Tipologia y Ciclo de vida de los Datos. Practica 2

Autores: Alexis Germán Arroyo Peña y Gabriel Pulido de Torres

Enero 2021

Contents

1 Descripción del dataset. ¿Por qué es importante y qué pregunta/problema pretende responder?

Como objetivo de esta práctica se ha optado por analizar el conjunto de datos del titanic.

Este conjunto de datos incluye información acerca de los pasajeros que iban en el Titanic, en el que murieron 1500 personas en 1912. Disponemos tanto de información descriptiva del tipo de pasajero como un indicador de su supervivencia al hundimiento o no. La pregunta que nos marcamos como objetivo es: ¿las variables aportadas son suficientes para crear un modelo predictivo que prediga la supervivencia de un determinado pasajero?

El conjunto de datos del Titanic permite entrenar modelos predictivos supervisados. Contiene un número manejable de datos y de atributos de distintos tipos. Además, es un conjunto que no está limpio (por ejemplo, contiene elementos nulos) lo que obligará a poner en práctica técnicas aprendidas de limpieza de datos.

Estructura del dataset

A continuación, realizamos un primer contacto con el conjunto de datos, visualizando su estructura.

```
# Cargamos el fichero de datos
data <- read.csv('train.csv',stringsAsFactors = FALSE, na.strings = "")
dim.data.frame(data)
```

```
## [1] 891  12
```

```
# Verificamos la estructura del conjunto de datos
str(data)
```

```
## 'data.frame':    891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : chr  "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
##  $ Sex        : chr  "male" "female" "female" "female" ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : chr  "A/5 21171" "PC 17599" "STON/O2. 3101282" "113803" ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : chr  NA "C85" NA "C123" ...
##  $ Embarked   : chr  "S" "C" "S" "S" ...
```

El conjunto de datos está formado 12 variables y 891 observaciones.

Descripción del conjunto de datos:

- Passengerld: Contador de pasajeros del 1 al 891.
- Survived: nos indica si el pasajero sobrevivió o no (0= No, 1= Si).
- pClass: nos indica la clase a la que pertenece el ticket, 1=1st (clase alta), 2=2nd (clase media) y 3=3rd (clase baja).
- Name: nombre completo del pasajero.
- Sex: sexo del pasajero (Female o Male).
- Age: edad del pasajero
- SibSp: número de hermanos/hermanas, hermanastros/hermanastros y marido o esposa del pasajero que también iban a bordo.

- Parch: número de hijas, hijos, padre y madre del pasajero a bordo del Titanic.
- Ticket: número del ticket del pasajero.
- Fare: la tarifa del pasajero en dólares.
- Cabin: código identificativo de la cabina.
- Embarked: el puerto en el que embarcó el pasajero (C = Cherbourg, Q =Queenstown, S = Southampton).

2 Integración y selección de los datos de interés a analizar.

El conjunto de datos que se proporciona en Kaggle está dividido en dos partes, un conjunto de datos de train, y un conjunto de datos de test. La diferencia principal es que mientras que en el conjunto de datos de train disponemos del valor de la variable Survived, en el de test no disponemos de esa variable.

Como parte de los modelos que vamos a diseñar son modelos supervisados, a la hora de validarlos no podemos usar el conjunto de datos de test, por lo que hemos optado trabajar únicamente con los de train sin fusionarlos.

En el apartado anterior leímos los datos en un dataframe (al que llamamos "data"). Recordamos su dimensionalidad y comprobamos los nombres y tipos de las variables.

```
## [1] 891  12
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##   "integer"   "integer"   "integer" "character" "character"   "numeric"
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##   "integer"   "integer" "character"   "numeric" "character" "character"
```

Comprobamos si hay registros duplicados usando el comando unique()

```
data_unique <- unique(data)
dim.data.frame(data_unique)
```

```
## [1] 891  12
```

```
remove(data_unique)
```

La dimensionalidad ha quedado exactamente igual que cuando cargamos el conjunto de datos. Esto nos hace ver que no existen registros duplicados.

Revisamos la estructura de nuestro dataset:

```
summary(data)
```

```
##   PassengerId       Survived          Pclass          Name
##  Min.   :  1.0   Min.   :0.0000   Min.   :1.000   Length:891
##  1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000   Class :character
##  Median :446.0   Median :0.0000   Median :3.000   Mode  :character
##  Mean   :446.0   Mean   :0.3838   Mean   :2.309
##  3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000
##  Max.   :891.0   Max.   :1.0000   Max.   :3.000
##
##      Sex                 Age            SibSp           Parch
##  Length:891         Min.   : 0.42   Min.   :0.000   Min.   :0.0000
##  Class :character   1st Qu.:20.12   1st Qu.:0.000   1st Qu.:0.0000
##  Mode  :character   Median :28.00   Median :0.000   Median :0.0000
##                     Mean   :29.70   Mean   :0.523   Mean   :0.3816
##                     3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000
```

```
##                     Max.   :80.00   Max.   :8.000   Max.   :6.0000
##                     NA's   :177
##     Ticket               Fare           Cabin             Embarked
##  Length:891         Min.   :  0.00   Length:891         Length:891
##  Class :character   1st Qu.:  7.91   Class :character   Class :character
##  Mode  :character   Median : 14.45   Mode  :character   Mode  :character
##                     Mean   : 32.20
##                     3rd Qu.: 31.00
##                     Max.   :512.33
##
```

```
df_status(data)
```

```
##       variable q_zeros p_zeros q_na  p_na q_inf p_inf      type unique
## 1  PassengerId       0    0.00    0  0.00     0     0   integer    891
## 2     Survived     549   61.62    0  0.00     0     0   integer      2
## 3       Pclass       0    0.00    0  0.00     0     0   integer      3
## 4         Name       0    0.00    0  0.00     0     0 character    891
## 5          Sex       0    0.00    0  0.00     0     0 character      2
## 6          Age       0    0.00  177 19.87     0     0   numeric     88
## 7        SibSp     608   68.24    0  0.00     0     0   integer      7
## 8        Parch     678   76.09    0  0.00     0     0   integer      7
## 9       Ticket       0    0.00    0  0.00     0     0 character    681
## 10        Fare      15    1.68    0  0.00     0     0   numeric    248
## 11       Cabin       0    0.00  687 77.10     0     0 character    147
## 12    Embarked       0    0.00    2  0.22     0     0 character      3
```

Podemos visualizar una muestra de los primeros registros de nuestro dataframe:

```
head(data, 10)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
1	0	3	Braund, Mr. Owen Harris	male	22	1	0
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0
3	1	3	Heikkinen, Miss. Laina	female	26	0	0
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0
5	0	3	Allen, Mr. William Henry	male	35	0	0
6	0	3	Moran, Mr. James	male	NA	0	0
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0

Existen variables que carecen de interés por ser identificadoras de cada registro. Estamos haciendo referencia a "PassengerId", "Ticket" y "Name". No nos interesa tener unívocamente identificado cada caso para realizar algún tipo de análisis por lo que no aportan nada a nuestro estudio.

2.1 PersonasTicket y Price

Antes de eliminar la variable **Ticket**, nos va a servir para conocer el precio unitario que han pagado los pasajeros, ya que la variable **Fare** es la tarifa pagada en el ticket, pero dentro del mismo ticket pueden incluirse a varias personas. Inicialmente, vamos a obtener el número de personas que están incluidas en un mismo ticket.

```
nrow(table(data$Ticket, data$Fare))
```

```
## [1] 681
```

```
length(unique(data$Ticket))
```

```
## [1] 681
```

```
dt <- data %>% group_by(Ticket, Fare) %>%
  filter(row_number() == 1)

kbl(head(dt, n = 20), booktabs = T) %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NA	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NA	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NA	S
6	0	3	Moran, Mr. James	male	NA	0	0	330877	8.4583	NA	Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.0750	NA	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333	NA	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708	NA	C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1	1	PP 9549	16.7000	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.5500	C103	S
13	0	3	Saundercock, Mr. William Henry	male	20	0	0	A/5. 2151	8.0500	NA	S
14	0	3	Andersson, Mr. Anders Johan	male	39	1	5	347082	31.2750	NA	S
15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14	0	0	350406	7.8542	NA	S
16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55	0	0	248706	16.0000	NA	S
17	0	3	Rice, Master. Eugene	male	2	4	1	382652	29.1250	NA	Q
18	1	2	Williams, Mr. Charles Eugene	male	NA	0	0	244373	13.0000	NA	S
19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vandemoortele)	female	31	1	0	345763	18.0000	NA	S
20	1	3	Masselmani, Mrs. Fatima	female	NA	0	0	2649	7.2250	NA	C

Podemos obtener el precio por persona al dividir Fare entre los integrantes del ticket:

```
ticket_personas <- as.data.frame(data %>%
                                   group_by(Ticket) %>%
                                   dplyr::summarize(PersonasTicket=n()))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

Creamos un nuevo dataframe, "ticket_personas" al que asignamos las variables **Ticket** y **PersonasTicket**

```
df_status(ticket_personas)
```

```
##          variable q_zeros p_zeros q_na p_na q_inf p_inf      type unique
## 1          Ticket       0       0    0    0     0     0 character    681
## 2 PersonasTicket       0       0    0    0     0     0   integer      7
```

Este dataframe lo combinamos con el dataframe original a través del número de ticket para incorporar el número de personas al dataframe original. Tras lo cual podemos eliminar el dataframe ticket_personas.

```
data <- merge(data, ticket_personas, by = "Ticket")
remove(ticket_personas)
```

Hemos incluido a nuestros datos la variable nueva "PersonasTicket", la cual utilizaremos para dividir la tarifa del ticket "Fare" entre "PersonasTicket" para crear la variable "Price". Esta nueva variable representa el precio por persona que se paga en el billete.

```
data$Price <- data$Fare / data$PersonasTicket
```

Tras realizar esto, procedemos a eliminar todas las variables que no utilizaremos: PassengerId, Name, Ticket, PersonasTicket:

```
data <- select(data, -PassengerId, -Name, -Ticket, -PersonasTicket)
```

2.2 FamilySize

Podemos crear una nueva variable, **FamilySize**, que será la suma de **Parch** (número de padres / hijos a bordo) y **SibSp** (sibling: número de hermanos del pasajero, spouse: cónyuge del pasajero), más el propio viajero en cuestión (haciendo esta variable como mínimo igual a 1).

```
data$FamilySize = data$Parch + data$SibSp + 1
```

2.3 Revisión y conversion de tipos

Debemos convertir las variables **Sex**, **Survived**, **Pclass** y **Embarked** a "factor". Estas tienen un número finito de valores por lo que, aunque puedan ser numéricas, las discretizamos convirtiendo sus valores en factores.

```
data$Sex <- as.factor(data$Sex)
data$Survived <- as.factor(data$Survived)
data$Pclass <- as.factor(data$Pclass)
data$Embarked <- as.factor(data$Embarked)
```

Nos quedarían como variables numéricas: **Age**, **SubSp**, **Parch**, **FamilySize**, **Fare** y **Price**

```
summary(data)
```

```
##  Survived Pclass      Sex           Age            SibSp           Parch
##  0:549    1:216   female:314   Min.   : 0.42   Min.   :0.000   Min.   :0.0000
##  1:342    2:184   male  :577   1st Qu.:20.12   1st Qu.:0.000   1st Qu.:0.0000
##           3:491                Median :28.00   Median :0.000   Median :0.0000
##                                Mean   :29.70   Mean   :0.523   Mean   :0.3816
##                                3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000
##                                Max.   :80.00   Max.   :8.000   Max.   :6.0000
##                                NA's   :177
##       Fare           Cabin             Embarked       Price
##  Min.   :  0.00   Length:891         C   :168   Min.   :  0.000
##  1st Qu.:  7.91   Class :character   Q   : 77   1st Qu.:  7.763
##  Median : 14.45   Mode  :character   S   :644   Median :  8.850
##  Mean   : 32.20                      NA's:  2   Mean   : 17.789
##  3rd Qu.: 31.00                                 3rd Qu.: 24.288
##  Max.   :512.33                                 Max.   :221.779
##
##    FamilySize
##  Min.   : 1.000
##  1st Qu.: 1.000
##  Median : 1.000
##  Mean   : 1.905
##  3rd Qu.: 2.000
##  Max.   :11.000
##
```

3 Limpieza de los datos.

Ahora nos toca analizar aquellos valores que se presenten nulos y/o vacíos. Mostramos a continuación un resumen del estado de nuestras 8 variables actuales:

```
df_status(data)
```

```
##       variable q_zeros p_zeros q_na  p_na q_inf p_inf      type unique
## 1    Survived     549   61.62    0  0.00     0     0    factor      2
## 2      Pclass       0    0.00    0  0.00     0     0    factor      3
## 3         Sex       0    0.00    0  0.00     0     0    factor      2
## 4         Age       0    0.00  177 19.87     0     0   numeric     88
## 5       SibSp     608   68.24    0  0.00     0     0   integer      7
## 6       Parch     678   76.09    0  0.00     0     0   integer      7
## 7        Fare      15    1.68    0  0.00     0     0   numeric    248
## 8       Cabin       0    0.00  687 77.10     0     0 character    147
## 9    Embarked       0    0.00    2  0.22     0     0    factor      3
## 10      Price      15    1.68    0  0.00     0     0   numeric    248
## 11 FamilySize       0    0.00    0  0.00     0     0   numeric      9
```

3.1 Elementos nulos y ceros.

3.1.1 Corrección de elementos nulos

Vamos a analizar los elementos nulos que se encuentran en el dataset

En la tabla anterior observamos que el campo **Age** tiene un 19.87% de nulos, **Cabin** más de un 77% y **Embarked** tiene 2 valores nulos. Analizamos cada uno de ellos con más detalles para decidir las acciones a realizar.

3.1.1.1 Cabin

Por su número tan elevado de casos, un 77%, y por no ser significativa para nuestros objetivos, se opta por eliminar esta variable.

```
data <- select(data, -Cabin)
```

3.1.1.2 Embarked

Esta variable presenta 2 valores nulos. Al ser pocos casos, decidimos imputar el valor que más se repite: Embarked=S (Southampton)

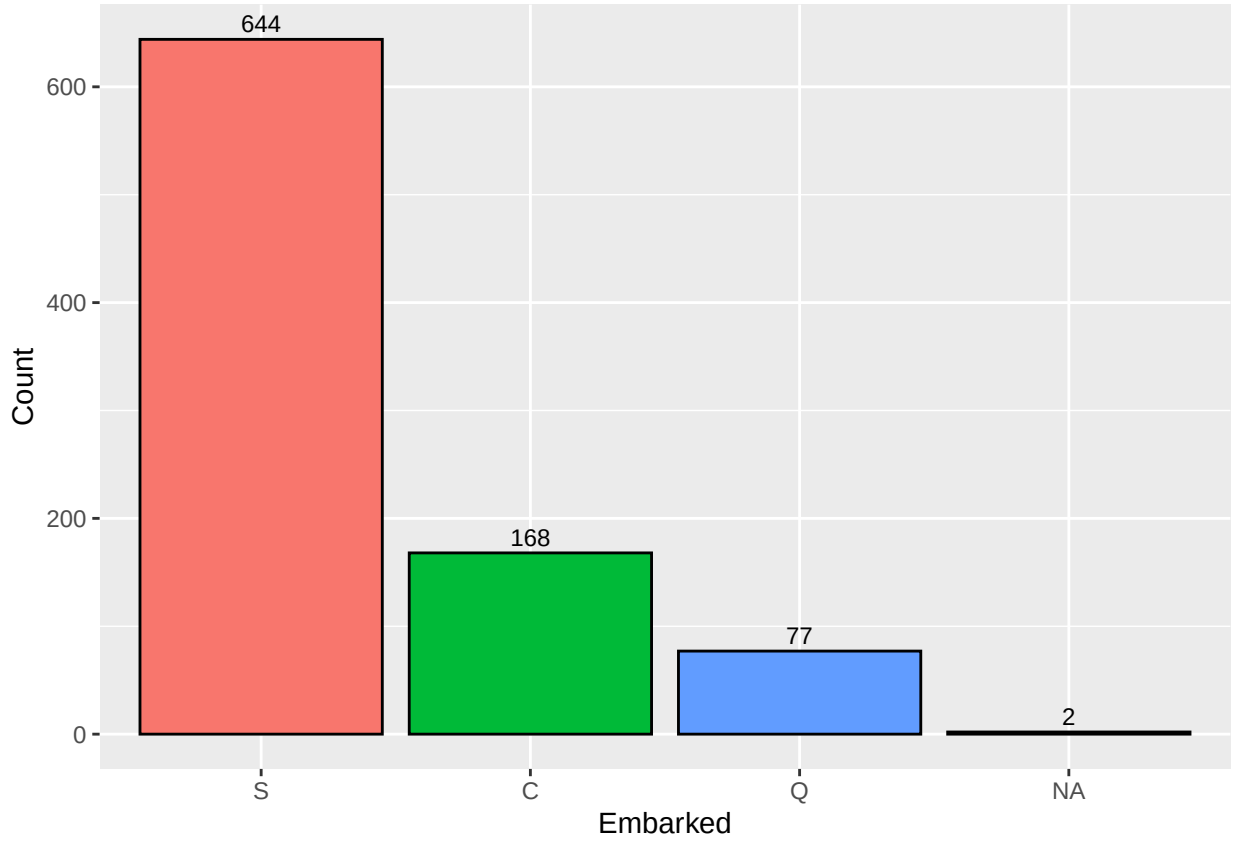
```
data %>% group_by(Embarked) %>% count(Embarked)
```

Embarked	n
C	168
Q	77
S	644
NA	2

```
data_Embarked <- sort(table(data$Embarked, useNA = "ifany"), decreasing = TRUE)
```

Gráficamente:

```
dat_plot <- as.data.frame(data_Embarked)
ggplot(dat_plot, aes(x=Var1, y=Freq, fill=Var1)) +
  geom_bar(stat = "identity", color = "black") +
  geom_text(aes(label=Freq), vjust = -0.4, color="black", size=3) +
  labs(x='Embarked', y='Count') +
  theme(legend.position = "none")
```




```
data$Embarked[is.na(data$Embarked)] <- names(data_Embarked[1])
```

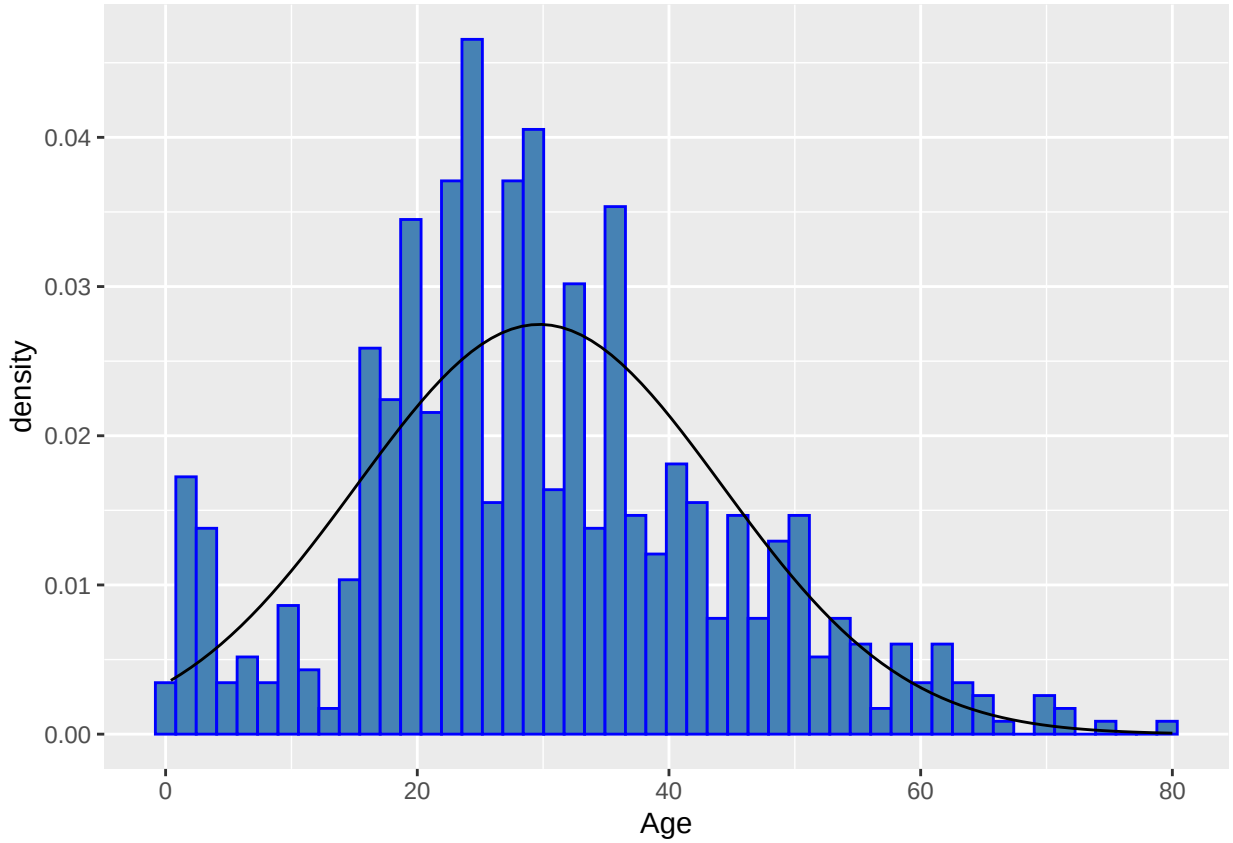
3.1.1.3 Age

Contiene concretamente 177 registros con valores nulos.

Analizamos la distribución de la variable **Age** teniendo en cuenta sólo los registros donde hay valores. Para ello creamos un nuevo dataset sin los registros nulos de Age:

```
data_NoNA <- data[which(!is.na(data$Age)),]
```

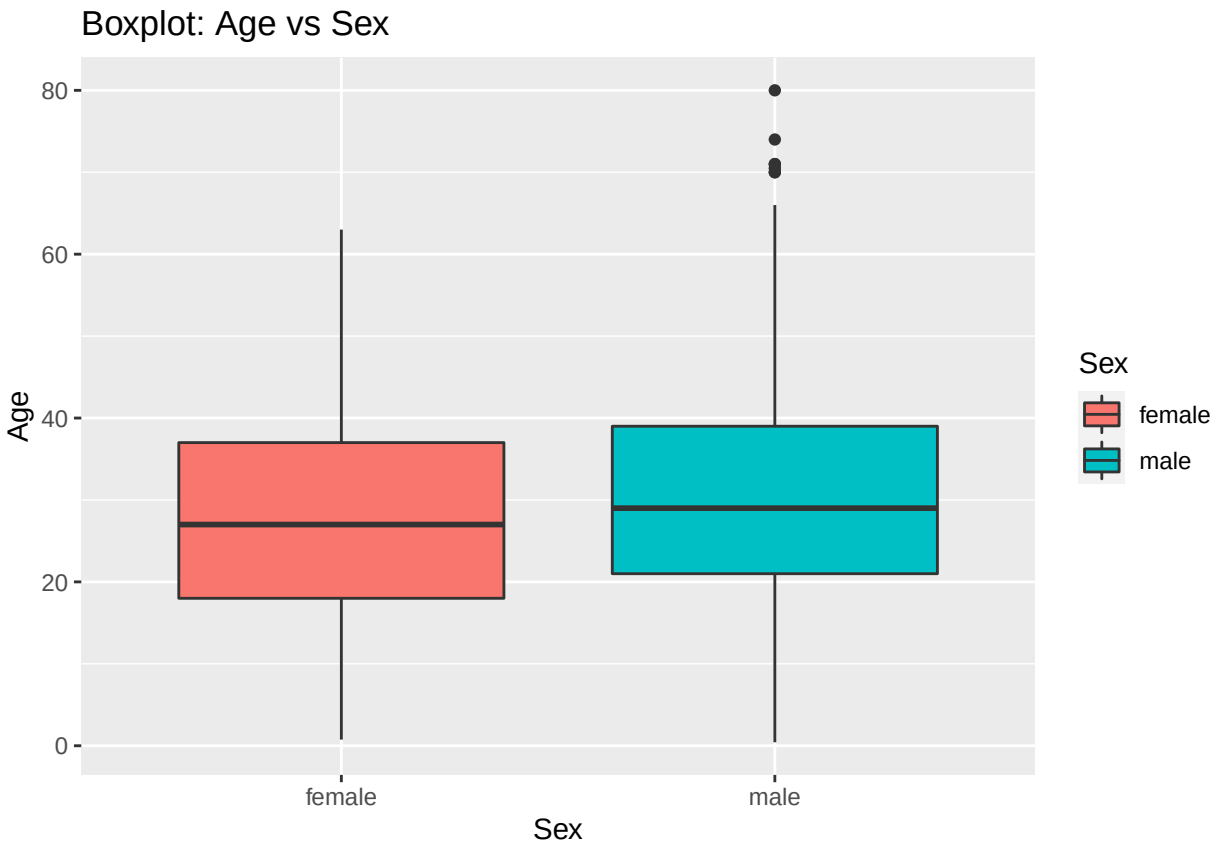
Observamos gráficamente como se distribuye la variable Age en este dataset:



Comprobamos posibles relaciones entre la edad y alguna otra variable. Para ello iremos analizando Age con las distintas variables.

3.1.1.3.1 Age vs Sex

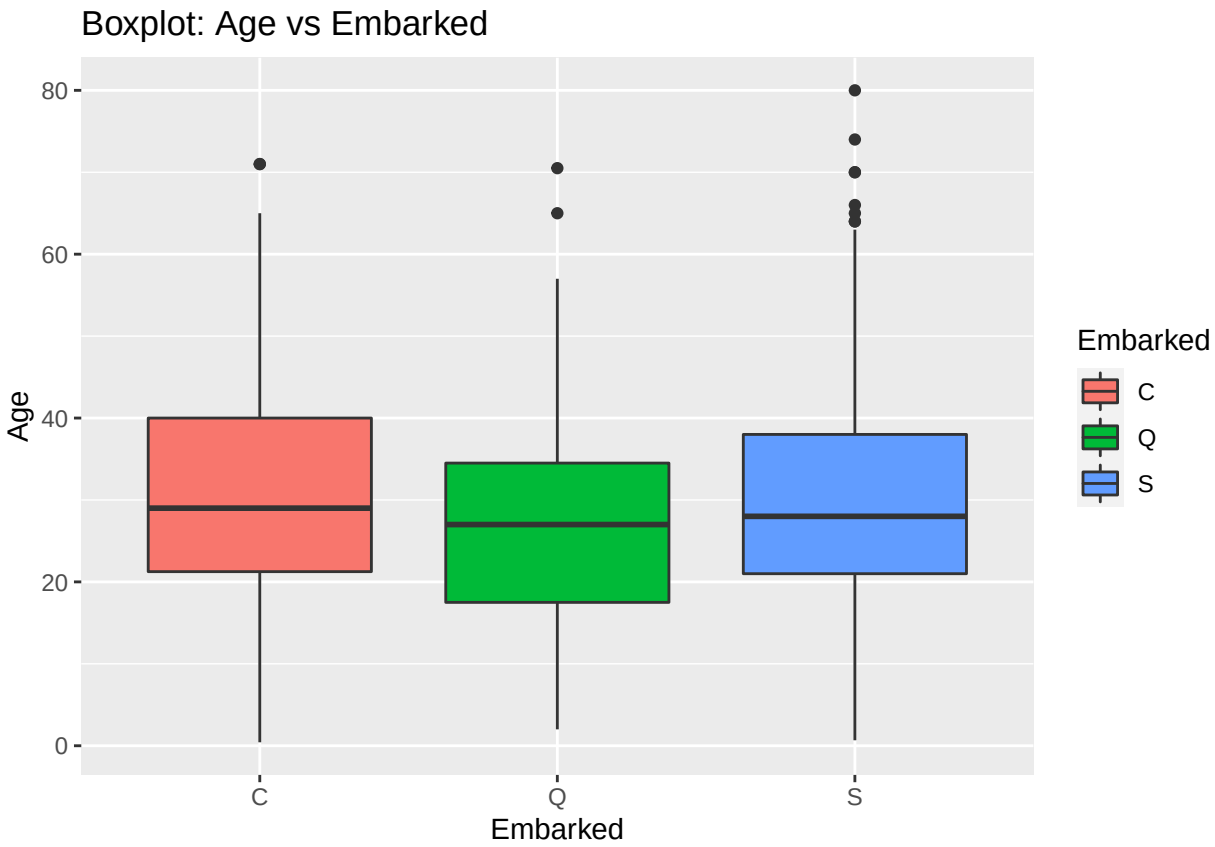
Observamos gráficamente la relación entre la edad y el género



No se aprecian diferencias significativas de edad en función del género

3.1.1.3.2 Age vs Embarked

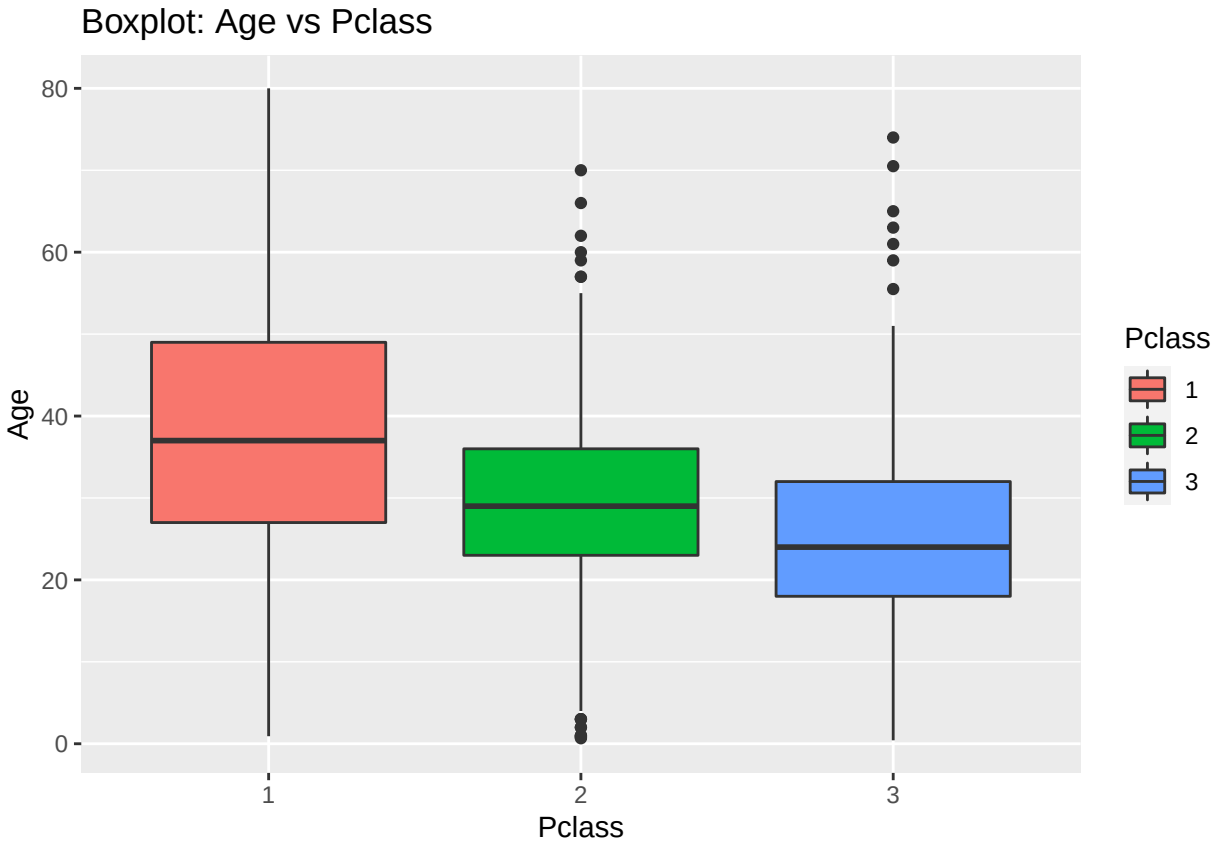
Observamos gráficamente la relación entre la edad y el puerto de Embarque



No se aprecian diferencias significativas con respecto a las medias de edad en cada una de las clases de **Embarked**.

3.1.1.3.3 Age vs Pclass

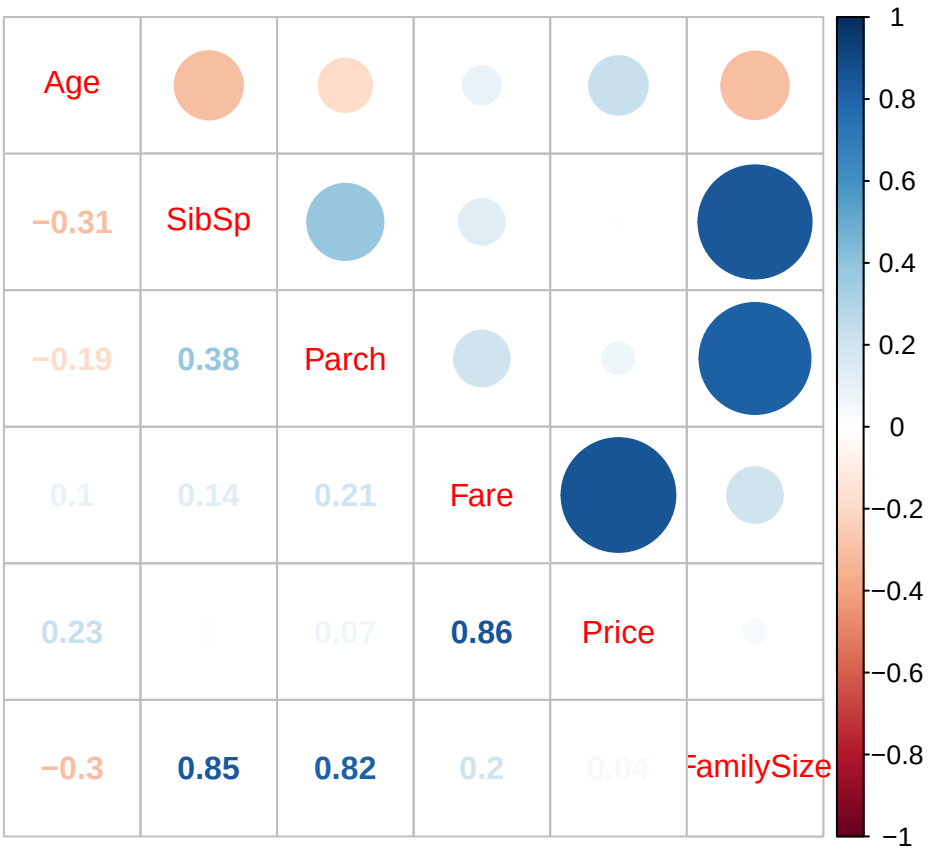
Observamos gráficamente la relación entre Age y Pclass:



Observamos una relación entre la edad y la clase en que viajaban los pasajeros: Los pasajeros de clase 1 (alta) tenían generalmente mayor edad que los de clase 2 (media) e igualmente sucede con los de clase 3 (baja). Lo cual es lógico (más edad más poder adquisitivo).

3.1.1.3.4 Age vs Variables numéricas

Vemos las distintas correlaciones de las variables numéricas



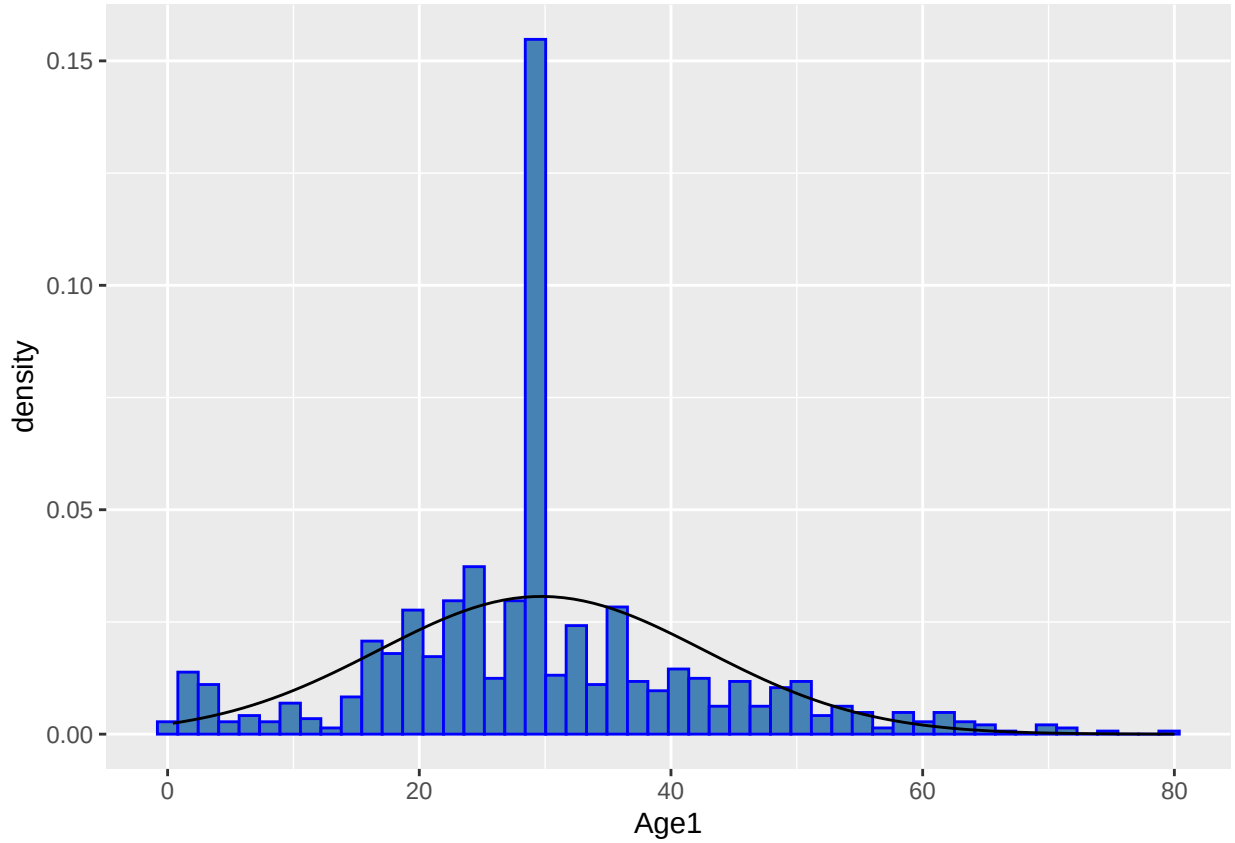
Observamos una correlación negativa con **SibSp** y con **Parch**.

3.1.1.3.5 Imputación de valores a Age

Vamos a realizar dos simulaciones diferentes de imputación y nos quedaremos con la mejor:

Caso 1: Imputar la media de Age a todos los elementos faltantes

```
data$Age1 <- data$Age
data$Age1[is.na(data$Age1)] <- mean(data_NoNA$Age)

ggplot(data, aes(Age1)) +
  geom_histogram(aes(y = ..density..), bins=50, fill="steelblue", color="blue") +
  stat_function(fun = dnorm, args = list(mean = mean(data$Age1), sd = sd(data$Age1)))
```



Caso 2: Imputando datos de Age**, con MICE (Multivariate Imputation via Chained Equations)**

En este caso se predicen los valores de **Age**, con el resto de valores observados (**Pclass**, **SibSp**, **Parch**, **Sex** y **Age**).

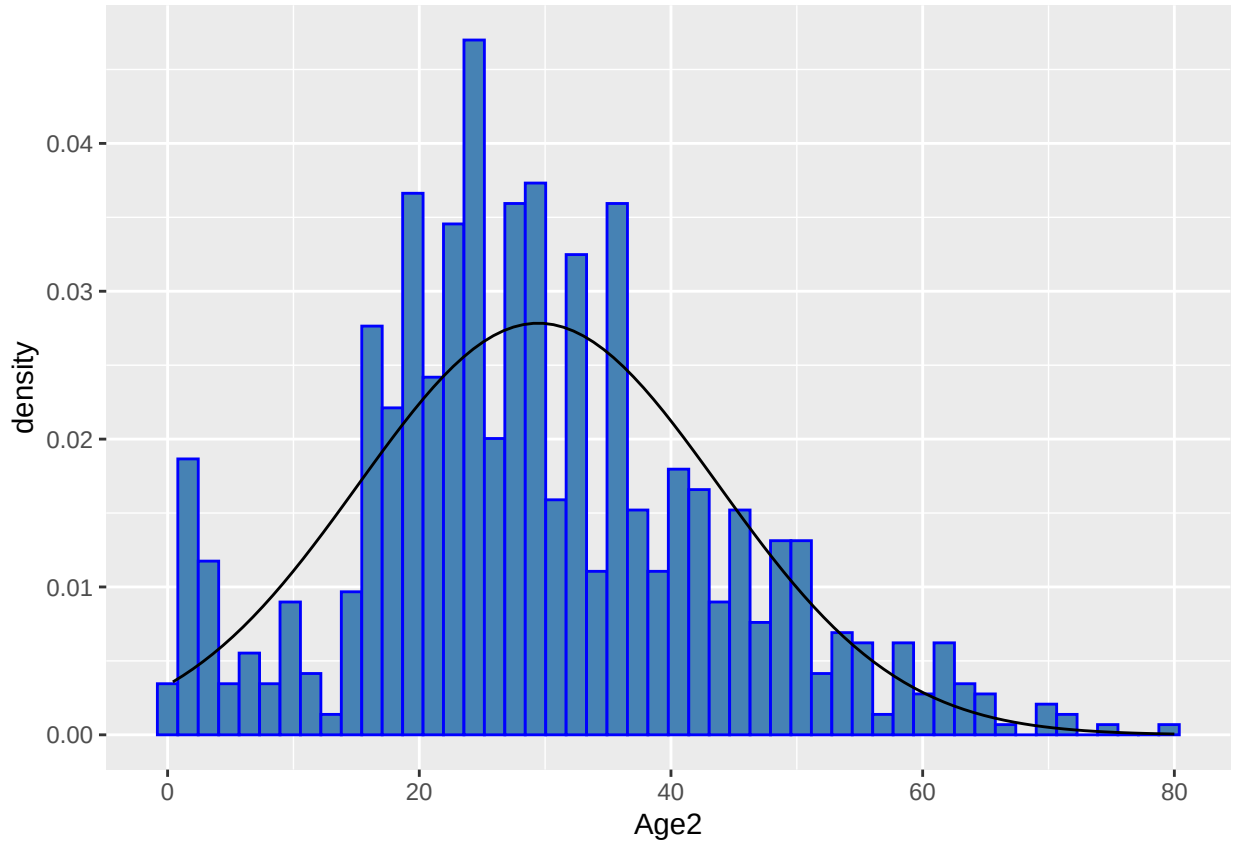
```
columnas <- c('Pclass', 'SibSp', 'Parch', 'Sex', 'Age')

mice_imputar <- mice(data = data[, columnas], method = "rf")
```

```
##
##  iter imp variable
##    1   1  Age
##    1   2  Age
##    1   3  Age
##    1   4  Age
##    1   5  Age
##    2   1  Age
##    2   2  Age
##    2   3  Age
##    2   4  Age
##    2   5  Age
##    3   1  Age
##    3   2  Age
##    3   3  Age
##    3   4  Age
##    3   5  Age
##    4   1  Age
```

```
##    4    2  Age
##    4    3  Age
##    4    4  Age
##    4    5  Age
##    5    1  Age
##    5    2  Age
##    5    3  Age
##    5    4  Age
##    5    5  Age
```

```
mice_completo <- mice::complete(mice_imputar)
data$Age2 <- data$Age
data$Age2[is.na(data$Age2)]<- mice_completo$Age[is.na(data$Age2)]
```

Observamos la gráfica:



Mostramos el resultado de las opciones de imputación más la original:

```
summary(data$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.42   20.12   28.00   29.70   38.00   80.00     177
```

```
summary(data$Age1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.42   22.00   29.70   29.70   35.00   80.00
```

```
summary(data$Age2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.42   20.00   28.00   29.45   38.00   80.00
```

Tras ver los resultados nos decantamos por la segunda opción (Age2), ya que la distribución se parece mucho más a la original. Asignamos y limpiamos el conjunto de datos:

```
data$Age <- data$Age2
data <- select(data, -Age1, -Age2)
```

3.1.2 Valores igual a cero

Recordamos los valores con ceros del dataset

```
##       variable q_zeros p_zeros q_na p_na q_inf p_inf    type unique
## 1     Survived     549   61.62    0    0     0     0  factor      2
## 2       Pclass       0    0.00    0    0     0     0  factor      3
## 3          Sex       0    0.00    0    0     0     0  factor      2
## 4          Age       0    0.00    0    0     0     0 numeric     88
## 5        SibSp     608   68.24    0    0     0     0 integer      7
## 6        Parch     678   76.09    0    0     0     0 integer      7
## 7         Fare      15    1.68    0    0     0     0 numeric    248
## 8     Embarked       0    0.00    0    0     0     0  factor      3
## 9        Price      15    1.68    0    0     0     0 numeric    248
## 10 FamilySize       0    0.00    0    0     0     0 numeric      9
```

Tenemos un alto número de valores cero tanto en **SibSp** como en **Parch**. No obstante, son valores válidos dentro del rango para estas variables ya que expresan el número de acompañantes del pasajero. Sorprende que hay algunos valores cero en **Fare**, aunque pudiera significar que esos tickets fueron gratis debido a algún sorteo o regalo.

3.1.3 Conclusión limpieza nulos y ceros

Observamos que hemos eliminado los valores nulos y tenemos un nuevo valor de media y mediana para **Age**.

```
summary(data)
```

```
##  Survived Pclass      Sex           Age            SibSp           Parch
##  0:549    1:216   female:314   Min.   : 0.42   Min.   :0.000   Min.   :0.0000
##  1:342    2:184   male  :577   1st Qu.:20.00   1st Qu.:0.000   1st Qu.:0.0000
##           3:491                Median :28.00   Median :0.000   Median :0.0000
##                                Mean   :29.45   Mean   :0.523   Mean   :0.3816
##                                3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000
##                                Max.   :80.00   Max.   :8.000   Max.   :6.0000
##       Fare        Embarked     Price           FamilySize
##  Min.   :  0.00   C:168    Min.   :  0.000   Min.   : 1.000
##  1st Qu.:  7.91   Q: 77    1st Qu.:  7.763   1st Qu.: 1.000
##  Median : 14.45   S:646    Median :  8.850   Median : 1.000
##  Mean   : 32.20            Mean   : 17.789   Mean   : 1.905
##  3rd Qu.: 31.00            3rd Qu.: 24.288   3rd Qu.: 2.000
##  Max.   :512.33            Max.   :221.779   Max.   :11.000
```

Observamos también los datos tras eliminar variables y tras imputar valores en aquellas variables que presentaban valores nulos.

```
df_status(data)
```

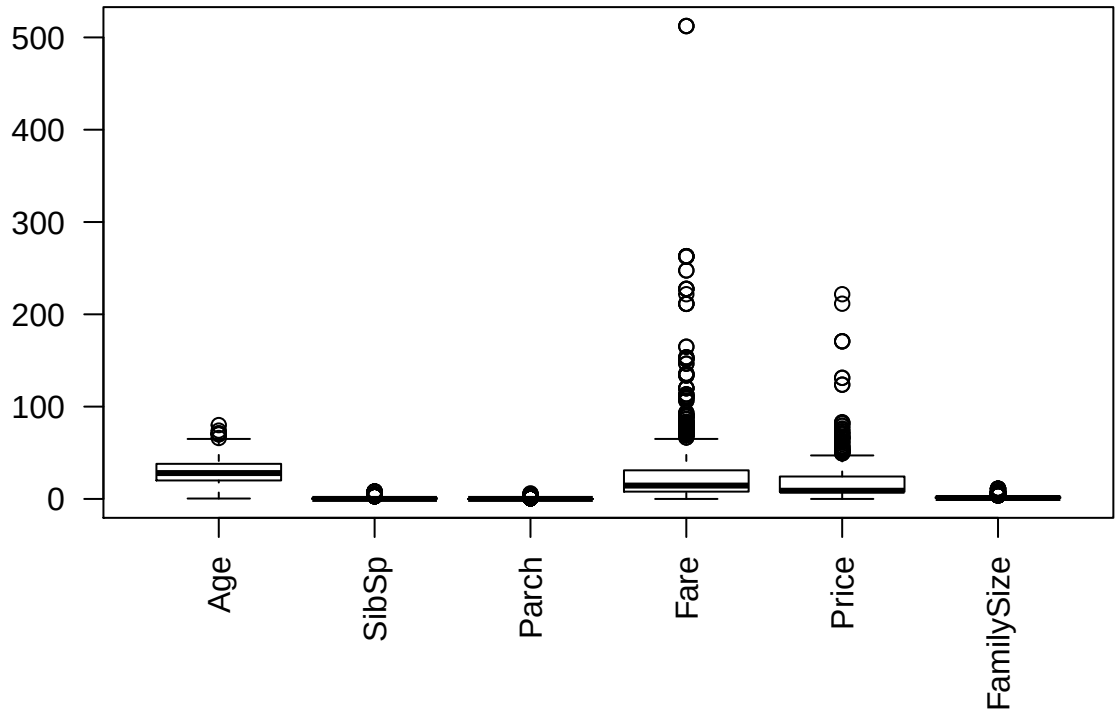
```
##       variable q_zeros p_zeros q_na p_na q_inf p_inf    type unique
## 1     Survived     549   61.62    0    0     0     0  factor      2
## 2       Pclass       0    0.00    0    0     0     0  factor      3
## 3          Sex       0    0.00    0    0     0     0  factor      2
## 4          Age       0    0.00    0    0     0     0 numeric     88
## 5        SibSp     608   68.24    0    0     0     0 integer      7
## 6        Parch     678   76.09    0    0     0     0 integer      7
## 7         Fare      15    1.68    0    0     0     0 numeric    248
## 8     Embarked       0    0.00    0    0     0     0  factor      3
## 9        Price      15    1.68    0    0     0     0 numeric    248
## 10 FamilySize       0    0.00    0    0     0     0 numeric      9
```

3.2 Identificación y tratamiento de valores extremos.

Se considera un valor extremo, outlier, a un valor fuera de rango. Son valores que se salen de la escala esperada visualizando el resto de las observaciones. En la actualidad, el criterio más habitual es considerar un valor extremo a aquel que se encuentra alejado de la media unas tres veces la desviación típica.

Gráfico boxplot variables numéricas

```
lista<-sapply(data, is.numeric)
data_num<-data[,lista]
boxplot(data_num,las=2)
```



```
var.continuas <- vector()
```

De todas las variables numéricas sólamente tres son contínuas: **Age**, **Price** y **Fare**

Analicemos esas tres variables por separado

3.2.1 Fare

El boxplot parece indicar que Fare tiene valores extremos. Los identificamos usando el criterio de tres veces la desviación típica:

```
data_out <- as.data.frame(data$Fare)
data_out$outlier <- FALSE
for (i in 1:ncol(data_out) - 1){
  columna = data_out[, i]
  if (is.numeric(columna)) {
    media = mean(columna)
    desviacion = sd(columna)
    data_out$outlier = (columna> (media+3*desviacion) | columna<(media-3*desviacion))
  }
}
table(data_out$outlier)
```

```
##
## FALSE  TRUE
##   871    20
```

Con este criterio tenemos identificados 20 posibles *outliers*.

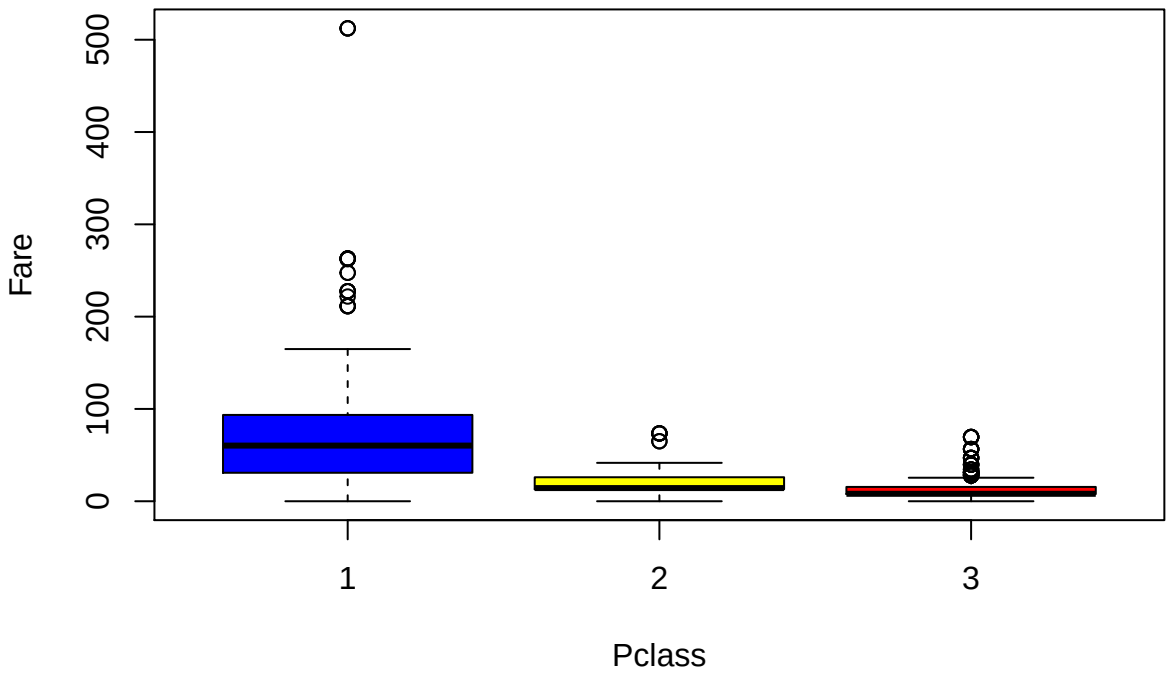
```
boxplot.stats(data$Fare)$out
```

```
##   [1]  86.5000  86.5000  86.5000  79.6500  79.6500  79.6500  75.2500 211.5000
##   [9]  80.0000  80.0000 120.0000 120.0000 120.0000 120.0000  66.6000  66.6000
##  [17] 151.5500 151.5500 151.5500 151.5500  83.1583  83.1583  76.2917  91.0792
##  [25]  91.0792  93.5000  93.5000  77.9583  77.9583  77.9583  79.2000 134.5000
##  [33] 134.5000 110.8833 110.8833 110.8833 110.8833  89.1042  89.1042  78.8500
##  [41]  78.8500  90.0000  90.0000  90.0000  90.0000 263.0000 263.0000 263.0000
##  [49] 263.0000 211.3375 211.3375 211.3375  81.8583 113.2750 113.2750 113.2750
##  [57]  77.2875  77.2875 164.8667 164.8667  78.2667  78.2667  83.4750  83.4750
##  [65]  69.5500  69.5500  69.5500  69.5500  69.5500  69.5500  69.5500  69.3000
##  [73]  69.3000 221.7792 247.5208 247.5208 146.5208 146.5208  76.7292  76.7292
##  [81]  76.7292 153.4625 153.4625 153.4625  79.2000  79.2000  79.2000  71.2833
##  [89]  82.1708  82.1708 262.3750 262.3750 133.6500 133.6500 512.3292 512.3292
##  [97] 512.3292  83.1583 227.5250 227.5250 227.5250 227.5250 108.9000 108.9000
## [105] 135.6333 135.6333 135.6333 106.4250 106.4250  73.5000  73.5000  73.5000
## [113]  73.5000  73.5000  71.0000  71.0000
```

Los valores están bien distribuidos, los más altos en las clases altas. Por todo esto, decidimos no realizar ninguna acción con estos *outliers* ya que incluso pueden estar aportando información importante:

Gráfico outlier Fare

```
boxplot(data$Fare~data$Pclass,xlab="Pclass",ylab="Fare",
        col=c("blue","yellow","red"))
```



3.2.2 Price

Hacemos un análisis similar con la variable Price:

```
data_out <- as.data.frame(data$Price)
data_out$outlier <- FALSE
for (i in 1:ncol(data_out) - 1){
  columna = data_out[, i]
  if (is.numeric(columna)) {
    media = mean(columna)
    desviacion = sd(columna)
    data_out$outlier = (columna> (media+3*desviacion) | columna<(media-3*desviacion))
  }
}
table(data_out$outlier)
```

```
##
## FALSE  TRUE
##   878    13
```

Con boxplot.stats analizamos los outliers, el valor máximo 221:

```
boxplot.stats(data$Price)$out
```

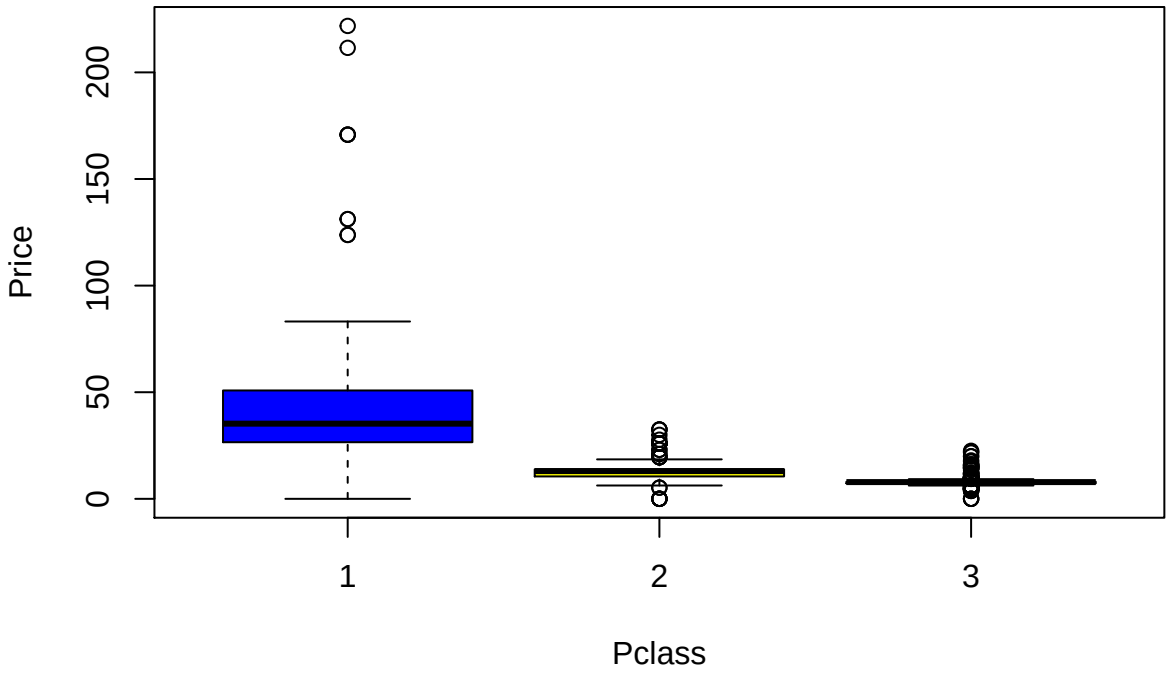
```
##  [1]  75.25000 211.50000  61.97920  50.00000  53.10000  52.55420  55.44170
##  [8]  51.47920  76.29170  79.20000  49.50000  67.25000  67.25000  51.86250
## [15]  51.86250  65.75000  65.75000  65.75000  65.75000  70.44583  70.44583
```

```
## [22]  70.44583  81.85830  82.43335  82.43335  52.00000  49.50420 221.77920
## [29] 123.76040 123.76040  73.26040  73.26040  51.15417  51.15417  51.15417
## [36]  79.20000  50.49580  61.37920  71.28330  59.40000 131.18750 131.18750
## [43]  49.50420  66.82500  66.82500 170.77640 170.77640 170.77640  83.15830
## [50]  56.88125  56.88125  56.88125  56.88125  54.45000  54.45000  63.35830
## [57]  53.21250  53.21250  61.17500
```

Los valores más altos están presentes en las clases altas:

Gráfico boxplot Price

```
boxplot(data$Price~data$Pclass,xlab="Pclass",ylab="Price",col=c("blue","yellow","red"))
```



Optamos por considerarlos válidos y no realizamos ninguna accion.

3.2.3 Age

Hacemos un análisis similar con la variable Age:

```
data_out <- as.data.frame(data$Age)
data_out$outlier <- FALSE
for (i in 1:ncol(data_out) - 1){
  columna = data_out[, i]
  if (is.numeric(columna)) {
    media = mean(columna)
    desviacion = sd(columna)
    data_out$outlier = (columna> (media+3*desviacion) | columna<(media-3*desviacion))
  }
```

```
}
table(data_out$outlier)
```

```
##
## FALSE  TRUE
##   889     2
```

Con boxplot.stats:

```
boxplot.stats(data$Age)$out
```

```
## [1] 80.0 74.0 70.5 66.0 70.0 71.0 71.0 70.0
```

Podríamos tener 2 valores outliers pero observando los valores devueltos por boxplot.stats no los vamos a considerar significativos.

4 Análisis de los datos.

4.1 Planificación de los análisis a aplicar.

Nos interesa poder realizar diferentes tipos de análisis en función de diferentes subconjuntos de datos, como puede ser el género, la clase en la que viajan los pasajeros, el puerto en el que han embarcado, grupos por edad, índice de supervivencia de los niños respecto a los adultos, etc.

4.1.1 Niños

Definimos una variable Child para aquellos registros en los que la edad sea inferior a 8 años.

```
edad_corte = 8
data$Child[data$Age <= edad_corte] <- 1
data$Child[data$Age > edad_corte] <- 0
data$Child <- as.factor((data$Child))
```

4.1.2 Género

Agrupamos por el género, creando una variable para cada uno de los géneros

```
Mujeres <- data[which(data$Sex == 'female'),]
Hombres <- data[which(data$Sex == 'male'),]
```

4.1.3 Lugar de embarque

Agrupamos por el lugar de embarque creando una variable por cada uno de los lugares

```
EmbarqueC <- data[which(data$Embarked == 'C'),]
EmbarqueQ <- data[which(data$Embarked == 'Q'),]
EmbarqueS <- data[which(data$Embarked == 'S'),]
```

4.1.4 Clase

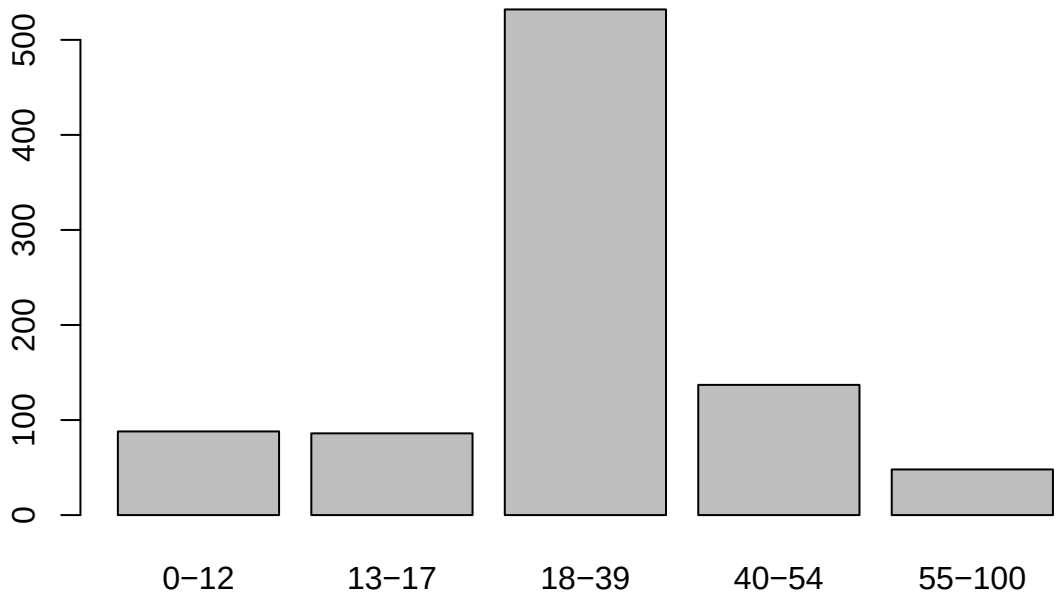
Agrupamos por clase

```
FirstClass <- data[which(data$Pclass == 1),]
SecondClass <- data[which(data$Pclass == 2),]
ThirdClass <- data[which(data$Pclass == 3),]
```

4.1.5 Edades

Vamos a discretizar los valores por grupos de edades, añadiendo una columna al dataset (AgeInterval)

```
##
##   0-12  13-17  18-39  40-54 55-100
##     88     86    532    137     48
```



4.2 Comprobación de la normalidad y homogeneidad de la varianza.

4.2.1 Normalidad

Algunos test estadísticos requieren que las variables que van a ser analizadas sigan una distribución normal, por tanto tenemos que conocer cuáles son las distribuciones de nuestras variables continuas. Estas son las variables **Age** y **Fare**. Además, hemos generado una nueva variable a partir de **Fare**, **Price**, y que, por tanto también es una variable cualitativa continua.

En general, la prueba de *Shapiro-Wilk* se considera una prueba muy potente para contrastar la normalidad de distribuciones. Se asume como hipótesis nula que la población sigue una distribución normal. Si el p-valor obtenido es inferior al nivel de significancia (normalmente $= 0{,}05$) entonces se rechaza la hipótesis nula (y por tanto se concluye que los datos no vienen de una distribución normal). En cambio, si el p-valor es superior al nivel de significancia, entonces no se puede rechazar la hipótesis nula y se asume que los datos siguen una distribución normal. Además, vamos a aplicar otros dos métodos, la prueba de *Anderson-Darling* y la prueba de *Kolmogorov-Smirnov* (conocida también como K-S)

Creamos un data frame para resumir los tests:

```
tabla.normalidad <- data.frame('variable' = character(),
                               'Test de Normalidad' = character(),
                               'Valor Estadístico' = numeric(),
                               'p-value' = numeric(),
                               stringsAsFactors = FALSE)
str(tabla.normalidad)
```

```
## 'data.frame':    0 obs. of  4 variables:
```

```
##  $ variable          : chr
##  $ Test.de.Normalidad: chr
##  $ Valor.Estadístico : num
##  $ p.value           : num
```

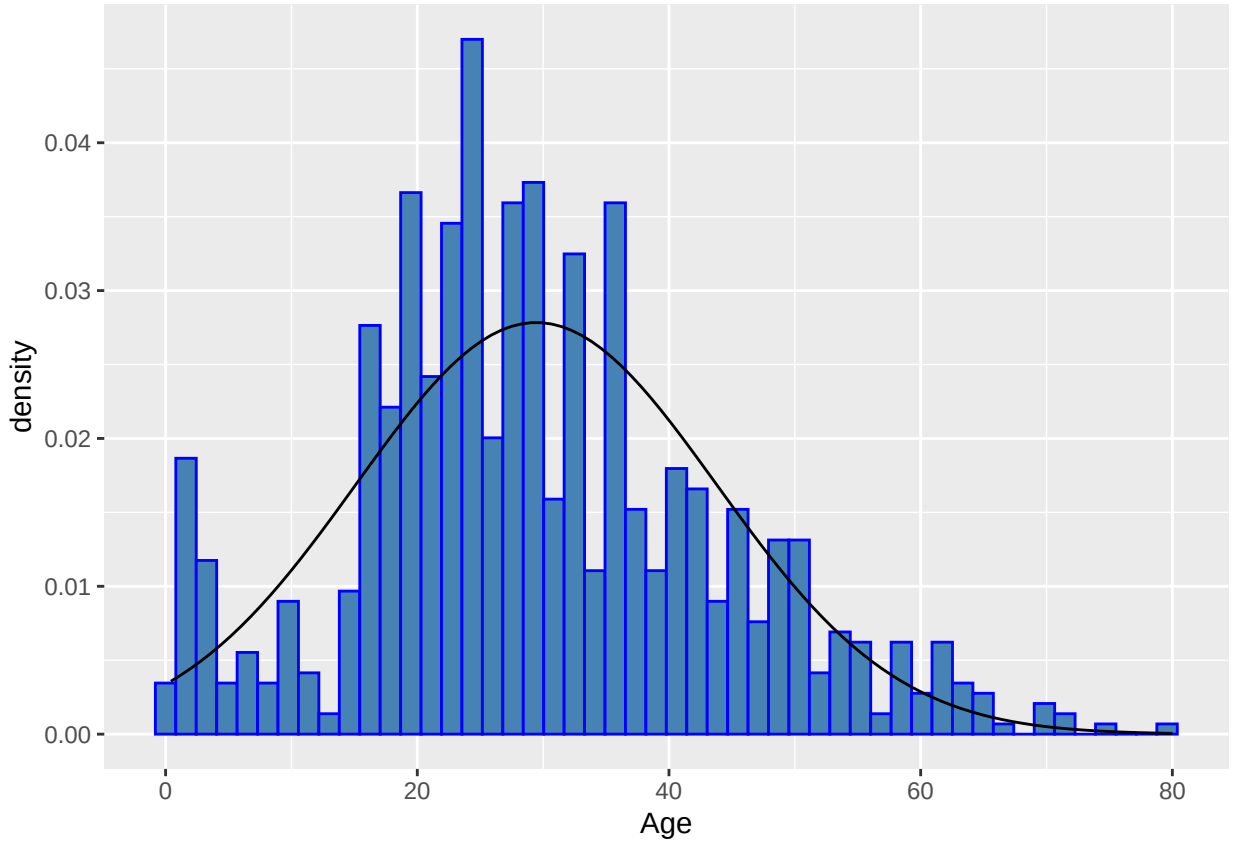
Ahora recorremos todas las variables continuas aplicando los tres tests y añadiéndolos al dataframe:

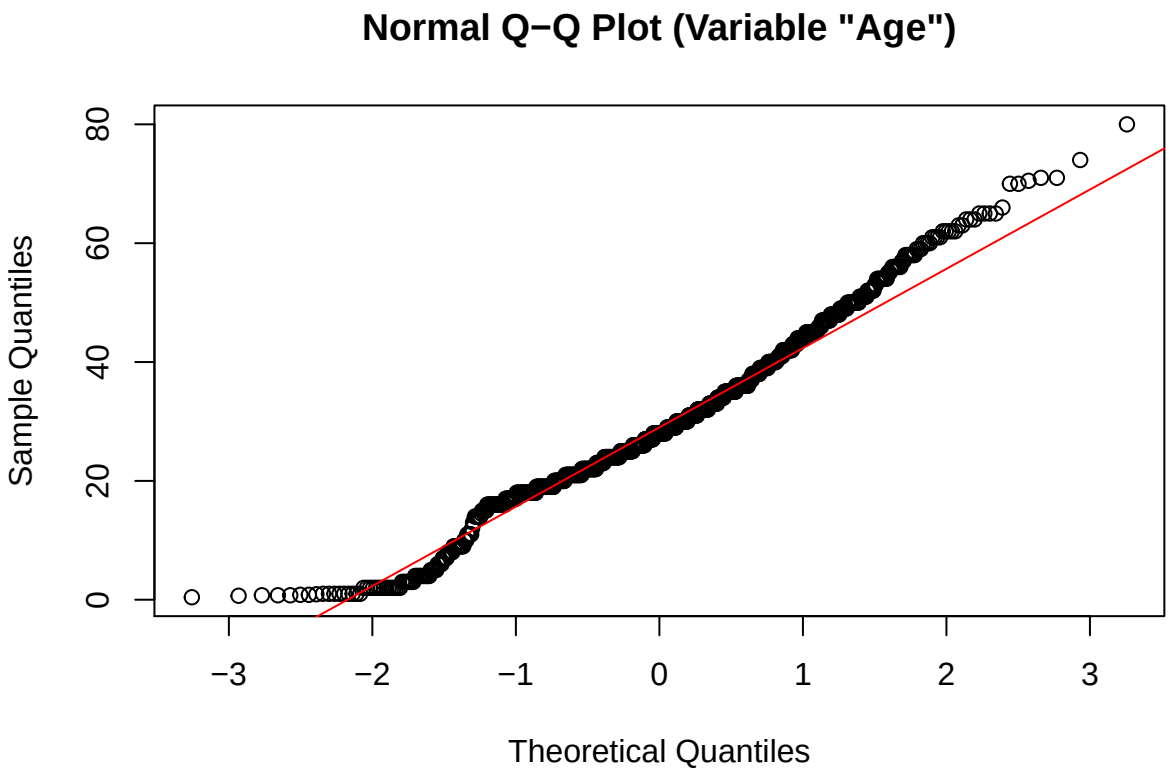
```
var.continuas <-c("Age", "Fare", "Price")

for (i in 1:length(var.continuas)){
  variable = var.continuas[i]
  #Test Shapiro-wil
  test = shapiro.test(data[,variable])
  tabla.normalidad[nrow(tabla.normalidad)+1,] = c(variable, test$method,
                                                  test$statistic, test$p.value)

  #Test Anderson-Darling
  test = ad.test(data[,variable])
  tabla.normalidad[nrow(tabla.normalidad)+1,] = c(variable, test$method,
                                                  test$statistic, test$p.value)

  #Test Kolmogorov-Smirnov
  test = ks.test(data[,variable], "pnorm", mean=mean(data[,variable]),
                 sd=sd(data[,variable]))
  tabla.normalidad[nrow(tabla.normalidad)+1,] = c(variable, test$method,
                                                  test$statistic, test$p.value)
}

knitr::kable(tabla.normalidad)
```

variable	Test.de.Normalidad	Valor.Estadístico	p.value
Age	Shapiro-Wilk normality test	0.981573339303633	3.59237516646611e-09
Age	Anderson-Darling normality test	4.76443331454129	8.36895246089196e-12
Age	One-sample Kolmogorov-Smirnov test	0.062646394633216	0.0018355008067592
Fare	Shapiro-Wilk normality test	0.521891302117355	1.08404452322613e-43
Fare	Anderson-Darling normality test	122.169627214592	3.7e-24
Fare	One-sample Kolmogorov-Smirnov test	0.281848040985975	0
Price	Shapiro-Wilk normality test	0.566484900244852	3.00259150054125e-42
Price	Anderson-Darling normality test	110.325199878766	3.7e-24
Price	One-sample Kolmogorov-Smirnov test	0.26829589819932	0

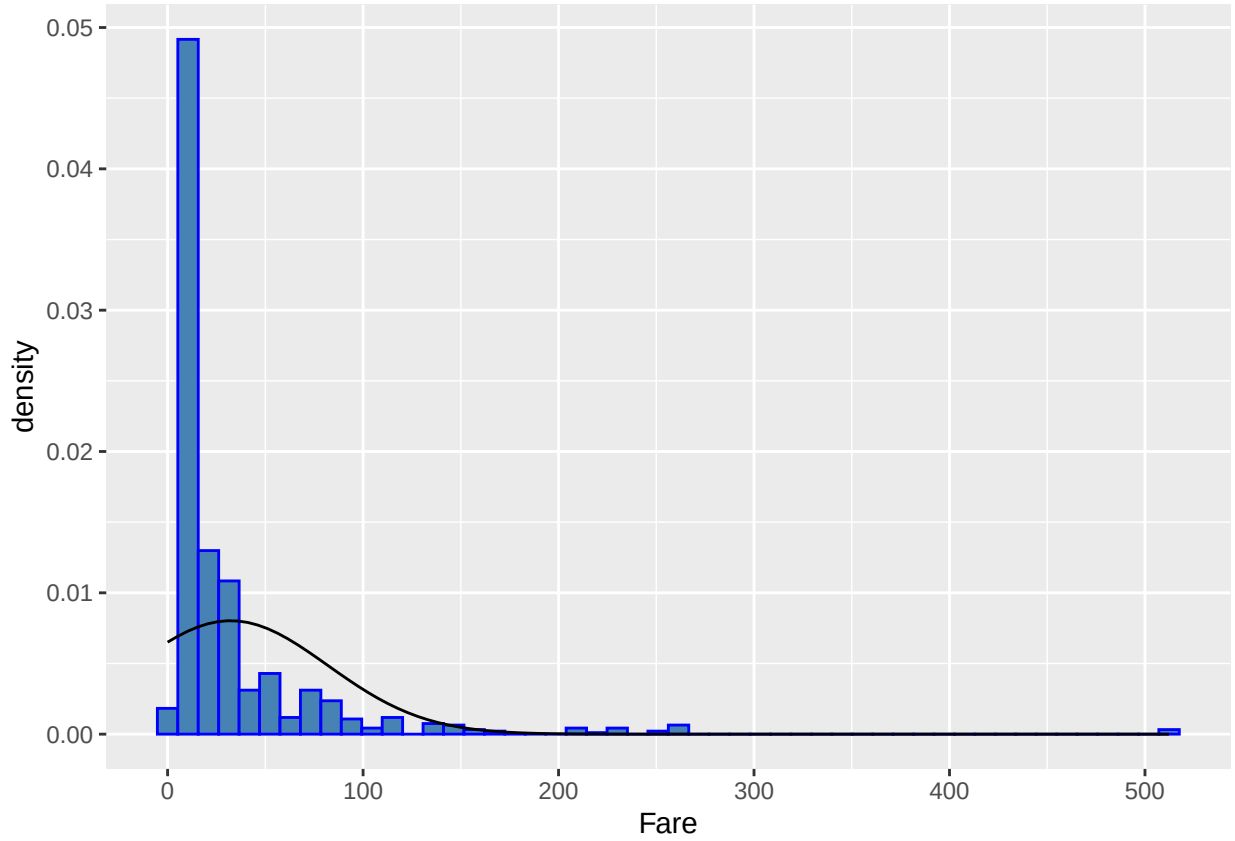
En todos los casos el *p-value* ha sido inferior a 0.05 y, por lo tanto, se rechaza la hipótesis nula que implicaba que la variable sigue una distribución normal. Ninguna de las 3 variables sigue una distribución normal.

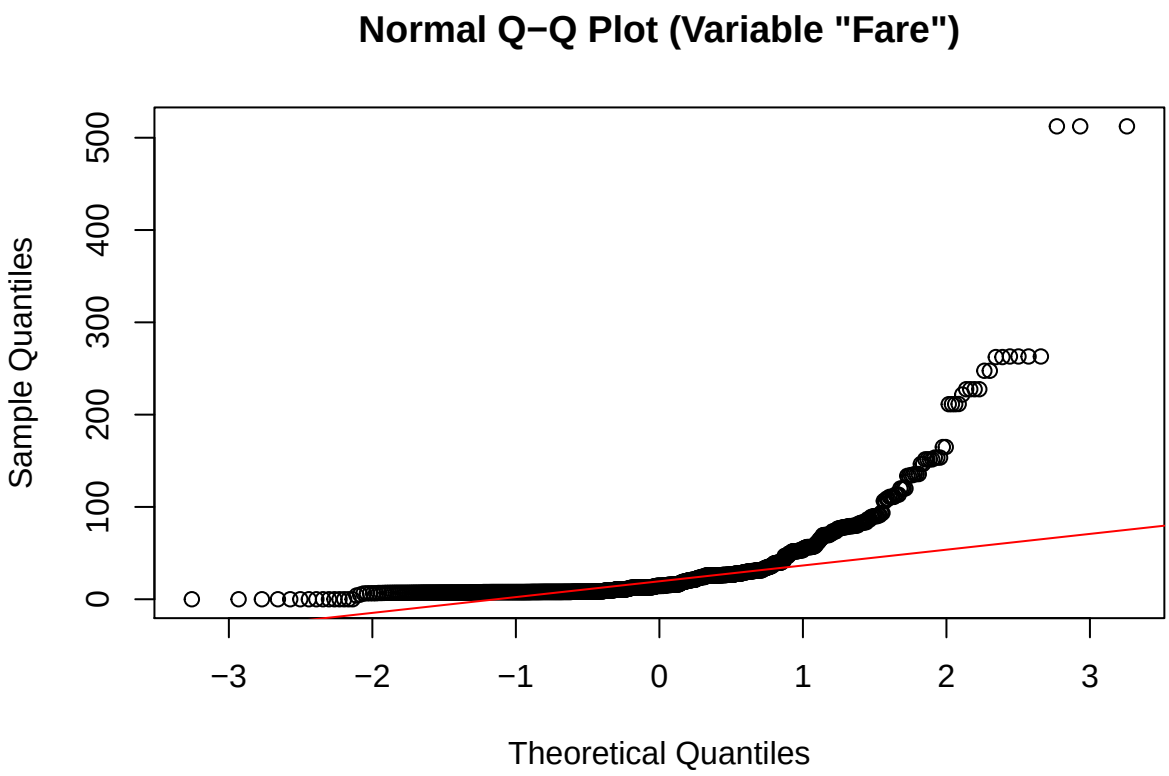
Revisamos gráficamente la distribución de cada una de las variables usando su histograma, curva de densidad y gráficas Q-Q


density
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
Age

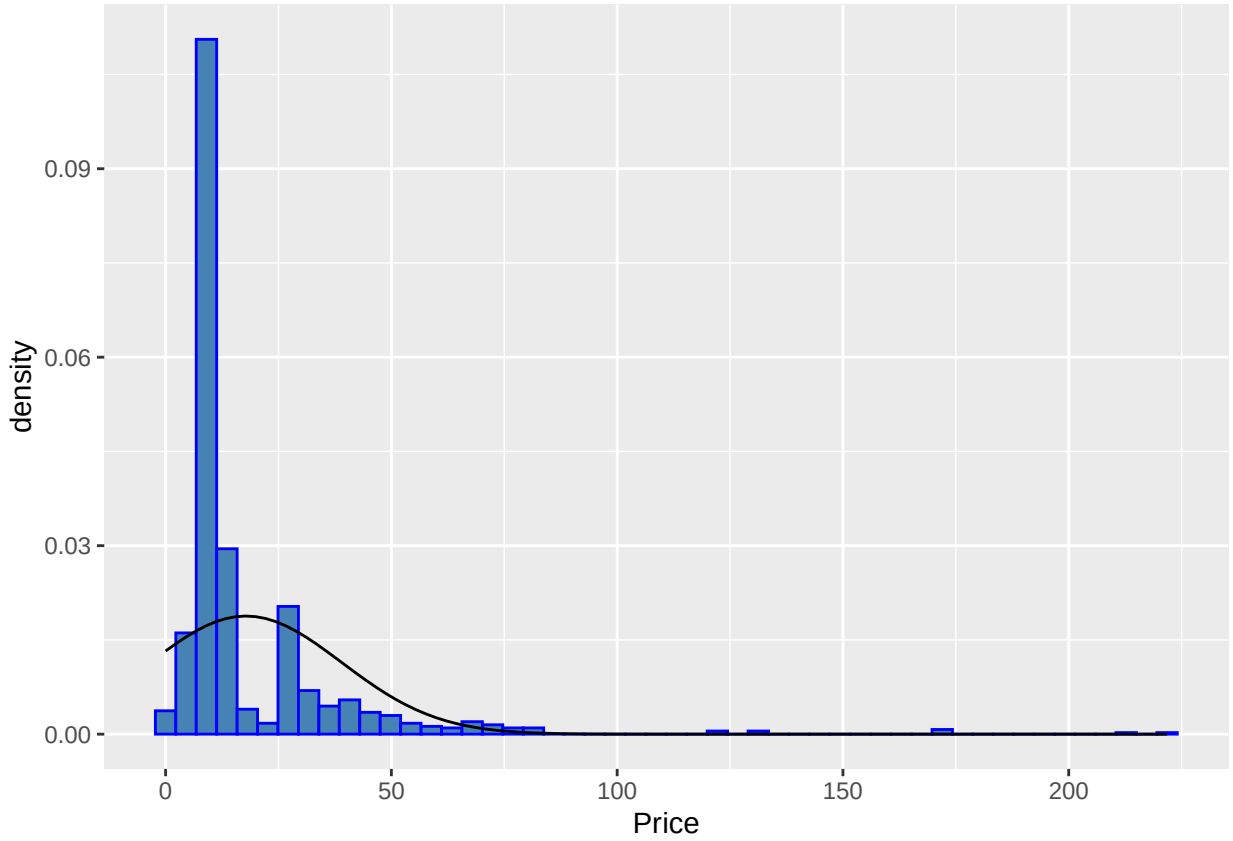

Normal Q−Q Plot (Variable "Age")
Sample Quantiles
Theoretical Quantiles
0
20
40
60
80
−3
−2
−1
0
1
2
3

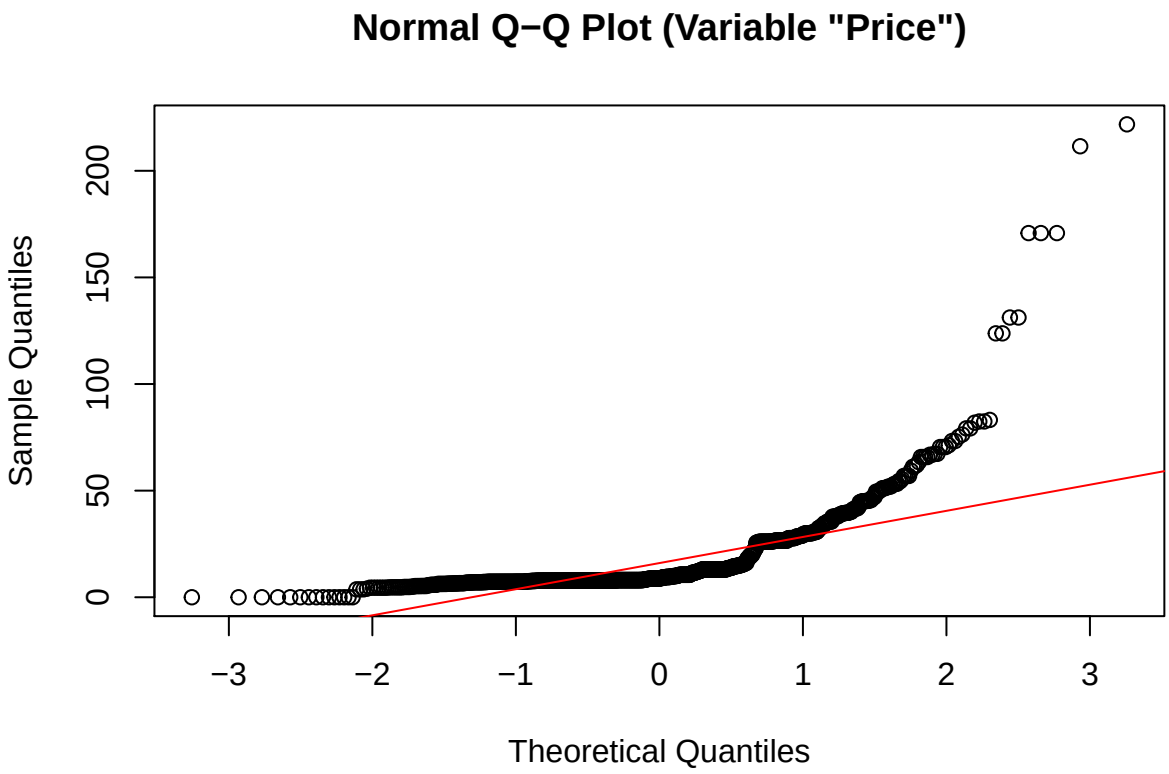
```
## [1] "Age"
```


0.05
0.04
0.03
0.02
0.01
0.00
density
0
100
200
300
400
500
Fare


Normal Q−Q Plot (Variable "Fare")
Sample Quantiles
0
100
200
300
400
500
−3
−2
−1
0
1
2
3
Theoretical Quantiles

```
## [1] "Fare"
```


density
0.09
0.06
0.03
0.00
0
50
100
150
200
Price




```
## [1] "Price"
```

Observamos que la variable **Age** está próxima a una distribución normal por lo cual podemos utilizar el test de *Levene* para la comprobación de varianzas. Cuando comparamos varianzas lo que estamos comprobando es que las varianzas entre los grupos a comparar son iguales. La hipótesis nula asume la igualdad de varianzas en los diferentes grupos de datos, con lo que si el p-value obtenido es inferior al nivel de significancia (generalmente alpha = 0,05) se rechaza la hipótesis nula y se concluye que hay heterocedasticidad.

4.2.1.1 Age

4.2.1.1.1 Age con Survived

Primero, comprobamos la homogeneidad de varianzas de la edad en los grupos de supervivientes y no supervivientes.

```
leveneTest(data = data, Age ~ Survived, center = mean)
```

	Df	F value	Pr(>F)
group	1	0.1051012	0.7458681
	889	NA	NA

En este caso el p-value es superior a 0.05 y por tanto asumimos que hay **homogeneidad de varianzas** entre los grupos

4.2.1.1.2 Age con Embarked

Comprobamos el comportamiento de la varianza cuando se trata de la variable edad **Age** con **Embarked**.

```
leveneTest(data = data, Age ~ Embarked, center = mean)
```

	Df	F value	Pr(>F)
group	2	1.053172	0.3492647
	888	NA	NA

Hay **homogeneidad de varianzas** de **Age** en los grupos que define la variable **Embarked**

4.2.1.1.3 Age con Pclass

Comprobamos el comportamiento de la varianza cuando se trata de la variable edad **Age** con **Pclass**.

```
leveneTest(data = data, Age ~ Pclass, center = mean)
```

	Df	F value	Pr(>F)
group	2	7.208846	0.0007841
	888	NA	NA

Hay **heterogeneidad de varianzas** entre las muestras de **Age** cuando se agrupan por **Pclass**

4.2.1.1.4 Age con Sex

Comprobamos el comportamiento de la varianza cuando se trata de la variable edad **Age** con respecto a **Sex**.

```
leveneTest(data = data, Age ~ Sex, center = mean)
```

	Df	F value	Pr(>F)
group	1	0.1382756	0.71009
	889	NA	NA

Hay **homogeneidad de varianzas** para *Age* cuando está agrupado por *Sex*

4.2.1.1.5 Resumen Age

Resumimos en una tabla los resultados de homogeneidad de la varianza de Age con respecto a los distintos atributos:

Age con respecto a	resultado
Survived	homogeneidad
Embarked	homogeneidad
Pclass	heterogeneidad
Sex	homogeneidad

4.3 Aplicación de pruebas estadísticas para comparar los grupos de datos.

4.3.1 Análisis de relaciones entre variables

Vamos a analizar las relaciones entre variables para aclarar la dependencia entre ellas y como pueden afectar a las posibilidades de supervivencia:

4.3.1.1 Age y Sex

Nos gustaría analizar las diferencias entre la media de edad de mujeres y de hombres, utilizando el test paramétrico *t-test de Student*, que requiere que las muestras a comparar sigan una distribución normal. Comprobemos primero que la variable **Age** sigue una distribución normal:

```
var.test(Mujeres$Age, Hombres$Age)
```

```
##
##  F test to compare two variances
##
## data:  Mujeres$Age and Hombres$Age
```

```
## F = 0.93176, num df = 313, denom df = 576, p-value = 0.4845
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.7689208 1.1355235
## sample estimates:
## ratio of variances
##          0.9317562
```

El resultado nos indica que las varianzas de edad en los grupos de mujeres y hombres son iguales.

Comparamos ahora las medias de los 2 grupos, utilizando *t-Test* indicando que las varianzas de los grupos son iguales. La hipótesis nula será que no hay diferencias significativas entre la media de edades para hombres y mujeres.

```
t.test(Mujeres$Age, Hombres$Age, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Mujeres$Age and Hombres$Age
## t = -1.7381, df = 889, p-value = 0.08254
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.7147053  0.2254088
## sample estimates:
## mean of x mean of y
##  28.31847  30.06312
```

El valor de p-values es inferior a 0.05, por lo que rechazamos la hipótesis nula. Es decir, **la media de edad por género tiene una diferencia significativa**.

Queremos comprobar ahora si la media de edad de las mujeres es menor que la de los hombres. Por lo tanto, la hipótesis nula corresponde a que la edad de las mujeres es mayor o igual a la de los hombres:

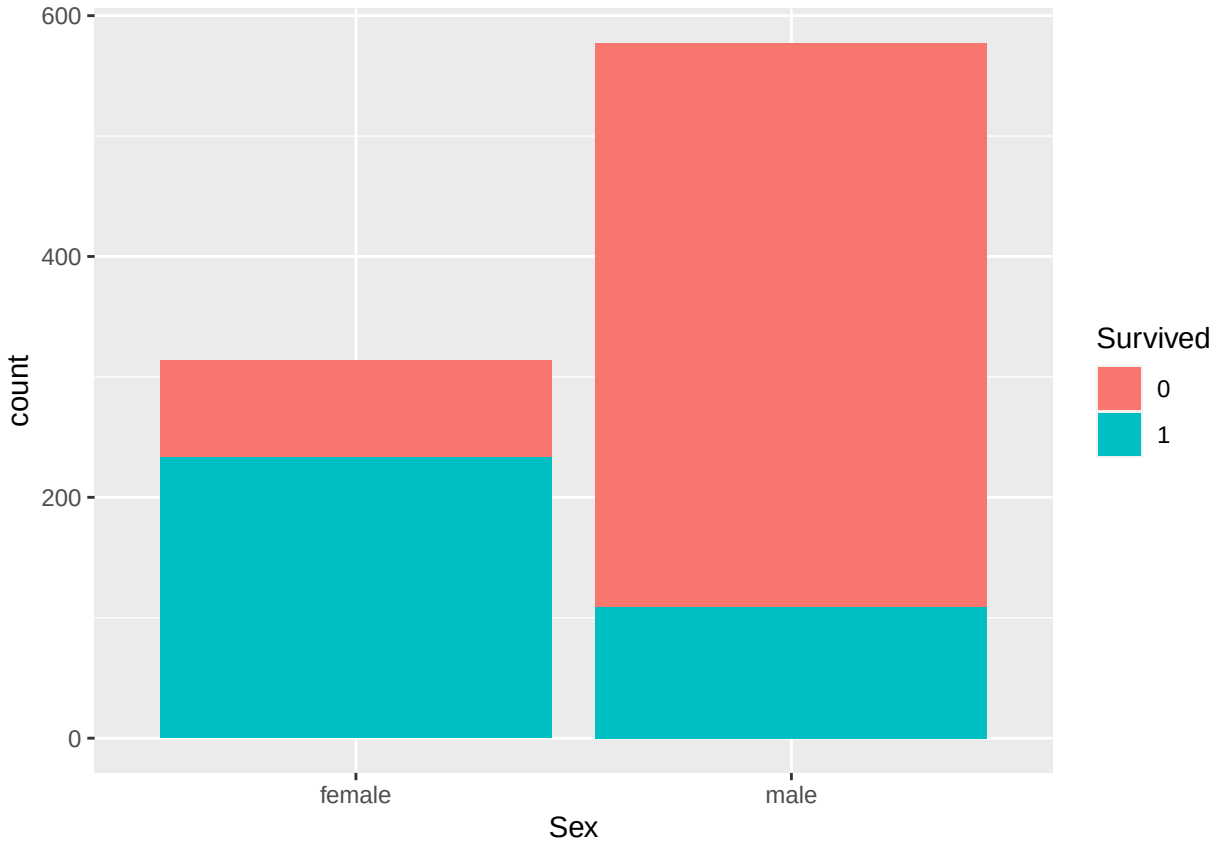
```
t.test(Mujeres$Age, Hombres$Age, alternative = "less", var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Mujeres$Age and Hombres$Age
## t = -1.7381, df = 889, p-value = 0.04127
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##        -Inf -0.0918529
## sample estimates:
## mean of x mean of y
##  28.31847  30.06312
```

Con un p-value menor de 0.5 rechazamos la hipótesis nula, es decir, podemos afirmar que la media de edad de las mujeres es inferior a la media de edad de los hombres.

4.3.1.2 Survival y Sex

Mostramos gráficamente la relación entre las dos variables:



Parece indicar que ser mujer implicaba mas posibilidades de supervivencia que ser hombre. Vamos a confirmar esto realizando algunos contrastes de hipótesis.

Primero, vamos a comprobar la relación entre la supervivencia y el género, analizando si dependiendo de si se era mujer u hombre las posibilidades de supervivencia cambiaban significativamente. Para hacerlo aplicamos el *test exacto de Fisher* que analiza tablas de contingencia. En este caso la hipótesis nula es que la proporción de mujeres que mueren coincide con la proporción de hombre que mueren en el accidente del Titanic.

```
fisher.test(table(data$Sex, data$Survived))
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table(data$Sex, data$Survived)
## p-value < 2.2e-16
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.0575310 0.1138011
## sample estimates:
## odds ratio
## 0.08128333
```

Con un p-valor < 0.05 rechazamos la hipótesis nula, es decir: **hay diferente proporción de supervivencia entre hombres y mujeres.**

Una vez confirmado que el género si que importaba, vamos a ver quien tiene más probabilidades de supervivencia, si las mujeres o los hombres. La hipótesis nula sería que la proporción de mujeres que mueren es mayor que la de los hombres.

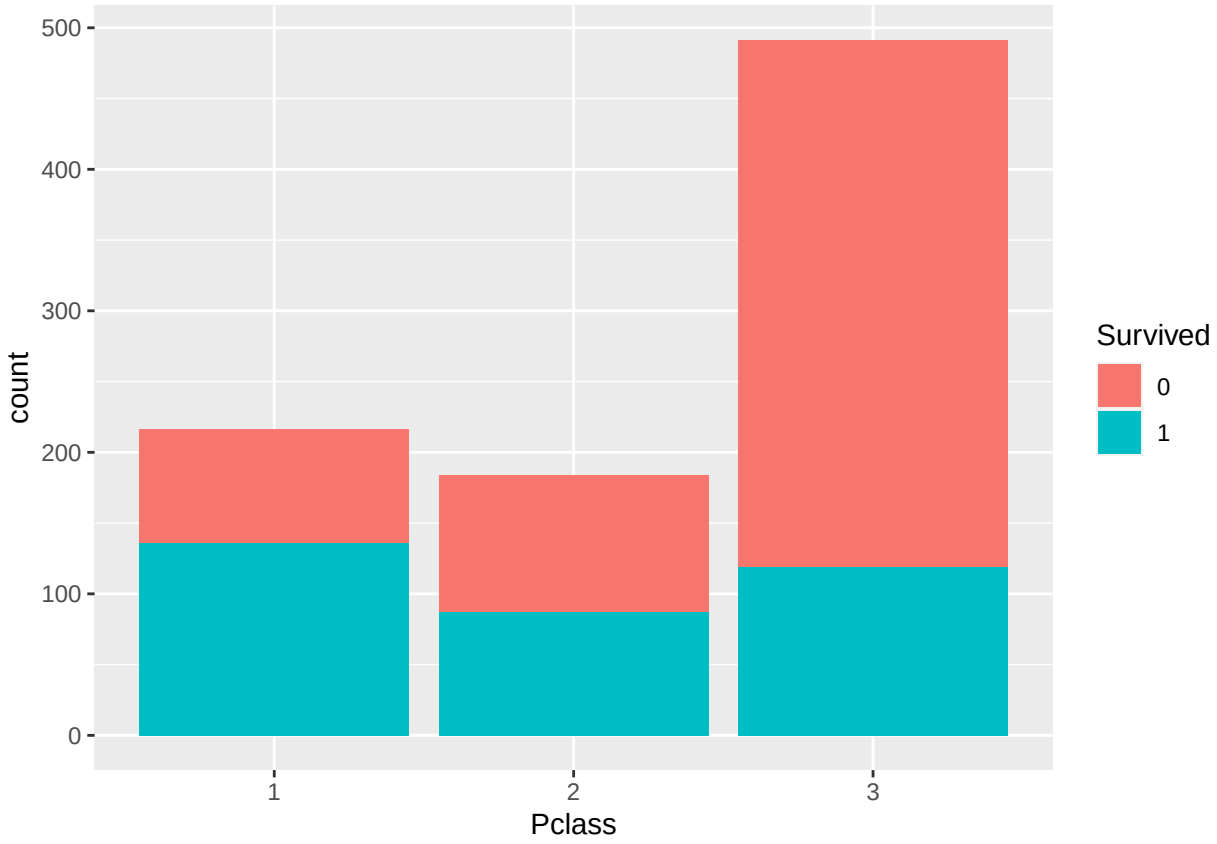
```
fisher.test(table(data$Sex, data$Survived), alternative = 'less')
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table(data$Sex, data$Survived)
## p-value < 2.2e-16
## alternative hypothesis: true odds ratio is less than 1
## 95 percent confidence interval:
##  0.0000000 0.1081391
## sample estimates:
## odds ratio
## 0.08128333
```

El valor de p-value inferior a 0.05) con lo que se cumple la hipótesis alternativa: **la proporción de mujeres que mueren es inferior a la de los hombres**.

4.3.1.3 Survival y Pclass

Mostramos gráficamente la relación entre las dos variables:



El porcentaje de supervivencia por clases disminuye desde la primera hasta la tercera clase, las clases con mas superviencia eran las mejores. Para comprobarlo utilizamos el test *Chi-Cuadrado* (son variables categóricas). Aquí la hipótesis nula nos dice que las variables son independientes.

```
test_chisq <- chisq.test(data$Pclass, data$Survived)
test_chisq
```

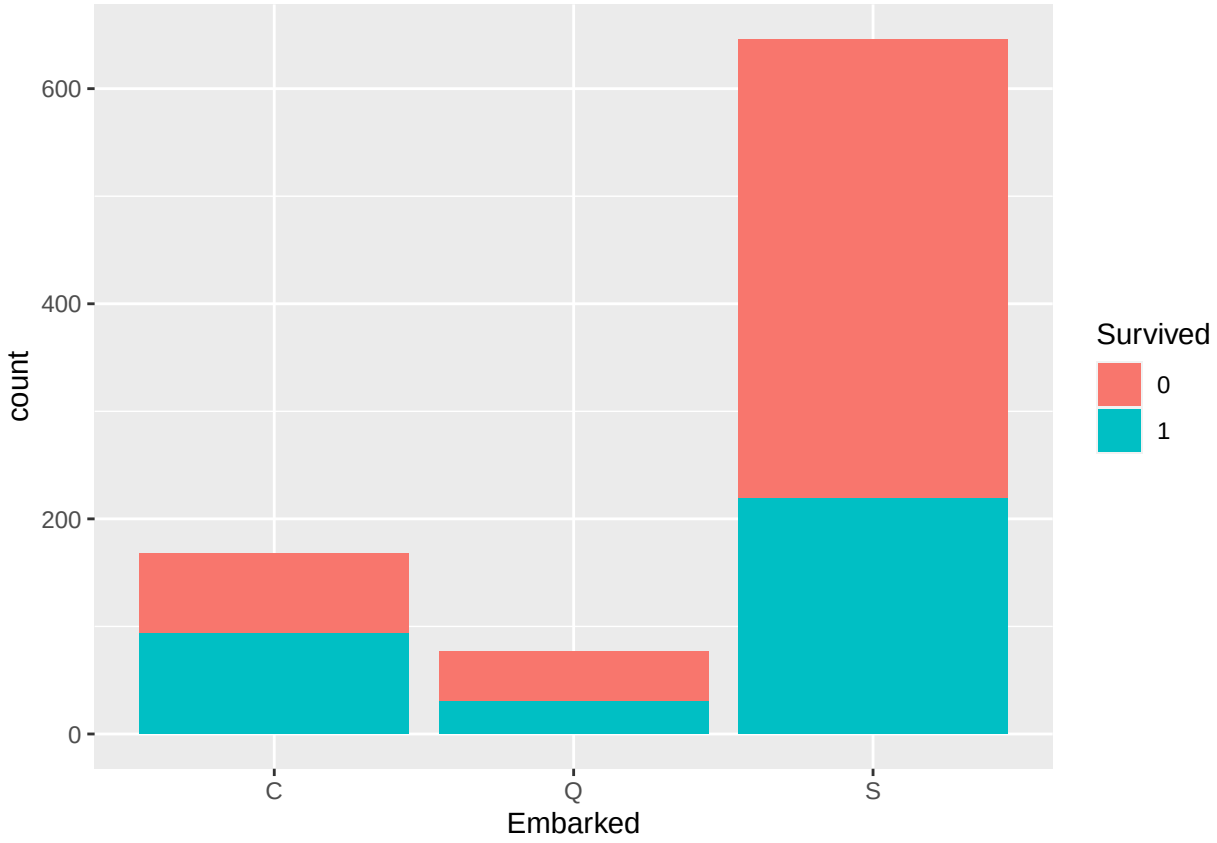
```
##
```

```
##  Pearson's Chi-squared test
##
## data:  data$Pclass and data$Survived
## X-squared = 102.89, df = 2, p-value < 2.2e-16
```

El valor de p-value es inferior a 0.05, es decir, rechazamos la hipótesis nula, y por tanto afirmamos que las variables no son independientes. Esto quiere decir que **el grado de supervivencia era distinto dependiendo de la clase en la que estuvieses**.

4.3.1.4 Survived y Embarked

Mostramos gráficamente la relación entre las dos variables:



Parece indicar que los embarcados en Southampton tenían menos posibilidades de sobrevivir (tal vez por el número de clase al que accedían). Utilizamos *Chi-Squared* para analizar la relación entre las variables **Embarked** y **Survived**.

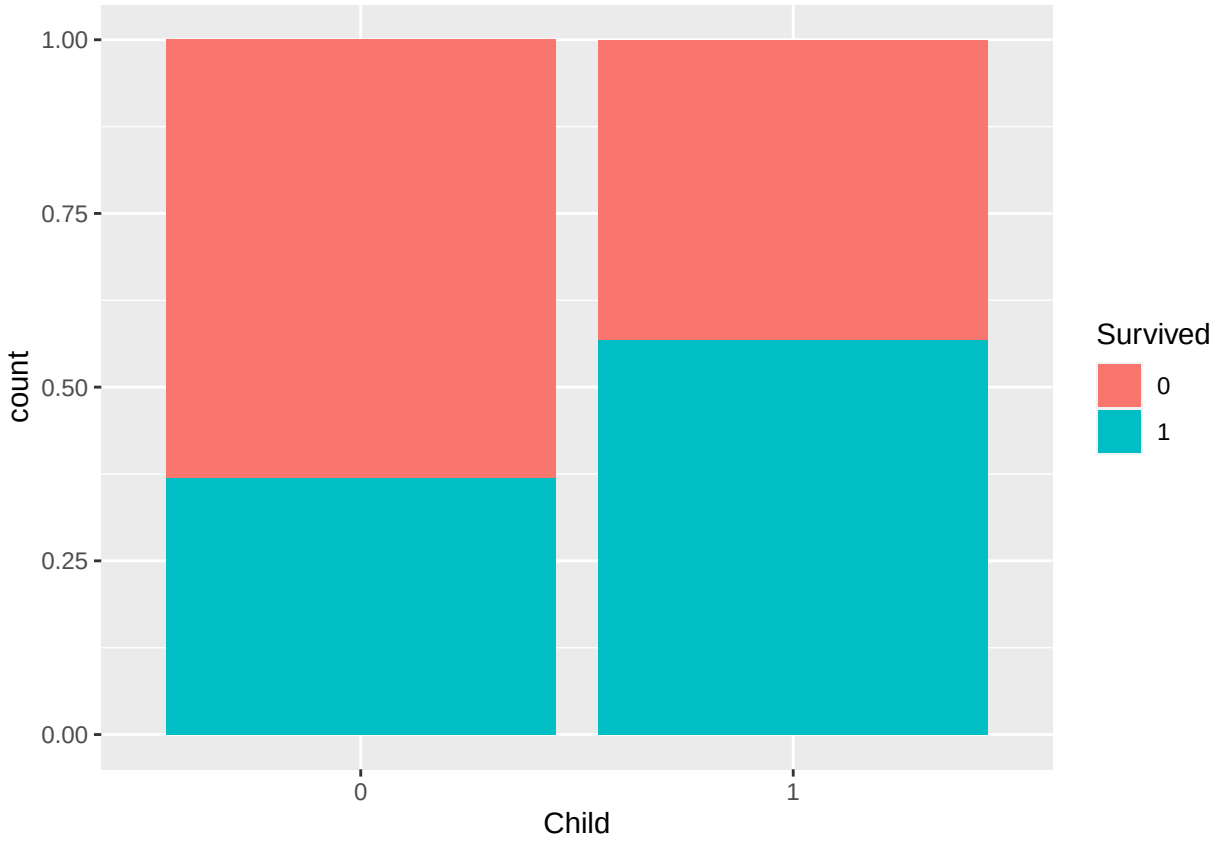
```
test_chisq <- chisq.test(data$Embarked, data$Survived)
test_chisq
```

```
##
##  Pearson's Chi-squared test
##
## data:  data$Embarked and data$Survived
## X-squared = 25.964, df = 2, p-value = 2.301e-06
```

Se rechaza la hipótesis nula, por lo tanto **las variables son dependientes**.

4.3.1.5 Survived y Child

Mostramos gráficamente la relación entre las dos variables, siendo **Child** una variable que corresponde a los niños de edad igual o inferior a 8 años:



Parece que hay más esperanza de supervivencia en ese caso. Aplicamos el *test de Fisher* para comprobar si tienen las mismas probabilidades de supervivencia que el resto de pasajeros.

```
fisher.test(table(data$Child, data$Survived))
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table(data$Child, data$Survived)
## p-value = 0.001657
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  1.315090 3.847717
## sample estimates:
## odds ratio
##   2.239271
```

Se rechaza la hipótesis nula, es decir que **las proporciones no coinciden**. Analizamos ahora si las probabilidades de supervivencia son mayores o menores. Para ello analizamos si la proporción de no-niños que no sobreviven es menor a la proporción de niños que no sobreviven:

```
fisher.test(table(data$Child, data$Survived), alternative = 'greater')
```

```
##
##  Fisher's Exact Test for Count Data
```

```
##
## data:  table(data$Child, data$Survived)
## p-value = 0.001204
## alternative hypothesis: true odds ratio is greater than 1
## 95 percent confidence interval:
##  1.424655      Inf
## sample estimates:
## odds ratio
##   2.239271
```

Se rechaza la hipótesis nula, y se acepta la alternativa, es decir que la **probabilidad de muerte de los no-niños es mayor que la de los niños**.

4.3.1.6 Survived y Age

Comprobamos ahora si la media de edad de las personas que murieron es similar con la media de edad que sobrevivieron, aplicando el test *t-Test*.

```
t.test(data$Age[which(data$Survived==0)], data$Age[which(data$Survived == 1)])
```

```
##
##  Welch Two Sample t-test
##
## data:  data$Age[which(data$Survived == 0)] and data$Age[which(data$Survived == 1)]
## t = 1.6097, df = 705.26, p-value = 0.1079
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3514748  3.5512763
## sample estimates:
## mean of x mean of y
##  30.06239  28.46249
```

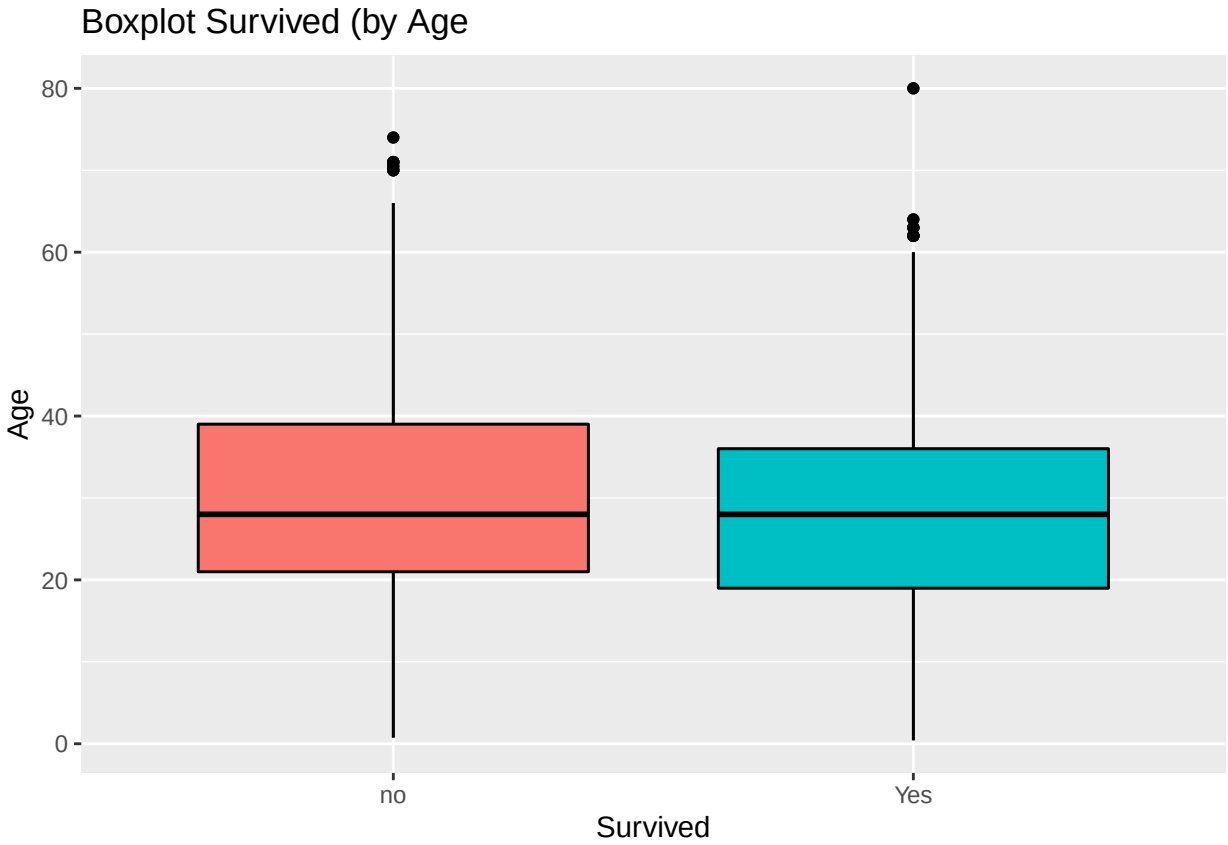
Se observa que el p-valor es inferior a 0.05, con lo que rechazamos la hipótesis nula (que las medias de edad son coincidentes para fallecidos y supervivientes), pero como ya sabíamos, la distribución de **Age** no sigue exactamente una normal, por lo que vamos a aplicar el *test no paramétrico U de Mann-Whitney*

```
wilcox.test(x = data$Age[which(data$Survived==0)],
            y = data$Age[which(data$Survived==1)],
            paired =FALSE
            )
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  data$Age[which(data$Survived == 0)] and data$Age[which(data$Survived == 1)]
## W = 97760, p-value = 0.2987
## alternative hypothesis: true location shift is not equal to 0
```

La prueba no-paramétrica nos devuelve un p-value por encima de 0.05 y por tanto no rechazamos la hipótesis nula, con lo que podemos decir que **la media de edad es similar entre los muertos y los supervivientes del Titanic**.

Gráficamente podemos comprobarlo mediante un Boxplot



Se puede apreciar que las distribuciones son muy parecidas.

4.3.2 Modelos

Vamos a aplicar diversos modelos predictivos con respecto a la variable **Survived**. Vamos a utilizar como variables explicativas las variables que traía por defecto el conjunto de datos original, menos las que hemos eliminado durante el proceso de transformación y limpieza de datos.

Creamos un conjunto de entrenamiento y un conjunto de test. Es decir,vamos a crear una partición, un 75% de datos para el entrenamiento, y el 25% restante para validar los modelos.

```
index_train <- createDataPartition(data$Survived, p=.75, list = FALSE)
data_train <-data[index_train,]
data_test <- data[-index_train, ]
```

Creamos una función que nos imprima correctamente las matrices de confusión a partir de un CrossTable.

```
# Función necesaria para imprimir la matriz de confusión a partir de un CrossTable
plot_matriz_confusion <- function(par_crosstable) {
  #Obtenemos la matriz de confusion en porcentaje
  datplotconfusion <- as.data.frame(par_crosstable[2]$prop.row)
  colnames(datplotconfusion) <- c('Observacion','Prediccion','Valor')
  datplotconfusion$Valor <- datplotconfusion$Valor*100
  #Ordenamos los factores de las clases
  datplotconfusion <- datplotconfusion %>% arrange(Observacion)
  clasesord <- sort(as.character(levels(datplotconfusion$Observacion)),decreasing = FALSE)
  datplotconfusion$Observacion <- factor(datplotconfusion$Observacion, levels = clasesord)
  #Ordenar las clases descendente (para el plot)
```

```
  clasesinv<-rev(as.character(unique(datplotconfusion$Observacion)))

  plot1 <- ggplot() +
    geom_tile(aes(x=Observacion, y=Prediccion,fill=Valor),
              data=datplotconfusion, color="black",size=0.1) +
    labs(x="Observación real (%)",y="Predicción (%)") +
    scale_y_discrete(limits=clasesinv)

  plot1 = plot1 +
    geom_text(aes(x=Observacion, y=Prediccion, label=sprintf("%.2f", Valor)),
              data=datplotconfusion, size=3, colour="black") +
    scale_fill_gradient(low="gray",high="red")

  #Obtenemos la matriz de confusion en numero de muestras
  datplotconfusion <- as.data.frame(par_crosstable[1]$t)
  colnames(datplotconfusion) <- c('Observacion','Prediccion','Valor')
  #Ordenamos los factores de las clases
  datplotconfusion <- datplotconfusion %>% arrange(Observacion)
  clasesord <- sort(as.character(levels(datplotconfusion$Observacion)),decreasing = FALSE)
  datplotconfusion$Observacion <- factor(datplotconfusion$Observacion, levels = clasesord)
  #Ordenar las clases descendente (para el plot)
  clasesinv<-rev(as.character(unique(datplotconfusion$Observacion)))
  plot2 <- ggplot() +
    geom_tile(aes(x=Observacion, y=Prediccion,fill=Valor),
              data=datplotconfusion, color="black",size=0.1) +
    labs(x="Observación real (nº)",y="Predicción (nº)") +
    scale_y_discrete(limits=clasesinv)

  plot2 = plot2 +
    geom_text(aes(x=Observacion, y=Prediccion, label=sprintf("%d", Valor)),
              data=datplotconfusion, size=3, colour="black") +
    scale_fill_gradient(low="gray",high="red")
  return(grid.arrange(plot1, plot2, ncol=2))
}
```

Creamos un dataframe en donde iremos guardando los distintos valores de precisión y errores de clasificación.

```
tabla.modelos <- data.frame('Modelo' = character(),
                            'Precisión' = numeric(),
                            'Errores' = numeric(),
                            stringsAsFactors = FALSE)

str(tabla.modelos)
```

```
## 'data.frame':    0 obs. of  3 variables:
##  $ Modelo   : chr
##  $ Precisión: num
##  $ Errores  : num
```

4.3.2.1 Modelo de regresión logística

Un modelo de regresión logística es un tipo de análisis de regresión que se utiliza para predecir el resultado de una variable categórica, en función de las variables independientes. Nuestra variable objetivo es **Survived**, que toma los valores de 0 o 1 (1 cuando el pasajero sobrevivió al accidente y 0 en caso contrario).

4.3.2.1.1 Primer modelo glm

Creamos un primer modelo glm1 con las variables "original": **Pclass**, **SibSp**, **Parch**, **Sex**, **Age**, **Fare** y **Embarked**

```
modelo_glm1 <- glm(formula=Survived~ Pclass+SibSp+Parch+Sex+Age+Fare+Embarked, data = data_train, family
summary(modelo_glm1)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + SibSp + Parch + Sex + Age +
##     Fare + Embarked, family = binomial(link = "logit"), data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4459  -0.6849  -0.4251   0.6291   2.3410
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.612678   0.519877   6.949 3.68e-12 ***
## Pclass2     -0.739694   0.334114  -2.214 0.026836 *
## Pclass3     -2.040088   0.334118  -6.106 1.02e-09 ***
## SibSp       -0.236448   0.121225  -1.950 0.051119 .
## Parch       -0.125800   0.133279  -0.944 0.345229
## Sexmale     -2.566677   0.227643 -11.275  < 2e-16 ***
## Age         -0.026784   0.007825  -3.423 0.000619 ***
## Fare         0.002248   0.002626   0.856 0.391945
## EmbarkedQ   -0.249211   0.433601  -0.575 0.565462
## EmbarkedS   -0.472015   0.269107  -1.754 0.079430 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 891.19  on 668  degrees of freedom
## Residual deviance: 613.33  on 659  degrees of freedom
## AIC: 633.33
##
## Number of Fisher Scoring iterations: 5
```

Se puede observar que tanto las variables **Fare** como **Parch** no aportan nada a la hora de predecir la variable **Survived**. En el caso de la variable Sex, al tener 2 valores se ha creado la variable **Sexmale**, es decir la parte correspondiente a hombres, mientras que la parte de mujeres forma parte del Intercept del modelo. Concretamente vemos que **Sexmale** es una variable significativa, pero con valor negativo (junto con **Pclass3**). Eso significa que esas variables contribuyen negativamente de cara a la supervivencia.

Ahora usamos el conjunto de datos de test para validar nuestro modelo. (Para este primer modelo mostramos el código como ejemplo)

```
predict_glm1 <- predict(object=modelo_glm1, newdata=data_test, type = "response")
r1 <- pROC::roc(response=data_test$Survived, predictor = predict_glm1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
threshold_r1 <- pROC::coords(r1, "best", ret = "threshold", transpose="FALSE")
predict_glm1 <- if_else(predict_glm1 < threshold_r1[1,1], 0, 1)
```

```
mat.confusion_glm1 <- table(data_test$Survived, predict_glm1)
mat.confusion_glm1
```

```
##    predict_glm1
##       0   1
##   0 124  13
##   1  19  66
```

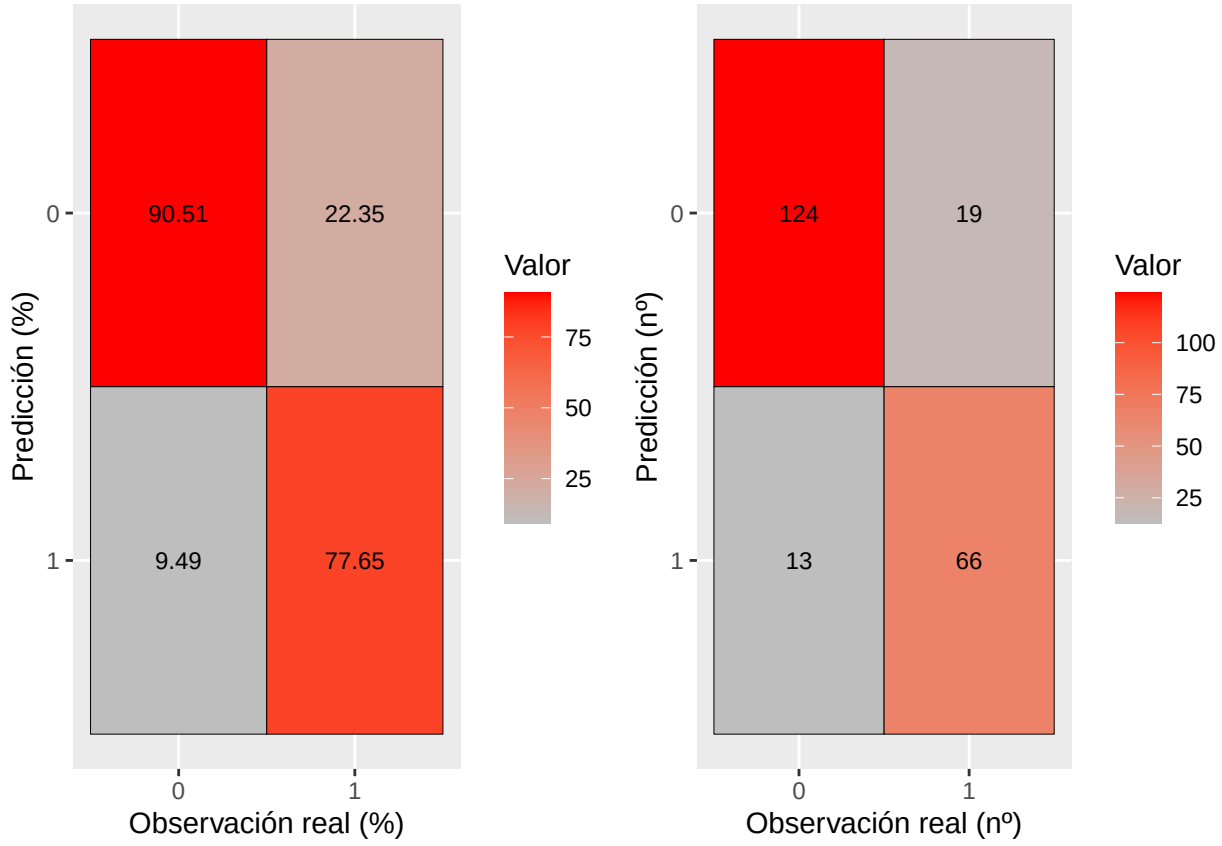
```
pct.correcto_glm1 <- 100 * sum(diag(mat.confusion_glm1)) / sum(mat.confusion_glm1)
print(sprintf("El %% de registros correctamente clasificados es: %.4f %%",
              pct.correcto_glm1))
```

```
## [1] "El % de registros correctamente clasificados es: 85.5856 %"
```

```
csstab_glm1 <- CrossTable(data_test$Survived, predict_glm1,
                          prop.chisq  = FALSE, prop.c = FALSE, prop.r =FALSE,
                          dnn = c('Reality', 'Prediction'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       124 |        13 |       137 |
##              |     0.559 |     0.059 |           |
## -------------|-----------|-----------|-----------|
##            1 |        19 |        66 |        85 |
##              |     0.086 |     0.297 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       143 |        79 |       222 |
## -------------|-----------|-----------|-----------|
##
##
```

```
plot_matriz_confusion(csstab_glm1)
```




```
tabla.modelos[nrow(tabla.modelos)+1,] = c("glm1", pct.correcto_glm1, sum(mat.confusion_glm1)-sum(diag(ma
```

4.3.2.1.2 Segundo modelo glm

Construimos un segundo modelo (glm2), similar al anterior pero esta vez partiendo de los resultados del modelo anterior, le vamos a quitar aquellas variables que vimos que no eran significativas para el modelo de predicción. Por lo tanto, tendremos un modelo2 cuyas variables a utilizar serán **Pclass**, **SibSp**, **Sex** y **Age**.

```
modelo_glm2 <- glm(formula=Survived~ Pclass+SibSp+Sex+Age, data = data_train, family = binomial(link =
summary(modelo_glm2)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + SibSp + Sex + Age, family = binomial(link = "logit"),
##     data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5245  -0.6737  -0.4310   0.6257   2.3881
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.471193   0.423769   8.191 2.59e-16 ***
## Pclass2     -1.017832   0.292607  -3.478 0.000504 ***
## Pclass3     -2.276724   0.272902  -8.343  < 2e-16 ***
## SibSp       -0.272582   0.113608  -2.399 0.016425 *
## Sexmale     -2.546428   0.218050 -11.678  < 2e-16 ***
```

```
## Age         -0.027118   0.007803  -3.476 0.000510 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 891.19  on 668  degrees of freedom
## Residual deviance: 618.65  on 663  degrees of freedom
## AIC: 630.65
##
## Number of Fisher Scoring iterations: 5
```

Se comprueba que todas las variables utilizadas son variables significativas para el modelo. Además, el modelo parece haber mejorado un poco en cuanto a la predicción:

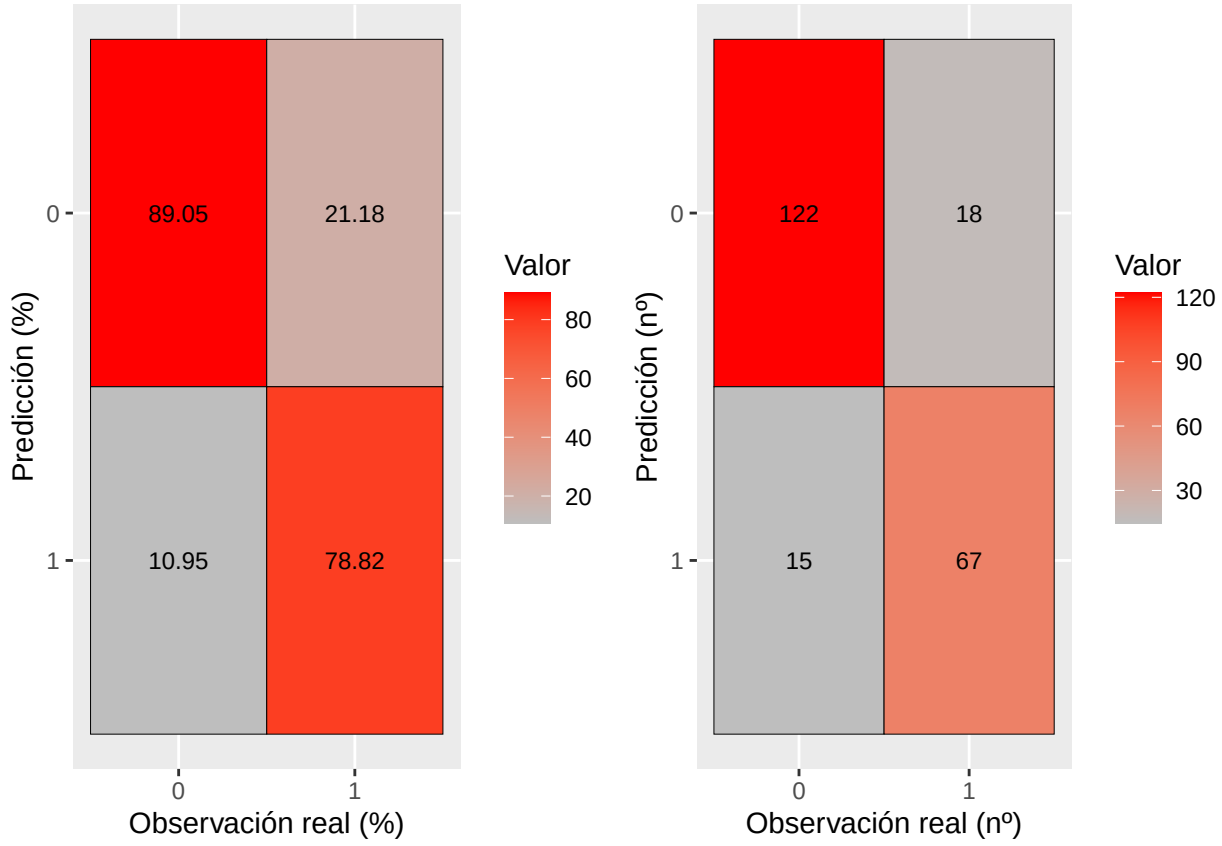
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
##    predict_glm2
##       0   1
##   0 122  15
##   1  18  67
```

```
## [1] "El % de registros correctamente clasificados es: 85.1351 %"
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       122 |        15 |       137 |
##              |     0.550 |     0.068 |           |
## -------------|-----------|-----------|-----------|
##            1 |        18 |        67 |        85 |
##              |     0.081 |     0.302 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       140 |        82 |       222 |
## -------------|-----------|-----------|-----------|
##
##
```



4.3.2.1.3 Tercer modelo glm

Construimos un tercer modelo (glm3), pero esta vez vamos a intentar utilizar alguna de las variables que hemos construido para ayudar a predecir la supervivencia. Para este caso vamos a utilizar las variables nuevas **Child** y también **FamilySize**. Además, vamos a utilizar tres variables originales: **Pclass**, **Sex** y **Age**.

```
modelo_glm3 <- glm(formula=Survived~ Sex+Pclass+Age+Child+FamilySize,, data = data_train, family = bino
summary(modelo_glm3)
```

```
##
## Call:
## glm(formula = Survived ~ Sex + Pclass + Age + Child + FamilySize,
##     family = binomial(link = "logit"), data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7131  -0.7012  -0.4318   0.5935   2.3558
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.295556   0.478909   6.881 5.93e-12 ***
## Sexmale     -2.685475   0.229168 -11.718  < 2e-16 ***
## Pclass2     -0.964359   0.293376  -3.287  0.00101 **
## Pclass3     -2.185323   0.274059  -7.974 1.54e-15 ***
## Age         -0.013287   0.008945  -1.485  0.13743
## Child1       1.315074   0.450494   2.919  0.00351 **
```

```
## FamilySize  -0.232295   0.075458  -3.078  0.00208 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 891.19  on 668  degrees of freedom
## Residual deviance: 609.73  on 662  degrees of freedom
## AIC: 623.73
##
## Number of Fisher Scoring iterations: 5
```

Al igual que pasaba en el modelo2, todas las variables utilizadas han resultado significativas, y además el AIC del modelo ha mejorado ligeramente. Además la capacidad predictiva del modelo ha aumentado un poco:

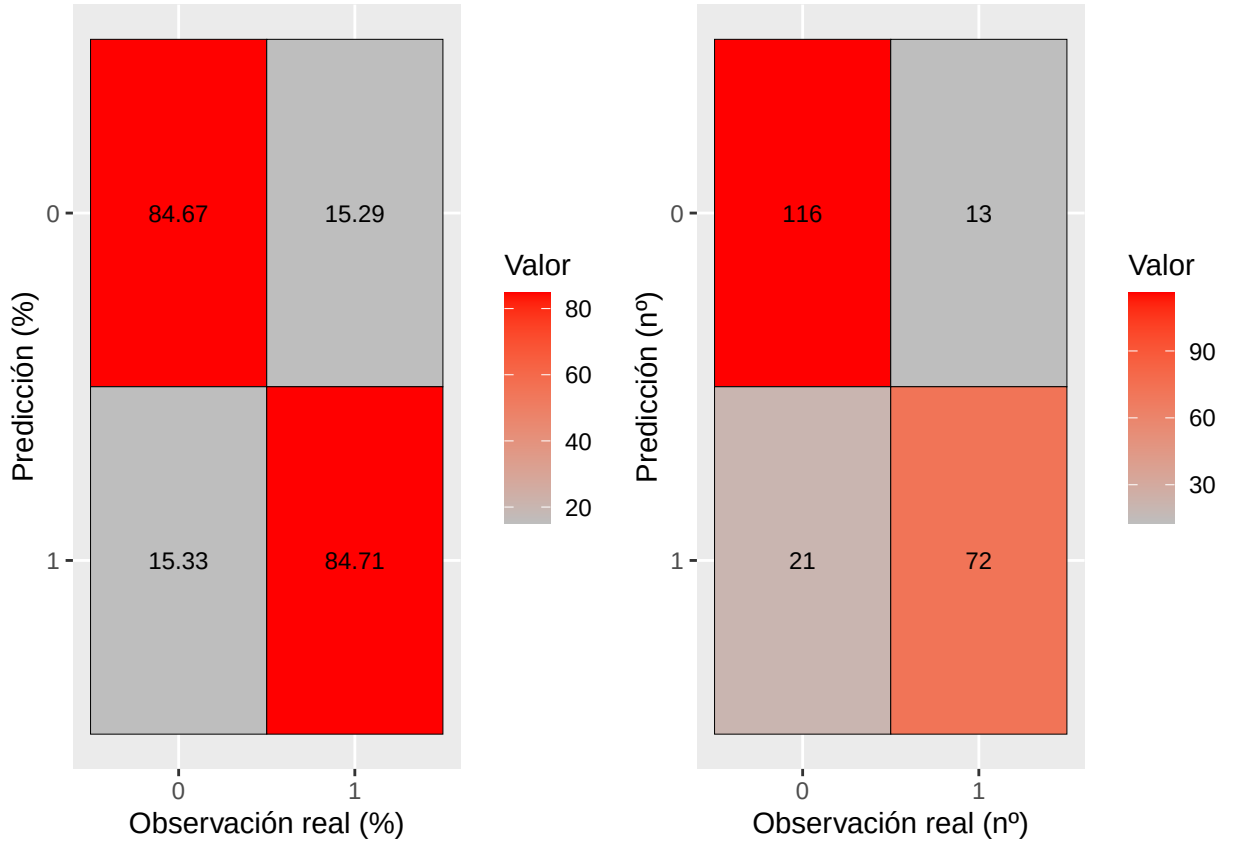
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
##    predict_glm3
##       0   1
##   0 116  21
##   1  13  72
```

```
## [1] "El % de registros correctamente clasificados es: 84.6847 %"
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       116 |        21 |       137 |
##              |     0.523 |     0.095 |           |
## -------------|-----------|-----------|-----------|
##            1 |        13 |        72 |        85 |
##              |     0.059 |     0.324 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       129 |        93 |       222 |
## -------------|-----------|-----------|-----------|
##
##
```



4.3.2.1.4 Resumen modelos regresion

Los diferentes modelos se han comportado de la siguiente forma:

```
knitr::kable(tabla.modelos)
```

Modelo	Precisión	Errores
glm1	85.5855855855856	32
glm2	85.1351351351351	33
glm3	84.6846846846847	34

4.3.2.2 Árboles de decisión

Los árboles de decisión son modelos muy sencillos de construir. Utilizaremos los "Recursive Partitioning and Regression Tree" que proporciona la libreria `rpart`

4.3.2.2.1 Primer árbol de decisión

Creamos un primer árbol con los parámetros básicos

```
tctrl <- caret::trainControl(method = "repeatedcv",
                             number=10, repeats = 3)
model_tree1 <- caret::train(Survived~Sex+FamilySize+Child+Age+Pclass,
                            data=data_train,
                            method="rpart",
                            trControl = tctrl)
summary(model_tree1)
```

```
## Call:
## (function (formula, data, weights, subset, na.action = na.rpart,
```

```
##     method, model = FALSE, x = FALSE, y = TRUE, parms, control,
##     cost, ...)
## {
##     Call <- match.call()
##     if (is.data.frame(model)) {
##         m <- model
##         model <- FALSE
##     }
##     else {
##         indx <- match(c("formula", "data", "weights", "subset"),
##             names(Call), nomatch = 0)
##         if (indx[1] == 0)
##             stop("a 'formula' argument is required")
##         temp <- Call[c(1, indx)]
##         temp$na.action <- na.action
##         temp[[1]] <- quote(stats::model.frame)
##         m <- eval.parent(temp)
##     }
##     Terms <- attr(m, "terms")
##     if (any(attr(Terms, "order") > 1))
##         stop("Trees cannot handle interaction terms")
##     Y <- model.response(m)
##     wt <- model.weights(m)
##     if (any(wt < 0))
##         stop("negative weights not allowed")
##     if (!length(wt))
##         wt <- rep(1, nrow(m))
##     offset <- model.offset(m)
##     X <- rpart.matrix(m)
##     nobs <- nrow(X)
##     nvar <- ncol(X)
##     if (missing(method)) {
##         method <- if (is.factor(Y) || is.character(Y))
##             "class"
##         else if (inherits(Y, "Surv"))
##             "exp"
##         else if (is.matrix(Y))
##             "poisson"
##         else "anova"
##     }
##     if (is.list(method)) {
##         mlist <- method
##         method <- "user"
##         init <- if (missing(parms))
##             mlist$init(Y, offset, wt = wt)
##         else mlist$init(Y, offset, parms, wt)
##         keep <- rpartcallback(mlist, nobs, init)
##         method.int <- 4
##         parms <- init$parms
##     }
##     else {
##         method.int <- pmatch(method, c("anova", "poisson", "class",
##             "exp"))
##         if (is.na(method.int))
```

```
##             stop("Invalid method")
##         method <- c("anova", "poisson", "class", "exp")[method.int]
##         if (method.int == 4)
##             method.int <- 2
##         init <- if (missing(parms))
##             get(paste("rpart", method, sep = "."), envir = environment())(Y,
##                 offset, , wt)
##         else get(paste("rpart", method, sep = "."), envir = environment())(Y,
##             offset, parms, wt)
##         ns <- asNamespace("rpart")
##         if (!is.null(init$print))
##             environment(init$print) <- ns
##         if (!is.null(init$summary))
##             environment(init$summary) <- ns
##         if (!is.null(init$text))
##             environment(init$text) <- ns
##     }
##     Y <- init$y
##     xlevels <- .getXlevels(Terms, m)
##     cats <- rep(0, ncol(X))
##     if (!is.null(xlevels))
##         cats[match(names(xlevels), colnames(X))] <- unlist(lapply(xlevels,
##             length))
##     extraArgs <- list(...)
##     if (length(extraArgs)) {
##         controlargs <- names(formals(rpart.control))
##         indx <- match(names(extraArgs), controlargs, nomatch = 0)
##         if (any(indx == 0))
##             stop(gettextf("Argument %s not matched", names(extraArgs)[indx ==
##                 0]), domain = NA)
##     }
##     controls <- rpart.control(...)
##     if (!missing(control))
##         controls[names(control)] <- control
##     xval <- controls$xval
##     if (is.null(xval) || (length(xval) == 1 && xval == 0) ||
##         method == "user") {
##         xgroups <- 0
##         xval <- 0
##     }
##     else if (length(xval) == 1) {
##         xgroups <- sample(rep(1:xval, length = nobs), nobs, replace = FALSE)
##     }
##     else if (length(xval) == nobs) {
##         xgroups <- xval
##         xval <- length(unique(xgroups))
##     }
##     else {
##         if (!is.null(attr(m, "na.action"))) {
##             temp <- as.integer(attr(m, "na.action"))
##             xval <- xval[-temp]
##             if (length(xval) == nobs) {
##                 xgroups <- xval
##                 xval <- length(unique(xgroups))
```

```
##             }
##             else stop("Wrong length for 'xval'")
##         }
##         else stop("Wrong length for 'xval'")
##     }
##     if (missing(cost))
##         cost <- rep(1, nvar)
##     else {
##         if (length(cost) != nvar)
##             stop("Cost vector is the wrong length")
##         if (any(cost <= 0))
##             stop("Cost vector must be positive")
##     }
##     tfun <- function(x) if (is.matrix(x))
##         rep(is.ordered(x), ncol(x))
##     else is.ordered(x)
##     labs <- sub("^`(.*)`$", "\\1", attr(Terms, "term.labels"))
##     isord <- unlist(lapply(m[labs], tfun))
##     storage.mode(X) <- "double"
##     storage.mode(wt) <- "double"
##     temp <- as.double(unlist(init$parms))
##     if (!length(temp))
##         temp <- 0
##     rpfit <- .Call(C_rpart, ncat = as.integer(cats * !isord),
##         method = as.integer(method.int), as.double(unlist(controls)),
##         temp, as.integer(xval), as.integer(xgroups), as.double(t(init$y)),
##         X, wt, as.integer(init$numy), as.double(cost))
##     nsplit <- nrow(rpfit$isplit)
##     ncat <- if (!is.null(rpfit$csplit))
##         nrow(rpfit$csplit)
##     else 0
##     if (nsplit == 0)
##         xval <- 0
##     numcp <- ncol(rpfit$cptable)
##     temp <- if (nrow(rpfit$cptable) == 3)
##         c("CP", "nsplit", "rel error")
##     else c("CP", "nsplit", "rel error", "xerror", "xstd")
##     dimnames(rpfit$cptable) <- list(temp, 1:numcp)
##     tname <- c("<leaf>", colnames(X))
##     splits <- matrix(c(rpfit$isplit[, 2:3], rpfit$dsplit), ncol = 5,
##         dimnames = list(tname[rpfit$isplit[, 1] + 1], c("count",
##             "ncat", "improve", "index", "adj")))
##     index <- rpfit$inode[, 2]
##     nadd <- sum(isord[rpfit$isplit[, 1]])
##     if (nadd > 0) {
##         newc <- matrix(0, nadd, max(cats))
##         cvar <- rpfit$isplit[, 1]
##         indx <- isord[cvar]
##         cdir <- splits[indx, 2]
##         ccut <- floor(splits[indx, 4])
##         splits[indx, 2] <- cats[cvar[indx]]
##         splits[indx, 4] <- ncat + 1:nadd
##         for (i in 1:nadd) {
##             newc[i, 1:(cats[(cvar[indx])[i]])] <- -as.integer(cdir[i])
```

```
##             newc[i, 1:ccut[i]] <- as.integer(cdir[i])
##         }
##         catmat <- if (ncat == 0)
##             newc
##         else {
##             cs <- rpfit$csplit
##             ncs <- ncol(cs)
##             ncc <- ncol(newc)
##             if (ncs < ncc)
##                 cs <- cbind(cs, matrix(0, nrow(cs), ncc - ncs))
##             rbind(cs, newc)
##         }
##         ncat <- ncat + nadd
##     }
##     else catmat <- rpfit$csplit
##     if (nsplit == 0) {
##         frame <- data.frame(row.names = 1, var = "<leaf>", n = rpfit$inode[,
##             5], wt = rpfit$dnode[, 3], dev = rpfit$dnode[, 1],
##             yval = rpfit$dnode[, 4], complexity = rpfit$dnode[,
##                 2], ncompete = 0, nsurrogate = 0)
##     }
##     else {
##         temp <- ifelse(index == 0, 1, index)
##         svar <- ifelse(index == 0, 0, rpfit$isplit[temp, 1])
##         frame <- data.frame(row.names = rpfit$inode[, 1], var = tname[svar +
##             1], n = rpfit$inode[, 5], wt = rpfit$dnode[, 3],
##             dev = rpfit$dnode[, 1], yval = rpfit$dnode[, 4],
##             complexity = rpfit$dnode[, 2], ncompete = pmax(0,
##                 rpfit$inode[, 3] - 1), nsurrogate = rpfit$inode[,
##                 4])
##     }
##     if (method.int == 3) {
##         numclass <- init$numresp - 2
##         nodeprob <- rpfit$dnode[, numclass + 5]/sum(wt)
##         temp <- pmax(1, init$counts)
##         temp <- rpfit$dnode[, 4 + (1:numclass)] %*% diag(init$parms$prior/temp)
##         yprob <- temp/rowSums(temp)
##         yval2 <- matrix(rpfit$dnode[, 4 + (0:numclass)], ncol = numclass +
##             1)
##         frame$yval2 <- cbind(yval2, yprob, nodeprob)
##     }
##     else if (init$numresp > 1)
##         frame$yval2 <- rpfit$dnode[, -(1:3), drop = FALSE]
##     if (is.null(init$summary))
##         stop("Initialization routine is missing the 'summary' function")
##     functions <- if (is.null(init$print))
##         list(summary = init$summary)
##     else list(summary = init$summary, print = init$print)
##     if (!is.null(init$text))
##         functions <- c(functions, list(text = init$text))
##     if (method == "user")
##         functions <- c(functions, mlist)
##     where <- rpfit$which
##     names(where) <- row.names(m)
```

```
##     ans <- list(frame = frame, where = where, call = Call, terms = Terms,
##         cptable = t(rpfit$cptable), method = method, parms = init$parms,
##         control = controls, functions = functions, numresp = init$numresp)
##     if (nsplit)
##         ans$splits = splits
##     if (ncat > 0)
##         ans$csplit <- catmat + 2
##     if (nsplit)
##         ans$variable.importance <- importance(ans)
##     if (model) {
##         ans$model <- m
##         if (missing(y))
##             y <- FALSE
##     }
##     if (y)
##         ans$y <- Y
##     if (x) {
##         ans$x <- X
##         ans$wt <- wt
##     }
##     ans$ordered <- isord
##     if (!is.null(attr(m, "na.action")))
##         ans$na.action <- attr(m, "na.action")
##     if (!is.null(xlevels))
##         attr(ans, "xlevels") <- xlevels
##     if (method == "class")
##         attr(ans, "ylevels") <- init$ylevels
##     class(ans) <- "rpart"
##     ans
## })(formula = .outcome ~ ., data = list(c(0, 0, 1, 1, 1, 1, 0,
## 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
## 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,
## 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
## 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
## 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
## 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1,
## 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
## 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1,
## 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
## 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
## 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
## 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1,
## 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
## 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
## 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
## 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
## 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
## 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0,
## 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
## 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
## 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
## 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
## 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
## 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
## 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
## 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
## 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1), c(1, 3, 3, 1, 1, 1, 2, 1, 2,
## 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 2, 1, 1, 1, 4,
## 4, 4, 1, 4, 4, 4, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 3, 2,
## 2, 1, 2, 3, 1, 1, 2, 1, 3, 1, 1, 1, 1, 1, 2, 2, 1, 3, 1, 1, 1,
## 1, 1, 1, 3, 1, 1, 1, 3, 3, 3, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 3,
## 2, 2, 1, 6, 6, 6, 1, 1, 2, 2, 1, 1, 1, 1, 1, 4, 4, 1, 1, 1, 2,
## 2, 1, 3, 3, 3, 4, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
## 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 2, 4, 1, 1, 1, 1, 1, 2, 2, 1, 1,
## 1, 2, 2, 1, 3, 1, 1, 1, 2, 2, 1, 1, 3, 3, 3, 1, 1, 1, 1, 1, 1,
## 2, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 2, 2, 3, 2, 3, 3, 3, 1, 2, 4,
## 4, 4, 1, 3, 3, 2, 1, 3, 3, 2, 1, 1, 1, 2, 3, 2, 2, 1, 3, 1, 1,
## 1, 1, 1, 2, 1, 1, 3, 1, 1, 1, 1, 1, 2, 2, 3, 4, 5, 6, 3, 3, 1,
## 2, 3, 3, 1, 1, 1, 2, 3, 3, 2, 4, 7, 6, 6, 6, 1, 2, 2, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 3, 3, 1, 1, 1, 3, 3,
## 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 7, 7, 7,
## 1, 3, 3, 1, 7, 7, 7, 7, 7, 7, 7, 2, 1, 6, 6, 6, 6, 6, 1, 1, 1,
## 1, 1, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5,
## 5, 5, 5, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 3, 1, 1, 1, 2, 2, 1,
## 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1, 1, 1, 1, 3, 3,
## 1, 2, 2, 1, 1, 1, 1, 1, 1, 3, 3, 2, 2, 1, 1, 2, 2, 3, 3, 2, 2,
## 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 6,
## 6, 6, 6, 1, 2, 1, 2, 1, 5, 5, 5, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 4, 4, 1, 1, 3, 3, 1, 1, 1, 3, 3,
## 1, 3, 3, 1, 1, 4, 4, 4, 3, 8, 8, 8, 8, 8, 8, 11, 11, 11, 11,
## 11, 2, 1, 3, 3, 3, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1,
## 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 2, 1, 5, 1, 3, 2, 1, 1, 1,
## 1, 2, 3, 2, 1, 1, 1, 2, 2, 2, 1, 2, 1, 3, 3, 3, 1, 1, 3, 3, 1,
## 1, 1, 1, 2, 1, 4, 4, 4, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1,
## 1, 1, 1, 1, 4, 5, 5, 5, 5, 1, 2, 1), c(0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
## 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
## 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
## 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
## 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), c(30, 39, 52, 47, 35, 28, 60,
## 47, 44, 16, 35, 26, 39, 2, 19, 16, 38, 28, 45.5, 45, 16, 28,
## 29, 27, 45, 65, 19, 62, 38, 14, 11, 36, 56, 25, 0.92, 2, 45,
## 52, 55, 35, 45, 35, 37, 27, 33, 27, 62, 29, 27, 37, 48, 25, 24,
## 56, 53, 58, 64, 25, 34, 52, 30, 36, 56, 32, 21, 63, 39, 65, 60,
## 31, 1, 32, 28, 26, 32, 40, 41, 21, 28, 39, 17, 49, 58, 49, 30,
## 49, 48, 31, 17, 56, 36, 26, 44, 35, 38, 23, 19, 24, 23, 48, 47,
## 48, 28, 27, 51, 28, 57, 27, 48, 24, 18, 45, 19, 36, 29, 36, 3,
## 2, 36.5, 4, 50, 28, 34, 18, 23, 48, 32, 25, 30, 18, 42, 25, 47,
## 28, 38, 14, 32.5, 25, 42, 28, 18, 38, 35, 28, 29, 43, 24, 34,
## 31, 23, 27, 25, 34, 24, 43, 34, 55, 6, 28, 52, 0.83, 28, 24,
## 50, 13, 41, 30, 30, 0.67, 24, 18, 30, 19, 39, 22, 40, 32, 0.42,
## 41, 17, 16, 45.5, 21, 27, 33, 17, 20, 24, 12, 14, 20, 27, 20,
## 10, 36, 20, 14.5, 5, 24, 0.75, 15, 25, 26, 44, 0.75, 9, 49, 26,
## 34.5, 30, 27, 61, 17, 18, 45, 23.5, 15, 11, 26, 36, 80, 32.5,
## 21, 35, 39, 34, 30, 54, 19, 23, 36, 34, 32, 4, 23, 54, 24, 3,
## 0.83, 18, 29, 60, 40, 23, 34, 18, 18, 37, 28, 32, 33, 17, 7,
## 41, 1, 39, 2, 34, 43, 21, 23, 40, 45, 29, 30, 17, 42, 17, 20,
## 20, 21, 27, 26, 32, 19, 22, 1, 15, 26, 29, 19, 26, 25, 16, 31,
## 50, 4, 65, 44, 26, 27, 24.5, 21, 29, 61, 36, 18, 16, 47, 20,
## 28, 36, 30, 21, 16, 29, 33, 24, 28, 29, 2, 74, 45, 33, 51, 19,
## 25, 45, 16, 25, 38, 3, 5, 9, 18, 34, 28, 22, 11, 2, 39, 39, 9,
## 4, 6, 25, 18, 40, 45, 4, 10, 9, 27, 26, 28, 20, 26, 4, 27, 1,
## 51, 42, 25, 19, 7, 25, 23, 19, 22, 25, 24, 36, 19, 35, 20, 35,
## 16, 47, 26, 20, 21, 19, 24, 9, 24, 25, 18, 33, 32, 28, 30, 36,
## 26, 40, 22, 44, 2, 29, 3, 8, 36, 18, 21, 24, 18, 27, 19, 17,
## 20, 22, 22, 18, 14, 31, 32, 31, 23, 16, 21, 33, 34, 44, 31, 30.5,
## 59, 34, 44, 35, 5, 21, 20, 32, 40, 30.5, 35, 19, 18, 33, 25,
## 24.5, 39, 62, 40.5, 43, 28, 25, 32, 40.5, 31, 45, 54, 52, 61,
## 47, 35, 45, 18, 41, 26, 19, 70.5, 38, 19, 38, 32, 24, 65, 31,
## 21, 21, 44, 25, 35, 5, 25, 16, 21, 27, 18, 29, 7, 4, 8, 2, 24,
## 29, 32, 6, 15, 18, 30, 9, 37, 18, 42, 28.5, 21, 24, 2, 19, 64,
## 33, 23, 16, 20, 33, 29, 22, 45, 32, 26, 55.5, 24, 42, 22, 22,
## 28.5, 26, 43, 18, 50, 47, 20, 30, 47, 40.5, 8, 24, 28, 43, 30,
## 36, 31, 26, 1, 66, 70, 35, 16, 29, 24, 21, 31, 31, 32, 8, 19,
## 40, 34, 36, 33, 5, 3, 11, 9, 14, 1, 43, 16, 20, 48, 19, 20, 25,
## 31, 35, 7, 43, 45, 50, 17, 19, 25, 36, 30, 40, 36, 35, 42, 24,
```

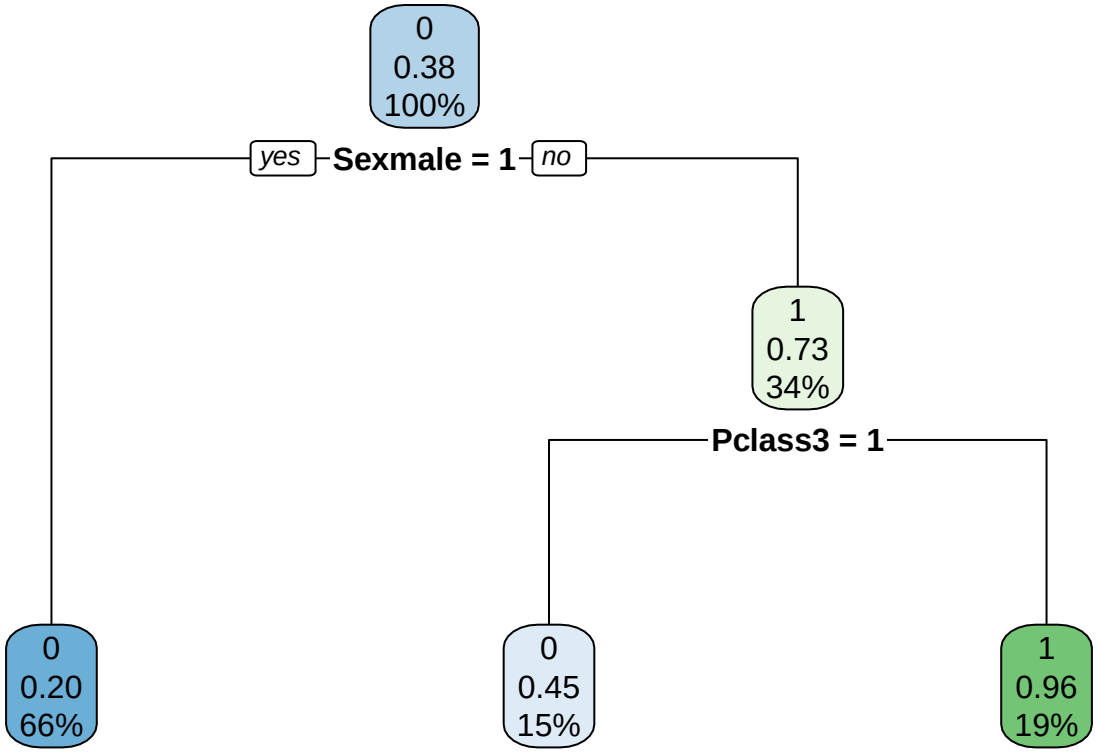
```
## 49, 49, 30, 24, 48, 58, 40, 38, 22, 16, 24, 51, 38, 50, 28, 35,
## 32, 21, 44, 50, 32, 6, 71, 35, 35, 36, 39, 18, 38, 42, 36, 18,
## 17, 23, 36, 50, 22, 24, 1, 31, 16, 45.5, 32, 21, 21, 18, 19,
## 17, 37, 23, 22, 25, 3, 16, 27, 36, 29, 17, 21, 38, 20, 35, 24,
## 25, 32, 22, 41, 21, 25, 16, 30, 44, 35, 17, 22, 21, 20, 32, 31,
## 39, 32, 21, 25, 28, 24, 26, 27, 23, 19, 50, 29, 18, 9, 21, 48,
## 16, 40, 46, 30), c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
## 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
## 0, 0, 0, 1), c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
## 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
## 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
## 0, 0), c(2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1,
## 2, 1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2,
## 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
## 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2,
## 2, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2,
## 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 1, 1, 1, 2,
## 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 2,
## 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1,
## 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1,
## 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1,
## 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2,
## 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
## 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2,
## 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1,
## 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,
## 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1,
## 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2,
## 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
## 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1,
## 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1,
## 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1,
## 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 1, 1,
## 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 2,
## 2, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1
## )), control = list(20, 7, 0, 4, 5, 2, 0, 30, 0))
##   n= 669
##
##           CP nsplit rel error
## 1 0.41245136      0 1.0000000
## 2 0.03891051      1 0.5875486
```

```
## 3 0.02529183      2 0.5486381
##
## Variable importance
##    Sexmale    Pclass3    Pclass2 FamilySize        Age     Child1
##         59         21          8          7          5          1
##
## Node number 1: 669 observations,    complexity param=0.4124514
##   predicted class=0  expected loss=0.3841555  P(node) =1
##     class counts:   412   257
##    probabilities: 0.616 0.384
##   left son=2 (441 obs) right son=3 (228 obs)
##   Primary splits:
##       Sexmale    < 0.5  to the right, improve=83.919140, (0 missing)
##       Pclass3    < 0.5  to the right, improve=39.775260, (0 missing)
##       FamilySize < 1.5  to the left,  improve=11.833780, (0 missing)
##       Pclass2    < 0.5  to the left,  improve= 5.533223, (0 missing)
##       Age        < 6.5  to the right, improve= 4.495378, (0 missing)
##   Surrogate splits:
##       FamilySize < 3.5  to the left,  agree=0.677, adj=0.053, (0 split)
##
## Node number 2: 441 observations
##   predicted class=0  expected loss=0.2040816  P(node) =0.6591928
##     class counts:   351    90
##    probabilities: 0.796 0.204
##
## Node number 3: 228 observations,    complexity param=0.03891051
##   predicted class=1  expected loss=0.2675439  P(node) =0.3408072
##     class counts:    61   167
##    probabilities: 0.268 0.732
##   left son=6 (102 obs) right son=7 (126 obs)
##   Primary splits:
##       Pclass3    < 0.5  to the right, improve=29.24667000, (0 missing)
##       FamilySize < 4.5  to the right, improve=11.65263000, (0 missing)
##       Pclass2    < 0.5  to the left,  improve= 7.94215600, (0 missing)
##       Age        < 18.5 to the left,  improve= 3.54994300, (0 missing)
##       Child1     < 0.5  to the right, improve= 0.09649123, (0 missing)
##   Surrogate splits:
##       Pclass2    < 0.5  to the left,  agree=0.715, adj=0.363, (0 split)
##       Age        < 26.5 to the left,  agree=0.658, adj=0.235, (0 split)
##       FamilySize < 4.5  to the right, agree=0.627, adj=0.167, (0 split)
##       Child1     < 0.5  to the right, agree=0.575, adj=0.049, (0 split)
##
## Node number 6: 102 observations
##   predicted class=0  expected loss=0.4509804  P(node) =0.1524664
##     class counts:    56    46
##    probabilities: 0.549 0.451
##
## Node number 7: 126 observations
##   predicted class=1  expected loss=0.03968254  P(node) =0.1883408
##     class counts:     5   121
##    probabilities: 0.040 0.960
```

Dibujamos el árbol

```
rpart.plot(model_tree1$finalModel)
```



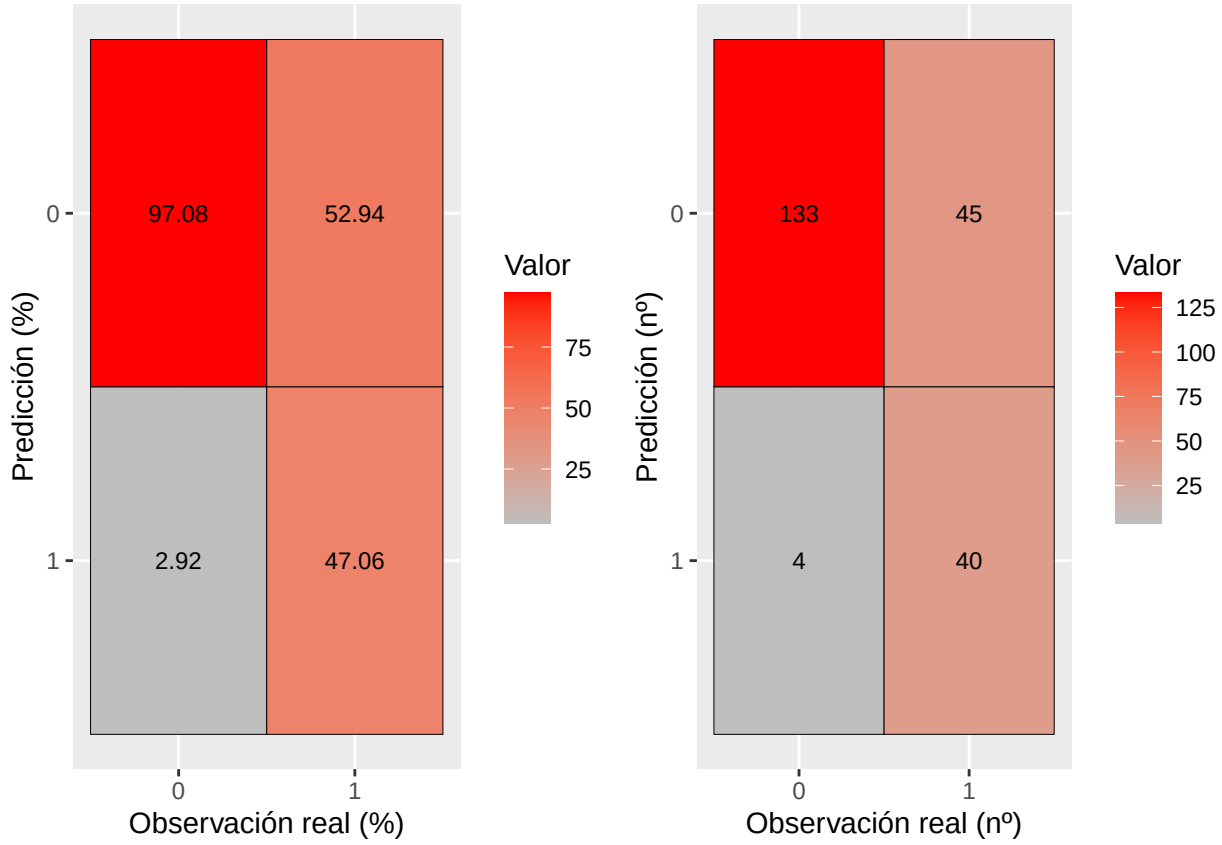
Realizamos la predicción con el modelo construido con rpart y el dataset test

```
##    predict_tree1
##       0   1
##   0 133   4
##   1  45  40
```

```
## [1] "El % de registros correctamente clasificados es: 77.9279 %"
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       133 |         4 |       137 |
```

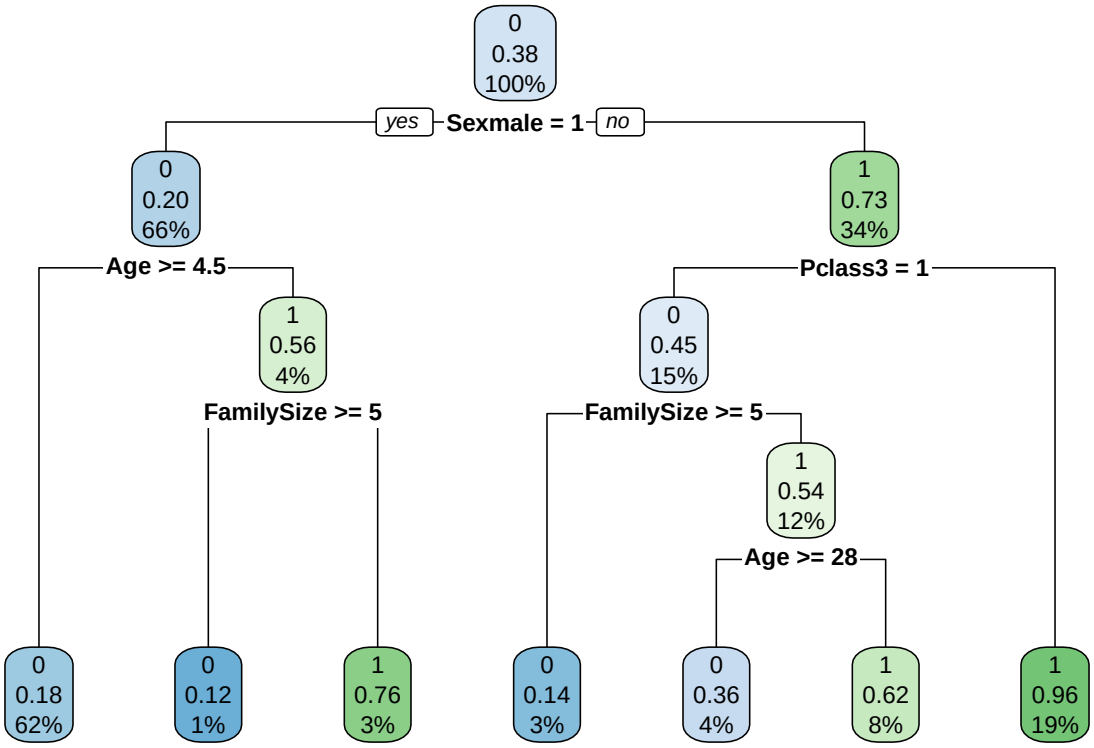
```
##              |     0.599 |     0.018 |           |
## -------------|-----------|-----------|-----------|
##            1 |        45 |        40 |        85 |
##              |     0.203 |     0.180 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       178 |        44 |       222 |
## -------------|-----------|-----------|-----------|
##
##
```



4.3.2.2.2 Segundo árbol de decisión

Manteniendo las mismas variables predictivas, pero cambiando la profundidad máxima del árbol (maxdepth) se pueden mejorar los resultados obtenidos:

```
tctrl <- caret::trainControl(method = "repeatedcv",
                             number=10, repeats = 3)
tgrid <- data.frame(maxdepth = seq(2,10,1))

model_tree2<- train(Survived ~ Sex+FamilySize+Child+Age+Pclass,
                    data=data_train,
                    trControl=tctrl,
                    tuneGrid=tgrid,
                    method="rpart2")
```




```
## Call:
## (function (formula, data, weights, subset, na.action = na.rpart,
##     method, model = FALSE, x = FALSE, y = TRUE, parms, control,
##     cost, ...)
## {
##     Call <- match.call()
##     if (is.data.frame(model)) {
##         m <- model
##         model <- FALSE
##     }
##     else {
##         indx <- match(c("formula", "data", "weights", "subset"),
##             names(Call), nomatch = 0)
##         if (indx[1] == 0)
##             stop("a 'formula' argument is required")
##         temp <- Call[c(1, indx)]
##         temp$na.action <- na.action
##         temp[[1]] <- quote(stats::model.frame)
##         m <- eval.parent(temp)
##     }
##     Terms <- attr(m, "terms")
##     if (any(attr(Terms, "order") > 1))
##         stop("Trees cannot handle interaction terms")
##     Y <- model.response(m)
##     wt <- model.weights(m)
##     if (any(wt < 0))
```

```
##         stop("negative weights not allowed")
##     if (!length(wt))
##         wt <- rep(1, nrow(m))
##     offset <- model.offset(m)
##     X <- rpart.matrix(m)
##     nobs <- nrow(X)
##     nvar <- ncol(X)
##     if (missing(method)) {
##         method <- if (is.factor(Y) || is.character(Y))
##             "class"
##         else if (inherits(Y, "Surv"))
##             "exp"
##         else if (is.matrix(Y))
##             "poisson"
##         else "anova"
##     }
##     if (is.list(method)) {
##         mlist <- method
##         method <- "user"
##         init <- if (missing(parms))
##             mlist$init(Y, offset, wt = wt)
##         else mlist$init(Y, offset, parms, wt)
##         keep <- rpartcallback(mlist, nobs, init)
##         method.int <- 4
##         parms <- init$parms
##     }
##     else {
##         method.int <- pmatch(method, c("anova", "poisson", "class",
##             "exp"))
##         if (is.na(method.int))
##             stop("Invalid method")
##         method <- c("anova", "poisson", "class", "exp")[method.int]
##         if (method.int == 4)
##             method.int <- 2
##         init <- if (missing(parms))
##             get(paste("rpart", method, sep = "."), envir = environment())(Y,
##                 offset, , wt)
##         else get(paste("rpart", method, sep = "."), envir = environment())(Y,
##             offset, parms, wt)
##         ns <- asNamespace("rpart")
##         if (!is.null(init$print))
##             environment(init$print) <- ns
##         if (!is.null(init$summary))
##             environment(init$summary) <- ns
##         if (!is.null(init$text))
##             environment(init$text) <- ns
##     }
##     Y <- init$y
##     xlevels <- .getXlevels(Terms, m)
##     cats <- rep(0, ncol(X))
##     if (!is.null(xlevels))
##         cats[match(names(xlevels), colnames(X))] <- unlist(lapply(xlevels,
##             length))
##     extraArgs <- list(...)
```

```
##     if (length(extraArgs)) {
##         controlargs <- names(formals(rpart.control))
##         indx <- match(names(extraArgs), controlargs, nomatch = 0)
##         if (any(indx == 0))
##             stop(gettextf("Argument %s not matched", names(extraArgs)[indx ==
##                 0]), domain = NA)
##     }
##     controls <- rpart.control(...)
##     if (!missing(control))
##         controls[names(control)] <- control
##     xval <- controls$xval
##     if (is.null(xval) || (length(xval) == 1 && xval == 0) ||
##         method == "user") {
##         xgroups <- 0
##         xval <- 0
##     }
##     else if (length(xval) == 1) {
##         xgroups <- sample(rep(1:xval, length = nobs), nobs, replace = FALSE)
##     }
##     else if (length(xval) == nobs) {
##         xgroups <- xval
##         xval <- length(unique(xgroups))
##     }
##     else {
##         if (!is.null(attr(m, "na.action"))) {
##             temp <- as.integer(attr(m, "na.action"))
##             xval <- xval[-temp]
##             if (length(xval) == nobs) {
##                 xgroups <- xval
##                 xval <- length(unique(xgroups))
##             }
##             else stop("Wrong length for 'xval'")
##         }
##         else stop("Wrong length for 'xval'")
##     }
##     if (missing(cost))
##         cost <- rep(1, nvar)
##     else {
##         if (length(cost) != nvar)
##             stop("Cost vector is the wrong length")
##         if (any(cost <= 0))
##             stop("Cost vector must be positive")
##     }
##     tfun <- function(x) if (is.matrix(x))
##         rep(is.ordered(x), ncol(x))
##     else is.ordered(x)
##     labs <- sub("^`(.*)`$", "\\1", attr(Terms, "term.labels"))
##     isord <- unlist(lapply(m[labs], tfun))
##     storage.mode(X) <- "double"
##     storage.mode(wt) <- "double"
##     temp <- as.double(unlist(init$parms))
##     if (!length(temp))
##         temp <- 0
##     rpfit <- .Call(C_rpart, ncat = as.integer(cats * !isord),
```

```
##         method = as.integer(method.int), as.double(unlist(controls)),
##         temp, as.integer(xval), as.integer(xgroups), as.double(t(init$y)),
##         X, wt, as.integer(init$numy), as.double(cost))
##     nsplit <- nrow(rpfit$isplit)
##     ncat <- if (!is.null(rpfit$csplit))
##         nrow(rpfit$csplit)
##     else 0
##     if (nsplit == 0)
##         xval <- 0
##     numcp <- ncol(rpfit$cptable)
##     temp <- if (nrow(rpfit$cptable) == 3)
##         c("CP", "nsplit", "rel error")
##     else c("CP", "nsplit", "rel error", "xerror", "xstd")
##     dimnames(rpfit$cptable) <- list(temp, 1:numcp)
##     tname <- c("<leaf>", colnames(X))
##     splits <- matrix(c(rpfit$isplit[, 2:3], rpfit$dsplit), ncol = 5,
##         dimnames = list(tname[rpfit$isplit[, 1] + 1], c("count",
##             "ncat", "improve", "index", "adj")))
##     index <- rpfit$inode[, 2]
##     nadd <- sum(isord[rpfit$isplit[, 1]])
##     if (nadd > 0) {
##         newc <- matrix(0, nadd, max(cats))
##         cvar <- rpfit$isplit[, 1]
##         indx <- isord[cvar]
##         cdir <- splits[indx, 2]
##         ccut <- floor(splits[indx, 4])
##         splits[indx, 2] <- cats[cvar[indx]]
##         splits[indx, 4] <- ncat + 1:nadd
##         for (i in 1:nadd) {
##             newc[i, 1:(cats[(cvar[indx])[i]])] <- -as.integer(cdir[i])
##             newc[i, 1:ccut[i]] <- as.integer(cdir[i])
##         }
##         catmat <- if (ncat == 0)
##             newc
##         else {
##             cs <- rpfit$csplit
##             ncs <- ncol(cs)
##             ncc <- ncol(newc)
##             if (ncs < ncc)
##                 cs <- cbind(cs, matrix(0, nrow(cs), ncc - ncs))
##             rbind(cs, newc)
##         }
##         ncat <- ncat + nadd
##     }
##     else catmat <- rpfit$csplit
##     if (nsplit == 0) {
##         frame <- data.frame(row.names = 1, var = "<leaf>", n = rpfit$inode[,
##             5], wt = rpfit$dnode[, 3], dev = rpfit$dnode[, 1],
##             yval = rpfit$dnode[, 4], complexity = rpfit$dnode[,
##                 2], ncompete = 0, nsurrogate = 0)
##     }
##     else {
##         temp <- ifelse(index == 0, 1, index)
##         svar <- ifelse(index == 0, 0, rpfit$isplit[temp, 1])
```

```
##         frame <- data.frame(row.names = rpfit$inode[, 1], var = tname[svar +
##             1], n = rpfit$inode[, 5], wt = rpfit$dnode[, 3],
##             dev = rpfit$dnode[, 1], yval = rpfit$dnode[, 4],
##             complexity = rpfit$dnode[, 2], ncompete = pmax(0,
##                 rpfit$inode[, 3] - 1), nsurrogate = rpfit$inode[,
##                 4])
##     }
##     if (method.int == 3) {
##         numclass <- init$numresp - 2
##         nodeprob <- rpfit$dnode[, numclass + 5]/sum(wt)
##         temp <- pmax(1, init$counts)
##         temp <- rpfit$dnode[, 4 + (1:numclass)] %*% diag(init$parms$prior/temp)
##         yprob <- temp/rowSums(temp)
##         yval2 <- matrix(rpfit$dnode[, 4 + (0:numclass)], ncol = numclass +
##             1)
##         frame$yval2 <- cbind(yval2, yprob, nodeprob)
##     }
##     else if (init$numresp > 1)
##         frame$yval2 <- rpfit$dnode[, -(1:3), drop = FALSE]
##     if (is.null(init$summary))
##         stop("Initialization routine is missing the 'summary' function")
##     functions <- if (is.null(init$print))
##         list(summary = init$summary)
##     else list(summary = init$summary, print = init$print)
##     if (!is.null(init$text))
##         functions <- c(functions, list(text = init$text))
##     if (method == "user")
##         functions <- c(functions, mlist)
##     where <- rpfit$which
##     names(where) <- row.names(m)
##     ans <- list(frame = frame, where = where, call = Call, terms = Terms,
##         cptable = t(rpfit$cptable), method = method, parms = init$parms,
##         control = controls, functions = functions, numresp = init$numresp)
##     if (nsplit)
##         ans$splits = splits
##     if (ncat > 0)
##         ans$csplit <- catmat + 2
##     if (nsplit)
##         ans$variable.importance <- importance(ans)
##     if (model) {
##         ans$model <- m
##         if (missing(y))
##             y <- FALSE
##     }
##     if (y)
##         ans$y <- Y
##     if (x) {
##         ans$x <- X
##         ans$wt <- wt
##     }
##     ans$ordered <- isord
##     if (!is.null(attr(m, "na.action")))
##         ans$na.action <- attr(m, "na.action")
##     if (!is.null(xlevels))
```

```
##         attr(ans, "xlevels") <- xlevels
##     if (method == "class")
##         attr(ans, "ylevels") <- init$ylevels
##     class(ans) <- "rpart"
##     ans
## })(formula = .outcome ~ ., data = list(c(0, 0, 1, 1, 1, 1, 0,
## 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
## 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,
## 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
## 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
## 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
## 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1,
## 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
## 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1,
## 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
## 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
## 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
## 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1,
## 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
## 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
## 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
## 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
## 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
## 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0,
## 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
## 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
## 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
## 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
## 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
## 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
## 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
## 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1), c(1, 3, 3, 1, 1, 1, 2, 1, 2,
## 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 2, 1, 1, 1, 4,
## 4, 4, 1, 4, 4, 4, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 3, 2,
## 2, 1, 2, 3, 1, 1, 2, 1, 3, 1, 1, 1, 1, 1, 2, 2, 1, 3, 1, 1, 1,
## 1, 1, 1, 3, 1, 1, 1, 3, 3, 3, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 3,
## 2, 2, 1, 6, 6, 6, 1, 1, 2, 2, 1, 1, 1, 1, 1, 4, 4, 1, 1, 1, 2,
## 2, 1, 3, 3, 3, 4, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
## 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 2, 4, 1, 1, 1, 1, 1, 2, 2, 1, 1,
## 1, 2, 2, 1, 3, 1, 1, 1, 2, 2, 1, 1, 3, 3, 3, 1, 1, 1, 1, 1, 1,
## 2, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 2, 2, 3, 2, 3, 3, 3, 1, 2, 4,
## 4, 4, 1, 3, 3, 2, 1, 3, 3, 2, 1, 1, 1, 2, 3, 2, 2, 1, 3, 1, 1,
## 1, 1, 1, 2, 1, 1, 3, 1, 1, 1, 1, 1, 2, 2, 3, 4, 5, 6, 3, 3, 1,
## 2, 3, 3, 1, 1, 1, 2, 3, 3, 2, 4, 7, 6, 6, 6, 1, 2, 2, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 3, 3, 1, 1, 1, 3, 3,
## 1, 1, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 7, 7, 7, 7,
## 1, 3, 3, 1, 7, 7, 7, 7, 7, 7, 7, 2, 1, 6, 6, 6, 6, 6, 1, 1, 1,
```

```
## 1, 1, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 5,
## 5, 5, 5, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 3, 1, 1, 1, 2, 2, 1,
## 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1, 1, 1, 1, 3, 3,
## 1, 2, 2, 1, 1, 1, 1, 1, 1, 3, 3, 2, 2, 1, 1, 2, 2, 3, 3, 2, 2,
## 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 6,
## 6, 6, 6, 1, 2, 1, 2, 1, 5, 5, 5, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 2, 2, 3, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 1, 1, 3, 3, 1, 1, 1, 3, 3,
## 1, 3, 3, 1, 1, 4, 4, 4, 3, 8, 8, 8, 8, 8, 8, 11, 11, 11, 11,
## 11, 2, 1, 3, 3, 3, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1,
## 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 2, 1, 5, 1, 3, 2, 1, 1, 1,
## 1, 2, 3, 2, 1, 1, 1, 2, 2, 2, 1, 2, 1, 3, 3, 3, 1, 1, 3, 3, 1,
## 1, 1, 1, 2, 1, 4, 4, 4, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1,
## 1, 1, 1, 1, 4, 5, 5, 5, 5, 1, 2, 1), c(0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
## 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
## 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
## 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
## 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), c(30, 39, 52, 47, 35, 28, 60,
## 47, 44, 16, 35, 26, 39, 2, 19, 16, 38, 28, 45.5, 45, 16, 28,
## 29, 27, 45, 65, 19, 62, 38, 14, 11, 36, 56, 25, 0.92, 2, 45,
## 52, 55, 35, 45, 35, 37, 27, 33, 27, 62, 29, 27, 37, 48, 25, 24,
## 56, 53, 58, 64, 25, 34, 52, 30, 36, 56, 32, 21, 63, 39, 65, 60,
## 31, 1, 32, 28, 26, 32, 40, 41, 21, 28, 39, 17, 49, 58, 49, 30,
## 49, 48, 31, 17, 56, 36, 26, 44, 35, 38, 23, 19, 24, 23, 48, 47,
```

```
## 48, 28, 27, 51, 28, 57, 27, 48, 24, 18, 45, 19, 36, 29, 36, 3,
## 2, 36.5, 4, 50, 28, 34, 18, 23, 48, 32, 25, 30, 18, 42, 25, 47,
## 28, 38, 14, 32.5, 25, 42, 28, 18, 38, 35, 28, 29, 43, 24, 34,
## 31, 23, 27, 25, 34, 24, 43, 34, 55, 6, 28, 52, 0.83, 28, 24,
## 50, 13, 41, 30, 30, 0.67, 24, 18, 30, 19, 39, 22, 40, 32, 0.42,
## 41, 17, 16, 45.5, 21, 27, 33, 17, 20, 24, 12, 14, 20, 27, 20,
## 10, 36, 20, 14.5, 5, 24, 0.75, 15, 25, 26, 44, 0.75, 9, 49, 26,
## 34.5, 30, 27, 61, 17, 18, 45, 23.5, 15, 11, 26, 36, 80, 32.5,
## 21, 35, 39, 34, 30, 54, 19, 23, 36, 34, 32, 4, 23, 54, 24, 3,
## 0.83, 18, 29, 60, 40, 23, 34, 18, 18, 37, 28, 32, 33, 17, 7,
## 41, 1, 39, 2, 34, 43, 21, 23, 40, 45, 29, 30, 17, 42, 17, 20,
## 20, 21, 27, 26, 32, 19, 22, 1, 15, 26, 29, 19, 26, 25, 16, 31,
## 50, 4, 65, 44, 26, 27, 24.5, 21, 29, 61, 36, 18, 16, 47, 20,
## 28, 36, 30, 21, 16, 29, 33, 24, 28, 29, 2, 74, 45, 33, 51, 19,
## 25, 45, 16, 25, 38, 3, 5, 9, 18, 34, 28, 22, 11, 2, 39, 39, 9,
## 4, 6, 25, 18, 40, 45, 4, 10, 9, 27, 26, 28, 20, 26, 4, 27, 1,
## 51, 42, 25, 19, 7, 25, 23, 19, 22, 25, 24, 36, 19, 35, 20, 35,
## 16, 47, 26, 20, 21, 19, 24, 9, 24, 25, 18, 33, 32, 28, 30, 36,
## 26, 40, 22, 44, 2, 29, 3, 8, 36, 18, 21, 24, 18, 27, 19, 17,
## 20, 22, 22, 18, 14, 31, 32, 31, 23, 16, 21, 33, 34, 44, 31, 30.5,
## 59, 34, 44, 35, 5, 21, 20, 32, 40, 30.5, 35, 19, 18, 33, 25,
## 24.5, 39, 62, 40.5, 43, 28, 25, 32, 40.5, 31, 45, 54, 52, 61,
## 47, 35, 45, 18, 41, 26, 19, 70.5, 38, 19, 38, 32, 24, 65, 31,
## 21, 21, 44, 25, 35, 5, 25, 16, 21, 27, 18, 29, 7, 4, 8, 2, 24,
## 29, 32, 6, 15, 18, 30, 9, 37, 18, 42, 28.5, 21, 24, 2, 19, 64,
## 33, 23, 16, 20, 33, 29, 22, 45, 32, 26, 55.5, 24, 42, 22, 22,
## 28.5, 26, 43, 18, 50, 47, 20, 30, 47, 40.5, 8, 24, 28, 43, 30,
## 36, 31, 26, 1, 66, 70, 35, 16, 29, 24, 21, 31, 31, 32, 8, 19,
## 40, 34, 36, 33, 5, 3, 11, 9, 14, 1, 43, 16, 20, 48, 19, 20, 25,
## 31, 35, 7, 43, 45, 50, 17, 19, 25, 36, 30, 40, 36, 35, 42, 24,
## 49, 49, 30, 24, 48, 58, 40, 38, 22, 16, 24, 51, 38, 50, 28, 35,
## 32, 21, 44, 50, 32, 6, 71, 35, 35, 36, 39, 18, 38, 42, 36, 18,
## 17, 23, 36, 50, 22, 24, 1, 31, 16, 45.5, 32, 21, 21, 18, 19,
## 17, 37, 23, 22, 25, 3, 16, 27, 36, 29, 17, 21, 38, 20, 35, 24,
## 25, 32, 22, 41, 21, 25, 16, 30, 44, 35, 17, 22, 21, 20, 32, 31,
## 39, 32, 21, 25, 28, 24, 26, 27, 23, 19, 50, 29, 18, 9, 21, 48,
## 16, 40, 46, 30), c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
## 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
## 0, 0, 0, 1), c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
## 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
## 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
## 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
## 0, 0), c(2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1,
## 2, 1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2,
## 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
## 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2,
## 2, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2,
## 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 1, 1, 1, 2,
## 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 2,
```

```
## 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1,
## 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1,
## 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1,
## 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2,
## 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
## 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2,
## 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1,
## 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1,
## 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,
## 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1,
## 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2,
## 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
## 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2,
## 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1,
## 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1,
## 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1,
## 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 1, 1,
## 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 2,
## 2, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1
## )), control = list(20, 7, 0.01, 4, 5, 2, 0, 6, 0))
##   n= 669
##
##           CP nsplit rel error
## 1 0.41245136      0 1.0000000
## 2 0.03891051      1 0.5875486
## 3 0.02529183      2 0.5486381
## 4 0.01750973      4 0.4980545
## 5 0.01000000      6 0.4630350
##
## Variable importance
##    Sexmale    Pclass3 FamilySize        Age    Pclass2     Child1
##         51         18         12         10          6          3
##
## Node number 1: 669 observations,    complexity param=0.4124514
##   predicted class=0  expected loss=0.3841555  P(node) =1
##     class counts:   412   257
##    probabilities: 0.616 0.384
##   left son=2 (441 obs) right son=3 (228 obs)
##   Primary splits:
##       Sexmale    < 0.5  to the right, improve=83.919140, (0 missing)
##       Pclass3    < 0.5  to the right, improve=39.775260, (0 missing)
##       FamilySize < 1.5  to the left,  improve=11.833780, (0 missing)
##       Pclass2    < 0.5  to the left,  improve= 5.533223, (0 missing)
##       Age        < 6.5  to the right, improve= 4.495378, (0 missing)
##   Surrogate splits:
##       FamilySize < 3.5  to the left,  agree=0.677, adj=0.053, (0 split)
##
## Node number 2: 441 observations,    complexity param=0.01750973
##   predicted class=0  expected loss=0.2040816  P(node) =0.6591928
```

```
##     class counts:   351    90
##    probabilities: 0.796 0.204
##   left son=4 (416 obs) right son=5 (25 obs)
##   Primary splits:
##       Age        < 4.5  to the right, improve=6.7145370, (0 missing)
##       Child1     < 0.5  to the left,  improve=5.2332210, (0 missing)
##       Pclass3    < 0.5  to the right, improve=4.4851890, (0 missing)
##       FamilySize < 1.5  to the left,  improve=2.8431800, (0 missing)
##       Pclass2    < 0.5  to the right, improve=0.1115739, (0 missing)
##   Surrogate splits:
##       Child1 < 0.5  to the left,  agree=0.98, adj=0.64, (0 split)
##
## Node number 3: 228 observations,    complexity param=0.03891051
##   predicted class=1  expected loss=0.2675439  P(node) =0.3408072
##     class counts:    61   167
##    probabilities: 0.268 0.732
##   left son=6 (102 obs) right son=7 (126 obs)
##   Primary splits:
##       Pclass3    < 0.5  to the right, improve=29.24667000, (0 missing)
##       FamilySize < 4.5  to the right, improve=11.65263000, (0 missing)
##       Pclass2    < 0.5  to the left,  improve= 7.94215600, (0 missing)
##       Age        < 18.5 to the left,  improve= 3.54994300, (0 missing)
##       Child1     < 0.5  to the right, improve= 0.09649123, (0 missing)
##   Surrogate splits:
##       Pclass2    < 0.5  to the left,  agree=0.715, adj=0.363, (0 split)
##       Age        < 26.5 to the left,  agree=0.658, adj=0.235, (0 split)
##       FamilySize < 4.5  to the right, agree=0.627, adj=0.167, (0 split)
##       Child1     < 0.5  to the right, agree=0.575, adj=0.049, (0 split)
##
## Node number 4: 416 observations
##   predicted class=0  expected loss=0.1826923  P(node) =0.6218236
##     class counts:   340    76
##    probabilities: 0.817 0.183
##
## Node number 5: 25 observations,    complexity param=0.01750973
##   predicted class=1  expected loss=0.44  P(node) =0.03736921
##     class counts:    11    14
##    probabilities: 0.440 0.560
##   left son=10 (8 obs) right son=11 (17 obs)
##   Primary splits:
##       FamilySize < 4.5  to the right, improve=4.452353, (0 missing)
##       Pclass3    < 0.5  to the right, improve=3.853333, (0 missing)
##       Pclass2    < 0.5  to the left,  improve=3.764444, (0 missing)
##       Age        < 2.5  to the left,  improve=0.320000, (0 missing)
##   Surrogate splits:
##       Age     < 3.5  to the right, agree=0.72, adj=0.125, (0 split)
##       Pclass3 < 0.5  to the right, agree=0.72, adj=0.125, (0 split)
##
## Node number 6: 102 observations,    complexity param=0.02529183
##   predicted class=0  expected loss=0.4509804  P(node) =0.1524664
##     class counts:    56    46
##    probabilities: 0.549 0.451
##   left son=12 (22 obs) right son=13 (80 obs)
##   Primary splits:
```

```
##       FamilySize < 4.5  to the right, improve=5.5529860, (0 missing)
##       Age        < 38.5 to the right, improve=2.6348040, (0 missing)
##       Child1     < 0.5  to the left,  improve=0.4764706, (0 missing)
##   Surrogate splits:
##       Age < 12.5 to the left,  agree=0.794, adj=0.045, (0 split)
##
## Node number 7: 126 observations
##   predicted class=1  expected loss=0.03968254  P(node) =0.1883408
##     class counts:     5   121
##    probabilities: 0.040 0.960
##
## Node number 10: 8 observations
##   predicted class=0  expected loss=0.125  P(node) =0.01195815
##     class counts:     7     1
##    probabilities: 0.875 0.125
##
## Node number 11: 17 observations
##   predicted class=1  expected loss=0.2352941  P(node) =0.02541106
##     class counts:     4    13
##    probabilities: 0.235 0.765
##
## Node number 12: 22 observations
##   predicted class=0  expected loss=0.1363636  P(node) =0.0328849
##     class counts:    19     3
##    probabilities: 0.864 0.136
##
## Node number 13: 80 observations,    complexity param=0.02529183
##   predicted class=1  expected loss=0.4625  P(node) =0.1195815
##     class counts:    37    43
##    probabilities: 0.462 0.537
##   left son=26 (25 obs) right son=27 (55 obs)
##   Primary splits:
##       Age        < 27.5 to the right, improve=2.29136400, (0 missing)
##       Child1     < 0.5  to the left,  improve=1.56756400, (0 missing)
##       FamilySize < 2.5  to the right, improve=0.01067393, (0 missing)
##
## Node number 26: 25 observations
##   predicted class=0  expected loss=0.36  P(node) =0.03736921
##     class counts:    16     9
##    probabilities: 0.640 0.360
##
## Node number 27: 55 observations
##   predicted class=1  expected loss=0.3818182  P(node) =0.08221226
##     class counts:    21    34
##    probabilities: 0.382 0.618
```

Realizamos la predicción con el modelo construido con rpart y el dataset test:

```
##    predict_tree2
##       0   1
##   0 123  14
##   1  28  57
```

```
## [1] "El % de registros correctamente clasificados es: 81.0811 %"
```

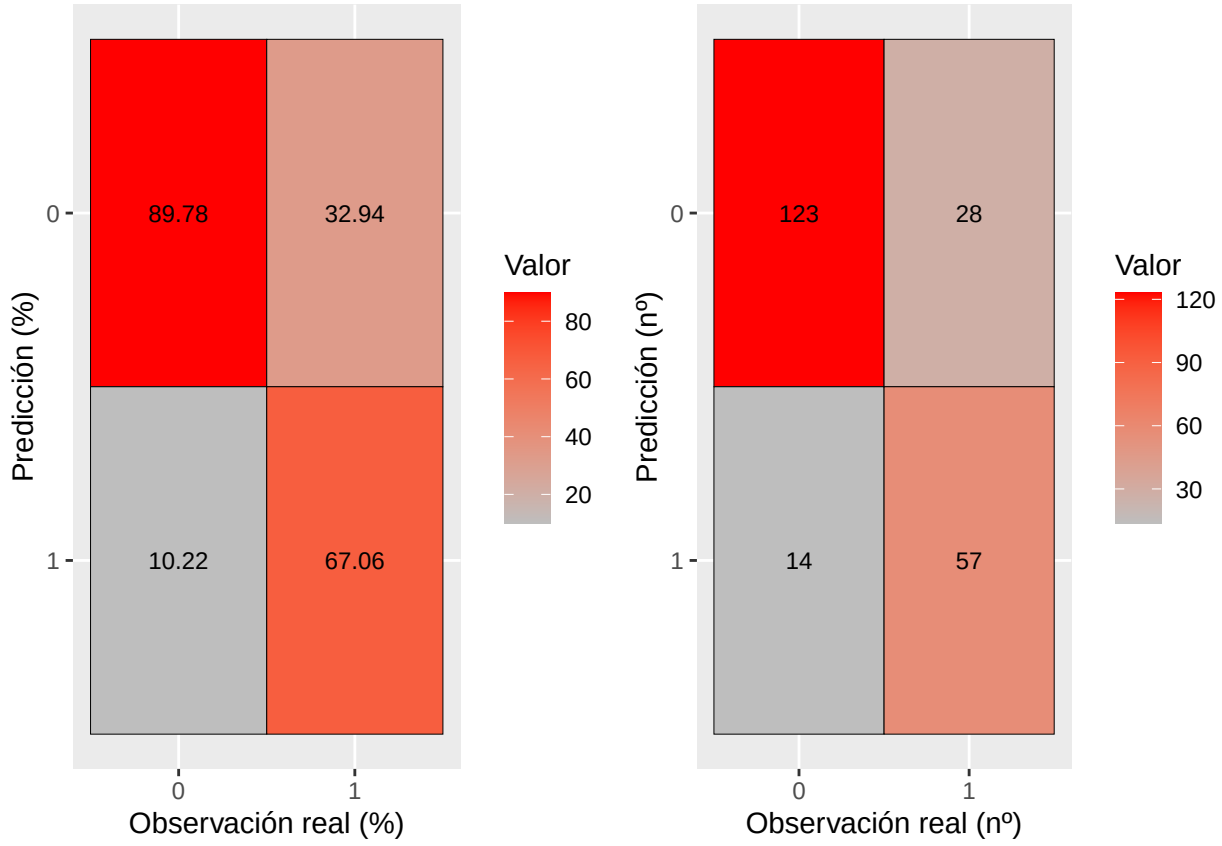
```
## [1] "El mejor parámetro para el árbol de clasificación ha sido: maxdepth=6"
```

Matriz de confusión

```
csstab_tree2 <- CrossTable(data_test$Survived, predict_tree2,
                           prop.chisq  = FALSE, prop.c = FALSE, prop.r =FALSE,
                           dnn = c('Reality', 'Prediction'))
```

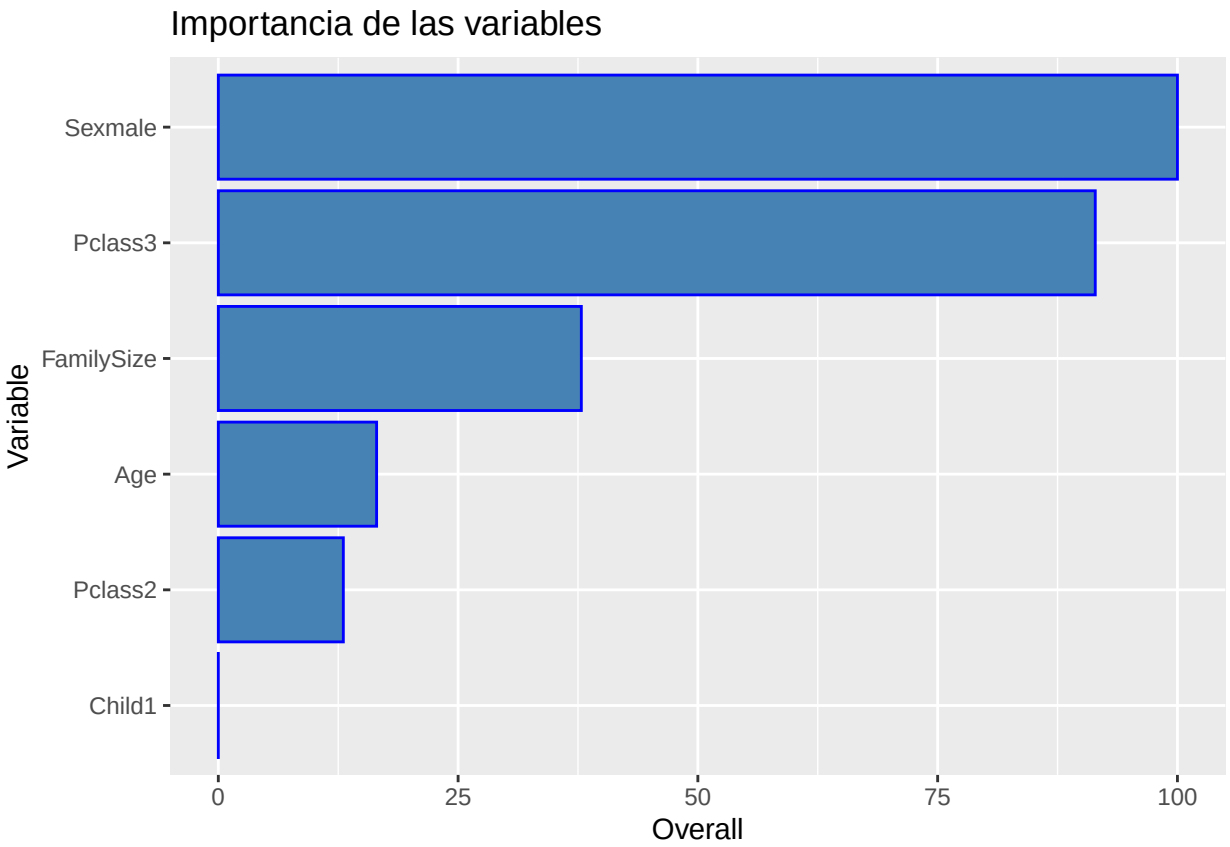
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       123 |        14 |       137 |
##              |     0.554 |     0.063 |           |
## -------------|-----------|-----------|-----------|
##            1 |        28 |        57 |        85 |
##              |     0.126 |     0.257 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       151 |        71 |       222 |
## -------------|-----------|-----------|-----------|
##
##
```

```
plot_matriz_confusion(csstab_tree2)
```



Importancia de las variables

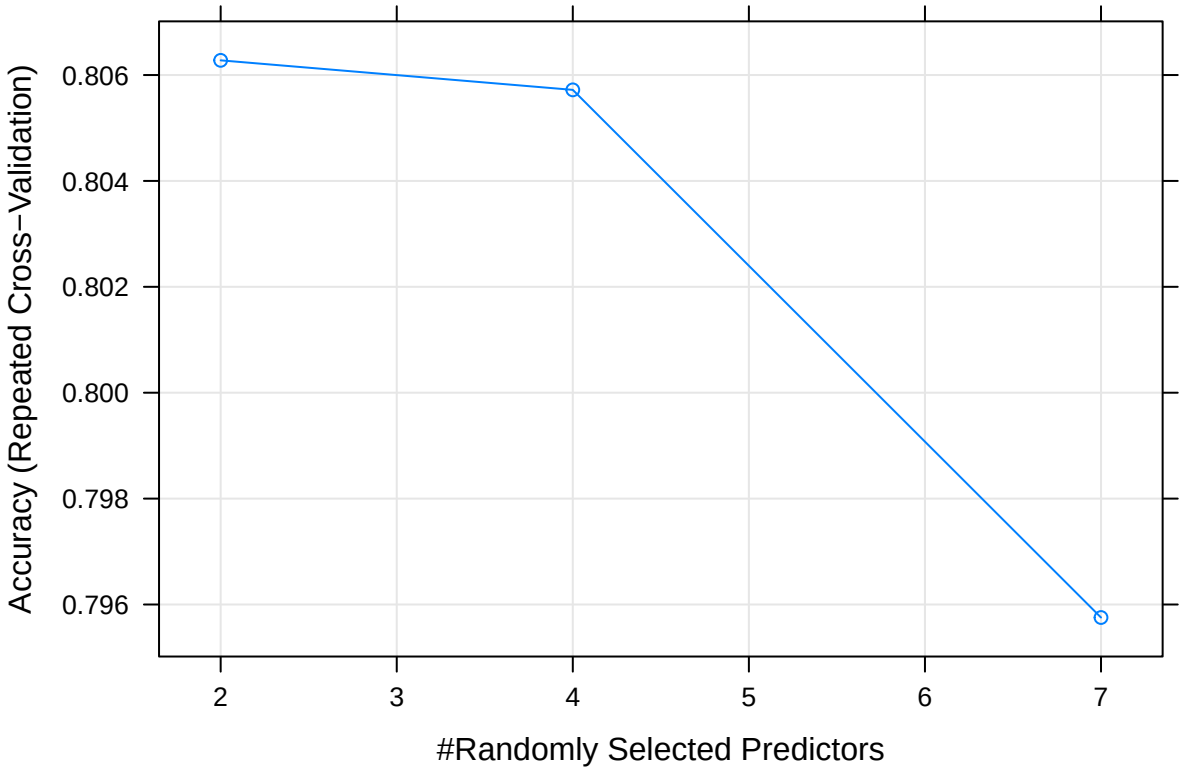
```
datosVarImp <- varImp(model_tree2)$importance %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  arrange(Overall)
datosVarImp$Variable <- reorder(datosVarImp$rowname, datosVarImp$Overall)
ggplot(datosVarImp)+
  geom_col(aes(x = Variable, y = Overall), fill="steelblue", color="blue")+
  coord_flip() + ggtitle("Importancia de las variables")
```



4.3.2.3 Random Forest

Construimos un Random Forest, una combinación de árboles de predicción. Cada árbol utiliza muestras con reemplazo del conjunto de datos que se le pasa al modelo. Cada uno utiliza valores diferentes de variables, para dar opciones a algunas variables que podrían quedar eclipsadas por otras con más relevancia.

```
tctrl <- caret::trainControl(method = "repeatedcv",
                             number=10, repeats = 3)
model_rf <- caret::train(Survived ~ Sex+Parch+SibSp+Child+Age+Pclass,
                         data = data_train,
                         method = "rf",
                         trControl = tctrl,
                         verbose = FALSE)
plot(model_rf)
```



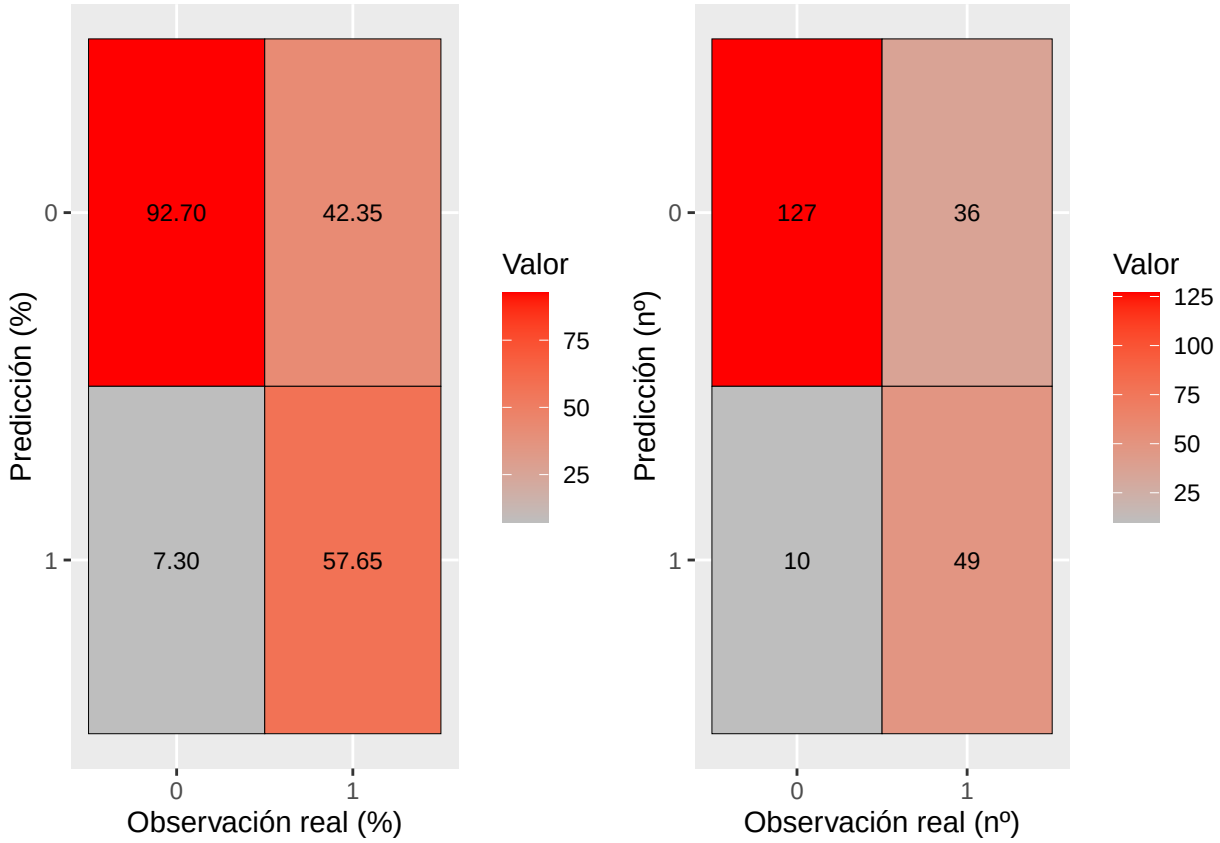
Matriz de confusión

```
##    predict_rf
##       0   1
##   0 127  10
##   1  36  49
```

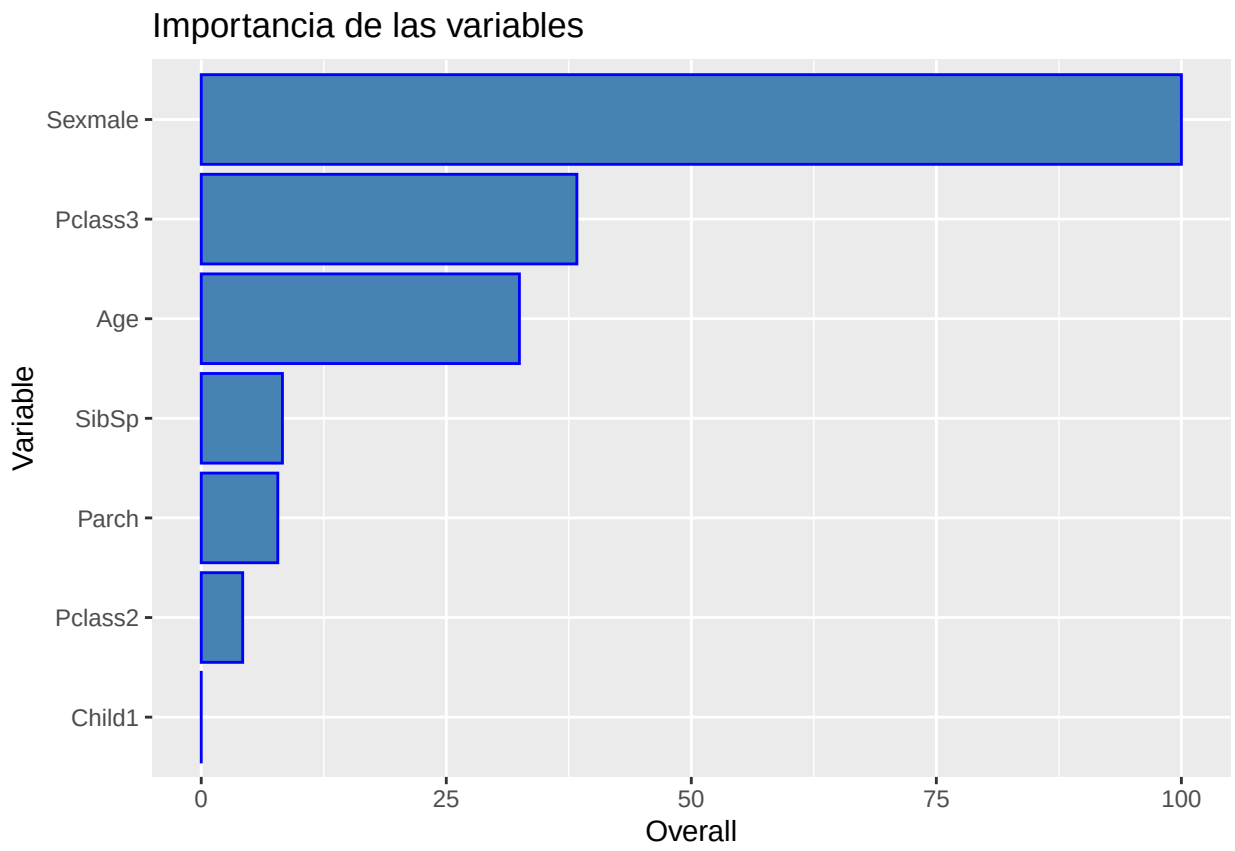
```
## [1] "El % de registros correctamente clasificados es: 79.2793 %"
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       127 |        10 |       137 |
##              |     0.572 |     0.045 |           |
## -------------|-----------|-----------|-----------|
##            1 |        36 |        49 |        85 |
```

```
##              |     0.162 |     0.221 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       163 |        59 |       222 |
## -------------|-----------|-----------|-----------|
##
##
```



Importancia de las variables



4.3.2.4 C5.0 C5.0 es el algoritmo sucesor de C4.5, publicado por John Ross Quinlan (1992), con el objetivo de crear árboles de clasificación. Entre sus características, destacan la capacidad para generar árboles de decisión simples, modelos basados en reglas, ensembles basados en boosting y asignación de distintos pesos a los errores. Este algoritmo ha resultado de una gran utilidad a la hora de crear modelos de clasificación y todas sus capacidades son accesibles mediante el paquete C50. Aunque comparte muchas características con los algoritmos de random forest y gradient boosting, cabe tener en cuenta algunas peculiaridades:

- La medida de pureza empleada para las divisiones del árbol es la entropía.
- El podado de los árboles se realiza por defecto, y el método empleado se conoce como pessimistic pruning.
- Los árboles se pueden convertir en modelos basados en reglas.
- Emplea un algoritmo de boosting más próximo a AdaBoost que a Gradient Boosting.
- Por defecto, el algoritmo de boosting se detiene si la incorporación de nuevos modelos no aporta un mínimo de mejora.
- Incorpora una estrategia para la selección de predictores (Winnowing) previo ajuste del modelo.
- Permite asignar diferente peso a cada tipo de error.

Creamos primero unas variables para simplificar el uso de los distintos modelos. - trainX corresponde a los atributos **Sex**, **Parch**, **SibSp**, **Child**, **Age**, **Pclass** del dataframe sin la variable Survived (para el conjunto de train) - trainy es el dataframe pero sólo con la variable Survived (para el conjunto de train)

- testX corresponde a los atributos **Sex**, **Parch**, **SibSp**, **Child**, **Age**, **Pclass** del dataframe sin la variable Survived (para el conjunto de train)
- testy es el dataframe pero sólo con la variable Survived (para el conjunto de test)

```
trainX <- data_train[, c("Sex","Parch","SibSp","Child","Age","Pclass")]
trainy <- data_train[, c("Survived")]
testX <- data_test[, c("Sex","Parch","SibSp","Child","Age","Pclass")]
testy <- data_test[, c("Survived")]
```

4.3.2.4.1 Importancia de los atributos

Definimos primero las métricas que vamos a usar en los árboles C5.0 a la hora de mostrar la importancia de los atributos:

- Usage: porcentaje de observaciones de entrenamiento que caen en todos los nodos generados tras una división en el que ha participado el predictor. Por ejemplo, en el caso de un árbol simple, el predictor empleado en la primera decisión recibe automáticamente una importancia del 100%, ya que todas las observaciones de entrenamiento caen en uno de los dos nodos hijos generados. Otros predictores puede que participen con frecuencia en divisiones, pero si en los nodos hijos solo caen unas pocas observaciones, su importancia puede ser próxima a cero. En el caso de boosting, se promedia la importancia de cada predictor en todos los árboles que forman el ensemble.
- Splits: porcentaje de divisiones en las que participa cada predictor. No tiene en cuenta la importancia de la división.

4.3.2.4.2 Primer modelo c5.0

Vamos a empezar realizando un primer modelo C5.0 con los valores por defecto y con las reglas:

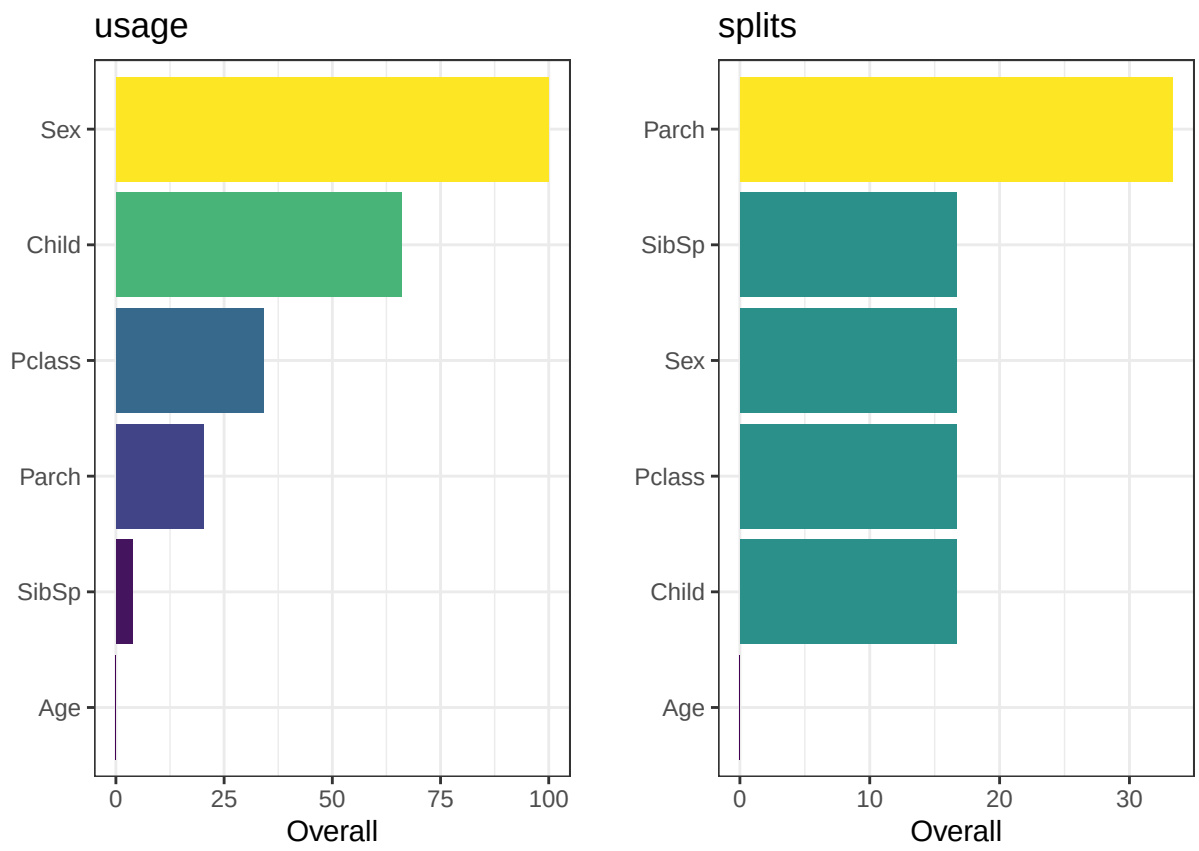
```
bc_tree <- C50::C5.0(trainX,
                     trainy,
                      rules=FALSE,
                      control = C5.0Control(seed = 123)
                )
summary(bc_tree)
```

```
##
## Call:
## C5.0.default(x = trainX, y = trainy, rules = FALSE, control =
##  C5.0Control(seed = 123))
##
##
## C5.0 [Release 2.07 GPL Edition]      Tue Jan  5 17:26:53 2021
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 669 cases (7 attributes) from undefined.data
##
## Decision tree:
##
## Sex = female:
## :...Pclass in {1,2}: 1 (126/5)
## :   Pclass = 3:
## :   :...Parch <= 0: 1 (58/25)
## :       Parch > 0: 0 (44/13)
## Sex = male:
## :...Child = 0: 0 (407/74)
##     Child = 1:
##     :...Parch <= 0: 0 (8)
##         Parch > 0:
```

```
##         :...SibSp <= 2: 1 (15)
##             SibSp > 2: 0 (11/1)
##
##
## Evaluation on training data (669 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##       7  118(17.6%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     382    30    (a): class 0
##      88   169    (b): class 1
##
##
##  Attribute usage:
##
##  100.00% Sex
##   65.92% Child
##   34.08% Pclass
##   20.33% Parch
##    3.89% SibSp
##
##
## Time: 0.0 secs
```

Aunque podríamos representar el árbol, nos han salido muchas reglas y es complicado de visualizar. Lo analizamos numéricamente.

Primero veamos visualmente la importancia de cada variable:

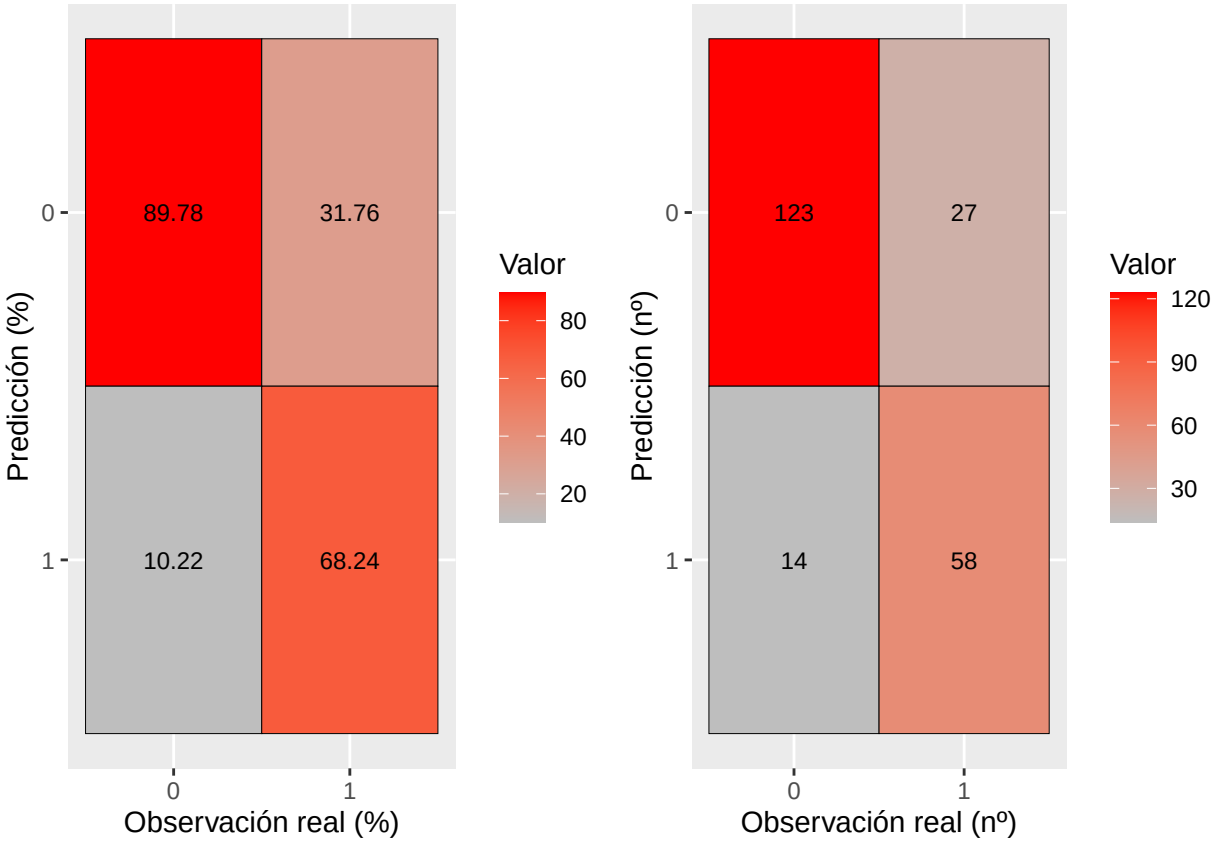


Está usando casi todas las variables, de ahí que nos salgan tantas reglas.

Vamos a ver la precisión del modelo

```
## [1] "La precisión del árbol C50 1 es: 81.5315 %"
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       123 |        14 |       137 |
##              |     0.820 |     0.194 |           |
##              |     0.554 |     0.063 |           |
## -------------|-----------|-----------|-----------|
##            1 |        27 |        58 |        85 |
##              |     0.180 |     0.806 |           |
```

```
##              |     0.122 |     0.261 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       150 |        72 |       222 |
##              |     0.676 |     0.324 |           |
## -------------|-----------|-----------|-----------|
##
##
```



4.3.2.4.3 Segundo modelo C5.0 con Winnowing

Winnowing consiste en aplicar un algoritmo de clasificación (algoritmo Winnow) a los atributos para eliminar aquellos que sean de poca ayuda.

Es una selección de atributos que se realiza antes de la creación del modelo.

El conjunto de datos se divide por la mitad aleatoriamente y se crea un modelo inicial. Cada predictor se va quitando por turno y se analiza el efecto sobre la calidad del modelo (usando la otra mitad de la división aleatoria)

Se marcan los predictores si su eliminación no incrementa la tasa de error. El modelo final se calcula usando todos los conjuntos de datos usando sólo los predictores no marcados.

Se puede realizar winnowing en C5.0 simplemente usando un flag.

```
bc_winno_tree <- C50::C5.0(trainX,
                           trainy,
                           rules   = TRUE,
                           control = C5.0Control(seed = 123, winnow = TRUE)
                     )
```

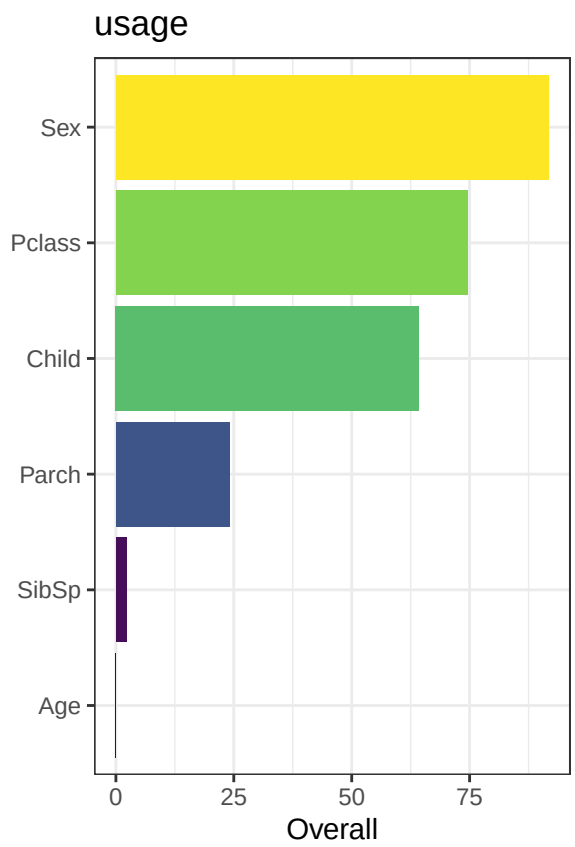
```
summary(bc_winno_tree)
```

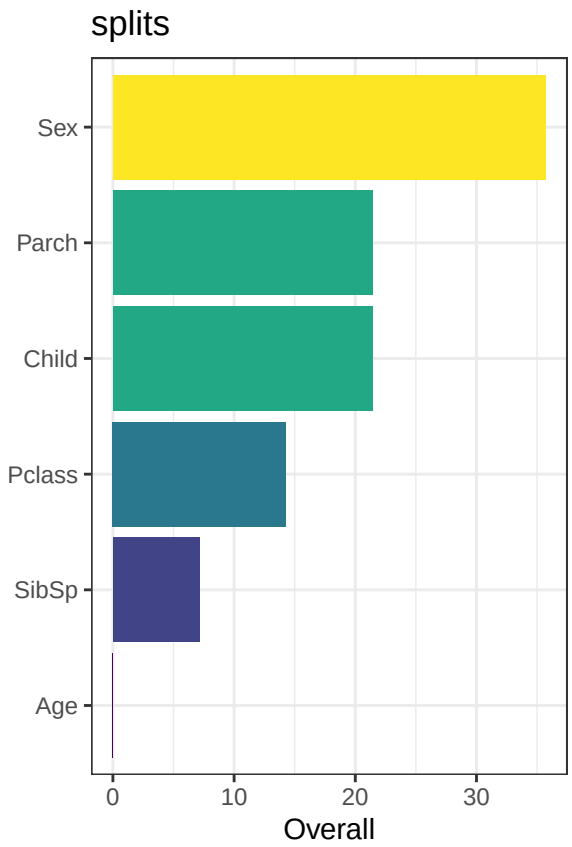
```
##
## Call:
## C5.0.default(x = trainX, y = trainy, rules = TRUE, control = C5.0Control(seed
##  = 123, winnow = TRUE))
##
##
## C5.0 [Release 2.07 GPL Edition]      Tue Jan  5 17:26:54 2021
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 669 cases (7 attributes) from undefined.data
##
## No attributes winnowed
##
## Rules:
##
## Rule 1: (8, lift 1.5)
##  Sex = male
##  Parch <= 0
##  Child = 1
##  ->  class 0  [0.900]
##
## Rule 2: (407/74, lift 1.3)
##  Sex = male
##  Child = 0
##  ->  class 0  [0.817]
##
## Rule 3: (373/86, lift 1.2)
##  Pclass = 3
##  ->  class 0  [0.768]
##
## Rule 4: (126/5, lift 2.5)
##  Sex = female
##  Pclass in {1, 2}
##  ->  class 1  [0.953]
##
## Rule 5: (15, lift 2.4)
##  Sex = male
##  Parch > 0
##  SibSp <= 2
##  Child = 1
##  ->  class 1  [0.941]
##
## Rule 6: (138/28, lift 2.1)
##  Sex = female
##  Parch <= 0
##  ->  class 1  [0.793]
##
## Default class: 0
##
##
```

```
## Evaluation on training data (669 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       6  118(17.6%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     382    30    (a): class 0
##      88   169    (b): class 1
##
##
##  Attribute usage:
##
##   91.78% Sex
##   74.59% Pclass
##   64.28% Child
##   24.07% Parch
##    2.24% SibSp
##
##
## Time: 0.0 secs
```

Parece que en este caso no es capaz de eliminar atributos (winnow)

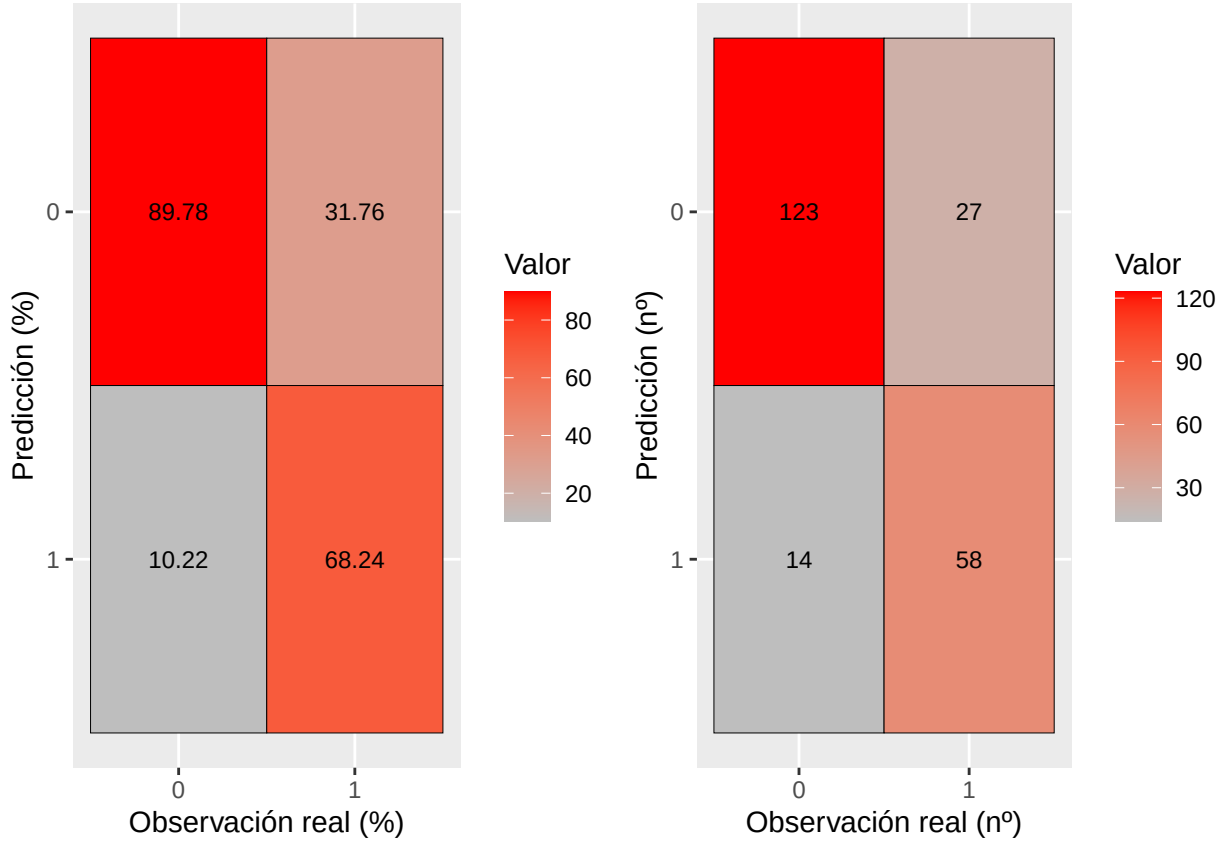
Vamos a ver cuantas variables ha utilizado ahora:






```
## [1] "La precisión del árbol C50 2 es: 81.5315 %"
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       123 |        14 |       137 |
##              |     0.820 |     0.194 |           |
##              |     0.554 |     0.063 |           |
## -------------|-----------|-----------|-----------|
##            1 |        27 |        58 |        85 |
##              |     0.180 |     0.806 |           |
##              |     0.122 |     0.261 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       150 |        72 |       222 |
```

```
##              |     0.676 |     0.324 |           |
## -------------|-----------|-----------|-----------|
##
##
```



No hemos mejorado los errores

4.3.2.4.4 Tercer modelo C5.0 usando boosting

La implementación del C5.0 dispone de la posibilidad de aplicar un *Boosting* similar a *AdaBoost*, que permite combinar varios árboles (técnica de *Ensemble*) para producir mejores modelos predictivos que si sólamente usasemos un árbol de decisión simple.

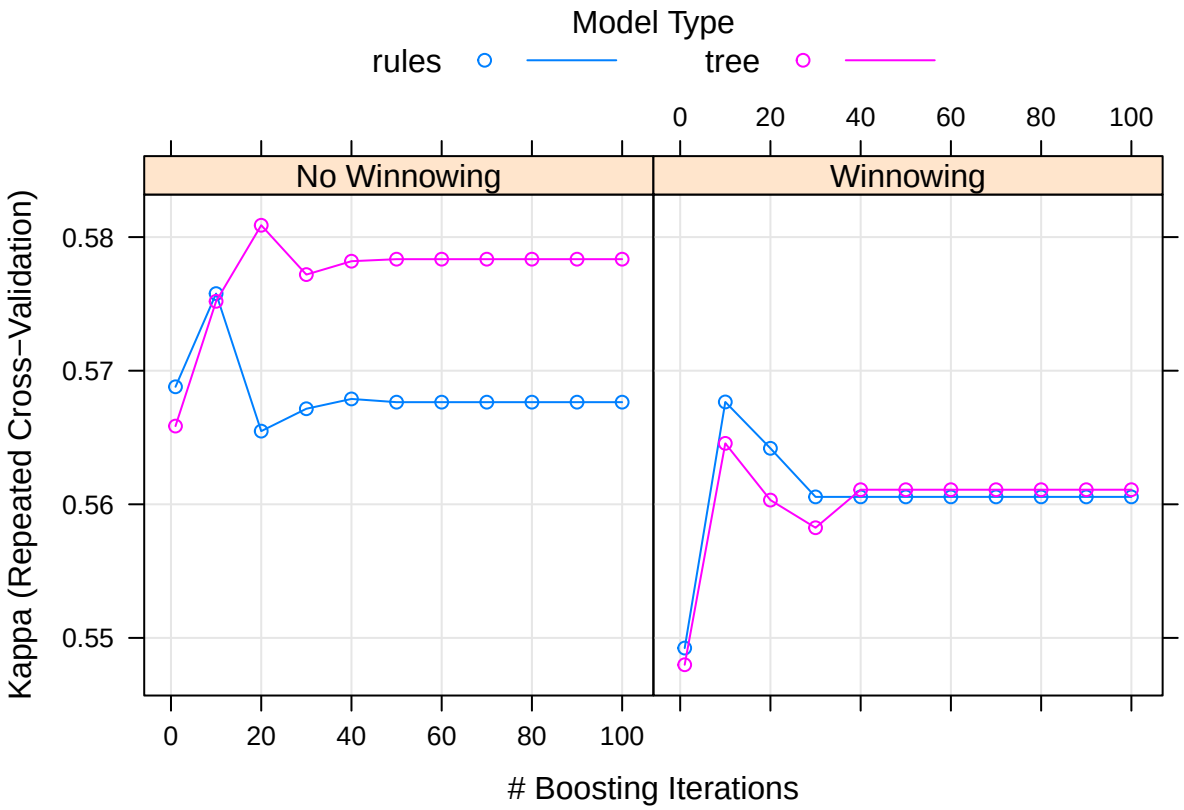
Para ello se utiliza el parámetro `trials`, en el que le indicamos cuantas iteraciones (que a la postre son árboles distintos) debe utilizar para generar el modelo *ensemble*.

Podríamos ir probando con distintos boosts variando el winnowing y los parámetros hasta encontrar el que mejor modelo nos generase. Es una labor tediosa que podemos simplificar usando la librería `caret`, que proporciona una forma de entrenar modelos modificando sus parámetros, vamos a usarlo para el C5.0 mostrando la comparativa posteriormente.

En este caso vamos a usar Cohen-Kappa como medición de calidad del modelo, y compararemos C5.0 generados con distintas iteraciones (boosting) combinándolo con winnowing y con rules/tree.

```
tuned <- train(trainX, trainy, method = "C5.0", tuneLength = 11,
        trControl = trainControl(method = "repeatedcv", repeats = 5, allowParallel = F),
        metric = "Kappa")
```

```
##  Accuracy     Kappa
## 0.8108108 0.5789759
```

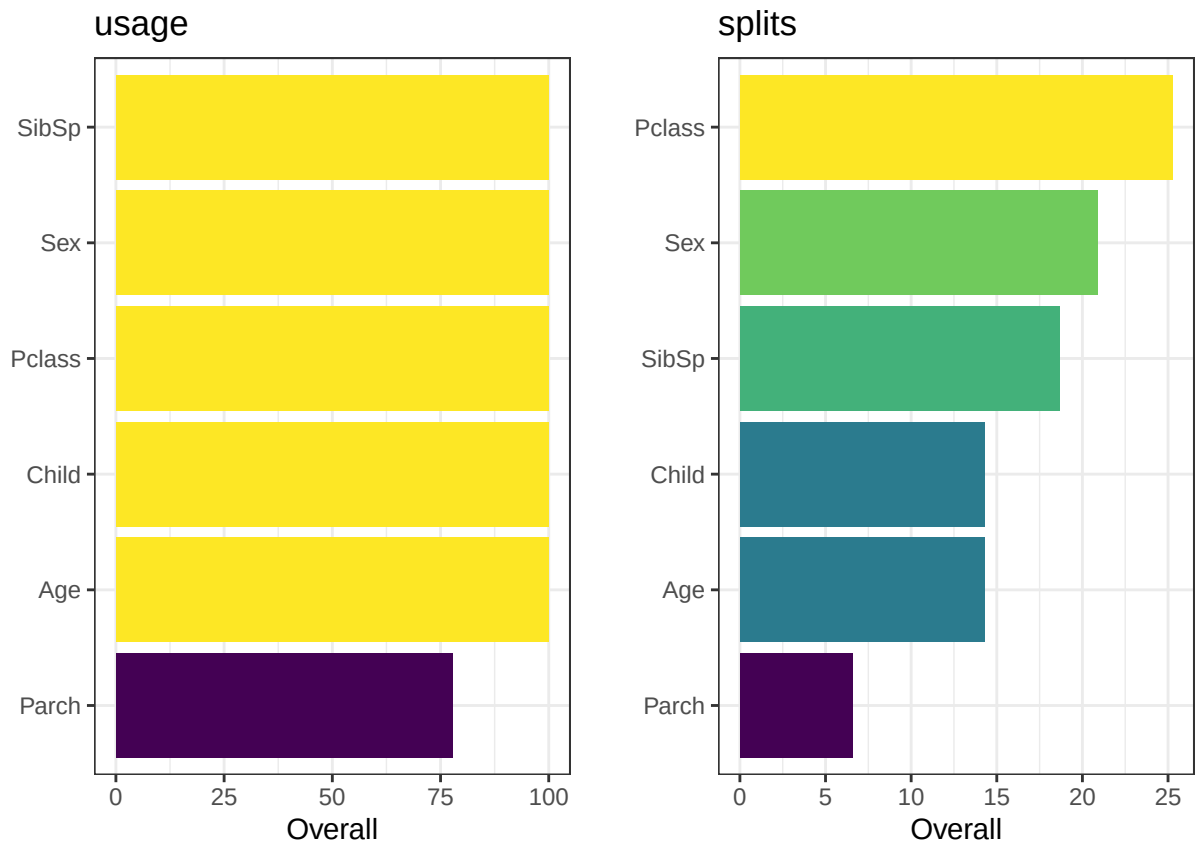


Los valores anteriores son orientativos y nos sirven para acotar más fácilmente los parámetros en los que buscar.

Vamos a crear el mejor árbol sin winnowing y boost, usando 20 trials y sin reglas:

```
bc_boost_tree <- C50::C5.0(
                      trainX,
                      trainy,
                      rules   = FALSE,
                      control = C5.0Control(seed = 123, winnow = FALSE),
                      trials = 20
                    )
```

Como antes, vemos la importancia de los atributos:

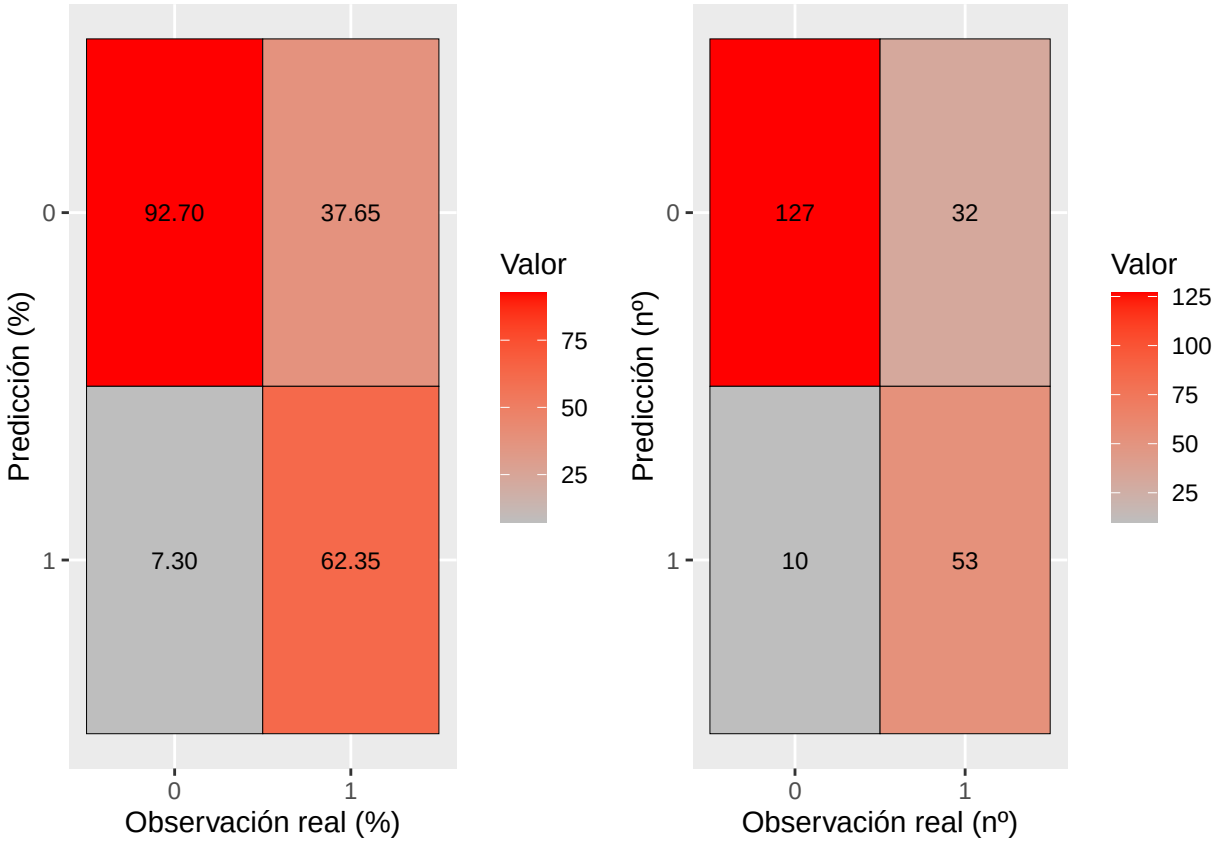


En este caso todos los atributos tienen un uso muy alto.

De nuevo calculamos matriz de confusión

```
## [1] "La precisión del árbol C50 3 es: 81.0811 %"
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  222
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       127 |        10 |       137 |
##              |     0.799 |     0.159 |           |
##              |     0.572 |     0.045 |           |
## -------------|-----------|-----------|-----------|
##            1 |        32 |        53 |        85 |
##              |     0.201 |     0.841 |           |
##              |     0.144 |     0.239 |           |
```

```
## -------------|-----------|-----------|-----------|
## Column Total |       159 |        63 |       222 |
##              |     0.716 |     0.284 |           |
## -------------|-----------|-----------|-----------|
##
##
```



5 Representación de los resultados a partir de tablas y gráficas.

Durante la resolución de la práctica hemos aprovechado las capacidades didácticas que ofrece el formato rmd. Se ha podido ir generando diversas gráficas y tablas en cada una de las secciones y explicando sus interpretaciones. Hemos incluido todo el código de la práctica a lo largo de este documento mostrándose los resultados de forma rápida y amena. Hemos utilizado diversos tipos de gráficas en nuestro análisis exploratorio de las variables, durante la creación de nuevas, durante la elaboración de hipótesis de correlaciones entre variables y a la hora de mostrar los resultados de nuestros modelos. Esto es especialmente relevante en el caso del análisis exploratorio pues a la hora de tomar decisiones de limpieza y de uso de variables, disponer de las gráficas en el momento de la decisión es muy útil.

Para entender mejor los modelos predictivos, se han ido también creando gráficas representando las tablas de confusión y extrayendo la precisión de los modelos.

En los árboles se han mostrado, cuando eran suficientemente pequeños, las representaciones gráficas de los mismos, y cuando no, se han mostrado datos y tablas numéricas explicando brevemente su utilidad.

A continuación, exponemos un resumen del comportamiento de los diferentes modelos predictivos creados a lo largo de la práctica:

```
knitr::kable(tabla.modelos)
```

Modelo	Precisión	Errores
glm1	85.5855855855856	32
glm2	85.1351351351351	33
glm3	84.6846846846847	34
Decision Tree 1	77.9279279279279	49
Decision Tree 2	81.0810810810811	42
Random Forest	79.2792792792793	46
C50 1	81.5315315315315	41
C50 2	81.5315315315315	41
C50 3	81.0810810810811	42

6 Resolución del problema. A partir de los resultados obtenidos, ¿cuáles son las conclusiones? ¿Los resultados permiten responder al problema?

Hemos realizado, sobre el conjunto de datos del Titanic, las fases de limpieza, depuración y análisis de datos (eliminando variables innecesarias, imputando datos faltantes, comprobando hipótesis estadísticas), hasta finalmente implementar varios modelos de predicción.

El problema que nos planteábamos era: Dado el conjunto de datos que disponemos, ¿podemos predecir la supervivencia o no de un pasajero? Podemos ser optimistas respecto al resultado.

Podemos diferenciar entre las conclusiones del análisis exploratorio y la capacidad de predecir de los modelos creados. Del análisis exploratorio podríamos responder a la pregunta: ¿Importaba el sexo, la edad, la clase social para sobrevivir al naufragio del Titanic? Todo parece indicar que si, que tuvieron mayor proporción de supervivencia las mujeres, los niños y los ricos. Obviamente, no en todos los casos fue así, por eso no hemos encontrado un modelo predictivo con un alto nivel de precisión

Hemos creado diferentes modelos predictivos y los resultados de precisión han sido discretos:

- Destaca una regresión logística con una precisión del 85.58% (glm1), usando datos enriquecidos con variables sintéticas con 32 errores de predicción con los datos de tests.
- Ninguno de Los árboles de decisión planteados ha conseguido mejorar ese registro, quedándose como mejores dos c5.0 en un 81.53% de precision y 41 errores. El uso de un conjunto mas limitado de

atributos tras el análisis previo, ha resultado en que la técnica de winnowing no mejore los resultados, pues todos los atributos son usados. Lo que si varía entre los distintos c5.0 es el balance entre los tipos de errores. Dado el tipo de problema que se plantea sería preferible quedarse con los modelos que fallan mas prediciendo más "no-supervivencia" (y luego si que sobreviven) que los que fallan más prediciendo mas supervivencia y luego no sobreviven.

- La distribución de los fallos en el modelo de regresión logística ha sido que realmente fallecen el 9,49% y el modelo predice que sobreviven (y sobreviven el 22.35% y el modelo predice lo contrario). Si buscamos un modelo que se equivoque menos en la predicción de que sobrevive y luego no lo haga, podríamos usar el primer árbol de decisión que solo tiene un 2.92% de error en ese caso a costa de tener un 52.94% en los casos que predice que fallecen y realmente sobrevive. Otra opción es parametrizar el C5.0 para que priorize tener menos errores del tipo predigo que fallece pero sobrevive. Aunque esto siempre será a costa de un mayor número de errores globales.

6.1 Exportación de los datos

Durante esta práctica se generan dos ficheros que estarán disponibles en el repositorio github. Estos son los siguientes:

- Un fichero "titanic_final.csv" que contiene las 12 variables con las que se ha trabajado, con la imputación de los datos realizada, con aquellas variables no utilizadas del dataset origen eliminadas, y con las transformaciones que hemos considerado necesarias.
- Un fichero "titanic_predict.csv", en el que además de las columnas que tiene el fichero anterior, tiene una columna nueva llamada "Survived_Predicted", que es la columna con la predicción de la variable "Survived" realizada con el modelo escogido.

```
fichero.nuevo <- "titanic_final.csv"
write.csv(data, file=fichero.nuevo, row.names = FALSE)
#Realizamos la predicción con el modelo construido con rpart y el dataset total
predict_fin <-predict(modelo_glm1, data)

#Añadimos la columna de la prediccion al conjunto de datos anterior
data$Survived_Predicted <- predict_fin
fichero.nuevo <- "titanic_predict.csv"
write.csv(data, file=fichero.nuevo, row.names = FALSE)
```

7 Tabla de contribuciones

Contribuciones	Firma
Investigación Previa	Alexis Germán Arroyo Peña, Gabriel Pulido de Torres
Redacción de las respuestas	Alexis Germán Arroyo Peña, Gabriel Pulido de Torres
Desarrollo código	Alexis Germán Arroyo Peña, Gabriel Pulido de Torres

8 Bibliografía y referencias

[1] Arboles de decision, Random Forest, Gradient Boosting y C5.0. [en línea], [sin fecha]. [Consulta: 2 enero 2021]. Disponible en: https://www.cienciadedatos.net/documentos/33_arboles_decision_random_forest_gradient_boosting_c50#C50.

[2] JARMAN, K.H., 2013. The Art of Data Analysis: How to Answer Almost Any Question Using Basic Statistics. S.l.: s.n. ISBN 9781118413357.

[3] MAT, L.S., OSWALDO, D., MIREIA, T. y GONZ, C., [sin fecha]. Introducción a la limpieza y análisis de los datos. ,

[4] MOLINA, L.C. y SANGÜESA I SOLÉ, R., [sin fecha]. Módulo 8: Evaluación de modelos. UOC.