1st Qu.: 0.100   1st Qu.:  4.80
##  Median : 858   Median :  3.80   Median : 0.800   Median : 14.00
##  Mean   :1062   Mean   : 12.88   Mean   : 4.049   Mean   : 27.62
##  3rd Qu.:1754   3rd Qu.: 20.00   3rd Qu.: 4.200   3rd Qu.: 51.20
##  Max.   :7900   Max.   :362.00   Max.   :93.900   Max.   :267.00
##  NA's   :2867   NA's   :2905     NA's   :3323     NA's   :2902
##      Sugars           Fiber          Proteins          Salt
##  Min.   :  0.00   Min.   : 0.000   Min.   : 0.000   Min.   :  0.000
##  1st Qu.:  0.90   1st Qu.: 1.000   1st Qu.: 1.200   1st Qu.:  0.040
##  Median :  3.80   Median : 2.600   Median : 4.300   Median :  0.280
##  Mean   : 12.19   Mean   : 4.115   Mean   : 6.213   Mean   :  1.071
##  3rd Qu.: 15.20   3rd Qu.: 5.200   3rd Qu.: 8.200   3rd Qu.:  1.040
##  Max.   :100.00   Max.   :94.800   Max.   :80.000   Max.   :100.000
##  NA's   :3315     NA's   :5393     NA's   :2902     NA's   :3340
##      Sodium          Alcohol            Zinc         Magnesium
##  Min.   : 0.000   Min.   : 0.000   Min.   :0.040   Min.   :   0.41
##  1st Qu.: 0.016   1st Qu.: 0.000   1st Qu.:1.540   1st Qu.:  14.65
##  Median : 0.110   Median : 0.000   Median :3.100   Median : 110.00
##  Mean   : 0.422   Mean   : 4.043   Mean   :3.702   Mean   : 161.60
##  3rd Qu.: 0.410   3rd Qu.: 5.500   3rd Qu.:5.500   3rd Qu.: 232.00
##  Max.   :39.370   Max.   :40.000   Max.   :8.400   Max.   :2094.00
##  NA's   :3340     NA's   :8026     NA's   :8198    NA's   :8116
```

```
##      Omega3
##  Min.   : 0.000
##  1st Qu.: 0.140
##  Median : 0.318
##  Mean   : 4.311
##  3rd Qu.: 2.500
##  Max.   :52.000
##  NA's   :8182
```

3.2 Análisis de elementos vacíos o nulos.

Se analiza a continuación la existencia de vacíos en el conjunto de datos.

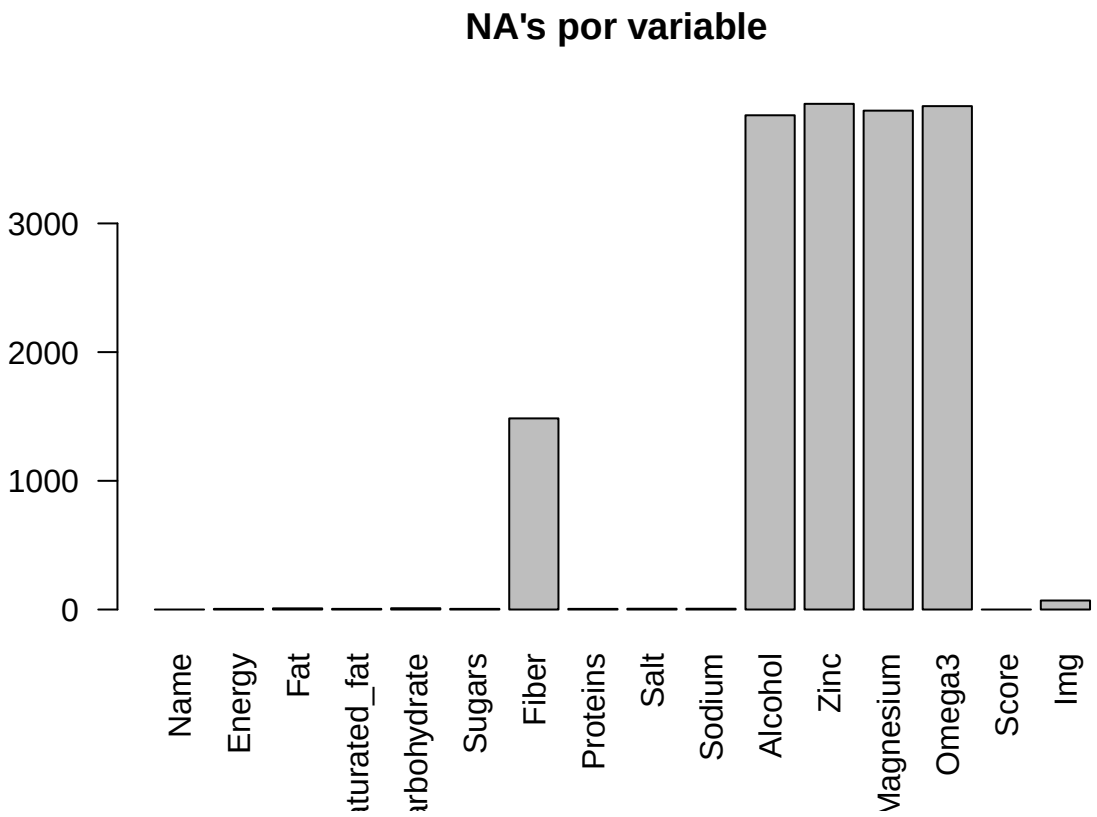
```
# NA's
na.data <- sapply(data, function(y) sum(length(which(is.na(y)))))
# Representación
barplot(na.data, main="NA's por variable",las= 2)
```



Se observa la existencia de numerosos casos de valores desconocidos en todas las variables del conjunto de datos, en primer lugar se detecta que la variable **score**, que se corresponde con la clase a clasificar en nuestro estudio, cuenta con alrededor el 50% de las observaciones sin valor conocido, al no tener utilidad las observaciones que no se pueden clasificar se opta por la eliminación de estas observaciones. Hay cuatro variables **alcohol**, **Zinc**, **Magnesium** y **Omega3** que cuenta con prácticamente el 100% de valores ausentes. A continuación se muestra la existencia de valores vacíos tras eliminar las observaciones que cuentan con NA en la variable **Score**

```
# Elimina observaciones Score = Na
data_clean  <- subset(data, Score != "NA")
```

```r
x_na.data <- sapply(data_clean, function(y) sum(length(which(is.na(y)))))
barplot(x_na.data, main="NA's por variable ",las= 2)
```



Se observa que con la eliminiación de las observaciones con variable **Score = NA** se han eliminado también la mayoría de vacíos en las variables **Energy, Fat, Saturated_fat, Carbohidrate, Sugars Proteins, Salt y Sodium**, lo cual nos facilita la decisión de eliminación de variables con altos casos de presencia de NA. A partir de estos datos se decide eliminar las variables que tiene alrededor del 100% de NA's, **alcohol**, **Zinc**, **Nagnesium** y **Omega3** y con un porcentaje muy elevado también **Fiber**, donde probablemente no daría buenos resultados la imputación a aplicar debido a la elevada ausencia de valores. Se elimina también las variables **Img** y **Name**, que no tienen interés para el posterior estudio.

Se eliminan a continuación las variables indicadas y se muestra el número de observaciones con valores Na por columna:

```r
# Eliminación variables Alcohol, Zinc, Magnesium, Omega3, Fiber, Img y Name
data_clean <- data_clean[, -c(1, 7, 11:14, 16)]

sapply(data_clean, function(y) sum(length(which(is.na(y)))))
```

```
##        Energy           Fat Saturated_fat  Carbohydrate        Sugars
##             5             9             5            10             5
##      Proteins          Salt        Sodium         Score
##             5             6             6             0
```

Existen varios métodos para tratar los casos en los qué existen valores ausentes en los datos, se puede ver que son muy pocos los casos en los qué esto pasa, la variable con mayor número de valores ausuentes es **Carbohydrate** que apenas tiene 10 casos. Se podría eliminar las observaciones ya que representan un porcentaje ínfimo respecto al total que son 3939. Lo ideal sería en este caso imputar los valores por grupos de productos, ya que existen artículos como la sal que contienen grandes cantidades de sodio y otro tipo de productos que no contienen, pero no existe una categorización de los alimentos que se pueda utilizar.

Por ello, se imputará los casos existentes mediante el método kNN, método basado en distancias, que se basa en la identificación de las k observaciones más cercanas al valor a tratar e imputa la media ponderada basada en estas observaciones. A continuación se puede visualizar diferentes medidas de dispersión y tendencia central de las distribuciones de cada variable y se observa, como era de esperar, que las varaciones han sido mínimas.

```
library(VIM)
library(psych)

# Imputación
imp_data_clean <- kNN(data_clean)[1:8]

# Cáculo estmadores
f <- function(x) {
      c(Mean = mean(x, na.rm = TRUE), Median = median(x, na.rm = TRUE),
        SD = sd(x, na.rm = TRUE), RIC = IQR(x, na.rm = TRUE))
}

# Tablas comparación estimadores
est_table<-sapply(data_clean[1:8], f)
knitr::kable(est_table, caption = "Estimadores con datos sin imputar")
```

Table 1: Estimadores con datos sin imputar

	Energy	Fat	Saturated_fat	Carbohydrate	Sugars	Proteins	Salt	Sodium
Mean	1076.8076	13.19637	4.049343	27.77568	12.21384	5.947491	0.8434391	0.3320697
Median	837.0000	4.00000	0.785000	15.00000	3.80000	4.050000	0.2400000	0.0940000
SD	872.3688	18.40992	7.219499	26.31427	17.51131	6.453834	2.7443792	1.0804665
RIC	1572.7500	20.30000	4.000000	45.50000	15.00000	6.700000	0.9860000	0.3880000

```
est_table<-sapply(imp_data_clean, f)
knitr::kable(est_table, caption = "Estimadores con datos imputados")
```

Table 2: Estimadores con datos imputados

	Energy	Fat	Saturated_fat	Carbohydrate	Sugars	Proteins	Salt	Sodium
Mean	1077.2407	13.17815	4.044838	27.83241	12.23274	5.948573	0.843703	0.3317893
Median	838.3000	4.00000	0.760000	15.00000	3.80000	4.100000	0.250000	0.0940000
SD	871.8995	18.39348	7.216022	26.33450	17.50822	6.449807	2.742296	1.0796784
RIC	1571.0000	20.30000	4.000000	45.65000	15.00000	6.700000	0.986000	0.3880000

```
data_clean[1:8] <-imp_data_clean
```

3.3 Identificación y tratamiento de valores extremos.

A continuación se trata de detectar la existencia de valores extremos en las variables, pueden ser varias las causas que los provocan, y por otro lado pueden causar errores en análisis posteriores como correlaciones o en modelos de predicción. Etiquetar como valor extremo o outlier un valor requiere un análisis más profundo, ya que, como se ha comentado antes un producto como la sal tendrá una gran cantidad de sodio y será normal comparado con otros productos similares pero en este caso se tienen en cuenta todos los productos

disponibles sin categorizar, lo cual puede identificar valores extremos ciertas observaciones y llevar a su descarte provocando pérdida de información valiosa.

```
# N valores extremos
out_f <- function(x) {
      length(boxplot.stats(x)$out)
}

nout_table<-sapply(data_clean[1:8], out_f)
knitr::kable(nout_table, caption = "Numero de valores detectados como extreme scores")
```

Table 3: Numero de valores detectados como extreme scores

	x
Energy	2
Fat	132
Saturated_fat	567
Carbohydrate	0
Sugars	431
Proteins	241
Salt	229
Sodium	229

De la tabla 3 se desprende el gran número de valores detectados como outliers, si se optara por la eliminación de las observaciones como método de tratamiento se perdería gran parte de los datos. Este elevado número se debe a las características de los productos existentes, ya comentado anteriormente, como ejemplo se observa en la tabla 3 las variables **Salt** y **Sodium** que tienen el mismo número de outliers (229), lo cual nos indica que estos productos tienen estas características en los nutrientes que los diferencian del resto catalogándolos como outliers, y sean no posibles errores que sucedieran en la toma de datos. Es por esto que se decide mantener las observaciones, también teniendo en cuenta que los datos provienen de una base de datos pública que se actualiza mediante la actividad de los usuarios y es fácil que los posibles errores en medidas se corrijan regularmente. En anexo 1 se encuentra el listado de todos los valores detectados como outliers por variable.

3.4 Almacenamiento de datos procesados

Los datos tratados y limpios se encuentran ya listos para ser utilizados en los análisis de clasificació previstos, se almacenan en el archivo **es__food__nutrients__clean.csv**

```
write.csv(data_clean, "es_food_nutrients_clean.csv")
```

4 Análisis de los datos

4.1 Planificación del análisis a aplicar (Selección de datos).

Se analizará el conjuto total de datos preprocesados, para la llevar a cabo el modelo de clasificación se dividará los datos en dos grupos, entrenamiento y test, se destina el 70% de las observaciones para entrenamiento y el 30% restante para test de la clasificación y así evaluar su precisión. Previamente se codifica la clases **Score** a valores numéricos, “A”=“5”, “B”=“4”, “C”=“3”, “D”=“2”, “E”=“1”, a continuación se normalizan los datos mediante técnica min-max como requiere el modelo, rango de valores entre 0 y 1.

```
library(plyr)
# Codififcación clase Score
data_m <- data_clean
data_m$Score <- as.numeric(revalue(data_m$Score,
                                   c("A"="5", "B"="4","C"="3", "D"="2", "E"="1")))
# Escalado
maxs <- apply(data_m, 2, max)
mins <- apply(data_m, 2, min)
scaled.data_m <- as.data.frame(scale(data_m, center = mins, scale = maxs - mins))

# Preparación conjuntos 70 train-30 test
samplesize = 0.70 * nrow(data_m)
set.seed(40)
indexA = sample( seq_len ( nrow ( data_m)), size = samplesize )
data.train = data_m[ indexA,]
data.test = data_m[ -indexA,]

scaled.data.train = scaled.data_m[ indexA,] #70%
scaled.data.test = scaled.data_m[ -indexA,] #30%
```

4.2 Comprobación de la normalidad y homogeneidad de la varianza.

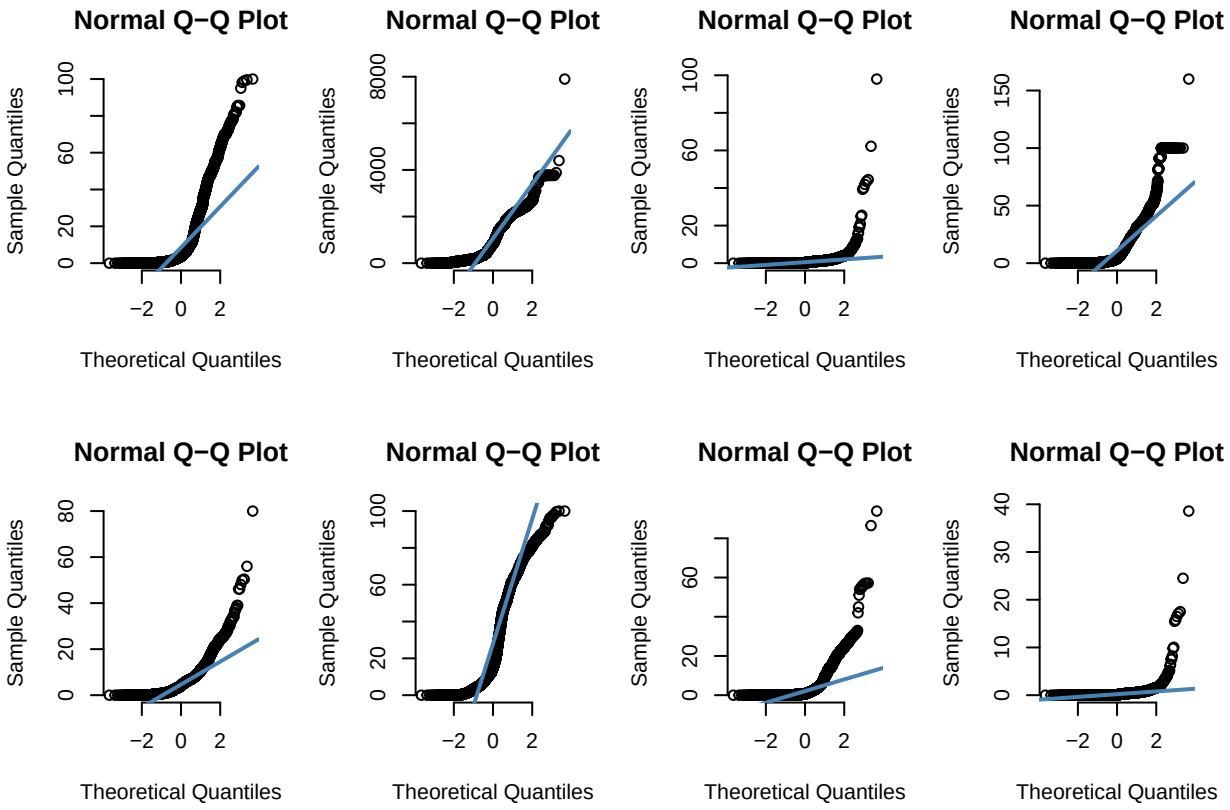
Para comprobar la normalidad de las distribuciones de las diferentes variables se usa el método Shapiro-Wilk test, se muestran a continuación los resultados del p.value obtenidos del test de las variables, en todos los casos es menor al nivel de significación, lo cual indicaría la no normalidad de las distribuciones de las muestras. Aspecto que también se puede observar al analizar los gráficos QQ norm de cada una de las variables.

```
library(ggplot2)

# p.value variables
test_f <- function(x) {
      shapiro.test(x)$p.value
}

t_table<-sapply(data_clean[1:8], test_f)
t_table
```

```
##        Energy           Fat Saturated_fat  Carbohydrate        Sugars
##  1.909263e-44  6.047025e-63  7.483175e-70  3.626867e-50  2.371539e-63
##      Proteins          Salt        Sodium
##  1.093567e-57  3.751707e-84  3.725906e-84
```

```
par(mfrow=c(2,4))

qqnorm(data_clean$Sugars, pch = 1, frame = FALSE)
qqline(data_clean$Sugars, col = "steelblue", lwd = 2)
qqnorm(data_clean$Energy, pch = 1, frame = FALSE)
qqline(data_clean$Energy, col = "steelblue", lwd = 2)
qqnorm(data_clean$Salt, pch = 1, frame = FALSE)
qqline(data_clean$Salt, col = "steelblue", lwd = 2)
qqnorm(data_clean$Fat, pch = 1, frame = FALSE)
qqline(data_clean$Fat, col = "steelblue", lwd = 2)
qqnorm(data_clean$Proteins, pch = 1, frame = FALSE)
qqline(data_clean$Proteins, col = "steelblue", lwd = 2)
```

```
qqnorm(data_clean$Carbohydrate, pch = 1, frame = FALSE)
qqline(data_clean$Carbohydrate, col = "steelblue", lwd = 2)
qqnorm(data_clean$Saturated_fat, pch = 1, frame = FALSE)
qqline(data_clean$Saturated_fat, col = "steelblue", lwd = 2)
qqnorm(data_clean$Sodium, pch = 1, frame = FALSE)
qqline(data_clean$Sodium, col = "steelblue", lwd = 2)
```



Respecto a la varianza se realiza a continuación el test levene, ya que este no rquiere normalidad en los datos.Se evalua en el test la hipotesis que la varianza de las poblaciones analizadas es igual (homogeneidad de varianza), si el p-value es menor a 0.05 entonces se rechaza la hipótesis y se puede afirmar que no son homogeneas.

Se obtiene a continuación el p-value de los test de cada variable del conjunto con respecto a la variable clase Score, a la vista de estos se descarta la hipótesis nula indicando que las varianzas no son iguales en todos los casos:

```
library(car)

lev_f <- function(x) {
      leveneTest(x ~ Score, data = data_clean)$`Pr(>F)`[1]
}
lev_table<-sapply(data_clean[1:8], lev_f)
lev_table
```

```
##        Energy           Fat Saturated_fat  Carbohydrate        Sugars
##  1.908535e-33 3.702857e-149 3.123356e-247  4.451903e-42 1.978465e-177
##      Proteins          Salt        Sodium
##  2.296145e-10  2.080936e-15  1.755560e-15
```

4.3 Pruebas estadísticas

Una vez elegida la topografia y las características de la red neuronal se entrena con el conjunto datos de entrenamiento anteriormente seleccionado.

```
library(neuralnet)

#Formula
feats <- names(data_m[1:8])
f <- paste(feats,collapse=' + ')
f <- paste("Score ~",f)
f <- as.formula(f)

nn <- neuralnet(f,scaled.data.train,hidden=c(4,2),lifesign = "full", threshold = 0.05
                , linear.output=FALSE, stepmax=1e6)
```

```
## hidden: 4, 2    thresh: 0.05    rep: 1/1    steps:    1000   min thresh: 0.419122248
##                                                     2000   min thresh: 0.1821459657
##                                                     3000   min thresh: 0.123312959
##                                                     4000   min thresh: 0.123312959
##                                                     5000   min thresh: 0.123312959
##                                                     6000   min thresh: 0.08557996114
##                                                     7000   min thresh: 0.05241915304
##                                                     7556   error: 33.80571 time: 8.46 secs
```

5 Resultados

Se representan a continuación los resultados obtenidos una vez aplicado el modelo de clasificación de los alimentos en base a sus nutrientes. Se evalúa la precisión de las predicciones mediante la matriz de confusión siguiente:

```
library(NeuralNetTools)

predict_test <- compute(nn,scaled.data.test[1:8])

predict_test_ = predict_test$net.result * (max(data_m$Score) - min(data_m$Score)) +
  min(data_m$Score)
test.r <- (scaled.data.test$Score)*(max(data_m$Score)-min(data_m$Score))+min(data_m$Score)

predict_testnn_<- sapply(predict_test_,round,digits=0)
knitr::kable(table(data.test$Score,predict_testnn_), caption = "Matriz de confusion modelo nn")
```

Table 4: Matriz de confusion modelo nn

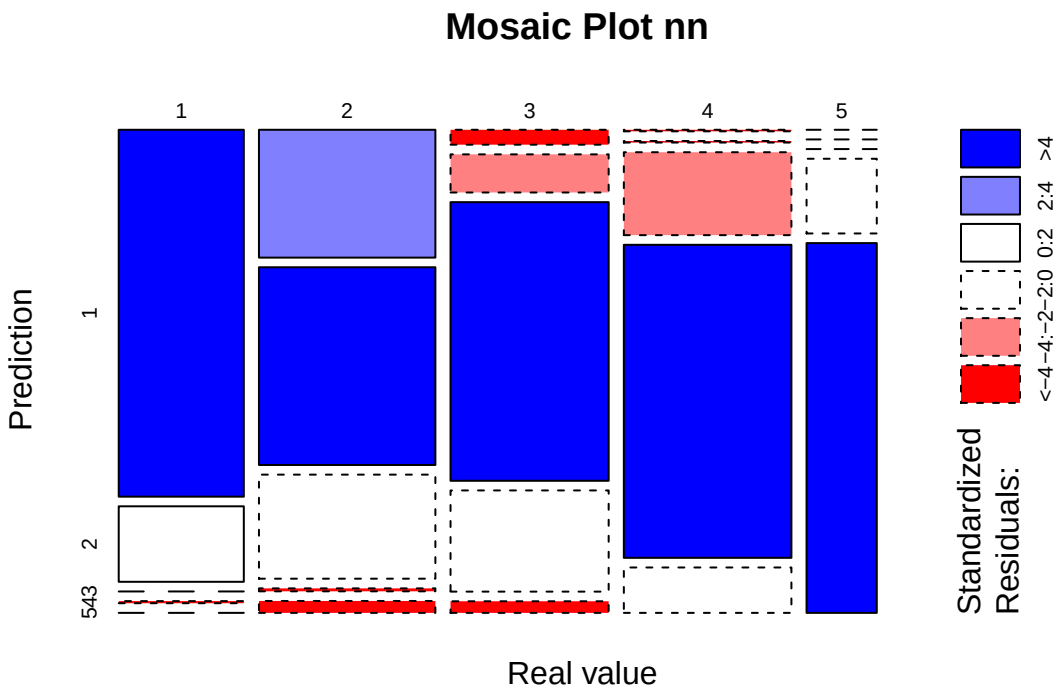
1	2	3	4	5
175	86	9	1	0
36	133	23	1	0
0	70	168	53	0
1	2	61	200	20
0	8	7	29	99

De la matriz de confusión de la predicción se desprenden los siguientes resultados:

- $ERR = \frac{407}{1182} = 0.344$ (missclassification error)
- $ACC = 1 - ERR = 0.656$ (Classification accuracy)

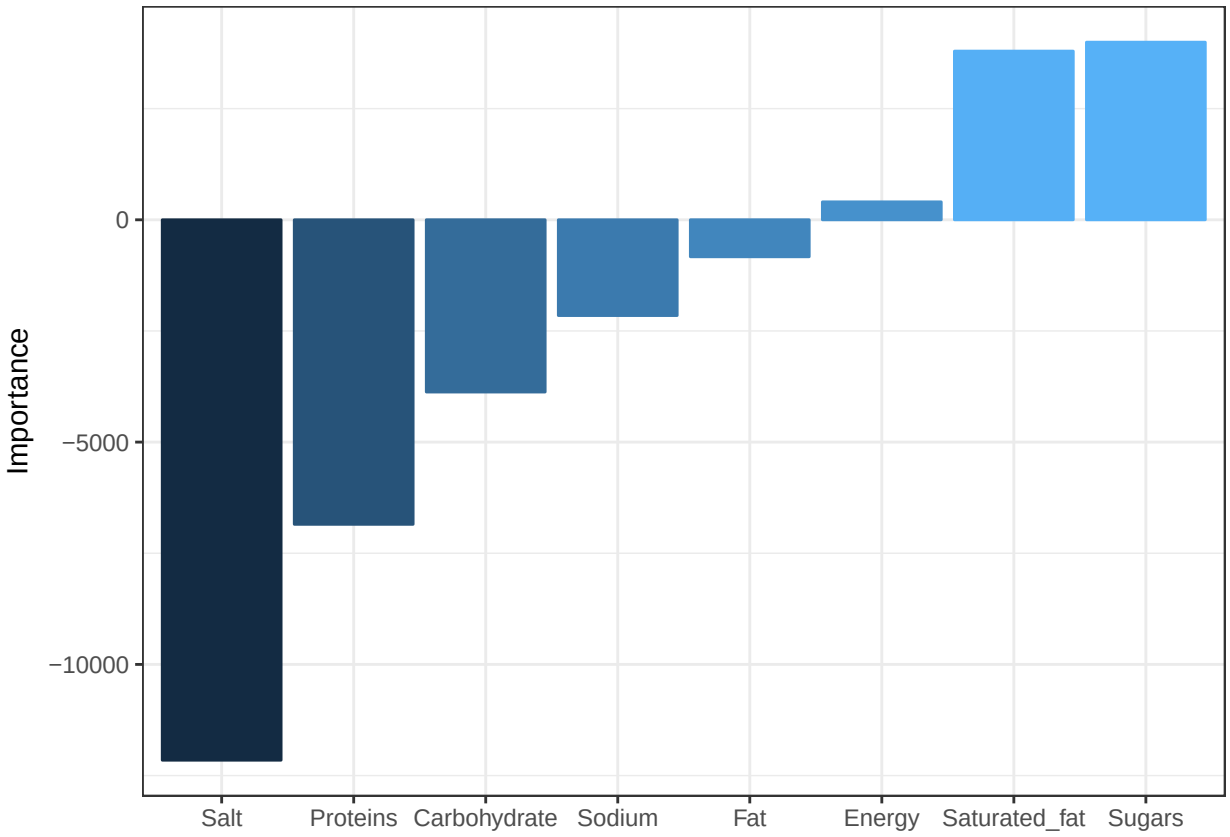
Lo cual indica una precisión del 65.6% en la clasificación.

```
mosaicplot(table(predict_testnn_, data.test$Score), xlab='Real value', ylab='Prediction',
           main='Mosaic Plot nn ', shade = T)
```



En el siguente gráfico se puede observar la importancia de cada una de las variables en el modelo a la hora de clasificar un alimento como saludable o no, es decir que son más influyentes o menos, las más influyentes son las variables **Salt**, **Proteins**, **Saturated_fat** y **Sugars**,y por otro lado las menos influyentes son las variables **Energy** y **Fat**

```
olden(nn)
```



6 Conclusiones

A la vista de los resultados se puede decir que el clasificador no es bueno, con una precisión de clasificación de un 65%, es muy alta la tasa de error que se producce. Se observa que los errores de clasificación en su mayoría se producen en categorías contiguas, siendo 30 del total los errores de clasificación más allá de la categoría contigua, por ejemplo que un producto sea de calificación nutricional A y se clasifique como C, D o E.

Será necesario investigar la mejora del modelo para aumentar la precisión, ya sea incluyendo las variables que se eliminaron en fases anteriores por falta de datos, mejorando el proceso de imputación de valores ausentes (que son muchos) o evaluando otros algoritmos de clasificación que se ajusten mejor a las características de esta tarea.

7 Anéxos:

7.1 Anexo 1: Valores extremos identificados por variable.

```
boxplot.stats(data_clean$Energy)$out
```

```
## [1] 7900 4402
```

```
boxplot.stats(data_clean$Fat)$out
```

```
##   [1] 160.00  52.00  52.30  55.00  70.70  55.00  56.50  63.00  67.40 100.00
##  [11]  52.00  53.70  91.00  65.00  69.10  54.80  63.80  52.00  51.88 100.00
```

```
##  [21]  92.00  72.10  82.00 100.00  64.50 100.00  52.00  91.00 100.00  57.00
##  [31] 100.00  60.00  60.00  51.90  52.40  92.00  53.00  58.00  91.00  91.60
##  [41]  52.50 100.00  99.90  55.00  82.00  61.00  60.00  82.00  81.00  53.00
##  [51]  56.30 100.00  91.00  53.00  59.10  55.00  52.00  82.00  62.50  54.90
##  [61]  91.00  65.00  56.10  60.00  93.00  81.00  92.00  57.00 100.00  92.00
##  [71]  61.80  54.40 100.00  91.50  66.40  91.00  51.50  71.00  60.30  55.20
##  [81]  55.20  60.70  56.50  54.80 100.00 100.00  91.60 100.00 100.00 100.00
##  [91] 100.00  91.50  71.00  91.80  82.00  61.00 100.00  99.88 100.00  54.00
## [101] 100.00  56.00  62.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00
## [111] 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00  82.00
## [121] 100.00  82.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00
## [131] 100.00  60.00
```

```
boxplot.stats(data_clean$Saturated_fat)$out
```

```
##   [1] 23.70 20.00 29.00 23.00 26.70 11.00 26.00 15.00 30.00 19.50 10.60
##  [12] 29.00 10.60 13.20 13.20 17.30 19.00 12.00 29.30 30.70 23.00 15.50
##  [23] 12.00 86.50 20.20 13.00 11.70 24.00 19.20 31.00 14.30 18.00 20.00
##  [34] 23.00 13.00 12.00 24.80 32.20 10.20 13.00 14.00 13.00 13.00 26.80
##  [45] 14.00 21.00 11.00 25.41 21.00 14.00 18.00 24.00 28.00 19.40 19.10
##  [56] 22.00 23.20 24.00 26.00 11.00 32.32 17.30 16.80 14.70 12.00 12.00
##  [67] 15.00 19.00 10.10 24.80 17.00 13.00 29.00 57.00 10.60 23.20 27.00
##  [78] 10.60 25.00 17.00 23.20 18.00 12.00 17.00 12.50 14.00 11.00 57.20
##  [89] 25.00 28.00 32.00 14.00 10.20 31.00 17.00 13.00 11.00 12.00 14.00
## [100] 17.00 15.00 13.00 10.40 11.00 10.20 11.00 27.30 13.40 20.80 24.00
## [111] 21.60 15.00 18.00 19.20 14.00 23.50 11.50 24.00 26.60 13.60 16.70
## [122] 19.00 24.00 16.60 18.00 14.00 23.00 25.00 26.00 27.60 23.00 22.00
## [133] 16.20 18.60 22.70 23.30 18.00 16.00 15.00 11.00 14.00 19.50 15.60
## [144] 20.00 10.40 20.90 14.00 24.00 13.74 30.70 25.20 15.00 15.00 21.00
## [155] 12.60 14.90 23.00 13.00 13.80 17.80 33.00 16.50 11.70 28.30 21.00
## [166] 56.00 14.60 11.00 13.00 45.00 16.00 20.00 15.00 18.00 15.00 23.00
## [177] 19.10 12.00 30.00 25.00 12.00 14.00 12.50 29.00 29.20 30.90 19.50
## [188] 23.00 27.00 23.50 29.80 29.60 30.80 23.00 16.00 14.00 11.00 13.00
## [199] 55.00 21.00 21.00 13.00 55.00 21.00 25.00 21.00 13.70 17.30 12.20
## [210] 15.60 15.00 21.00 21.00 18.00 23.00 16.00 14.00 21.00 14.00 10.50
## [221] 11.00 15.00 16.00 14.00 16.90 18.00 23.20 23.10 11.00 15.00 20.80
## [232] 14.00 15.00 15.00 26.00 15.00 26.00 15.00 21.00 17.00 16.00 32.00
## [243] 13.00 57.00 22.30 26.30 23.00 12.90 23.00 12.00 15.00 12.10 11.90
## [254] 14.00 11.00 19.00 13.40 24.00 22.70 11.00 11.60 26.10 23.40 26.00
## [265] 26.00 16.10 13.00 18.30 22.40 26.00 10.80 10.80 20.20 18.30 17.00
## [276] 17.00 17.00 17.00 22.00 22.00 23.58 15.00 16.50 16.50 21.00 22.50
## [287] 22.60 13.30 24.00 20.30 13.00 26.52 10.80 17.60 24.00 17.00 18.00
## [298] 15.00 18.00 21.00 11.00 13.70 14.60 23.00 16.70 21.00 17.00 12.35
## [309] 18.00 18.00 17.90 25.00 14.60 20.90 20.90 23.70 19.30 19.30 18.80
## [320] 26.10 20.90 24.10 15.00 29.00 13.40 13.00 21.10 22.30 19.00 19.00
## [331] 25.70 23.50 21.10 30.70 54.00 13.00 22.60 13.40 27.00 16.00 13.00
## [342] 11.00 11.00 13.60 11.00 11.00 12.00 12.00 11.00 20.00 19.80 21.00
## [353] 20.00 17.00 20.00 29.30 18.50 16.00 15.90 13.90 14.00 14.30 15.00
## [364] 16.00 13.30 10.70 24.80 13.40 10.30 12.50 16.00 18.00 11.00 11.00
## [375] 11.00 11.00 18.10 13.00 15.00 15.00 12.60 14.00 18.50 13.00 21.00
## [386] 21.70 12.80 27.30 19.00 21.00 21.00 13.40 26.00 19.70 24.00 24.00
## [397] 21.00 18.20 15.00 14.00 27.00 20.50 22.00 27.00 20.00 25.50 18.90
## [408] 26.20 23.00 25.00 25.00 31.70 29.00 20.00 28.00 20.00 27.00 28.00
## [419] 30.00 21.00 20.00 16.80 17.20 18.40 20.60 22.60 22.60 11.00 15.10
## [430] 14.00 11.90 11.40 13.00 15.00 11.00 12.50 12.90 13.40 13.70 11.20
```

```
## [441] 10.20 23.00 17.20 13.00 13.00 13.00 13.00 11.90 14.30 14.00 20.40
## [452] 15.60 11.00 22.00 14.00 15.50 19.90 24.00 14.30 21.40 18.50 11.00
## [463] 17.00 19.00 15.30 11.00 12.00 14.00 13.30 14.00 10.60 23.70 21.00
## [474] 56.00 20.30 11.00 11.80 17.00 16.00 22.00 18.00 42.00 13.00 14.00
## [485] 20.00 12.00 12.00 28.00 21.00 15.00 15.00 11.00 15.40 11.00 12.20
## [496] 93.90 14.00 20.00 12.70 19.10 14.80 18.00 20.30 17.10 30.00 14.20
## [507] 23.00 24.00 16.00 14.00 16.00 16.80 20.00 14.00 22.00 17.00 17.00
## [518] 26.10 19.93 15.00 14.20 15.90 13.16 15.30 18.00 20.00 19.00 21.00
## [529] 14.00 14.00 14.00 19.00 16.00 14.00 14.00 14.00 14.00 14.00 14.00
## [540] 14.00 14.00 14.00 14.00 14.00 14.00 14.00 51.00 14.00 20.00 11.70
## [551] 11.40 54.00 15.00 14.00 14.00 14.00 14.00 14.00 14.00 14.00 14.00
## [562] 14.00 11.40 15.00 11.00 14.00 12.70
```

```
boxplot.stats(data_clean$Carbohydrate)$out
```

```
## numeric(0)
```

```
boxplot.stats(data_clean$Sugars)$out
```

```
##   [1]  56.00  50.60  56.80  52.80  68.00  56.30  67.80  47.00  56.30  81.50
##  [11]  48.30  48.30  41.20  48.00  52.00  50.00  48.00  48.00  45.00  40.00
##  [21]  46.00  47.20  65.20  54.90  46.10  47.00  81.00  46.00  72.10  47.10
##  [31]  43.40  42.70  48.50  47.10  51.60  39.60  55.00  56.00  46.70  70.00
##  [41]  56.30  75.70  78.00  74.10  75.70  51.00  42.50  99.60  53.90  49.50
##  [51]  50.00  62.00  45.00  85.60  41.00  42.00  70.00  51.00  43.70  42.70
##  [61]  51.00  62.70  68.00  57.10  50.00  70.00  63.00  51.10  47.10  44.00
##  [71]  47.30  49.70  61.00  53.60  41.00  46.00  50.06  46.70  47.00  99.00
##  [81]  48.00  46.60  63.00  44.20  39.00  50.00  77.20  57.80  50.00  49.00
##  [91]  72.00  66.00  50.00  52.00  45.00  46.00  48.00  55.00  40.00  50.00
## [101]  69.00  47.10  77.70  82.00  68.50  72.00  55.00  58.00  41.00  50.00
## [111]  59.00  61.00  49.40  70.00  45.00  45.00  47.10  40.90  48.00  52.00
## [121]  40.00  44.10  39.00  70.00  39.30  40.00  70.00  66.00  69.00  69.00
## [131]  58.10  70.00  42.00  70.00  70.00  65.00  65.00  57.00  42.60  39.30
## [141]  57.00  57.00  57.00  50.60  38.70  48.30  45.30  61.80  39.00  44.00
## [151]  55.00  57.06  77.20  59.20  52.00  42.50  78.30  61.30  42.00  42.00
## [161]  47.00  51.00  60.00  56.00  48.00  50.10  51.40  61.00  56.00  61.00
## [171]  68.00  44.00  75.10  80.90  62.80  65.10  38.70  43.00  43.00  54.60
## [181]  46.00  53.00  47.00  47.00  53.60  53.00  57.00  52.00  48.00  62.00
## [191]  49.00  43.00  50.60  50.00  50.00  55.00  62.60  49.00  51.05  75.00
## [201]  82.00  76.00  68.00  40.00  40.00  40.00  41.60  48.00  48.00  66.00
## [211]  64.70  74.00  57.00  49.82  52.40  56.50  49.00  39.70  56.00  55.00
## [221]  55.00  56.00  55.00  60.50  52.60  62.10  56.00  56.00  55.00  55.00
## [231]  52.30  49.00  48.00  59.30  53.00  55.00  40.50  49.00  50.40  47.00
## [241]  70.00  80.00  58.00  46.96  57.00  49.82  73.00  61.40  40.30  55.00
## [251]  48.00  40.00  55.00  66.00  51.70  42.70  47.00  57.50  48.30  48.20
## [261]  40.20  39.90  46.10  48.10  48.10  48.30  52.00  43.90  48.00  42.00
## [271]  47.40  46.90  71.80  61.40  71.30  43.90  39.20  67.00  71.10  72.00
## [281]  65.00  43.00  43.00  42.00  52.00  44.00  46.00  66.00  54.00  63.30
## [291]  62.90  61.00  45.50  66.80  50.40  46.60  61.40  39.40  43.10  67.00
## [301]  40.00  41.00  63.50  63.50  63.50  50.40  58.00  71.00  59.00  48.00
## [311]  50.00  41.00  49.30  44.00  58.00  46.40  39.00  50.40  52.00  50.30
## [321]  42.20  44.00  45.00  45.00  41.00  42.00  52.40  50.00  42.00  51.90
## [331]  55.70  95.00  69.00  42.20  52.10  47.00  53.40  70.00  47.90  47.90
## [341]  40.00  76.60  50.00  50.00  46.80  46.80  38.70  45.70  46.70  57.20
## [351]  46.80  42.80  46.90  40.70  43.20  46.50  47.80  53.00  52.00  55.60
```

```
## [361]  47.50  49.20  55.00  56.80  56.30  50.00  65.10  43.30  53.70  51.00
## [371]  46.00  46.00  47.30  50.00  46.70  40.00 100.00  73.00  62.00  69.00
## [381]  50.00  42.90  85.00  39.00  85.60  41.00  39.80  51.40  52.60  55.00
## [391]  77.20  56.00  63.00  45.80  47.50  40.30  74.10  67.90  71.60  44.20
## [401]  85.60  40.58  50.00  67.10  98.20  45.30  46.70  40.70  58.00  65.00
## [411]  43.00  53.00  38.70  57.00  53.50  42.00  45.30  45.30  45.30  45.30
## [421]  45.30  45.30  45.30  45.30  45.30  45.30  45.30  45.30  71.40  41.10
## [431]  38.70
```

```
boxplot.stats(data_clean$Proteins)$out
```

```
##   [1] 21.00 80.00 26.50 32.00 18.50 27.00 19.00 27.90 24.40 19.00 22.20
##  [12] 22.90 19.70 22.00 21.00 25.40 27.77 34.00 19.00 21.00 19.00 22.03
##  [23] 25.50 21.00 24.89 20.00 19.00 21.50 22.30 24.80 18.20 23.00 20.80
##  [34] 21.00 22.00 24.40 32.00 19.30 25.30 29.00 27.00 48.10 18.40 26.00
##  [45] 20.90 20.30 24.50 20.00 33.00 30.20 25.00 24.50 24.50 25.00 50.00
##  [56] 34.00 39.00 18.30 30.00 28.00 24.91 24.00 50.40 23.90 20.00 25.80
##  [67] 18.60 24.40 22.20 25.40 21.00 20.90 24.60 23.40 30.10 27.00 27.00
##  [78] 24.00 22.50 21.60 20.80 21.00 21.90 18.70 46.10 23.70 26.50 21.70
##  [89] 26.00 25.00 26.00 36.00 23.00 20.00 19.00 28.00 21.90 30.00 18.50
## [100] 21.00 21.00 18.20 32.00 24.90 20.47 24.00 22.35 24.90 23.00 23.00
## [111] 18.10 22.80 22.30 22.00 18.55 26.00 26.00 18.50 19.20 19.70 21.50
## [122] 24.40 19.00 19.30 22.00 20.00 23.30 24.00 21.00 32.20 20.70 37.30
## [133] 20.90 24.00 26.00 19.56 18.55 21.33 24.30 20.30 19.00 19.34 19.90
## [144] 20.00 37.00 21.90 22.00 25.00 21.60 27.00 20.70 24.50 28.40 26.70
## [155] 26.00 23.60 22.00 18.20 20.70 19.30 18.20 19.30 23.00 29.80 33.00
## [166] 20.20 20.90 18.20 20.20 20.50 22.30 27.10 24.30 34.10 20.50 21.40
## [177] 21.00 21.00 20.70 21.90 22.00 21.20 21.20 22.90 24.70 24.40 18.80
## [188] 21.00 21.00 20.00 30.00 30.00 18.20 23.00 56.00 19.00 20.00 22.10
## [199] 22.00 18.40 33.89 22.60 27.00 30.50 18.70 20.00 26.00 21.80 23.90
## [210] 21.80 27.00 25.20 27.50 20.00 33.00 25.00 20.80 22.90 23.00 21.00
## [221] 22.00 25.00 23.00 28.00 25.70 19.30 46.10 22.00 20.90 25.00 39.00
## [232] 24.00 24.20 24.50 24.00 20.88 25.10 29.10 38.00 19.00 24.00
```

```
boxplot.stats(data_clean$Salt)$out
```

```
##   [1] 19.300  2.500  3.370  4.000  2.600  3.100  4.000  2.700  3.200  2.670
##  [11] 40.000  3.000  2.500 10.000  8.204 24.900  4.000  5.600  3.600  4.700
##  [21]  2.900  4.400  3.500 16.400  3.230  5.290 10.250  3.000  5.750  9.906
##  [31]  3.023  3.000  3.200  4.000  4.000 25.500  6.500  2.800 13.000  4.000
##  [41]  5.080  3.100  2.800  3.400  3.500  4.000  2.890  3.500  4.000  2.500
##  [51]  2.500  2.500  8.128  5.500  2.500  4.000  4.000  2.700 15.700  7.400
##  [61]  3.000  3.556  3.000  2.800  2.500  3.050  3.900  4.990  4.000 19.120
##  [71]  2.500 12.400  3.250  2.500  2.500  2.600 41.910  4.300  3.000 43.500
##  [81]  4.800  3.000  2.540  3.500  3.580  6.800  3.000  3.556  6.350  5.334
##  [91]  3.400  3.400  4.000  2.500  2.500  2.800  4.300  3.810  5.200  2.500
## [101]  6.900  3.800  9.600  4.064  9.144  4.572  5.100  3.000  3.050  3.300
## [111]  3.270  3.048  3.500  4.300  3.988  7.260  8.128  6.858 11.176 12.192
## [121]  3.400  3.300  2.500  5.000  2.667  2.667 11.853  3.800 20.828  2.940
## [131]  2.540  2.540  3.300  3.000  2.500  3.500  3.048  2.540  2.540  7.112
## [141]  2.900  4.318  6.400  4.318  3.700  2.500 44.450  2.540  3.810  3.556
## [151] 62.230  2.540  3.302  5.200  5.700  2.591  4.064  4.233  3.556  2.540
## [161]  4.762  2.794  3.302  4.064  4.318  4.064  4.064  3.810  2.963  4.572
## [171]  3.598  2.646  3.048  2.540  2.646  2.794  8.954  2.540  3.581  4.064
## [181]  2.794  3.492  3.810  3.757  2.540  3.302  4.521  8.992  7.620 13.000
```

```
## [191]  4.880  5.100 39.300  2.700  6.400 98.000  5.000  3.500  6.480  3.900
## [201]  3.600  3.100  3.950  2.700  2.900  2.800  9.500  2.500  4.216  2.700
## [211]  6.200  6.300  3.800  3.300  8.000  2.500  3.100  3.100  4.000  2.600
## [221]  3.800  4.699  4.250  4.000  3.000  2.900  9.000  2.769  2.540
```

```
boxplot.stats(data_clean$Sodium)$out
```

```
##   [1]  7.598  0.984  1.327  1.575  1.024  1.220  1.575  1.063  1.260  1.051
##  [11] 15.748  1.181  0.984  3.937  3.230  9.803  1.575  2.205  1.417  1.850
##  [21]  1.142  1.732  1.378  6.457  1.272  2.083  4.035  1.181  2.264  3.900
##  [31]  1.190  1.181  1.260  1.575  1.575 10.039  2.559  1.102  5.118  1.575
##  [41]  2.000  1.220  1.102  1.339  1.378  1.575  1.138  1.378  1.575  0.984
##  [51]  0.984  0.984  3.200  2.165  0.984  1.575  1.575  1.063  6.181  2.913
##  [61]  1.181  1.400  1.181  1.102  0.984  1.201  1.535  1.965  1.575  7.528
##  [71]  0.984  4.882  1.280  0.984  0.984  1.024 16.500  1.693  1.181 17.126
##  [81]  1.890  1.181  1.000  1.378  1.409  2.677  1.181  1.400  2.500  2.100
##  [91]  1.339  1.339  1.575  0.984  0.984  1.102  1.693  1.500  2.047  0.984
## [101]  2.717  1.496  3.780  1.600  3.600  1.800  2.008  1.181  1.201  1.299
## [111]  1.287  1.200  1.378  1.693  1.570  2.858  3.200  2.700  4.400  4.800
## [121]  1.339  1.299  0.984  1.969  1.050  1.050  4.667  1.496  8.200  1.157
## [131]  1.000  1.000  1.299  1.181  0.984  1.378  1.200  1.000  1.000  2.800
## [141]  1.142  1.700  2.520  1.700  1.457  0.984 17.500  1.000  1.500  1.400
## [151] 24.500  1.000  1.300  2.047  2.244  1.020  1.600  1.667  1.400  1.000
## [161]  1.875  1.100  1.300  1.600  1.700  1.600  1.600  1.500  1.167  1.800
## [171]  1.417  1.042  1.200  1.000  1.042  1.100  3.525  1.000  1.410  1.600
## [181]  1.100  1.375  1.500  1.479  1.000  1.300  1.780  3.540  3.000  5.118
## [191]  1.921  2.008 15.472  1.063  2.520 38.583  1.969  1.378  2.551  1.535
## [201]  1.417  1.220  1.555  1.063  1.142  1.102  3.740  0.984  1.660  1.063
## [211]  2.441  2.480  1.496  1.299  3.150  0.984  1.220  1.220  1.575  1.024
## [221]  1.496  1.850  1.673  1.575  1.181  1.142  3.543  1.090  1.000
```