Minería de datos: PEC3 - Clasificación con árboles de decisión

Autor: Nombre estudiante

Diciembre 2023

Contents

Recursos básicos

Esta Prueba de Evaluación Continuada (PEC) cubre principalmente el material didáctico de modelos supervisados y evaluación de modelos.

Complementarios:

- Material docente "Creación y evaluación de modelos no supervisados" proporcionado por la UOC.
- Fichero titanic.csv.

- R package C5.0 (Decision Trees and Rule-Based Models): https://cran.r-project.org/web/packages/C50/index.html
- Fichero de "German Credit": credit.csv (se obtuvo de https://www.kaggle.com/shravan3273/credit-approval)

La descripción de las variables se puede ver en https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

La variable "default" es el target siendo 1 = "No default" y 2 = "Default". Se deben utilizar estos datos para la realización de los ejercicios.

Ejemplo ilustrativo

En este ejercicio vamos a seguir los pasos del ciclo de vida de un proyecto de minería de datos, para el caso de un algoritmo de clasificación usaremos un árbol de decisión, **que es el algoritmo supervisado que vamos a tratar en esta asignatura**. Primero y a modo de ejemplo sencillo lo haremos con el archivo titanic.csv, que se encuentra adjunto en el aula. Este archivo contiene un registro por cada pasajero que viajaba en el Titanic. En las variables se caracteriza si era hombre o mujer, adulto o menor (niño), en qué categoría viajaba o si era miembro de la tripulación. Se mostrará un ejemplo sencillo de solución con estos datos pero los alumnos deberéis responder a las preguntas de la rúbrica para otro conjunto: German Credit. Para este conjunto, tomaréis como referencia la variable "default" que indica el impago de créditos.

Objetivos:

- Estudiar los datos, por ejemplo: ¿Número de registros del fichero? ¿Distribuciones de valores por variables? ¿Hay campos mal informados o vacíos?
- Preparar los datos. En este caso ya están en el formato correcto y no es necesario discretizar ni generar atributos nuevos. Hay que elegir cuáles son las variables que se utilizarán para construir el modelo y cuál es la variable que clasifica. En este caso la variable por la que clasificaremos es el campo de si el pasajero sobrevivia o no.
- Instalar, si es necesario, el paquete C5.0 Se trata de una implementación más moderna del algoritmo ID3 de Quinlan. Tiene los principios teóricos del ID3 más la poda automática. Con este paquete generar un modelo de minería.
- ¿Cuál es la calidad del modelo?
- Generar el árbol gráfico.
- Generar y extraer las reglas del modelo.
- En función del modelo, el árbol y las reglas: ¿Cuál es el conocimiento que obtenemos?
- Probar el modelo generado presentándole nuevos registros. ¿Clasifica suficientemente bien?

A continuación, se plantean los puntos a realizar en la PEC 3 y, tomando como ejemplo el conjunto de datos de Titanic, se obtendrán, a modo de ejemplo, algunos resultados que pretender servir a modo de inspiración para los estudiantes. Los estudiantes deberán utilizar el conjunto de datos de "German Credit Data" que se pueden conseguir en este enlace: https://www.kaggle.com/shravan3273/credit-approval

Este recurso puede ser útil para profundizar sobre el paquete IML: https://uc-r.github.io/iml-pkg

Revisión de los datos, extracción visual de información y preparación de los datos

Carga de los datos:

```
data<-read.csv("./titanic-1.csv",header=T,sep=",")
attach(data)
```

Análisis inicial

Empezaremos haciendo un breve análisis de los datos ya que nos interesa tener una idea general de los datos que disponemos.

Exploración de la base de datos

Primero calcularemos las dimensiones de nuestra base de datos y analizaremos qué tipos de atributos tenemos.

Para empezar, calculamos las dimensiones de la base de datos mediante la función dim(). Obtenemos que disponemos de 2201 registros o pasajeros (filas) y 4 variables (columnas).

```
dim(data)
```

```
## [1] 2201    4
```

¿Cuáles son esas variables? Gracias a la función str() sabemos que las cuatro variables son categóricas o discretas, es decir, toman valores en un conjunto finito. La variable CLASS hace referencia a la clase en la que viajaban los pasajeros (1^a, 2^a, 3^a o crew), AGE determina si era adulto o niño (Adulto o Menor), la variable SEX si era hombre o mujer (Hombre o Mujer) y la última variable (SURVIVED) informa si el pasajero murió o sobrevivió en el accidente (Muere o Sobrevive).

```
str(data)
```

```
## 'data.frame':    2201 obs. of  4 variables:
##  $ CLASS   : chr  "1a" "1a" "1a" "1a" ...
##  $ AGE     : chr  "Adulto" "Adulto" "Adulto" "Adulto" ...
##  $ SEX     : chr  "Hombre" "Hombre" "Hombre" "Hombre" ...
##  $ SURVIVED: chr  "Sobrevive" "Sobrevive" "Sobrevive" "Sobrevive" ...
```

Vemos que las variables están definidas como carácter, así que las transformamos a tipo factor.

```
data[] <- lapply(data, factor)
str(data)
```

```
## 'data.frame':    2201 obs. of  4 variables:
##  $ CLASS   : Factor w/ 4 levels "1a","2a","3a",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ AGE     : Factor w/ 2 levels "Adulto","Menor": 1 1 1 1 1 1 1 1 1 1 ...
##  $ SEX     : Factor w/ 2 levels "Hombre","Mujer": 1 1 1 1 1 1 1 1 1 1 ...
##  $ SURVIVED: Factor w/ 2 levels "Muere","Sobrevive": 2 2 2 2 2 2 2 2 2 2 ...
```

Es de gran interés saber si tenemos muchos valores nulos (campos vacíos) y la distribución de valores por variables. Es por ello recomendable empezar el análisis con una visión general de las variables. Mostraremos para cada atributo la cantidad de valores perdidos mediante la función summary.

```
summary(data)
```

```
##   CLASS         AGE           SEX            SURVIVED
##  1a  :325   Adulto:2092   Hombre:1731   Muere    :1490
##  2a  :285   Menor : 109   Mujer : 470   Sobrevive: 711
##  3a  :706
##  crew:885
```

Como parte de la preparación de los datos, miraremos si hay valores missing.

```
missing <- data[is.na(data),]
dim(missing)
```

```
## [1] 0 4
```

Observamos fácilmente que no hay valores missing y, por tanto, no deberemos preparar los datos en este sentido. En caso de haberlos, habría que tomar decisiones para tratar los datos adecuadamente.

Disponemos por tanto de un data frame formado por cuatro variables categóricas sin valores nulos.

Visualización

Para un conocimiento mayor sobre los datos, tenemos a nuestro alcance unas herramientas muy valiosas: las herramientas de visualización. Para dichas visualizaciones, haremos uso de los paquetes ggplot2, gridExtra y grid de R.

```
if(!require(ggplot2)){
    install.packages('ggplot2', repos='http://cran.us.r-project.org')
    library(ggplot2)
}
```

```
## Loading required package: ggplot2
```

```
if(!require(ggpubr)){
    install.packages('ggpubr', repos='http://cran.us.r-project.org')
    library(ggpubr)
}
```

```
## Loading required package: ggpubr
```

```
if(!require(grid)){
    install.packages('grid', repos='http://cran.us.r-project.org')
    library(grid)
}
```

```
## Loading required package: grid
```

```
if(!require(gridExtra)){
    install.packages('gridExtra', repos='http://cran.us.r-project.org')
    library(gridExtra)
}
```

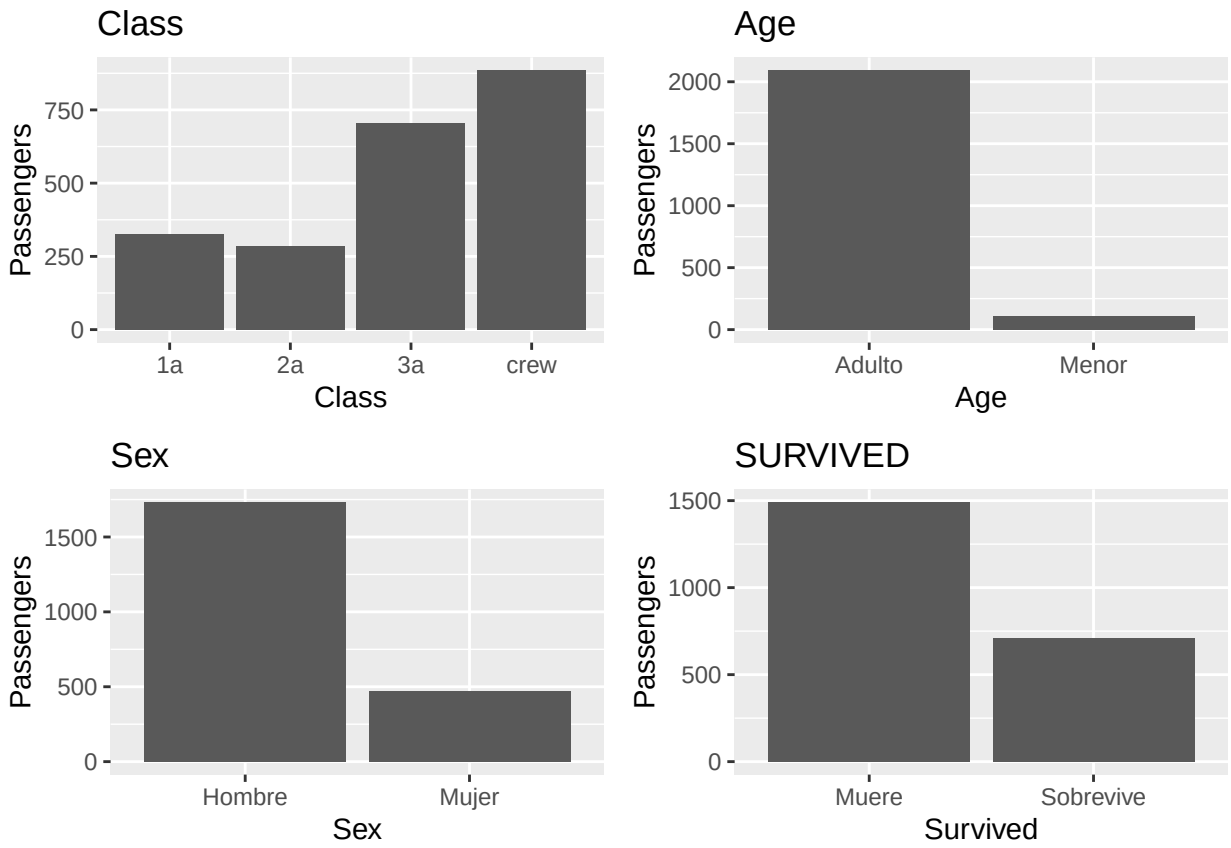
```
## Loading required package: gridExtra
```

```
if(!require(C50)){
    install.packages('C50', repos='http://cran.us.r-project.org')
    library(C50)
}
```

```
## Loading required package: C50
```

Siempre es importante analizar los datos que tenemos ya que las conclusiones dependerán de las características de la muestra.

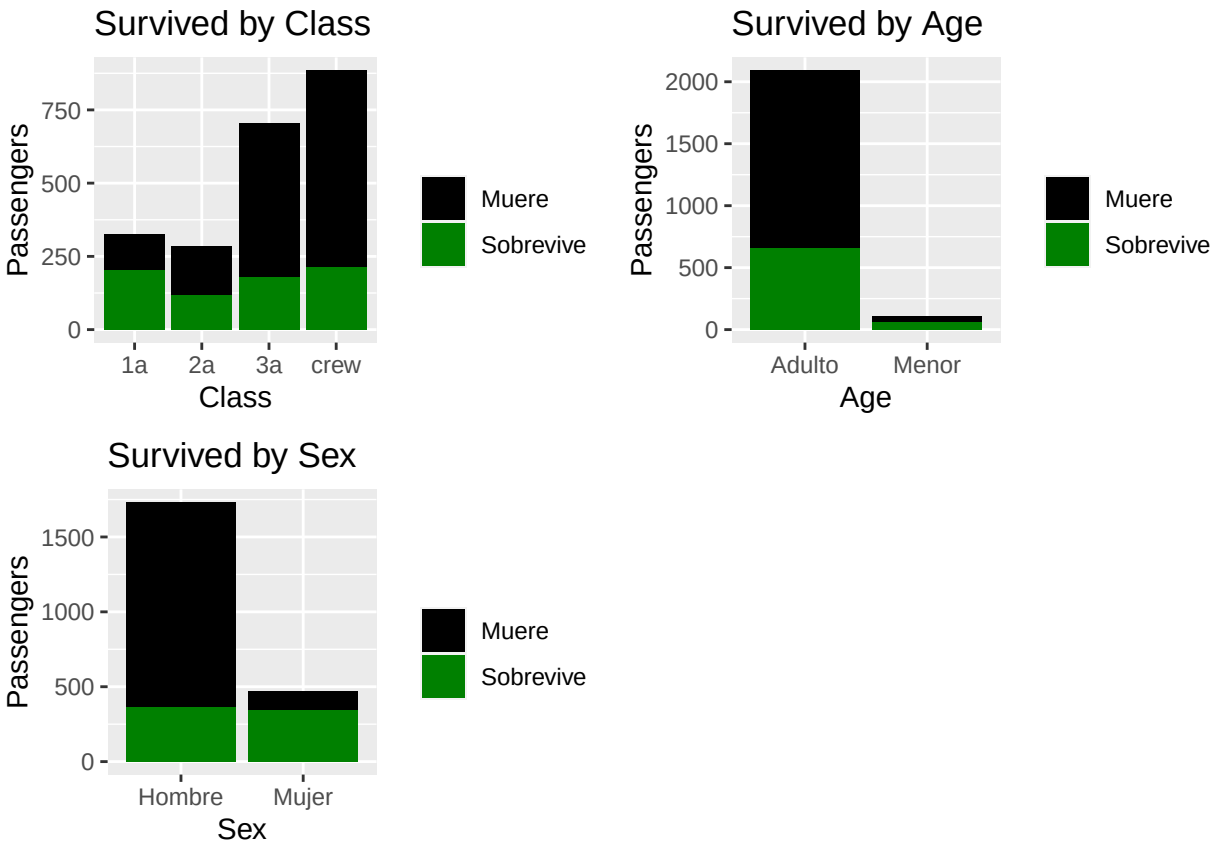
```
grid.newpage()
plotbyClass<-ggplot(data,aes(CLASS))+geom_bar() +labs(x="Class", y="Passengers")+ guides(fill=guide_leg
plotbyAge<-ggplot(data,aes(AGE))+geom_bar() +labs(x="Age", y="Passengers")+ guides(fill=guide_legend(ti
plotbySex<-ggplot(data,aes(SEX))+geom_bar() +labs(x="Sex", y="Passengers")+ guides(fill=guide_legend(ti
plotbySurvived<-ggplot(data,aes(SURVIVED))+geom_bar() +labs(x="Survived", y="Passengers")+ guides(fill=
grid.arrange(plotbyClass,plotbyAge,plotbySex,plotbySurvived,ncol=2)
```



Claramente vemos cómo es la muestra analizando la distribución de las variables disponibles. De cara a los informes, es mucho más interesante esta información que la obtenida en summary, que se puede usar para complementar.

Nos interesa describir la relación entre la supervivencia y cada uno de las variables mencionadas anteriormente. Para ello, por un lado graficaremos mediante diagramas de barras la cantidad de muertos y supervivientes según la clase en la que viajaban, la edad o el sexo. Por otro lado, para obtener los datos que estamos graficando utilizaremos el comando table para dos variables que nos proporciona una tabla de contingencia.

```
grid.newpage()
plotbyClass<-ggplot(data,aes(CLASS,fill=SURVIVED))+geom_bar() +labs(x="Class", y="Passengers")+ guides(
plotbyAge<-ggplot(data,aes(AGE,fill=SURVIVED))+geom_bar() +labs(x="Age", y="Passengers")+ guides(fill=gu
plotbySex<-ggplot(data,aes(SEX,fill=SURVIVED))+geom_bar() +labs(x="Sex", y="Passengers")+ guides(fill=gu
grid.arrange(plotbyClass,plotbyAge,plotbySex,ncol=2)
```



De estos gráficos obtenemos información muy valiosa que complementamos con las tablas de contingencia (listadas abajo). Por un lado, la cantidad de pasajeros que sobrevivieron es similar en hombres y mujeres (hombres: 367 y mujeres 344). No, en cambio, si tenemos en cuenta el porcentaje respecto a su sexo. Es decir, pese a que la cantidad de mujeres y hombres que sobrevivieron es pareja, viajaban más hombres que mujeres (470 mujeres y 1731 hombres), por lo tanto, la tasa de muerte en hombres es muchísimo mayor (el 78,79% de los hombres murieron mientras que en mujeres ese porcentaje baja a 26,8%).

En cuanto a la clase en la que viajaban, los pasajeros que viajaban en primera clase fueron los únicos que el porcentaje de supervivencia era mayor que el de mortalidad. El 62,46% de los viajeros de primera clase sobrevivió, el 41,4% de los que viajaban en segunda clase mientras que de los viajeros de tercera y de la tripulación solo sobrevivieron un 25,21% y 23,95% respectivamente. Para finalizar, destacamos que la presencia de pasajeros adultos era mucho mayor que la de los niños (2092 frente a 109) y que la tasa de supervivencia en niños fue mucho mayor (52,29% frente a 31,26%), no podemos obviar, en cambio, que los únicos niños que murieron fueron todos pasajeros de tercera clase (52 niños).

```
tabla_SST <- table(SEX, SURVIVED)
tabla_SST
```

```
##         SURVIVED
## SEX      Muere Sobrevive
##   Hombre  1364       367
##   Mujer    126       344
```

```
prop.table(tabla_SST, margin = 1)
```

```
##         SURVIVED
## SEX          Muere Sobrevive
##   Hombre 0.7879838 0.2120162
##   Mujer  0.2680851 0.7319149
```

```r
tabla_SCT <- table(CLASS,SURVIVED)
tabla_SCT
```

```
##       SURVIVED
## CLASS  Muere Sobrevive
##   1a     122       203
##   2a     167       118
##   3a     528       178
##   crew   673       212
```

```r
prop.table(tabla_SCT, margin = 1)
```

```
##       SURVIVED
## CLASS      Muere Sobrevive
##   1a   0.3753846 0.6246154
##   2a   0.5859649 0.4140351
##   3a   0.7478754 0.2521246
##   crew 0.7604520 0.2395480
```

```r
tabla_SAT <- table(AGE,SURVIVED)
tabla_SAT
```

```
##         SURVIVED
## AGE      Muere Sobrevive
##   Adulto  1438       654
##   Menor     52        57
```

```r
prop.table(tabla_SAT, margin = 1)
```

```
##         SURVIVED
## AGE          Muere Sobrevive
##   Adulto 0.6873805 0.3126195
##   Menor  0.4770642 0.5229358
```

```r
tabla_SAT.byClass <- table(AGE,SURVIVED,CLASS)
tabla_SAT.byClass
```

```
## , , CLASS = 1a
##
##         SURVIVED
## AGE      Muere Sobrevive
##   Adulto   122       197
##   Menor      0         6
##
## , , CLASS = 2a
##
##         SURVIVED
## AGE      Muere Sobrevive
##   Adulto   167        94
##   Menor      0        24
##
## , , CLASS = 3a
##
##         SURVIVED
## AGE      Muere Sobrevive
##   Adulto   476       151
##   Menor     52        27
```

```
##
## , , CLASS = crew
##
##          SURVIVED
## AGE       Muere Sobrevive
##   Adulto   673       212
##   Menor      0         0
```

Test estadísticos de significancia

Los resultados anteriores muestran los datos de forma descriptiva, podemos añadir algún test estadístico para validar el grado de significancia de la relación. La librería "DescTools" nos permite instalarlo fácilmente.

```
if(!require(DescTools)){
    install.packages('DescTools', repos='http://cran.us.r-project.org')
    library(DescTools)
}
```

```
## Loading required package: DescTools
```

```
Phi(tabla_SST)
```

```
## [1] 0.4556048
```

```
CramerV(tabla_SST)
```

```
## [1] 0.4556048
```

```
Phi(tabla_SAT)
```

```
## [1] 0.09757511
```

```
CramerV(tabla_SAT)
```

```
## [1] 0.09757511
```

```
Phi(tabla_SCT)
```

```
## [1] 0.2941201
```

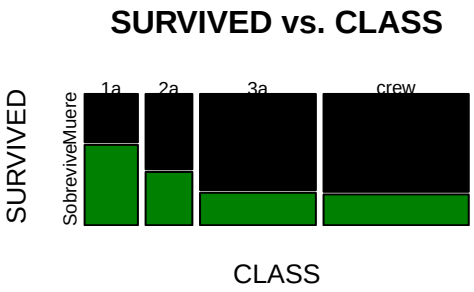
```
CramerV(tabla_SCT)
```

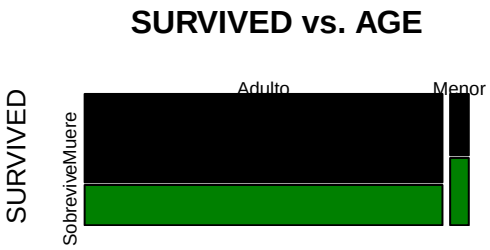
```
## [1] 0.2941201
```

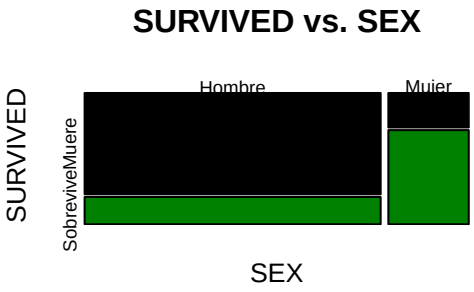
Valores de la V de Cramér (https://en.wikipedia.org/wiki/Cramér%27s_V) y Phi (https://en.wikipedia.org/wiki/Phi_coefficient) entre 0.1 y 0.3 nos indican que la asociación estadística es baja, y entre 0.3 y 0.5 se puede considerar una asociación media. Finalmente, si los valores fueran superiores a 0.5 (no es el caso), la asociación estadística entre las variables sería alta. Como se puede apreciar, los valores de Phi y V coinciden. Esto ocurre en el contexto de analizar tablas de contingencia 2x2.

Una alternativa interesante a las barras de diagramas, es el plot de las tablas de contingencia. Obtenemos la misma información pero para algunos receptores puede resultar más visual.

```
par(mfrow=c(2,2))
plot(tabla_SCT, col = c("black","#008000"), main = "SURVIVED vs. CLASS")
plot(tabla_SAT, col = c("black","#008000"), main = "SURVIVED vs. AGE")
plot(tabla_SST, col = c("black","#008000"), main = "SURVIVED vs. SEX")
```







Nuestro objetivo es crear un árbol de decisión que permita analizar qué tipo de pasajero del Titanic tenía probabilidades de sobrevivir o no. Por lo tanto, la variable por la que clasificaremos es el campo de si el pasajero sobrevivió o no. De todas maneras, al imprimir las primeras (con head) y últimas 10 (con tail) filas nos damos cuenta de que los datos están ordenados.

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

```
##    CLASS    AGE    SEX  SURVIVED
## 1     1a Adulto Hombre Sobrevive
## 2     1a Adulto Hombre Sobrevive
## 3     1a Adulto Hombre Sobrevive
## 4     1a Adulto Hombre Sobrevive
## 5     1a Adulto Hombre Sobrevive
## 6     1a Adulto Hombre Sobrevive
## 7     1a Adulto Hombre Sobrevive
## 8     1a Adulto Hombre Sobrevive
## 9     1a Adulto Hombre Sobrevive
## 10    1a Adulto Hombre Sobrevive
```

```
tail(data,10)
```

```
##      CLASS    AGE   SEX  SURVIVED
## 2192  crew Adulto Mujer Sobrevive
## 2193  crew Adulto Mujer Sobrevive
## 2194  crew Adulto Mujer Sobrevive
## 2195  crew Adulto Mujer Sobrevive
## 2196  crew Adulto Mujer Sobrevive
## 2197  crew Adulto Mujer Sobrevive
## 2198  crew Adulto Mujer Sobrevive
## 2199  crew Adulto Mujer     Muere
## 2200  crew Adulto Mujer     Muere
## 2201  crew Adulto Mujer     Muere
```

Preparación de los datos para el modelo

Para la futura evaluación del árbol de decisión, es necesario dividir el conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba. El conjunto de entrenamiento es el subconjunto del conjunto original de datos utilizado para construir un primer modelo; y el conjunto de prueba, el subconjunto del conjunto original de datos utilizado para evaluar la calidad del modelo.

Lo más correcto será utilizar un conjunto de datos diferente del que utilizamos para construir el árbol, es decir, un conjunto diferente del de entrenamiento. No hay ninguna proporción fijada con respecto al número relativo de componentes de cada subconjunto, pero la más utilizada acostumbra a ser 2/3 para el conjunto de entrenamiento y 1/3, para el conjunto de prueba.

La variable por la que clasificaremos es el campo de si el pasajero sobrevivió o no, que está en la cuarta columna. De esta forma, tendremos un conjunto de datos para el entrenamiento y uno para la validación

```
set.seed(666)
y <- data[,4]
X <- data[,1:3]
```

De forma dinámica podemos definir una forma de separar los datos en función de un parámetro. Así, definimos un parámetro que controla el split de forma dinámica en el test.

```
split_prop <- 3
indexes = sample(1:nrow(data), size=floor(((split_prop-1)/split_prop)*nrow(data)))
trainX<-X[indexes,]
trainy<-y[indexes]
testX<-X[-indexes,]
testy<-y[-indexes]
```

Después de una extracción aleatoria de casos es altamente recomendable efectuar un análisis de datos mínimo para asegurarnos de no obtener clasificadores sesgados por los valores que contiene cada muestra. En este caso, verificaremos que la proporción del supervivientes es más o menos constante en los dos conjuntos:

```
summary(trainX);
```

```
##   CLASS         AGE           SEX
##  1a  :208   Adulto:1395   Hombre:1153
##  2a  :185   Menor :  72   Mujer : 314
##  3a  :477
##  crew:597
```

```
summary(trainy)
```

```
##     Muere Sobrevive
##       997       470
```

```
summary(testX)
```

```
##   CLASS         AGE          SEX
##  1a  :117   Adulto:697   Hombre:578
##  2a  :100   Menor : 37   Mujer :156
##  3a  :229
##  crew:288
```

```
summary(testy)
```

```
##     Muere Sobrevive
##       493       241
```

Verificamos fácilmente que no hay diferencias graves que puedan sesgar las conclusiones.

Creación del modelo, calidad del modelo y extracción de reglas

Se crea el árbol de decisión usando los datos de entrenamiento (no hay que olvidar que la variable outcome es de tipo factor):

```
trainy <-  as.factor(trainy)
model <- C50::C5.0(trainX, trainy,rules=TRUE )
summary(model)
```

```
##
## Call:
## C5.0.default(x = trainX, y = trainy, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Feb  4 22:52:34 2024
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 1467 cases (4 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (1153/243, lift 1.2)
##  SEX = Hombre
##  ->  class Muere  [0.789]
##
## Rule 2: (477/123, lift 1.1)
##  CLASS = 3a
##  ->  class Muere  [0.741]
##
## Rule 3: (178/15, lift 2.8)
##  CLASS in {1a, 2a, crew}
##  SEX = Mujer
##  ->  class Sobrevive  [0.911]
##
## Default class: Muere
##
##
## Evaluation on training data (1467 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       3  322(21.9%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     982    15    (a): class Muere
##     307   163    (b): class Sobrevive
##
##
##  Attribute usage:
```

```
##
##   90.73% SEX
##   44.65% CLASS
##
##
## Time: 0.0 secs
```

Errors muestra el número y porcentaje de casos mal clasificados en el subconjunto de entrenamiento. El árbol obtenido clasifica erróneamente 322 de los 1467 casos dados, una tasa de error del 21.9%.

A partir del árbol de decisión de dos hojas que hemos modelado, se pueden extraer las siguientes reglas de decisión (gracias a rules=TRUE podemos imprimir las reglas directamente):

SEX = "Hombre" → Muere. Validez: 78,9%

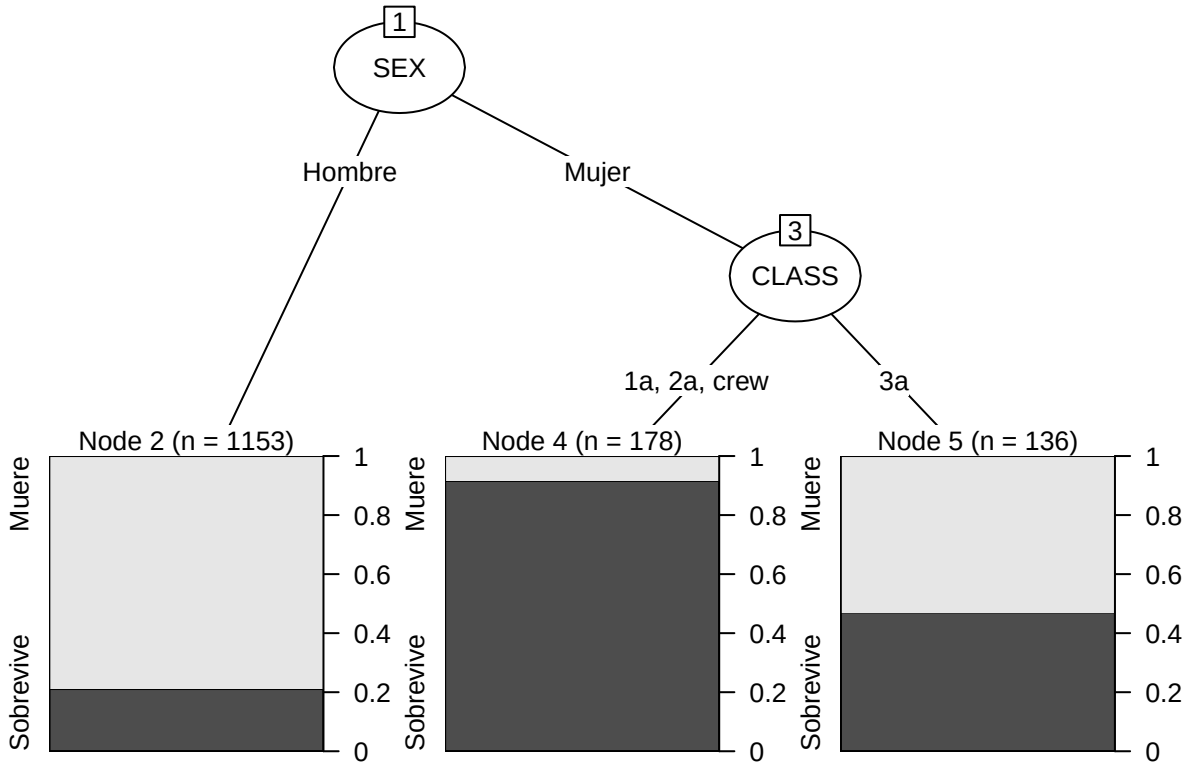
CLASS "3ª" → Muere. Validez: 74,1%

CLASS "1ª", "2ª", "crew" y SEX = "Mujer" → Sobrevive. Validez: 91,1%

Por tanto, podemos concluir que el conocimiento extraído y cruzado con el análisis visual se resume en "las mujeres y los niños primero a excepción de que fueras de 3ª clase".

A continuación, mostramos el árbol obtenido.

```
model <- C50::C5.0(trainX, trainy)
plot(model,gp = gpar(fontsize = 9.5))
```



Validación del modelo con los datos reservados

Una vez tenemos el modelo, podemos comprobar su calidad prediciendo la clase para los datos de prueba que nos hemos reservado al principio.

```
predicted_model <- predict( model, testX, type="class" )
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_model == testy) / length(predicted_
```

```
## [1] "La precisión del árbol es: 78.8828 %"
```

Cuando hay pocas clases, la calidad de la predicción se puede analizar mediante una matriz de confusión que identifica los tipos de errores cometidos.

```
mat_conf<-table(testy,Predicted=predicted_model)
mat_conf
```

```
##            Predicted
## testy       Muere Sobrevive
##   Muere       488         5
##   Sobrevive   150        91
```

Otra manera de calcular el porcentaje de registros correctamente clasificados usando la matriz de confusión:

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

```
## [1] "El % de registros correctamente clasificados es: 78.8828 %"
```

Además, tenemos a nuestra disposición el paquete gmodels para obtener información más completa:

```
if(!require(gmodels)){
    install.packages('gmodels', repos='http://cran.us.r-project.org')
    library(gmodels)
}
```

```
## Loading required package: gmodels
```

```
## Registered S3 method overwritten by 'gdata':
##   method         from
##   reorder.factor DescTools
```

```
CrossTable(testy, predicted_model,prop.chisq  = FALSE, prop.c = FALSE, prop.r =FALSE,dnn = c('Reality',
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  734
##
##
##              | Prediction
##      Reality |     Muere | Sobrevive | Row Total |
## -------------|-----------|-----------|-----------|
##        Muere |       488 |         5 |       493 |
##              |     0.665 |     0.007 |           |
## -------------|-----------|-----------|-----------|
##    Sobrevive |       150 |        91 |       241 |
##              |     0.204 |     0.124 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       638 |        96 |       734 |
## -------------|-----------|-----------|-----------|
##
```

##

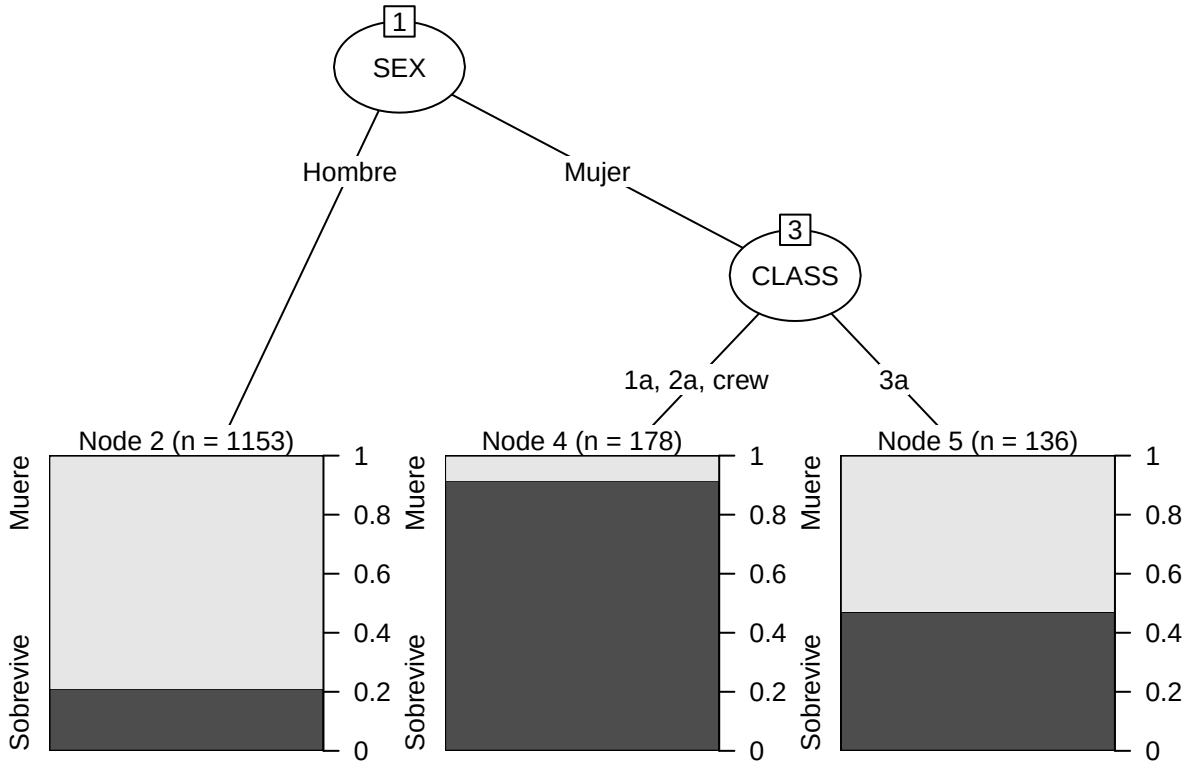
Prueba con una variación u otro enfoque algorítmico

Variaciones del paquete C5.0

En este apartado buscaremos probar con las variaciones que nos ofrece el paquete C5.0 para analizar cómo afectan a la creación de los árboles generados. Existen muchas posibles variaciones con otras funciones que podéis investigar. La idea es seguir con el enfoque de árboles de decisión explorando posibles opciones. Una vez tengamos un método alternativo, debemos analizar cómo se modifica el árbol y cómo afecta a la capacidad predictiva en el conjunto de test.

A continuación, utilizamos otro enfoque para comparar los resultados: incorpora como novedad "adaptative boosting", basado en el trabajo Rob Schapire and Yoav Freund (1999). La idea de esta técnica es generar varios clasificadores, con sus correspondientes arboles de decisión y su ser de reglas. Cuando un nuevo caso va a ser clasificado, cada clasificador vota cual es la clase predicha. Los votos son sumados y determina la clase final.

```
modelo2 <- C50::C5.0(trainX, trainy, trials = 10)
plot(modelo2,gp = gpar(fontsize = 9.5))
```



En este caso, dada la simplicidad del conjunto de ejemplo, no se aprecian diferencias, pero aparecerán en datos de mayor complejidad y modificando el parámetro "trials" se puede intentar mejorar los resultados.

Vemos a continuación cómo son las predicciones del nuevo árbol:

```
predicted_model2 <- predict( modelo2, testX, type="class" )
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_model2 == testy) / length(predicte
```

```
## [1] "La precisión del árbol es: 75.0681 %"
```

Observamos como se modifica levemente la precisión del modelo a mejor.

```
mat_conf<-table(testy,Predicted=predicted_model2)
mat_conf
```

```
##            Predicted
## testy       Muere Sobrevive
##   Muere       438        55
##   Sobrevive   128       113
```

Otra manera de calcular el porcentaje de registros correctamente clasificados usando la matriz de confusión:

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

```
## [1] "El % de registros correctamente clasificados es: 75.0681 %"
```

El algoritmo C5.0 incorpora algunas opciones para ver la importancia de las variables (ver documentación para los detalles entre los dos métodos):

```
importancia_usage <- C50::C5imp(modelo2, metric = "usage")
importancia_splits <- C50::C5imp(modelo2, metric = "splits")
importancia_usage
```

```
##       Overall
## CLASS  100.00
## SEX    100.00
## AGE     93.73
```

```
importancia_splits
```

```
##       Overall
## CLASS      40
## SEX        40
## AGE        20
```

Curiosamente y aunque el conjunto de datos es muy sencillo, se aprecian diferencias en los métodos de importancia de las variables. Se recomienda en vuestro ejercicio mejorar la visualización de los resultados con la función ggplo2 o similar.

Interpretación de las variables en las predicciones.

Nos interesa saber para las predicciones que variable son las que tienen más influencia. Así, probaremos con un enfoque algorítmico de Random Forest y obtendremos métricas de interpretabilidad con la librería IML (https://cran.r-project.org/web/packages/iml/iml.pdf). As:

```
if(!require(randomForest)){
  install.packages('randomForest',repos='http://cran.us.r-project.org')
  library(randomForest)
}
```

```
## Loading required package: randomForest
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:gridExtra':
##
```

```
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
if(!require(iml)){
  install.packages('iml', repos='http://cran.us.r-project.org')
  library(iml)
}
```

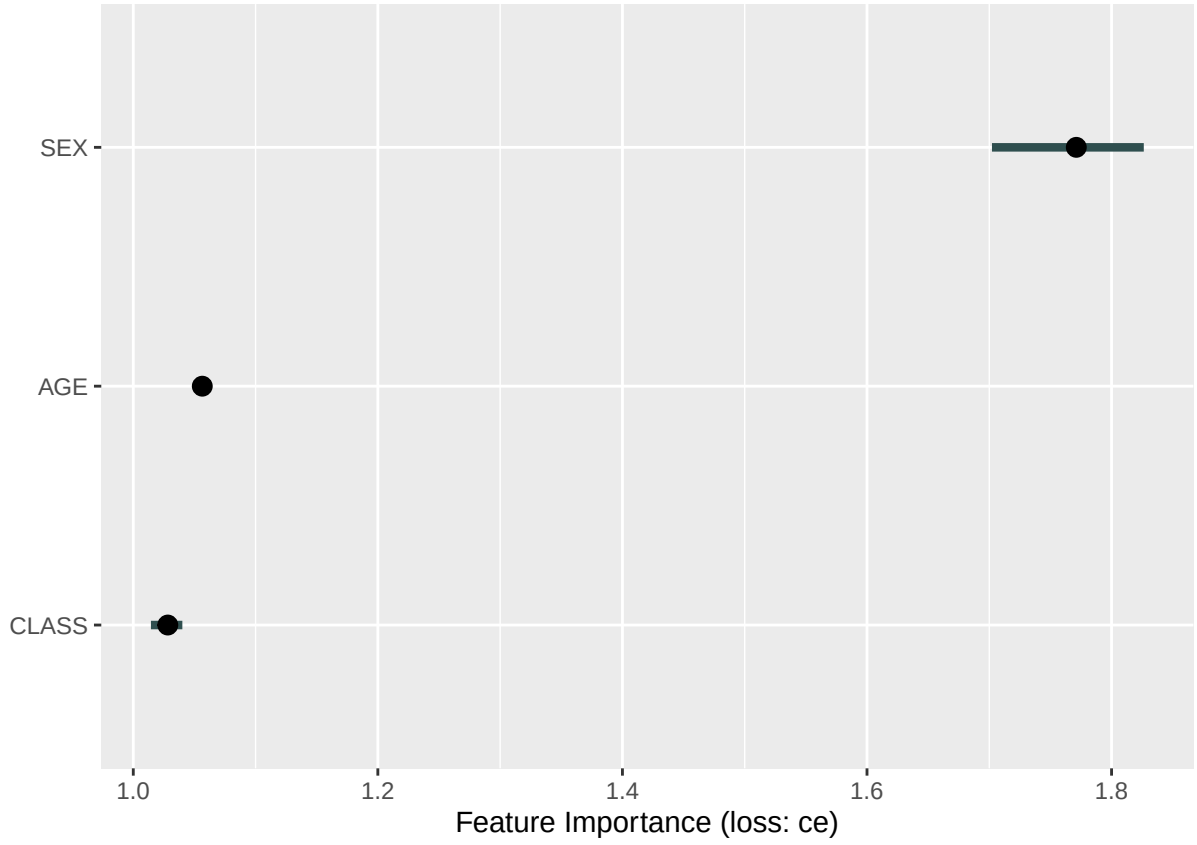
```
## Loading required package: iml
```

Empezamos ejecutado un Random Forest:

```
train.data <- as.data.frame(cbind(trainX,trainy))
colnames(train.data)[4] <- "SURVIVED"
rf <-  randomForest(SURVIVED ~ ., data = train.data, ntree = 50)
```

Podemos medir y graficar la importancia de cada variable para las predicciones del random forest con *FeatureImp*. La medida se basa funciones de pérdida de rendimiento que en nuestro caso será con el objetivo de clasificación ("ce").

```
X <- train.data[which(names(train.data) != "SURVIVED")]
predictor <- Predictor$new(rf, data = X, y = train.data$SURVIVED)
imp <- FeatureImp$new(predictor, loss = "ce")
plot(imp)
```




```
imp$results
```

```
##   feature importance.05 importance importance.95 permutation.error
```

```
## 1     SEX      1.702194   1.771160      1.826332         0.3851397
## 2     AGE      1.053292   1.056426      1.062069         0.2297205
## 3   CLASS      1.014420   1.028213      1.040125         0.2235855
```

Adicionalmente, podemos también dibujar los efectos locales acumulados (ALE) de la variable usando la libreria *patchwork*:

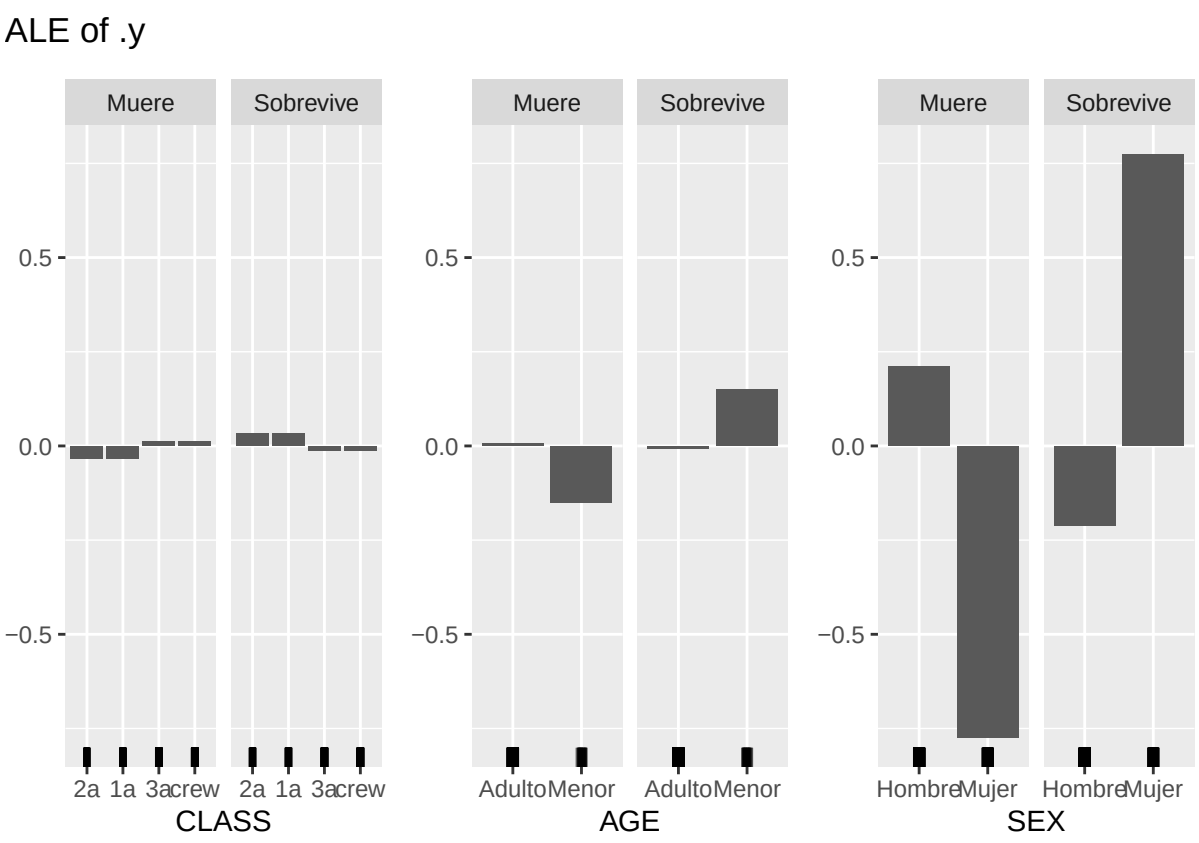
```
if(!require(patchwork)){
    install.packages('patchwork',repos='http://cran.us.r-project.org')
    library(patchwork)
}
```

```
## Loading required package: patchwork
```

```
##
## Attaching package: 'patchwork'
```

```
## The following object is masked from 'package:MASS':
##
##     area
```

```
effs <- FeatureEffects$new(predictor)
plot(effs)
```



Como podemos ver, el género es la variable con más importancia para las predicciones, siendo las mujeres mucho más propensas a sobrevivir. Nota: Se espera que los alumnos profundicen en la función de cara a la resolución de los ejercicios.

Enunciado del ejercicio

Para el conjunto de datos German Credit, los alumnos deben completar aquí la solución a la PEC3 que consiste de los siguientes apartados. Notad que se detalla el contenido necesario para cada apartado en la Sección 4 (Rúbrica).

El formato de entrega es como en las anteriores PECs: **usernameestudiant-PECn.html** (o PDF/Word) y el código **Rmd**.

Se debe entregar la PEC en el buzón de entregas del aula, como en las anteriores PECs.

Realizar un primer análisis descriptivo y de correlaciones. Es importante en este apartado entender bien los datos antes de seguir con los análisis posteriores. Lista todo lo que te haya sorprendido de los datos

En esta sección se va a llevar a cabo el análisis de los datos que vamos a utilizar. Primero se empezará con un análisis más general, para luego ir profundizando, según los datos vayan mostrando más información y características.

Como siempre, primero se comienza cargando los datos en el siguiente *chunk* de código y se visualizan las dimensiones del juego de datos:

```
nombre_archivo = './credit-2.csv'
df_original <- read.csv(nombre_archivo)
dim(df_original)
```

```
## [1] 1000   21
```

```
head(df_original)
```

```
##   checking_balance months_loan_duration credit_history   purpose amount
## 1           < 0 DM                    6       critical  radio/tv   1169
## 2       1 - 200 DM                   48         repaid  radio/tv   5951
## 3          unknown                   12       critical education   2096
## 4           < 0 DM                   42         repaid furniture   7882
## 5           < 0 DM                   24        delayed car (new)   4870
## 6          unknown                   36         repaid education   9055
##   savings_balance employment_length installment_rate personal_status
## 1         unknown           > 7 yrs                4     single male
## 2        < 100 DM         1 - 4 yrs                2          female
## 3        < 100 DM         4 - 7 yrs                2     single male
## 4        < 100 DM         4 - 7 yrs                2     single male
## 5        < 100 DM         1 - 4 yrs                3     single male
## 6         unknown         1 - 4 yrs                2     single male
##   other_debtors residence_history                property age installment_plan
## 1          none                 4             real estate  67             none
## 2          none                 2             real estate  22             none
## 3          none                 3             real estate  49             none
## 4     guarantor                 4 building society savings  45             none
## 5          none                 4            unknown/none  53             none
## 6          none                 4            unknown/none  35             none
##    housing existing_credits default dependents telephone foreign_worker
## 1      own                2       1          1       yes            yes
## 2      own                1       2          1      none            yes
## 3      own                1       1          2      none            yes
## 4 for free                1       1          2      none            yes
## 5 for free                2       2          2      none            yes
```

```
## 6 for free               1       1          2       yes            yes
##                  job
## 1   skilled employee
## 2   skilled employee
## 3 unskilled resident
## 4   skilled employee
## 5   skilled employee
## 6 unskilled resident
```

```
tail(df_original)
```

```
##      checking_balance months_loan_duration credit_history    purpose amount
## 995           unknown                   12         repaid  car (new)   2390
## 996           unknown                   12         repaid  furniture   1736
## 997            < 0 DM                   30         repaid car (used)   3857
## 998           unknown                   12         repaid   radio/tv    804
## 999            < 0 DM                   45         repaid   radio/tv   1845
## 1000       1 - 200 DM                   45       critical car (used)   4576
##      savings_balance employment_length installment_rate personal_status
## 995          unknown           > 7 yrs                4     single male
## 996         < 100 DM         4 - 7 yrs                3          female
## 997         < 100 DM         1 - 4 yrs                4   divorced male
## 998         < 100 DM           > 7 yrs                4     single male
## 999         < 100 DM         1 - 4 yrs                4     single male
## 1000    101 - 500 DM        unemployed                3     single male
##      other_debtors residence_history                property age
## 995           none                 3                   other  50
## 996           none                 4              real estate  31
## 997           none                 4 building society savings  40
## 998           none                 4                   other  38
## 999           none                 4             unknown/none  23
## 1000          none                 4                   other  27
##      installment_plan  housing existing_credits default dependents telephone
## 995              none      own                1       1          1       yes
## 996              none      own                1       1          1      none
## 997              none      own                1       1          1       yes
## 998              none      own                1       1          1      none
## 999              none for free                1       2          1       yes
## 1000             none      own                1       1          1      none
##      foreign_worker                     job
## 995             yes        skilled employee
## 996             yes      unskilled resident
## 997             yes mangement self-employed
## 998             yes        skilled employee
## 999             yes        skilled employee
## 1000            yes        skilled employee
```

```
# aAhora hacemos una copia del DF
df_copia = df_original
```

Como se puede observar por el resultado de arriba, el juego de datos alberga 1000 registros y 21 variables, i.e., 21 columnas. En el caso de este juego de datos, y por lo que se ha podido ver arriba con las funciones `head(·)` y `tail(·)`, los 1000 registros se pueden corresponder con los clientes de un banco. Digo se "puede", porque he investigado acerca del *dataset* en el link proporcionado, pero en la página de Kaggle no hay ninguna descripción a cerca de los datos, ya sea respecto a su origen o la información que representan cada

una de sus columnas.

Ya sabemos el número de registros (1000) y el número de variables que caracterizan a cada uno de ellos (21). Ahora hay que investigar que tipo de variables, representan cada una de las 21 columans del juego de datos, para ello se hace uso una función que hemos utilizado en las tres entregas anteriores de la asignatura: `str(·)`. Véase a continuación la información extraída por esta función, a partir de nuestro juego de datos:

```
str(df_original)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ checking_balance    : chr  "< 0 DM" "1 - 200 DM" "unknown" "< 0 DM" ...
##  $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credit_history      : chr  "critical" "repaid" "critical" "repaid" ...
##  $ purpose             : chr  "radio/tv" "radio/tv" "education" "furniture" ...
##  $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ savings_balance     : chr  "unknown" "< 100 DM" "< 100 DM" "< 100 DM" ...
##  $ employment_length   : chr  "> 7 yrs" "1 - 4 yrs" "4 - 7 yrs" "4 - 7 yrs" ...
##  $ installment_rate    : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ personal_status     : chr  "single male" "female" "single male" "single male" ...
##  $ other_debtors       : chr  "none" "none" "none" "guarantor" ...
##  $ residence_history   : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ property            : chr  "real estate" "real estate" "real estate" "building society savings" .
##  $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ installment_plan    : chr  "none" "none" "none" "none" ...
##  $ housing             : chr  "own" "own" "own" "for free" ...
##  $ existing_credits    : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ default             : int  1 2 1 1 2 1 1 1 1 2 ...
##  $ dependents          : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone           : chr  "yes" "none" "none" "none" ...
##  $ foreign_worker      : chr  "yes" "yes" "yes" "yes" ...
##  $ job                 : chr  "skilled employee" "skilled employee" "unskilled resident" "skilled em
```

Ahora se obtienen las métricas más generales de cada una de las variables con la función `summary(·)`, véase el siguiente *chunk* de código, implementando esta función:

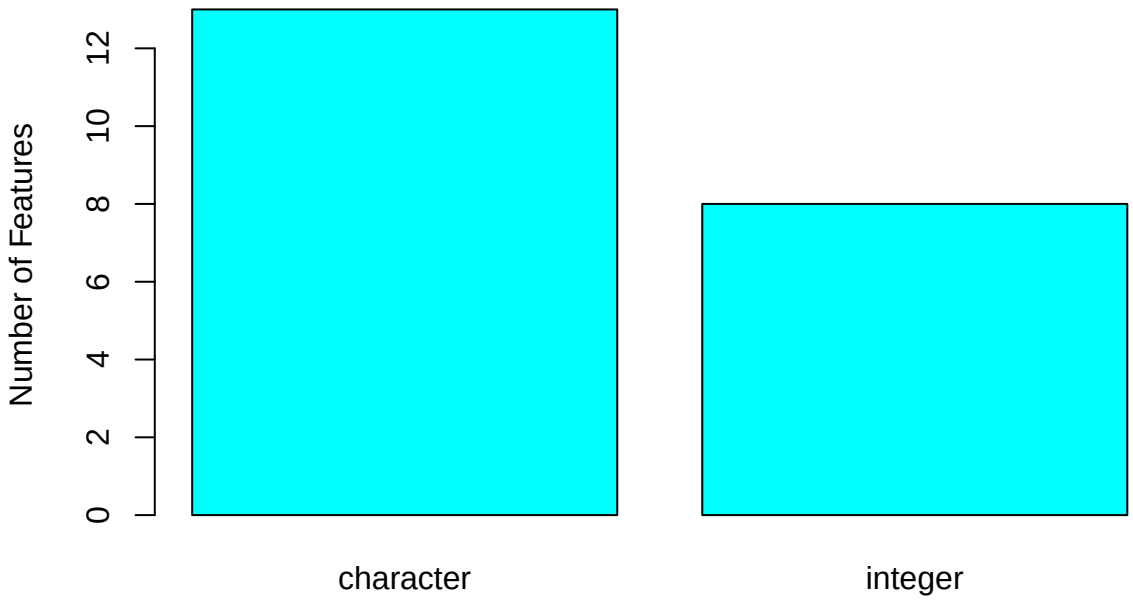
```
summary(df_original)
```

```
##  checking_balance   months_loan_duration credit_history       purpose
##  Length:1000        Min.   : 4.0         Length:1000        Length:1000
##  Class :character   1st Qu.:12.0         Class :character   Class :character
##  Mode  :character   Median :18.0         Mode  :character   Mode  :character
##                     Mean   :20.9
##                     3rd Qu.:24.0
##                     Max.   :72.0
##      amount      savings_balance    employment_length  installment_rate
##  Min.   :  250   Length:1000        Length:1000        Min.   :1.000
##  1st Qu.: 1366   Class :character   Class :character   1st Qu.:2.000
##  Median : 2320   Mode  :character   Mode  :character   Median :3.000
##  Mean   : 3271                                         Mean   :2.973
##  3rd Qu.: 3972                                         3rd Qu.:4.000
##  Max.   :18424                                         Max.   :4.000
##  personal_status    other_debtors      residence_history   property
##  Length:1000        Length:1000        Min.   :1.000     Length:1000
##  Class :character   Class :character   1st Qu.:2.000     Class :character
##  Mode  :character   Mode  :character   Median :3.000     Mode  :character
##                                        Mean   :2.845
```

```
##                                        3rd Qu.:4.000
##                                        Max.   :4.000
##       age        installment_plan     housing          existing_credits
##  Min.   :19.00   Length:1000        Length:1000        Min.   :1.000
##  1st Qu.:27.00   Class :character   Class :character   1st Qu.:1.000
##  Median :33.00   Mode  :character   Mode  :character   Median :1.000
##  Mean   :35.55                                         Mean   :1.407
##  3rd Qu.:42.00                                         3rd Qu.:2.000
##  Max.   :75.00                                         Max.   :4.000
##     default      dependents     telephone         foreign_worker
##  Min.   :1.0   Min.   :1.000   Length:1000        Length:1000
##  1st Qu.:1.0   1st Qu.:1.000   Class :character   Class :character
##  Median :1.0   Median :1.000   Mode  :character   Mode  :character
##  Mean   :1.3   Mean   :1.155
##  3rd Qu.:2.0   3rd Qu.:1.000
##  Max.   :2.0   Max.   :2.000
##      job
##  Length:1000
##  Class :character
##  Mode  :character
##
##
##
```

Como se puede observar, el juego de datos solo contempla dos tipos de variables: `chr` e `int`, en el siguiente *chunk*, se estudia cual de las dos variables es más común en el *dataset*.

```
tipos_de_datos <- function(df_original) {
  res <- lapply(df_original, class)
  res_frame <- data.frame(unlist(res))
  barplot(table(res_frame), main="Tipos de datos", col="cyan", ylab="Number of Features")
}

tipos_de_datos(df_original)
```



Como se puede observar por el gráfico de arriba, hay más columnas/variables de tipo `chr` (13) que de tipo `int` (8), por lo que habrá que ver como se van a tratar a la hora de representarlas o modificarlas, a fin de extaer información de la relación entre ellas y las variables de tipo `int`. Pues, a veces, considero que es mucho más fácil trabajar con variables numéricas que con variables de tipo `chr` pues en el primer tipo, uno puede calcular medias, medianas, rangos de valores, y consecuentemente, simplificar información a partir de un conjunto de números. Mientras que los métodos que se acaban de mencionar no siempre aplican a variables de tipo `chr` ya que estas contienen texto y para poder aplicar dichos métodos, o relacionar este tipo de variables con unas de otro tipo, hay que transformarlas a otro formato. De hecho, creo que trabajar con variables numéricas, potencia el análisis del juego de datos, obligando al programador a indagar más y más, hasta pulir cada una de las variables, mientras que una variable `chr` no ofrece tantas posibilidades a nivel de transformaciones. Esto no quiere decir, que las variables `chr` sean 'peores'. De hecho, en muchas ocasiones, estas ayudan a simplificar y pueden aumentar la velocidad con la que se extrae conocimiento, es por esto, que para poder simplificar y facilitar la comprensón del conocimiento e información que un juego de datos puede contener, muchas veces, hay que transformar variables numéricas a variables de tipo `chr`, e.g., SI o NO, BIEN o MAL, NEGRO o BLANCO, ALTO o MEDIO o BAJO.

Debido a la explicación anterior y conectando con uno de los *chunks* en un ejemplo de esta PEC, con el fin de facilitar y hacer más eficiente el análisis estadístico de las variables, se van a transformar las variables de tipo `chr` a variables categóricas. Es pertinente mencionar, que a parte de lo ya explicado, esta transformación trae numerosas ventajas, relacionadas ya no solo con el análisis estadístico, sino con los níveles de valores de la variable, o la visualización de la misma. Es importante señalar además, que al convertir una variable `chr` a categórica, los níveles de esta nueva variable se asignarán de manera alfabética, a no ser que se determine lo contrario, pues en caso de que los niveles de valores sean relevanteso importantes, el programador podrá modificarlos. Véase dicha transformación y la posterior verificación, en el siguiente *chunk*

```
df_original[] <- lapply(df_original, factor)
str(df_original)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4 1 1 4 4 3 4 3 ...
##  $ months_loan_duration: Factor w/ 33 levels "4","5","6","7",..: 3 30 9 27 18 24 18 24 9 22 ...
##  $ credit_history      : Factor w/ 5 levels "critical","delayed",..: 1 5 1 5 2 5 5 5 5 1 ...
##  $ purpose             : Factor w/ 10 levels "business","car (new)",..: 8 8 5 6 2 5 6 3 8 2 ...
```

```
##  $ amount              : Factor w/ 921 levels "250","276","338",..: 143 771 391 849 735 870 534 814 5
##  $ savings_balance     : Factor w/ 5 levels "< 100 DM","> 1000 DM",..: 5 1 1 1 1 5 4 1 2 1 ...
##  $ employment_length   : Factor w/ 5 levels "> 7 yrs","0 - 1 yrs",..: 1 3 4 4 3 3 1 3 4 5 ...
##  $ installment_rate    : Factor w/ 4 levels "1","2","3","4": 4 2 2 2 3 2 3 2 2 4 ...
##  $ personal_status     : Factor w/ 4 levels "divorced male",..: 4 2 4 4 4 4 4 4 1 3 ...
##  $ other_debtors       : Factor w/ 3 levels "co-applicant",..: 3 3 3 2 3 3 3 3 3 3 ...
##  $ residence_history   : Factor w/ 4 levels "1","2","3","4": 4 2 3 4 4 4 4 2 4 2 ...
##  $ property            : Factor w/ 4 levels "building society savings",..: 3 3 3 1 4 4 1 2 3 2 ...
##  $ age                 : Factor w/ 53 levels "19","20","21",..: 49 4 31 27 35 17 35 17 43 10 ...
##  $ installment_plan    : Factor w/ 3 levels "bank","none",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ housing             : Factor w/ 3 levels "for free","own",..: 2 2 2 1 1 1 2 3 2 2 ...
##  $ existing_credits    : Factor w/ 4 levels "1","2","3","4": 2 1 1 1 2 1 1 1 1 2 ...
##  $ default             : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...
##  $ dependents          : Factor w/ 2 levels "1","2": 1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone           : Factor w/ 2 levels "none","yes": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreign_worker      : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ job                 : Factor w/ 4 levels "mangement self-employed",..: 2 2 4 2 2 4 2 1 4 1 ...
```

Gracias a esta transformación, a la hora de aplicar la función `srt(·)` uno puede ver los posibles valores que todas las variables pueden tomar, no obstante esto se estudiará más adelante, a nivel individual para cada variable, y para cada relación de variables. Por ahora continuamos con un análisis más general.

Ya conocidos los dos tipos de variables, y su proporción, procedemos a extraer los valores más comúnes que estas toman, así como sus estadísticas más generales, con la ayuda de la función `summary(·)`, véase el siguiente *chunk*

```
cat("\n Ahora se muestra el summary después de la transformación\n\n")
```

```
##
##  Ahora se muestra el summary después de la transformación
```

```
summary(df_original)
```

```
##    checking_balance months_loan_duration                credit_history
##  < 0 DM    :274     24     :184          critical              :293
##  > 200 DM  : 63     12     :179          delayed               : 88
##  1 - 200 DM:269     18     :113          fully repaid          : 40
##  unknown   :394     36     : 83          fully repaid this bank: 49
##                     6      : 75          repaid                :530
##                     15     : 64
##                     (Other):302
##        purpose        amount         savings_balance  employment_length
##  radio/tv  :280   1258   :  3   < 100 DM     :603    > 7 yrs   :253
##  car (new) :234   1262   :  3   > 1000 DM    : 48    0 - 1 yrs :172
##  furniture :181   1275   :  3   101 - 500 DM :103    1 - 4 yrs :339
##  car (used):103   1393   :  3   501 - 1000 DM: 63    4 - 7 yrs :174
##  business  : 97   1478   :  3   unknown      :183    unemployed: 62
##  education : 50   433    :  2
##  (Other)   : 55   (Other):983
##  installment_rate      personal_status      other_debtors residence_history
##  1:136            divorced male: 50    co-applicant: 41   1:130
##  2:231            female       :310    guarantor   : 52   2:308
##  3:157            married male : 92    none        :907   3:149
##  4:476            single male  :548                       4:413
##
##
```

```
##
##                      property        age      installment_plan     housing
##  building society savings:232   27     : 51   bank  :139       for free:108
##  other                   :332   26     : 50   none  :814       own     :713
##  real estate             :282   23     : 48   stores: 47       rent    :179
##  unknown/none            :154   24     : 44
##                                 28     : 43
##                                 25     : 41
##                                 (Other):723
##  existing_credits default dependents telephone  foreign_worker
##  1:633            1:700   1:845      none:596   no : 37
##  2:333            2:300   2:155      yes :404   yes:963
##  3: 28
##  4:  6
##
##
##
##                       job
##  mangement self-employed:148
##  skilled employee       :630
##  unemployed non-resident: 22
##  unskilled resident     :200
##
##
##
```

Teniendo esta información, y debido a la falta de información descriptiva del juego de datos en su web de kaggle: https://www.kaggle.com/shravan3273/credit-approval, a continuación se van a interpretar y describir las variables de la mejor manera posible:

- `checking_balance`: es más que probable, que esta variable determine el saldo deudor del cliente. En el caso de esta variable: $checking\_balance \in (\backslash\!\!> 0\ \text{DM}'', \backslash\!\!> 200\ \text{DM}'', \backslash\ 1 - 200\ \text{DM}'', \backslash\ unknown'')$. Como se ha podido comprobar en Wikipedia: Marco alemán (Deutsche Mark), DM hace referencia a la moneda del marco alemán.
- `months_loan_duration`: el número de meses de duración del crédito. Por lo tanto: $months\_loan\_duration \in \mathbb{R}_+$
- `credit_history`: historial de créditos del cliente. En este caso: $credit\_history \in (\backslash critical'', \backslash\ delayed'', \backslash\ fully\ repaid''$
- `purpose`: propósito del crédito, e.g., coches (nuevos o usados), televisiones, radios, mobiliario, educación, etc.
- `amount`: cantidad de crédito en marcos alemanes (DM).
- `savings_balance`: cantidad de ahorros del cliente. Por lo tanto: $savings\_balance \in (\backslash\!\!< 100\ \text{DM}'', \backslash\!\!> 1000\ \text{DM}'', \backslash\ 101 - 500\ \text{DM}'', \backslash\ 501 - 1000\ \text{DM}'', \backslash\ unknown'')$
- `employment_length`: cantidad de tiempo empleado (años). Por lo tanto: $employment\_length \in (\backslash\!\!> 7\ yrs'', \backslash\ 0 - 1\ yrs'', \backslash\ 1 - 4\ yrs'', \backslash\ 4 - 7\ yrs'', \backslash\ unemployed'')$
- `installment_rate`: cuota de pago o tasa de fraccionamiento del crédito. Es decir, en cuantos meses ha dividido el pago del crédito (imagino que son meses). Por lo tanto: $installment\_rate \in (1, 2, 3, 4)$
- `personal_status`: estado civil del cliente. Por lo tanto: $personal\_status \in (\backslash divorced\ male'', \backslash\ female'', \backslash\ married\ mal$
- `other_debtors`: esta variable proporciona información a cerca de las personas o entidades asociadas con las obligaciones de deuda del prestatario. En este caso, esta variable toma tres valores; ''co-applicant'', "guarantor" o ''none'':''co-applicant" en español, co-deudores, indica que el cliente paga el crédito junto con otra persona, e..g, familiar, amigo, pareja, etc. Luego ''guarantor" en castellano, garante: indica que un tercero proporciona una garantia de pago del préstamo. Y por último, ''none" indica que es el propio cliente, el que asume el pago entero del préstamo.
- `residence_history`: es muy probable que esta variable se refiera al número de direcciones anteriores del cliente. El número de lugares donde ha vivido el cliente durante un período específico. Esta

información es utilizada por instituciones financieras para verificar la identidad de un individuo y evaluar su solvencia crediticia. Esta variable por lo tanto: $residence\_history \in (1, 2, 3, 4)$

- `property`: la propiedad del cliente. Por lo tanto: $property \in (\backslash building\ society\ savings'', \backslash other'', \backslash real\ state'', \backslash unkno$
- `age`: la edad del cliente. Por lo tanto: $age \in \mathbb{R}_+$
- `installment_plan`: como bien indica su traducción en castellano, esta variable indica el modo de pago del préstamo ofrecido al cliente.
- `housing`: tipo de alojamiento del cliente (gratis, en propiedad, alquilado), en este caso, $housing \in (\backslash for\ free'', \backslash own'', \backslash rent'')$
- `existing_credits`: el número de créditos que tiene el cliente. Por lo tanto: $existing\_credits \in \mathbb{R}_+$
- `default`: probablemente esta sea la principal variable que nos permita desplegar el modelo posteriormente, ya que tal y como se define en este enlace, el `default` es el término empleado en economía, para la situación de impago de la deuda, por parte del deudor: $default \in (1, 2)$
- `dependents`: esta variable, determina el número de personas que dependen del cliente que tiene el crédito, ya sean: hijos, conyúge, familiares, amigos, etc. Por lo tanto; $dependents \in \mathbb{R}_+$
- `telephone`: si el cliente tiene o no telefóno: $telephone \in (\backslash yes'', \backslash no'')$
- `foreign_worker`: se trata de un trabajador extranejero, ¿si o no?, $foreign\_worker \in (\backslash yes'', \backslash no'')$
- `job`: tipo de trabajo del cliente: $job \in (\backslash mangement\ \ self-employed'', \backslash skilled\ \ employee'', \backslash unemployed\ \ non-resident'', \backslash inskilled\ \ resident'')$

Aquellas variables (arriba) cuyos conjuntos a los que pertenecen, no se han especificado, es porque dichos conjuntos de valores son bastante grandes y por lo tanto pueden tomar muchos valores.

Ahora que ya conocemos más o menos las variables del *dataframe* procedemos a comprobar si existen valores vacíos o nulos en algunas de las columnas de la base de datos. Primero se van a comprobar si hay valores NULOS:

```
print("Valores NULOS dentro del df_original")
```

```
## [1] "Valores NULOS dentro del df_original"
```

```
colSums(is.na(df_original))
```

```
##     checking_balance months_loan_duration       credit_history
##                    0                    0                    0
##              purpose               amount      savings_balance
##                    0                    0                    0
##    employment_length     installment_rate      personal_status
##                    0                    0                    0
##        other_debtors    residence_history             property
##                    0                    0                    0
##                  age     installment_plan              housing
##                    0                    0                    0
##     existing_credits              default           dependents
##                    0                    0                    0
##            telephone       foreign_worker                  job
##                    0                    0                    0
```

Como se puede comprobar, no existe ningún valor NULO, por lo tanto, una tarea de procesamiento menos que acometer. Ahora que ya sabemos que no hay valores nulos, procedemos a comprobar si existen celdas/valores vacíos en el juego de datos:

```
print("Valores vacíos dentro del df_original")
```

```
## [1] "Valores vacíos dentro del df_original"
```

```
colSums(df_original == '')
```

```
##     checking_balance months_loan_duration       credit_history
```

```
##                   0                   0                   0
##             purpose              amount      savings_balance
##                   0                   0                   0
##    employment_length     installment_rate      personal_status
##                   0                   0                   0
##        other_debtors    residence_history            property
##                   0                   0                   0
##                 age     installment_plan             housing
##                   0                   0                   0
##     existing_credits             default          dependents
##                   0                   0                   0
##            telephone       foreign_worker                 job
##                   0                   0                   0
```

Aunque esta tarea la podemos automatizar un poco más, para no tener que revisar una a una las columnas, véase esta otra implementación:

```r
# Inicializamos los vectores donde vamos a guardar los resultados.
valores_vacios <- numeric(ncol(df_original))
valores_nulos <- numeric(ncol(df_original))

# Calculamos los valores vacíos y lso valores nulos en cada columna.
for (i in seq_along(df_original)) {
  valores_vacios[i] <- sum(is.na(df_original[, i]))
  valores_nulos[i] <- sum(df_original[, i] == "")
}

# Creamos otro DF con los resultados.
resultados_DF <- data.frame(
  Columna = colnames(df_original),
  ValoresVacios = valores_vacios,
  ValoresNulos = valores_nulos
)

# Sacamos por pantalla los resultados.
print(resultados_DF)
```

```
##                 Columna ValoresVacios ValoresNulos
## 1      checking_balance             0            0
## 2  months_loan_duration             0            0
## 3        credit_history             0            0
## 4               purpose             0            0
## 5                amount             0            0
## 6       savings_balance             0            0
## 7     employment_length             0            0
## 8      installment_rate             0            0
## 9       personal_status             0            0
## 10        other_debtors             0            0
## 11    residence_history             0            0
## 12             property             0            0
## 13                  age             0            0
## 14     installment_plan             0            0
## 15              housing             0            0
## 16     existing_credits             0            0
## 17              default             0            0
```

```
## 18           dependents             0             0
## 19            telephone             0             0
## 20      foreign_worker              0             0
## 21                  job             0             0
```

Efectivamente, no hay ningún valor nulo o vacío en el juego de datos. Como se puede comprobar, tampoco existe ningún valor vacío, por lo tanto, una tarea de procesado menos.

Una vez comprobados esto, vamos comprobar si existen registros duplicados, para ello, comparo todas las columnas del *dataframe*. Véase el siguiente *chunk* de código:

```
df_sin_duplicados <- df_original[!duplicated(df_original[,]) & !duplicated(df_original[,]), ] #Con el -
head(df_sin_duplicados)
```

```
##   checking_balance months_loan_duration credit_history   purpose amount
## 1           < 0 DM                    6       critical  radio/tv   1169
## 2       1 - 200 DM                   48         repaid  radio/tv   5951
## 3          unknown                   12       critical education   2096
## 4           < 0 DM                   42         repaid furniture   7882
## 5           < 0 DM                   24        delayed car (new)   4870
## 6          unknown                   36         repaid education   9055
##   savings_balance employment_length installment_rate personal_status
## 1         unknown           > 7 yrs                4     single male
## 2        < 100 DM         1 - 4 yrs                2          female
## 3        < 100 DM         4 - 7 yrs                2     single male
## 4        < 100 DM         4 - 7 yrs                2     single male
## 5        < 100 DM         1 - 4 yrs                3     single male
## 6         unknown         1 - 4 yrs                2     single male
##   other_debtors residence_history                property age installment_plan
## 1          none                 4             real estate  67            none
## 2          none                 2             real estate  22            none
## 3          none                 3             real estate  49            none
## 4     guarantor                 4 building society savings  45            none
## 5          none                 4             unknown/none  53            none
## 6          none                 4             unknown/none  35            none
##    housing existing_credits default dependents telephone foreign_worker
## 1      own                2       1          1       yes            yes
## 2      own                1       2          1      none            yes
## 3      own                1       1          2      none            yes
## 4 for free                1       1          2      none            yes
## 5 for free                2       2          2      none            yes
## 6 for free                1       1          2       yes            yes
##                  job
## 1   skilled employee
## 2   skilled employee
## 3 unskilled resident
## 4   skilled employee
## 5   skilled employee
## 6 unskilled resident
```

```
str(df_sin_duplicados)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4 1 1 4 4 3 4 3 ...
##  $ months_loan_duration: Factor w/ 33 levels "4","5","6","7",..: 3 30 9 27 18 24 18 24 9 22 ...
##  $ credit_history      : Factor w/ 5 levels "critical","delayed",..: 1 5 1 5 2 5 5 5 5 1 ...
```

```
##  $ purpose             : Factor w/ 10 levels "business","car (new)",..: 8 8 5 6 2 5 6 3 8 2 ...
##  $ amount              : Factor w/ 921 levels "250","276","338",..: 143 771 391 849 735 870 534 814 5
##  $ savings_balance     : Factor w/ 5 levels "< 100 DM","> 1000 DM",..: 5 1 1 1 1 5 4 1 2 1 ...
##  $ employment_length   : Factor w/ 5 levels "> 7 yrs","0 - 1 yrs",..: 1 3 4 4 3 3 1 3 4 5 ...
##  $ installment_rate    : Factor w/ 4 levels "1","2","3","4": 4 2 2 2 3 2 3 2 2 4 ...
##  $ personal_status     : Factor w/ 4 levels "divorced male",..: 4 2 4 4 4 4 4 4 1 3 ...
##  $ other_debtors       : Factor w/ 3 levels "co-applicant",..: 3 3 3 2 3 3 3 3 3 3 ...
##  $ residence_history   : Factor w/ 4 levels "1","2","3","4": 4 2 3 4 4 4 4 2 4 2 ...
##  $ property            : Factor w/ 4 levels "building society savings",..: 3 3 3 1 4 4 1 2 3 2 ...
##  $ age                 : Factor w/ 53 levels "19","20","21",..: 49 4 31 27 35 17 35 17 43 10 ...
##  $ installment_plan    : Factor w/ 3 levels "bank","none",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ housing             : Factor w/ 3 levels "for free","own",..: 2 2 2 1 1 1 2 3 2 2 ...
##  $ existing_credits    : Factor w/ 4 levels "1","2","3","4": 2 1 1 1 2 1 1 1 1 2 ...
##  $ default             : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...
##  $ dependents          : Factor w/ 2 levels "1","2": 1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone           : Factor w/ 2 levels "none","yes": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreign_worker      : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ job                 : Factor w/ 4 levels "mangement self-employed",..: 2 2 4 2 2 4 2 1 4 1 ...
```

Como se puede ver, se mantienen los 1000 registros, por lo tanto no hay ningun registro duplicado, es por esto que utilizaremos la variable `df_original` en vez de `df_sin_duplicados` para evitar cualquier error tonto.

A estas alturas, ya hemos comprobado que no existen registros duplicados, ni tampoco valores vacíos, asi como tampoco valores nulos. Es por esto que a continuación, se va a comenzar a relacionar las variables, para obtener más información, pero no solo acerca de los típicos comportamientos estadísticos, sino de cualquier dinámica que resulte de interés para la posterior aplicación del modelo.

Para poder relacionar las variables más fácilmente, y saber encontrar las que son potencialmente más interesantes, tenemos que montar gráficas que reflejen los datos que cada una de ellas alberga, por ello, a continuación se instalan algunas de las herramientas de representación, más útiles y prácticas dentro del lenguaje R. Para dichas visualizaciones, haremos uso de los paquetes ggplot2, gridExtra y grid. Por ello, se procede a su instalación (de nuevo para evitar errores, aunque ya lo hemos hecho antes para el ejemplo de esta PEC):

```
if(!require(ggplot2)){
    install.packages('ggplot2', repos='http://cran.us.r-project.org')
    library(ggplot2)
}
if(!require(ggpubr)){
    install.packages('ggpubr', repos='http://cran.us.r-project.org')
    library(ggpubr)
}
if(!require(grid)){
    install.packages('grid', repos='http://cran.us.r-project.org')
    library(grid)
}
if(!require(gridExtra)){
    install.packages('gridExtra', repos='http://cran.us.r-project.org')
    library(gridExtra)
}
if(!require(C50)){
    install.packages('C50', repos='http://cran.us.r-project.org')
    library(C50)
}
```

Ahora ya podemos empezar a representar los datos.

De todas las variables que hay, las variables que más me llaman la atención para empezar con el análisis serían:

- `checking_balance`: la cantidad de dinero en la cuenta de los clientes puede ser interesante para este análisis.
- `months_loan_duration`: la cantidad de meses que cada cliente ha pedido para pagar un préstamos puede dar indicios, acerca de si esa persona va a tender a no pagar el préstamo o a no terminar de pagarlo.
- `credit_history`: el histórico de una variable en cualquier ámbito es muy importante, más aún en este tipo de juego de datos, donde hay dinero de por medio. El historial crediticio de un cliente es imprescindible a la hora de determinar si se le puede dar o no un préstamo a alguien
- `personal_status`: esta variable a parte de dar información análitica acerca del número de integrantes de un núcleo familiar, también puede ofrecer inforación que puede ser utilizada con fines sociológicos.
- `other_debtors`: esto puede afectar positivamente a un cliente, en el caso de que los clientes tengan el soporte económico de tereceros, a fin de poder optar a un préstamo, no obstante también puede afectar de manera negativa en caso de que un cliente no comparta la deuda y tenga muchas deudas.
- `property`: una persona con propiedad es más apta a optar a un crédito y más propensa a poder afrontar un préstamo integro. Cuanta más propiedad tenga una persona, más probabilidad de que esta pague el préstamo.
- `age`: esta variable al igual que `personal_status` ofrece información sociologica, y puede llegar a interesarnos a la hora de saber si las personas mayores suelen tener mejor historial crediticio, si suelen compartir deuda, o si suelen tener mucho o poco dinero. No obstante, la edad no debería de ser un impedimento para obtener un crédito, o para pensar que una persona de avanzada edad no va a pagar un préstamo, pero si que es verdad que es muy dificíl otorgar un préstamo a una persona mayor, pues no se tiene la certeza de que sea capaz de pagar por completo la deuda, a no ser que sea capaz de compartirla con un tercero (esto creo que debería de verse en los análisis posteriores)
- `existing_credits`: Una persona con muchos créditos pudede no ser apto a la hora de acceder a un préstamo, es lógico, no se le puede dar un crédito a una persona que en el presente no es del todo solvente.
- `dependents`: Si muchas personas dependen de la economía de un cliente, puede situar en una posición con desventaja a dicho cliente, a no ser que este tenga mucho dinero, tenga un buen histórico crediticio o tenga mucha propiedad. Pues alfin y al cabo, las propiedades actúan como avales.
- `foreign_worker`: Por muy triste que puede parecer, y sin afán de hacer daño, la realidad demuestra que la gran parte de los trabajadores inmigrantes en un país, no tienen el mismo grado de solvencia que el de los autóctonos, y aunque esta variable no debería ser un detonante para denegar un préstamo, esta variable es muy influyente. Como ya se mencionó en una PEC/PAC anterior, es importante que los datos no estén sesgados, y esta práctica puede ser una oportunidad la contrario en este caso.
- `job`: el trabajo de una persona es imprescindible, para determinar su pertinencia a la hora de adquirir un préstamo, alfin y al cabo, la economía de un cliente viene dada por el tipo de trabajo que tiene, y aunque no todos los empleos son iguales, podríamos decir que existen rangos/familias laborales, que vienen preincipalmente determinadas por un rango salarial.
- `amount`: la cantidad del préstamo es también fundamental para este tipo de análisis. Pero antes deberíamos de discretizarla o simplificarla, porque puede tomar cualquier valor dentro del conjunto de números: $\mathbb{R}_+$.
- `default`: esta variable determina si el cliente se encuentra o no en situación de impago, y por ello es muy importante.

A continuación se respresentan las variables:

```
#Ahora se obtienen los posibles valores de cada una las 4 variables:
cat('Estos son los valores de checking_balance y sus ocurrencias:\n')
```

```
## Estos son los valores de checking_balance y sus ocurrencias:
```

```
table(df_original$checking_balance)
```

```
##
##     < 0 DM   > 200 DM 1 - 200 DM    unknown
##        274         63        269        394
```

```
cat('\nEstos son los valores de months_loan_duration y sus ocurrencias:\n')
```

```
##
## Estos son los valores de months_loan_duration y sus ocurrencias:
```

```
table(df_original$months_loan_duration)
```

```
##
##   4   5   6   7   8   9  10  11  12  13  14  15  16  18  20  21  22  24  26  27
##   6   1  75   5   7  49  28   9 179   4   4  64   2 113   8  30   2 184   1  13
##  28  30  33  36  39  40  42  45  47  48  54  60  72
##   3  40   3  83   5   1  11   5   1  48   2  13   1
```

```
cat('\nEstos son los valores de credit_history y sus ocurrencias:\n')
```

```
##
## Estos son los valores de credit_history y sus ocurrencias:
```

```
table(df_original$credit_history)
```

```
##
##                critical                 delayed           fully repaid
##                     293                      88                      40
## fully repaid this bank                 repaid
##                      49                     530
```

```
cat('\nEstos son los valores de personal_status y sus ocurrencias:\n')
```

```
##
## Estos son los valores de personal_status y sus ocurrencias:
```

```
table(df_original$personal_status)
```

```
##
## divorced male        female  married male   single male
##            50           310            92           548
```

```
cat('\nEstos son los valores de other_debtors y sus ocurrencias:\n')
```

```
##
## Estos son los valores de other_debtors y sus ocurrencias:
```

```
table(df_original$other_debtors)
```

```
##
## co-applicant    guarantor         none
##           41           52          907
```

```
cat('\nEstos son los valores de age y sus ocurrencias:\n')
```

```
##
## Estos son los valores de age y sus ocurrencias:
```

```
table(df_original$age)
```

```
##
## 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
##  2 14 14 27 48 44 41 50 51 43 37 40 38 34 33 32 40 39 29 24 21 25 17 22 17 17
## 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 70 74
## 15 18 17 12 14 12  8  9  7 10  8  3  9  5  3  6  7  2  8  5  5  5  3  3  1  4
## 75
##  2
```

```
cat('\nEstos son los valores de property y sus ocurrencias:\n')
```

```
##
## Estos son los valores de property y sus ocurrencias:
```

```
table(df_original$property)
```

```
##
## building society savings                    other              real estate
##                      232                      332                      282
##              unknown/none
##                      154
```

```
cat('\nEstos son los valores de existing_credits y sus ocurrencias:\n')
```

```
##
## Estos son los valores de existing_credits y sus ocurrencias:
```

```
table(df_original$existing_credits)
```

```
##
##   1   2   3   4
## 633 333  28   6
```

```
cat('\nEstos son los valores de dependents y sus ocurrencias:\n')
```

```
##
## Estos son los valores de dependents y sus ocurrencias:
```

```
table(df_original$dependents)
```

```
##
##   1   2
## 845 155
```

```
cat('\nEstos son los valores de months_loan_duration y sus ocurrencias:\n')
```

```
##
## Estos son los valores de months_loan_duration y sus ocurrencias:
```

```
table(df_original$foreign_worker)
```

```
##
##  no yes
##  37 963
```

```
cat('\nEstos son los valores de foreign_worker y sus ocurrencias:\n')
```

```
##
## Estos son los valores de foreign_worker y sus ocurrencias:
```

```
table(df_original$job)
```

```
##
## mangement self-employed        skilled employee unemployed non-resident
##                     148                     630                      22
##      unskilled resident
##                     200
```

```
cat('\nEstos son los valores de job y sus ocurrencias:\n')
```

```
##
## Estos son los valores de job y sus ocurrencias:
```

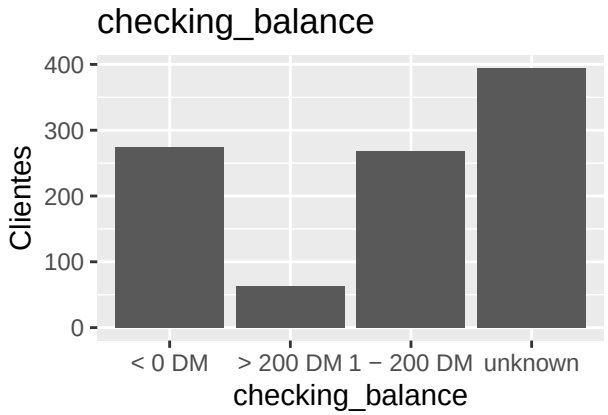
```
table(df_original$default)
```

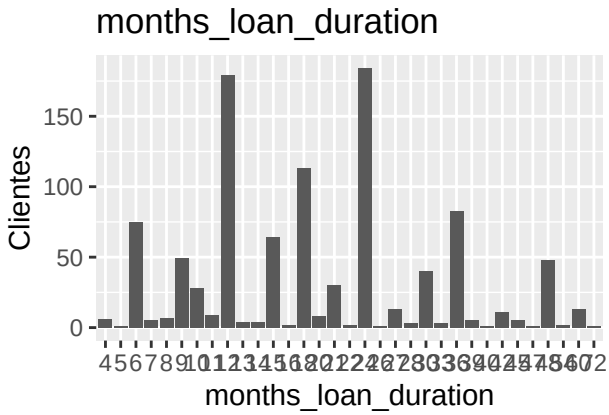
```
##
##   1   2
## 700 300
```

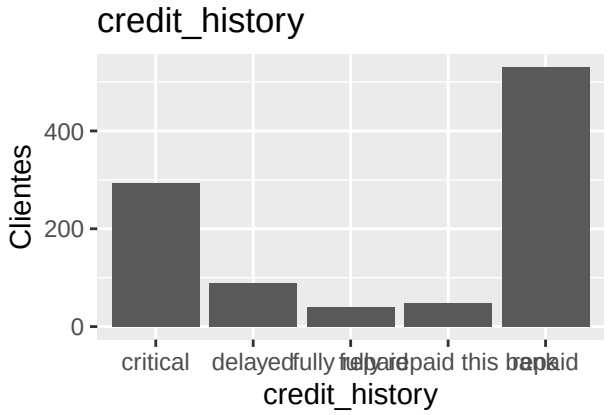
```
cat('\nEstos son los valores del default y sus ocurrencias:\n')
```

```
##
## Estos son los valores del default y sus ocurrencias:
```

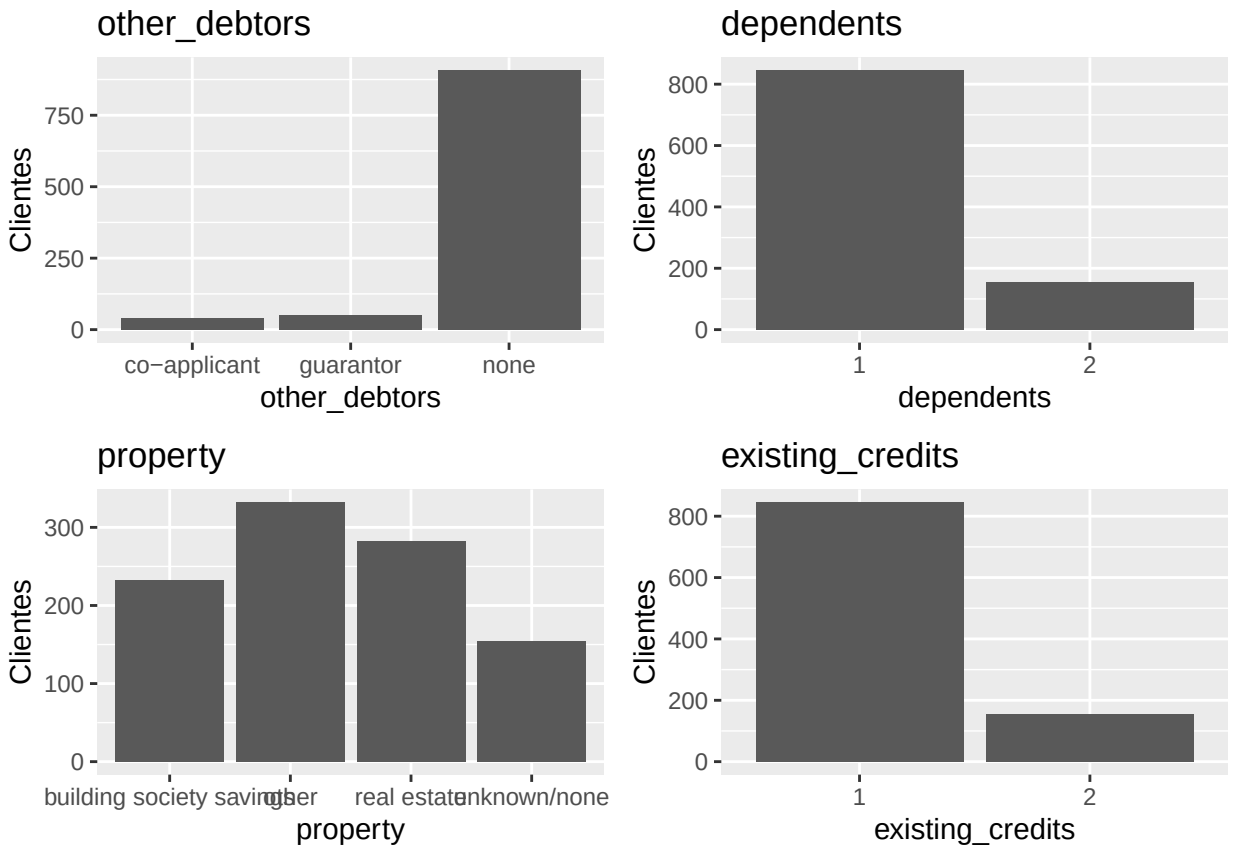
```
# grid.newpage()
plotbychecking_balance<-ggplot(df_original,aes(checking_balance))+geom_bar() +labs(x="checking_balance"
plotbymonths_loan_duration<-ggplot(df_original,aes(months_loan_duration))+geom_bar() +labs(x="months_loa
plotbycredit_history<-ggplot(df_original,aes(credit_history))+geom_bar() +labs(x="credit_history", y="Cl
plotbypersonal_status<-ggplot(df_original,aes(personal_status))+geom_bar() +labs(x="personal_status", y=
grid.arrange(plotbychecking_balance,plotbymonths_loan_duration,plotbycredit_history,plotbypersonal_statu
```



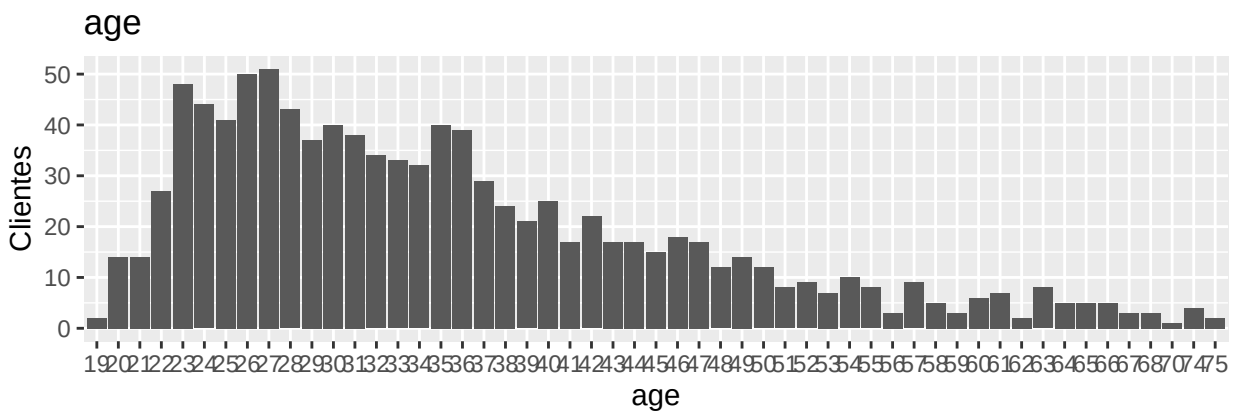


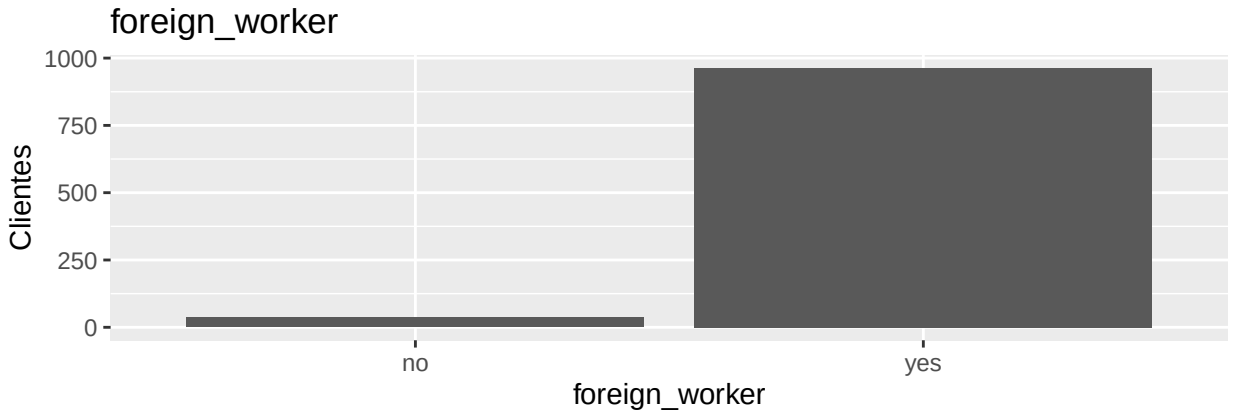



```
# Ahora representamos el segundo lote de gráficas.
plotby_other_debtors<-ggplot(df_original,aes(other_debtors))+geom_bar() +labs(x="other_debtors", y="Cli
plotby_dependents<-ggplot(df_original,aes(dependents))+geom_bar() +labs(x="dependents", y="Clientes")+
plotby_property<-ggplot(df_original,aes(property))+geom_bar() +labs(x="property", y="Clientes")+ guides
plotby_existing_credits<-ggplot(df_original,aes(dependents))+geom_bar() +labs(x="existing_credits", y="
grid.arrange(plotby_other_debtors,plotby_dependents,plotby_property,plotby_existing_credits,ncol=2)
```

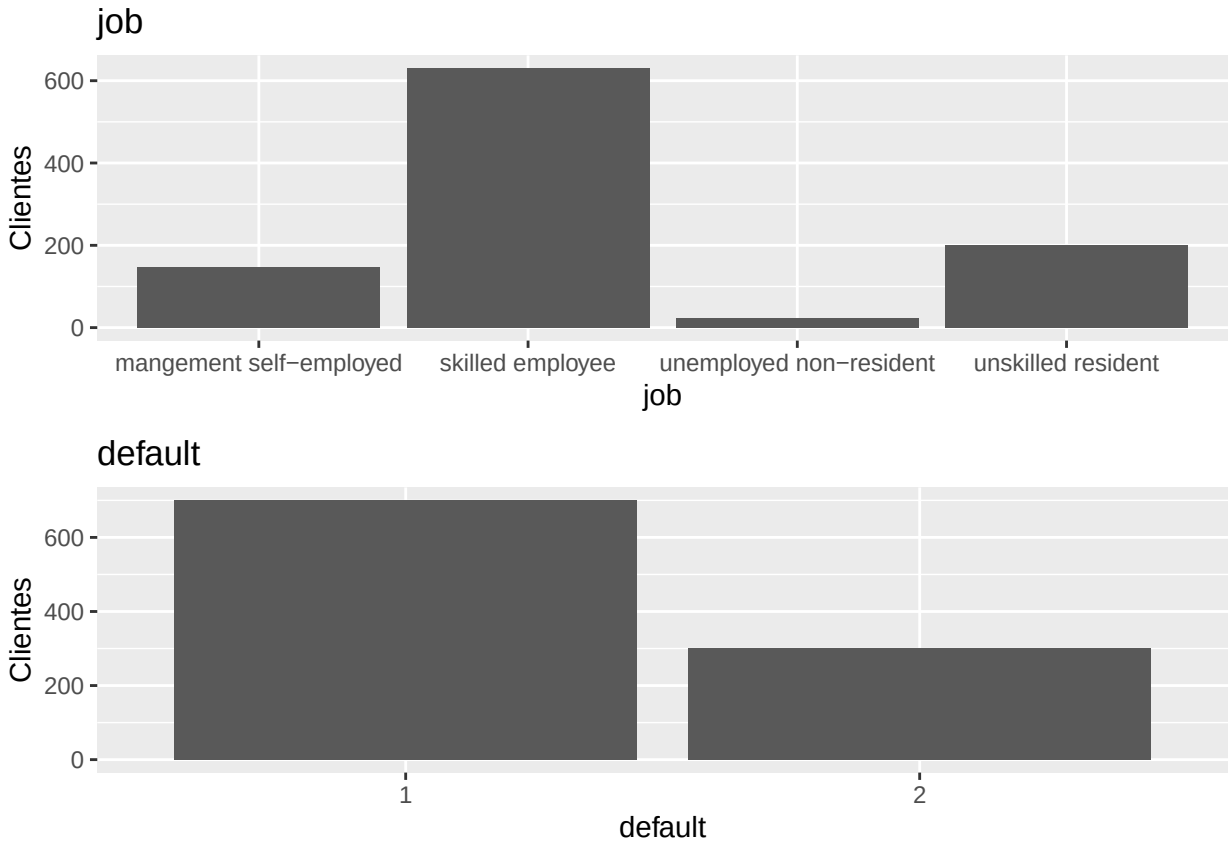



```
# Ahora representamos el tercer lote.
plotby_age<-ggplot(df_original,aes(age))+geom_bar() +labs(x="age", y="Clientes")+ guides(fill=guide_leg
plotby_foreign_worker<-ggplot(df_original,aes(foreign_worker))+geom_bar() +labs(x="foreign_worker", y="
grid.arrange(plotby_age,plotby_foreign_worker,ncol=1)
```





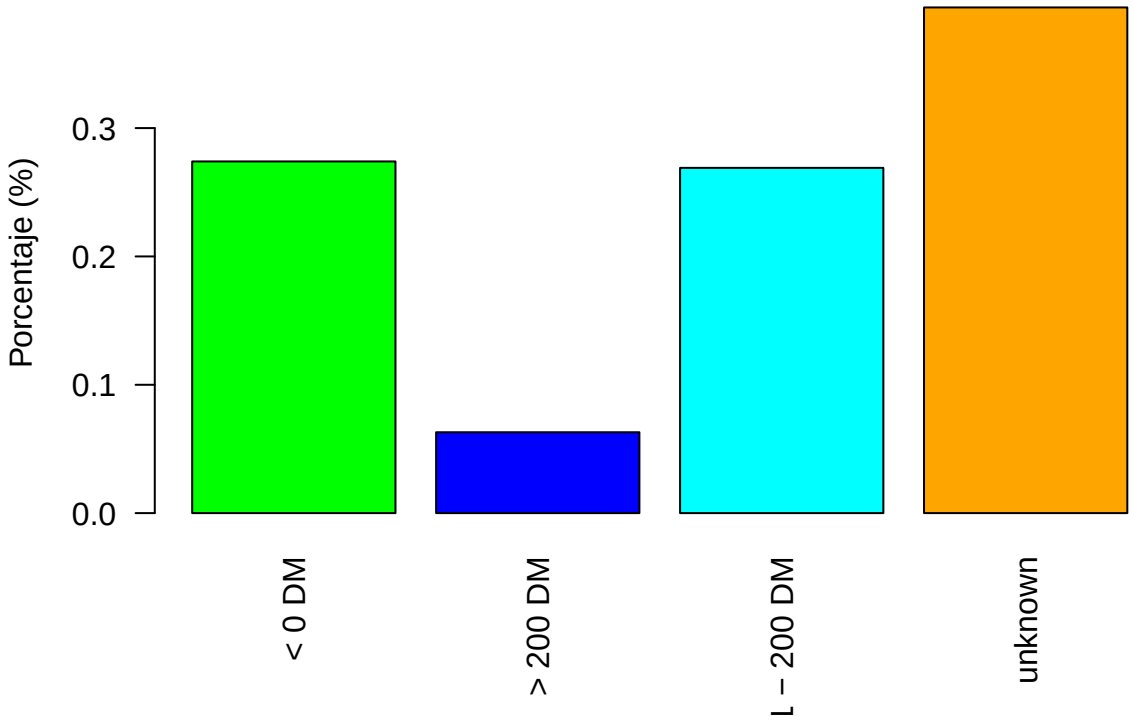

```
# Ahora representamos la última tanda de gráficas
plotby_job<-ggplot(df_original,aes(job))+geom_bar() +labs(x="job", y="Clientes")+ guides(fill=guide_leg
plotby_default<-ggplot(df_original,aes(default))+geom_bar() +labs(x="default", y="Clientes")+ guides(fi
grid.arrange(plotby_job,plotby_default,ncol=1)
```



Gracias a estas primeras representaciones, podemos visualizar gráficamente, los valores mayoritarios en cada una de las columnas. Vamos a dividir el siguiente análisis en cuatro partes, cada una de las partes se corresponde con cada una de las ventanas/bloques de gráficos obtenidos arriba.

En la primera tanda de gráficas (`checking_balance`, `months_loan_duration`, `credit_history`, `personal_status`), podemos observar como el `checking_balance` (i.e., saldo deudor) en la mayoría de los casos es desconocido, mientras que en la segunda posición se encuentran los valores "< 0 DM" y "1-200 DM". No obstante, en el anterior *chunk* donde calculábamos las ocurrencias, podemos observar como el segundo puesto en cuanto a ocurrencias de clientes con ese valor, es el rango de "< 0 DM" con 274 ocurrencias, frente a las 269 ocurrencias del valor "1-200 DM". En esta gráfica se observa una gran variabilidad en cuantoa a los datos, pues aunque mayoritariamente no se conozca el saldo de los clientes, el segundo y tercer puesto representan dos rangos que se encuentran en los dos extremos opuestos, de hecho, muy a grandes rasgos podriamos decir que más de la segunda mitad de clientes está formada por un cuarto de clientes que tienen un saldo muy bajo y otro cuarto que lo tienen muy alto. De todos modos, vamos a visualizarlo con porcentajes a continuación:

```
saldo <- table(df_original$checking_balance)
barplot(prop.table(saldo),col=c("green","blue","cyan","orange"), main=" Saldo deudor", ylab = "Porcenta
```



Como se puede comprobar, los dos rangos de valores mencionados (verde y cian) conforman un porcentaje notable de entre el total (28% aproximadamente)

```
summary(df_original$checking_balance)
```

```
##     < 0 DM   > 200 DM 1 - 200 DM    unknown
##        274         63        269        394
```

Volviendo al *chunk* relacionado con las gráficas de las variables más interesantes, puede verse como la duración de los préstamos rondan mayoritaariamente entre los 12 mese (1 año) y los 24 meses (2 años), nada extraño. Pero curiosamente le siguen los créditos de 27 meses y los de 6 meses, que son espacios temporales múltiplos de tres (trimestres). Luego, en la gráfica del `credit_history` se aprecia como la mayoría de clientes solo paga un poco del crédito (*bare paid*) o se encuentra en estado crítico. Es curioso ver como el menor porcentaje de los 4 posibles valores se corresponde con el grupo de clientes que pagan al completo el crédito. Más adelante en la gráfica del `personal_status` vemos claramente como la mayoría de los clientes/demandantes de crédito, son. hombres solteros, seguidos de las mujeres, que a su vez les siguen los hombres casados, y finalmente los hombres divorciados.

En la segunda tanda de gráficos, podemos ver como en la gráfica `other_debtors` dominan los clientes que no comparten deuda, y que asumen ellos mismos la misma de manera integra. Luego en el caso de las personas que dependen de los clientes, predominan aquellos clientes con una sola persona dependiente (en más de un 80%). En la tercera gráfica de este segundo lote, podemos observar como la mayoría de los clientes tienen otro tipo de propiedad diferente al resto de posibles valores, a este valor le siguen los clientes con bienes inmuebles (deberían de ser buenos candidatos, mientras que tengan una buena salud crediticia y un buen historial). Finalizando este conjunto de gráficas, y por último, se ve claramente como la mayoría de clientes (más de un 80%) que figuran en este juego de datos sobre créditos, cuentan con un crédito en curso, frente a un 18% aproximadamente.

Ya en el tercer lote de gráficos, podemos observar como la mayor parte de clientes son de: 27, 26, 23, 35, 30, 36, 31 años (de mayor a menor ocurrencia) . Seguidamente, en el gráfico `foreign_worker` observamos como

la mayoría son trabajadores extranjeros.

Terminando con el cuarto bloque de gráficos, en el gráfico `job` predominan los clientes cualificados, seguidos por los residentes no cualificados, que a su vez les siguen los administradores/autónomos (management self-employed). Por lo que podríamos decir que todos los clientes tienen un empleo estable, pero lo que los hace diferentes son sus entornos personales y familiares. Por último, la gráfica `default` indica los clientes que han cumplido y los que no han cumplido con el pago del crédito. En este caso, parece ser 700 clientes tienen que `default = 1` frente a los 300 restantes que tienen `default = 2`. Como no hay ninguna información al respecto de esta variable, he estado invesitgando, y no he encontrado nada en internet, por lo tanto he recurrido a ChatGPT, y este me ha respondido indicándome lo siguiente:

- `default = 1`. Podría indicar que el cliente ha incumplido o ha tenido problemas con el pago del préstamo. En este caso, 1 se consideraría como el valor que representa un incumplimiento.
- `default = 2`. Podría indicar que el cliente no ha incumplido, es decir, ha cumplido con sus obligaciones de pago según los términos del préstamo o crédito.

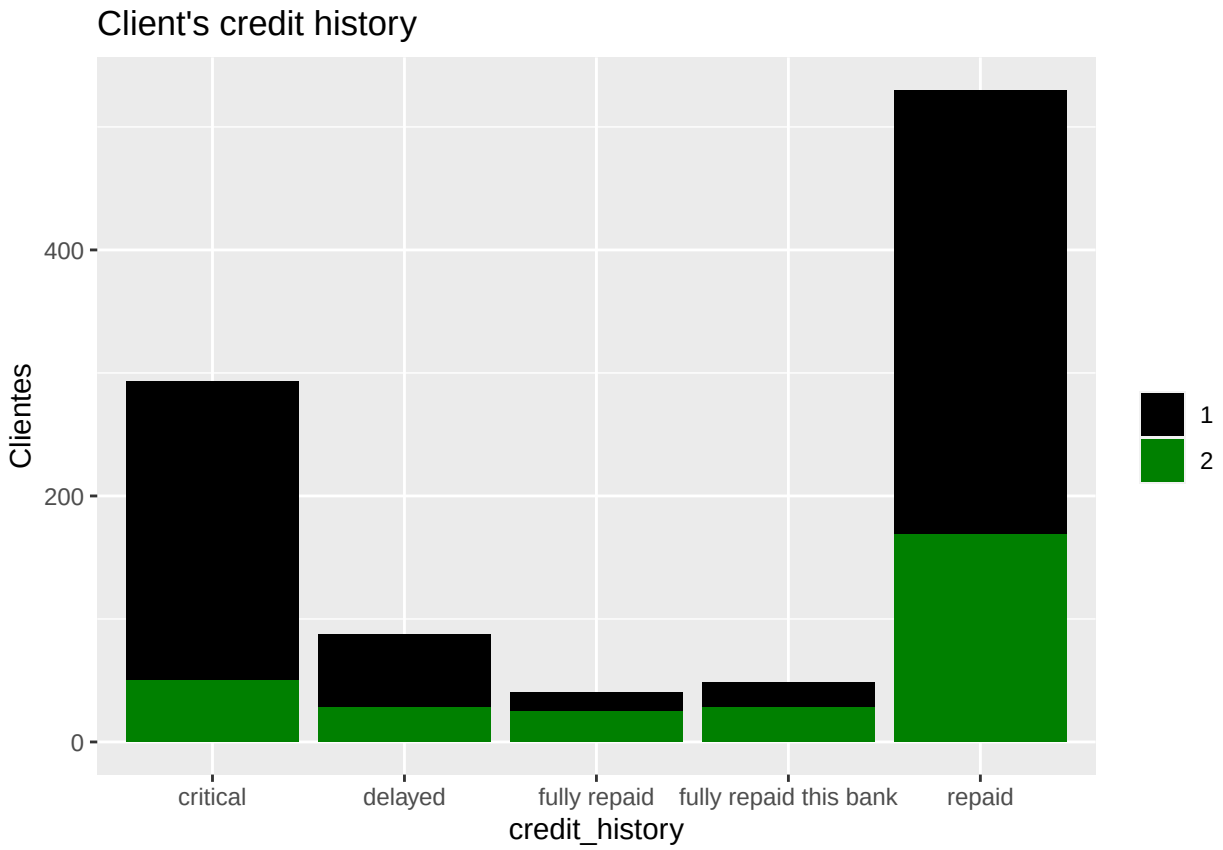
No obstante, como no estamos seguros de que esta sea la codificación de la variable, vamos a tener que relacionar esta variable con la de `credit_history`, esta tarea la vamos a llevar a cabo en el siguiente paso.

Habiendo realizado este primer análisis ya estamos en condiciones de relacionar las variables más influyentes con el resto del subconjunto que hemos extraído. Para este análisis de correlación, hemos destacado las variables más interesantes con un número de valores posibles razonables. Las variables que se han identificado son las siguientes:

- `default`
- `dependents`

Primero se va comenzar relacionando solamente la variable `default` con `credit_history` para poder determinar si la codificación de antes es correcta. Véase a continuación el código con el resultado que se ha obtenido:

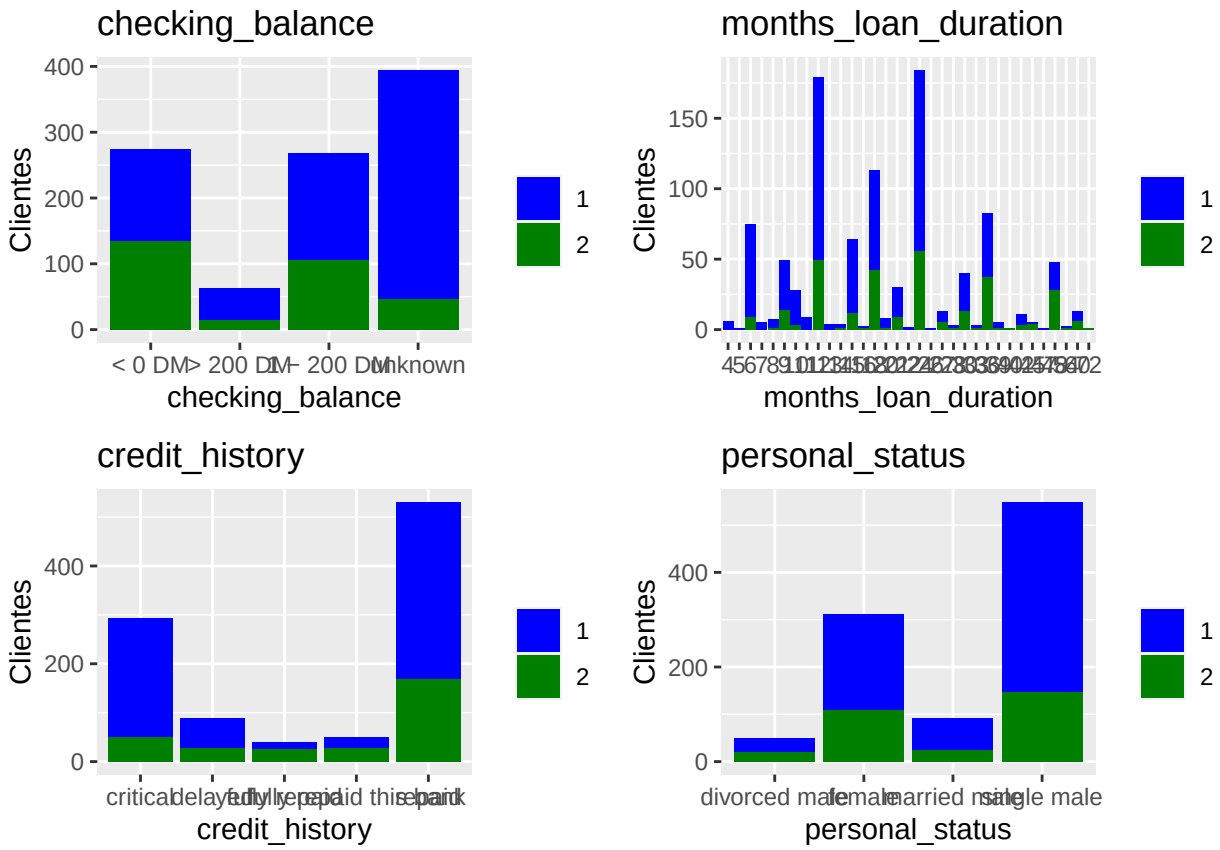
```
plotbyClass<-ggplot(df_original,aes(credit_history,fill=default))+geom_bar() +labs(x="credit_history", 
grid.arrange(plotbyClass, ncol=1)
```



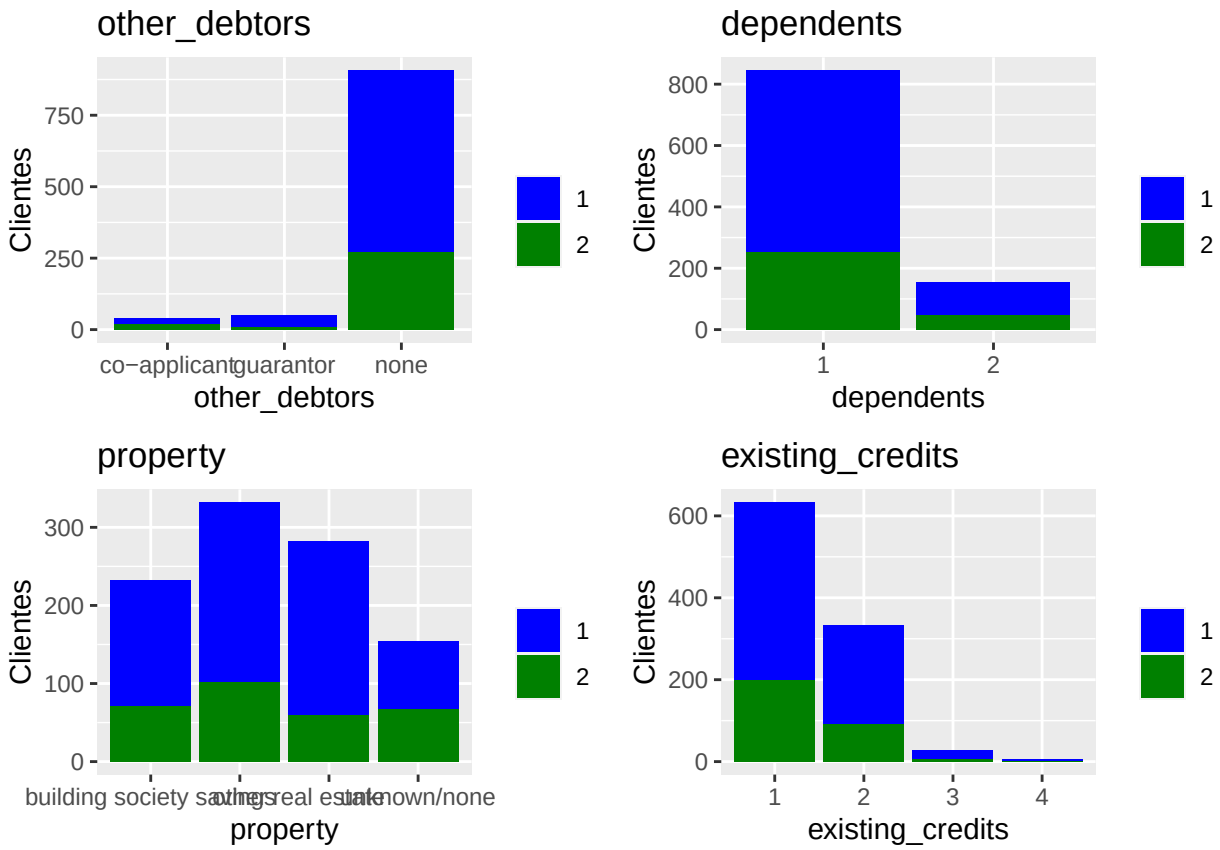
Estos datos que se acaban de obtener son muy importantes, ya que se está corroborando lo codificación de antes. Esto se debe a que para los clientes con $credit\_history =$" $critical$", el porcentaje de `default = 2`es mucho menor que el de `default = 1`, significando que cuando los clientes están en estado crediticio crítico, estos están en situación de `default` i.e., `default=1`. Esto que se acaba de explicar también aplica para los clientes que cumplen con que `credit_history = delayed`, pues de nuevo $\%(default = 1) > \%(default = 2)$. Y de manera contraria, los clientes que cumplen con el pago de su deuda, i.e., aquellos con `credit_history \in (``fully \hspace{1mm}'', ``fully\hspace{1mm}repaid\hspace{1mm}this\hspace{1mm}bank'')` cumplen con que el % de clientes con `default=2` es mayor que para aquellos que tienen `default=1`. Por lo tanto podemos afirmar que esta asignación es correcta y que efectivamente, los clientes que están en estado de default tienen que `default=1` mientras que aquellos que han cumplido con sus deudas, tienen que `default=2`. Cabe destacar que a pesar de que la codificación anterior aplica para todas los casos anteriores, para el caso de `credit_history = repaid` se ve como $\%(default = 1) > \%(default = 2)$ significando también, que los clientes que han pagado los créditos, se encuentran en situación de default pues `default=1` (mayoritariamente), no obstante, se sigue pensando que la codificación es correcta, por lo tanto la codificación permanece.

Ahora vamos a relacionar la vairable `default` con el resto de variables. Véase el siguiente *chunk* de código.

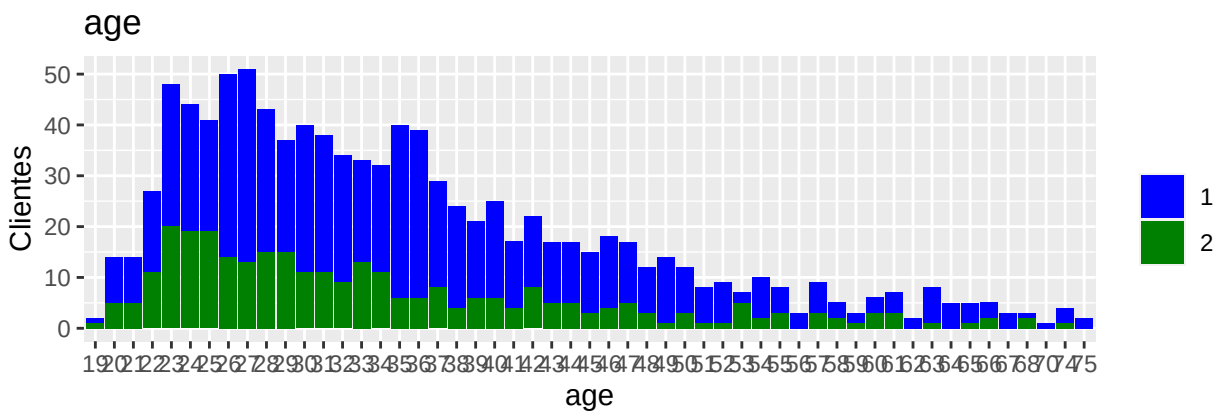
```
plotbychecking_balance<-ggplot(df_original,aes(checking_balance,fill=default))+geom_bar() +labs(x="checl
plotbymonths_loan_duration<-ggplot(df_original,aes(months_loan_duration,fill=default))+geom_bar() +labs
plotbycredit_history<-ggplot(df_original,aes(credit_history,fill=default))+geom_bar() +labs(x="credit_hi
plotbypersonal_status<-ggplot(df_original,aes(personal_status,fill=default))+geom_bar() +labs(x="persona
grid.arrange(plotbychecking_balance,plotbymonths_loan_duration,plotbycredit_history,plotbypersonal_statu
```

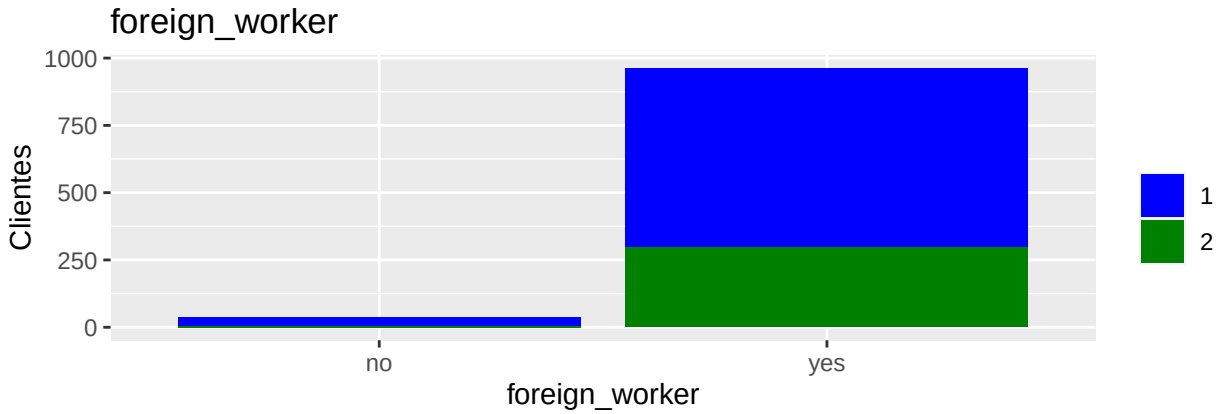



```
# Ahora representamos el segundo lote de gráficas.
plotby_other_debtors<-ggplot(df_original,aes(other_debtors,fill=default))+geom_bar() +labs(x="other_debt
plotby_dependents<-ggplot(df_original,aes(dependents,fill=default))+geom_bar() +labs(x="dependents", y="
plotby_property<-ggplot(df_original,aes(property,fill=default))+geom_bar() +labs(x="property", y="Client
plotby_existing_credits<-ggplot(df_original,aes(existing_credits,fill=default))+geom_bar() +labs(x="exis
grid.arrange(plotby_other_debtors,plotby_dependents,plotby_property,plotby_existing_credits,ncol=2)
```

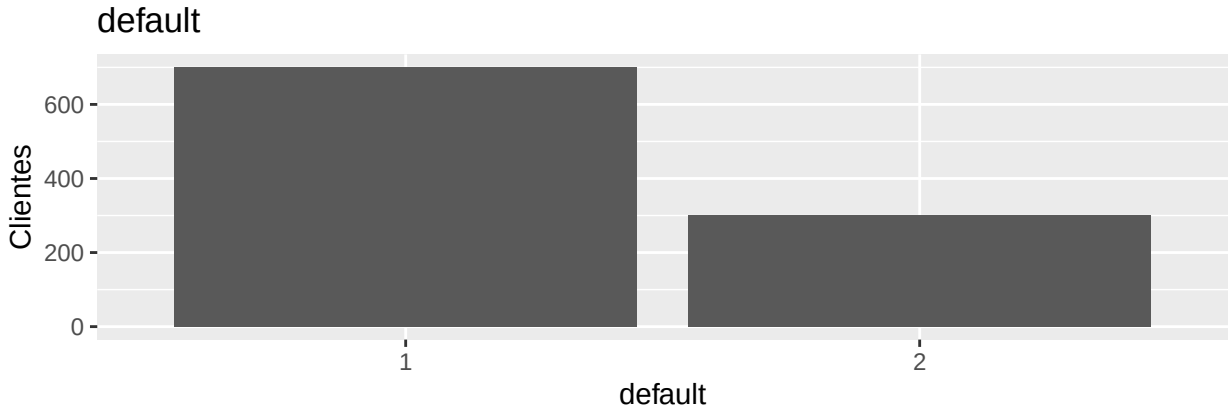



```
# Ahora representamos el tercer lote.
plotby_age<-ggplot(df_original,aes(age,fill=default))+geom_bar() +labs(x="age", y="Clientes")+ guides(f
plotby_foreign_worker<-ggplot(df_original,aes(foreign_worker,fill=default))+geom_bar() +labs(x="foreign_
grid.arrange(plotby_age,plotby_foreign_worker,ncol=1)
```






```
# Ahora representamos la última tanda de gráficas
plotby_job<-ggplot(df_original,aes(job,fill=default))+geom_bar() +labs(x="job", y="Clientes")+ guides(f
plotby_default<-ggplot(df_original,aes(default))+geom_bar() +labs(x="default", y="Clientes")+ guides(fi
grid.arrange(plotby_job,plotby_default,ncol=1)
```



Para poder analizar más en detalle y complementar los gráficos que se han obtenido, se van a calcular las tablas de contingencia, que nos van a perimitir poner números a los comportamientos que se observan en las gráficas. Véase el siguiente *chunk* de código.

```
cols = c('checking_balance', 'months_loan_duration', 'credit_history',
         'personal_status', 'other_debtors', 'dependents', 'property',
         'existing_credits', 'age', 'foreign_worker', 'job', 'default',
         'housing','purpose')

# Creamos otro dataframe solo con las columnas que qeueremos
sub_df_original <- df_original[, cols]

# Vamos a hacer uso de un bucle para poder crear todo más rápido
# Bucle para calcular tablas de contingencia
for (col in cols[-which(cols == 'default')]) {
  tabla_contingencia <- table(df_original[[col]], df_original$default)

  # Imprimimos la tabla de contingencia
  cat("Tabla de contingencia entre", col, "y default:\n")
  print(tabla_contingencia)
  cat("\n")

  # Ahora imprimimos la tabla de contingencia pero en proporciones
  cat("Tabla de contingencia (%) entre", col, "y default:\n")
  print(prop.table(tabla_contingencia, margin = 1))
  cat('-----------------------------------------------------\n')
}
```

```
## Tabla de contingencia entre checking_balance y default:
##
##                1   2
##   < 0 DM     139 135
##   > 200 DM    49  14
##   1 - 200 DM 164 105
##   unknown    348  46
##
## Tabla de contingencia (%) entre checking_balance y default:
##
##                       1         2
##   < 0 DM     0.5072993 0.4927007
##   > 200 DM   0.7777778 0.2222222
##   1 - 200 DM 0.6096654 0.3903346
##   unknown    0.8832487 0.1167513
## ----------------------------------------------------
## Tabla de contingencia entre months_loan_duration y default:
##
##        1   2
##   4    6   0
##   5    1   0
##   6   66   9
##   7    5   0
##   8    6   1
##   9   35  14
##   10  25   3
##   11   9   0
##   12 130  49
##   13   4   0
##   14   3   1
##   15  52  12
##   16   1   1
##   18  71  42
##   20   7   1
##   21  21   9
##   22   2   0
##   24 128  56
##   26   1   0
##   27   8   5
##   28   2   1
##   30  27  13
##   33   2   1
##   36  46  37
##   39   4   1
##   40   0   1
##   42   8   3
##   45   1   4
##   47   1   0
##   48  20  28
##   54   1   1
##   60   7   6
##   72   0   1
##
## Tabla de contingencia (%) entre months_loan_duration y default:
```

```
##
##              1         2
##   4  1.0000000 0.0000000
##   5  1.0000000 0.0000000
##   6  0.8800000 0.1200000
##   7  1.0000000 0.0000000
##   8  0.8571429 0.1428571
##   9  0.7142857 0.2857143
##   10 0.8928571 0.1071429
##   11 1.0000000 0.0000000
##   12 0.7262570 0.2737430
##   13 1.0000000 0.0000000
##   14 0.7500000 0.2500000
##   15 0.8125000 0.1875000
##   16 0.5000000 0.5000000
##   18 0.6283186 0.3716814
##   20 0.8750000 0.1250000
##   21 0.7000000 0.3000000
##   22 1.0000000 0.0000000
##   24 0.6956522 0.3043478
##   26 1.0000000 0.0000000
##   27 0.6153846 0.3846154
##   28 0.6666667 0.3333333
##   30 0.6750000 0.3250000
##   33 0.6666667 0.3333333
##   36 0.5542169 0.4457831
##   39 0.8000000 0.2000000
##   40 0.0000000 1.0000000
##   42 0.7272727 0.2727273
##   45 0.2000000 0.8000000
##   47 1.0000000 0.0000000
##   48 0.4166667 0.5833333
##   54 0.5000000 0.5000000
##   60 0.5384615 0.4615385
##   72 0.0000000 1.0000000
## ----------------------------------------------------
## Tabla de contingencia entre credit_history y default:
##
##                             1   2
##   critical                243  50
##   delayed                  60  28
##   fully repaid             15  25
##   fully repaid this bank   21  28
##   repaid                  361 169
##
## Tabla de contingencia (%) entre credit_history y default:
##
##                                    1         2
##   critical                 0.8293515 0.1706485
##   delayed                  0.6818182 0.3181818
##   fully repaid             0.3750000 0.6250000
##   fully repaid this bank   0.4285714 0.5714286
##   repaid                   0.6811321 0.3188679
## ----------------------------------------------------
```

```
## Tabla de contingencia entre personal_status y default:
##
##                   1   2
##   divorced male  30  20
##   female        201 109
##   married male   67  25
##   single male   402 146
##
## Tabla de contingencia (%) entre personal_status y default:
##
##                          1         2
##   divorced male 0.6000000 0.4000000
##   female        0.6483871 0.3516129
##   married male  0.7282609 0.2717391
##   single male   0.7335766 0.2664234
## ----------------------------------------------------
## Tabla de contingencia entre other_debtors y default:
##
##                   1   2
##   co-applicant  23  18
##   guarantor     42  10
##   none         635 272
##
## Tabla de contingencia (%) entre other_debtors y default:
##
##                         1         2
##   co-applicant 0.5609756 0.4390244
##   guarantor    0.8076923 0.1923077
##   none         0.7001103 0.2998897
## ----------------------------------------------------
## Tabla de contingencia entre dependents y default:
##
##       1   2
##   1 591 254
##   2 109  46
##
## Tabla de contingencia (%) entre dependents y default:
##
##             1         2
##   1 0.6994083 0.3005917
##   2 0.7032258 0.2967742
## ----------------------------------------------------
## Tabla de contingencia entre property y default:
##
##                               1   2
##   building society savings 161  71
##   other                    230 102
##   real estate              222  60
##   unknown/none              87  67
##
## Tabla de contingencia (%) entre property y default:
##
##                                    1         2
##   building society savings 0.6939655 0.3060345
```

```
##   other                    0.6927711 0.3072289
##   real estate              0.7872340 0.2127660
##   unknown/none             0.5649351 0.4350649
## ----------------------------------------------------
## Tabla de contingencia entre existing_credits y default:
##
##       1   2
##   1 433 200
##   2 241  92
##   3  22   6
##   4   4   2
##
## Tabla de contingencia (%) entre existing_credits y default:
##
##             1         2
##   1 0.6840442 0.3159558
##   2 0.7237237 0.2762763
##   3 0.7857143 0.2142857
##   4 0.6666667 0.3333333
## ----------------------------------------------------
## Tabla de contingencia entre age y default:
##
##       1  2
##   19  1  1
##   20  9  5
##   21  9  5
##   22 16 11
##   23 28 20
##   24 25 19
##   25 22 19
##   26 36 14
##   27 38 13
##   28 28 15
##   29 22 15
##   30 29 11
##   31 27 11
##   32 25  9
##   33 20 13
##   34 21 11
##   35 34  6
##   36 33  6
##   37 21  8
##   38 20  4
##   39 15  6
##   40 19  6
##   41 13  4
##   42 14  8
##   43 12  5
##   44 12  5
##   45 12  3
##   46 14  4
##   47 12  5
##   48  9  3
##   49 13  1
```

```
##   50  9  3
##   51  7  1
##   52  8  1
##   53  2  5
##   54  8  2
##   55  5  3
##   56  3  0
##   57  6  3
##   58  3  2
##   59  2  1
##   60  3  3
##   61  4  3
##   62  2  0
##   63  7  1
##   64  5  0
##   65  4  1
##   66  3  2
##   67  3  0
##   68  1  2
##   70  1  0
##   74  3  1
##   75  2  0
##
## Tabla de contingencia (%) entre age y default:
##
##               1          2
##   19 0.50000000 0.50000000
##   20 0.64285714 0.35714286
##   21 0.64285714 0.35714286
##   22 0.59259259 0.40740741
##   23 0.58333333 0.41666667
##   24 0.56818182 0.43181818
##   25 0.53658537 0.46341463
##   26 0.72000000 0.28000000
##   27 0.74509804 0.25490196
##   28 0.65116279 0.34883721
##   29 0.59459459 0.40540541
##   30 0.72500000 0.27500000
##   31 0.71052632 0.28947368
##   32 0.73529412 0.26470588
##   33 0.60606061 0.39393939
##   34 0.65625000 0.34375000
##   35 0.85000000 0.15000000
##   36 0.84615385 0.15384615
##   37 0.72413793 0.27586207
##   38 0.83333333 0.16666667
##   39 0.71428571 0.28571429
##   40 0.76000000 0.24000000
##   41 0.76470588 0.23529412
##   42 0.63636364 0.36363636
##   43 0.70588235 0.29411765
##   44 0.70588235 0.29411765
##   45 0.80000000 0.20000000
##   46 0.77777778 0.22222222
```

```
##   47 0.70588235 0.29411765
##   48 0.75000000 0.25000000
##   49 0.92857143 0.07142857
##   50 0.75000000 0.25000000
##   51 0.87500000 0.12500000
##   52 0.88888889 0.11111111
##   53 0.28571429 0.71428571
##   54 0.80000000 0.20000000
##   55 0.62500000 0.37500000
##   56 1.00000000 0.00000000
##   57 0.66666667 0.33333333
##   58 0.60000000 0.40000000
##   59 0.66666667 0.33333333
##   60 0.50000000 0.50000000
##   61 0.57142857 0.42857143
##   62 1.00000000 0.00000000
##   63 0.87500000 0.12500000
##   64 1.00000000 0.00000000
##   65 0.80000000 0.20000000
##   66 0.60000000 0.40000000
##   67 1.00000000 0.00000000
##   68 0.33333333 0.66666667
##   70 1.00000000 0.00000000
##   74 0.75000000 0.25000000
##   75 1.00000000 0.00000000
## ----------------------------------------------------
## Tabla de contingencia entre foreign_worker y default:
##
##         1   2
##   no   33   4
##   yes 667 296
##
## Tabla de contingencia (%) entre foreign_worker y default:
##
##               1         2
##   no  0.8918919 0.1081081
##   yes 0.6926272 0.3073728
## ----------------------------------------------------
## Tabla de contingencia entre job y default:
##
##                              1   2
##   mangement self-employed  97  51
##   skilled employee        444 186
##   unemployed non-resident  15   7
##   unskilled resident      144  56
##
## Tabla de contingencia (%) entre job y default:
##
##                                    1         2
##   mangement self-employed 0.6554054 0.3445946
##   skilled employee        0.7047619 0.2952381
##   unemployed non-resident 0.6818182 0.3181818
##   unskilled resident      0.7200000 0.2800000
## ----------------------------------------------------
```

```
## Tabla de contingencia entre housing y default:
##
##              1   2
##   for free  64  44
##   own      527 186
##   rent     109  70
##
## Tabla de contingencia (%) entre housing y default:
##
##                    1         2
##   for free 0.5925926 0.4074074
##   own      0.7391304 0.2608696
##   rent     0.6089385 0.3910615
## ------------------------------------------------------
## Tabla de contingencia entre purpose y default:
##
##                          1   2
##   business              63  34
##   car (new)            145  89
##   car (used)            86  17
##   domestic appliances    8   4
##   education             28  22
##   furniture            123  58
##   others                 7   5
##   radio/tv             218  62
##   repairs               14   8
##   retraining             8   1
##
## Tabla de contingencia (%) entre purpose y default:
##
##                                1         2
##   business             0.6494845 0.3505155
##   car (new)            0.6196581 0.3803419
##   car (used)           0.8349515 0.1650485
##   domestic appliances  0.6666667 0.3333333
##   education            0.5600000 0.4400000
##   furniture            0.6795580 0.3204420
##   others               0.5833333 0.4166667
##   radio/tv             0.7785714 0.2214286
##   repairs              0.6363636 0.3636364
##   retraining           0.8888889 0.1111111
## ------------------------------------------------------
```

Observando los resultados de arriba, podemos ver como para la primera tanda de gráficas, en la primera gráfica correspondiente a `cecking_balance` vemos un comportamiento curioso, y es que para los clientes con menos dinero en su cuenta, i.e., `checking_balance = "< 0 DM"` un poco más de la mitad no está en estado de *default*, i.e., `default = 1`, no osbtante se observa como según va subiendo el sueldo, esto cambia, es decir, que los clientes con `checking_balance = "1 - 200 DM"` son más propensos a acabar en *default* (60,96% frente a un 39,04%), puesto que $\%(default = 1) > \%(default = 2)$, y aun más propensos lo serán aquellos con `checking_balance = " > 200 DM"` ya que es un 77,78% con `default=1` frente a un 22,22% con `default=2`. En definitiva, más dinero en la cuenta, implica más probabilidad de acabar en *default*. Para la gráfica `months_loan_duration` vemos un comportamiento propocional entre la duración del pago del crédito, y la variable `default`, ya que a medida que aumenta el número de clientes con créditos con la duración de los meses destacados, la probabilidad de default baja, esto es lógico, pues cuantos más meses/plazos, más

capacidad se va a tener de pagar el crédito. Esto además se justifica, observando como según aumenta el número de meses, el porcentaje de defaults es menor al de los clientes que no están en dicho estado, i.e., $\%(default = 1) < \%(default = 2)$, la cantidad de meses que más clientes con `default=1` son los créditos de 9,10,12,15,18 y 24 meses con un 71'42%, 89'28%, 72'62%, 81'25%, 62'83% y un 69'57%. Luego para `personal_status` podemos constatar como la mayoría están en default.

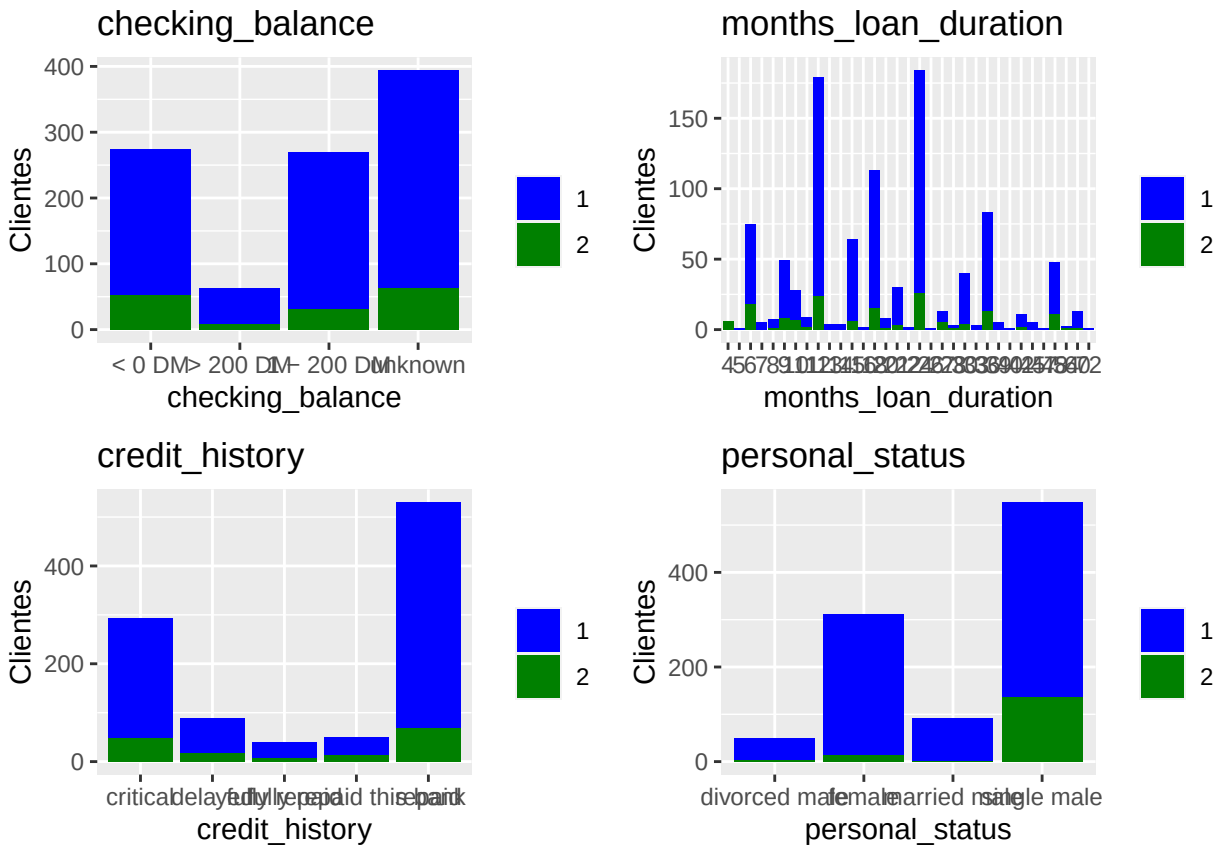
En la segunda tanda de gráficas, i.e., `other_debtors`, `dependents`, `property`, `existing_credits` observamos unas proporciones generalizadas muy similares. Es decir, existe un alto grado de propocionalidad de la variable `default` y el resto. Cabe destacar que también se observa un predominio de clientes en situación de `default` ya que en las 4 gráficas de esta segunda tanda se cumple que $\%(default = 1) > \%(default = 2)$. Esto puede verse claramente en el caso de la gráfica `other_debtors` dónde los porcentajes de `default=1` para los tres posibles valores de la variable `other_debtors`, son: 56'1% (co-applicant), 80'77% (guarantor) y 70% (none).

En la tercera tanda de gráficas (`age` y `foreign_worker`) se repite este comportamiento y de nuevo predomina el status de *default* entre los clientes. En el caso de la gráfica `age`, podemos ver claramente como las edades con más registros son las del conjunto $\{23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ y la media de clientes en situación de `default` en ese conjunto de edades, es del 71% aproximadamente.

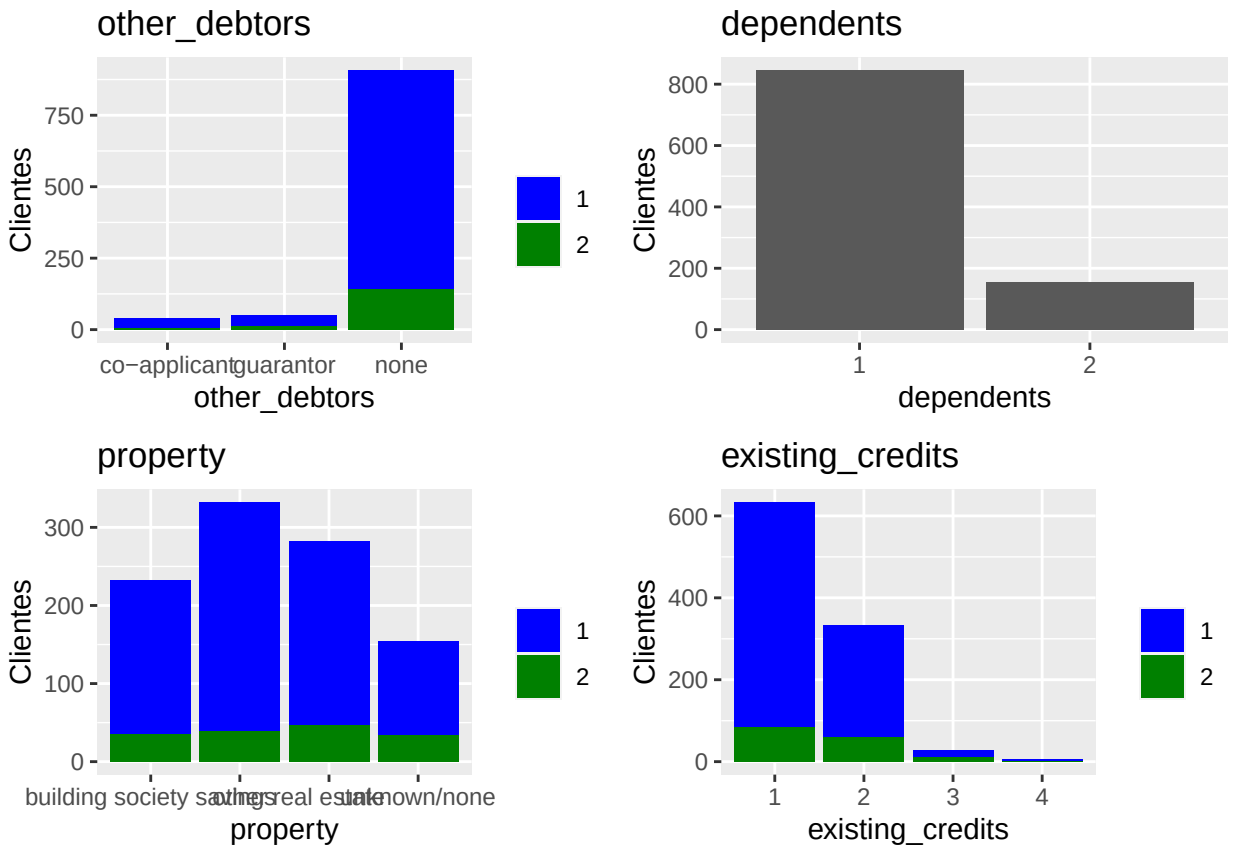
Por último, en la última gráfica, i.e., `job` se oberva el mismo comportamiento: predominio de `default=1` y proporcionalidad entre `default` y `job`. Proporcionalmente, en la gráfica de `job` los clientes cualificados son los que suelen acabar en situación de *default* ya que son 630 clientes de un total de 1000, donde dentro de esos 630 empleados 444 tienen `default=1` es decir, un 70'48%.

Ahora vamos a relacionar la variable `dependents` con el resto, véase el siguiente *chunk* de código:

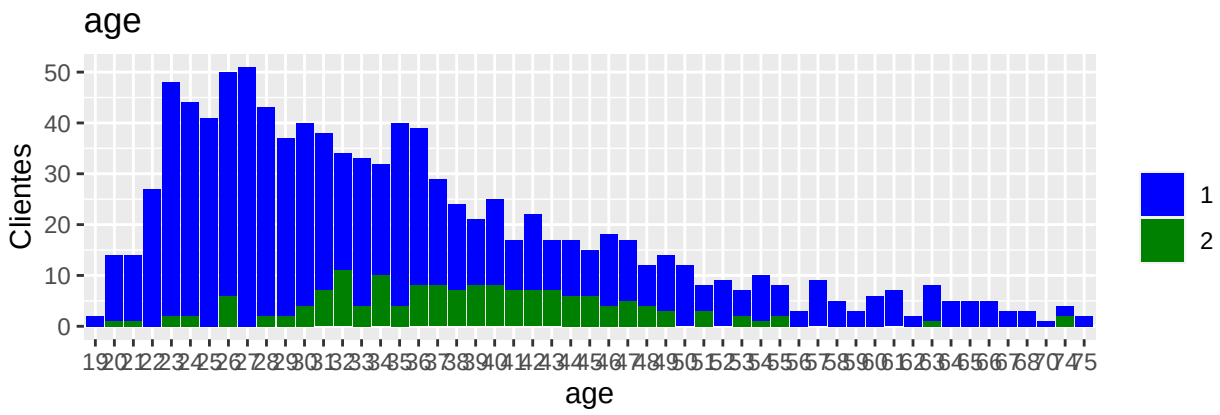
```
plotbychecking_balance<-ggplot(df_original,aes(checking_balance,fill=dependents))+geom_bar() +labs(x="cl
plotbymonths_loan_duration<-ggplot(df_original,aes(months_loan_duration,fill=dependents))+geom_bar() +la
plotbycredit_history<-ggplot(df_original,aes(credit_history,fill=dependents))+geom_bar() +labs(x="credi
plotbypersonal_status<-ggplot(df_original,aes(personal_status,fill=dependents))+geom_bar() +labs(x="pers
grid.arrange(plotbychecking_balance,plotbymonths_loan_duration,plotbycredit_history,plotbypersonal_statu
```

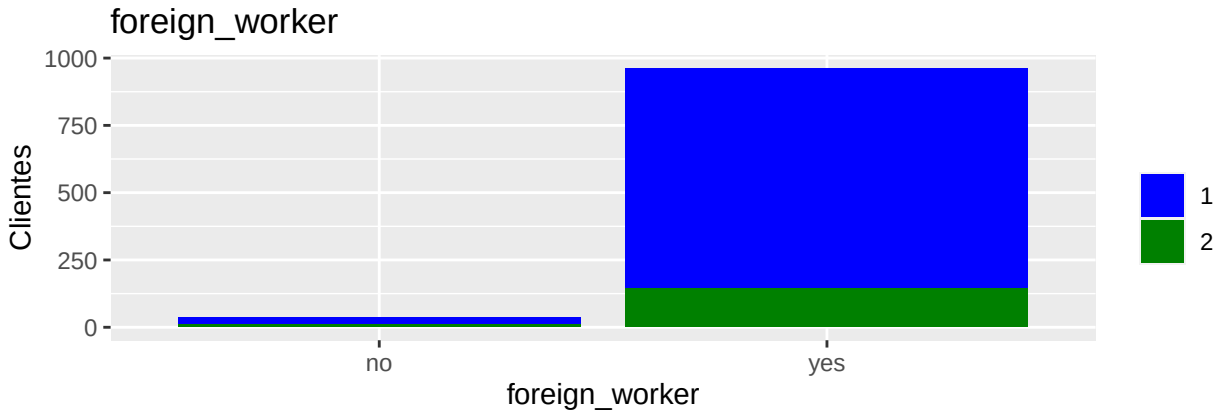



```
# Ahora representamos el segundo lote de gráficas.
plotby_other_debtors<-ggplot(df_original,aes(other_debtors,fill=dependents))+geom_bar() +labs(x="other_
plotby_dependents<-ggplot(df_original,aes(dependents))+geom_bar() +labs(x="dependents", y="Clientes")+
plotby_property<-ggplot(df_original,aes(property,fill=dependents))+geom_bar() +labs(x="property", y="Cl
plotby_existing_credits<-ggplot(df_original,aes(existing_credits,fill=dependents))+geom_bar() +labs(x="
grid.arrange(plotby_other_debtors,plotby_dependents,plotby_property,plotby_existing_credits,ncol=2)
```

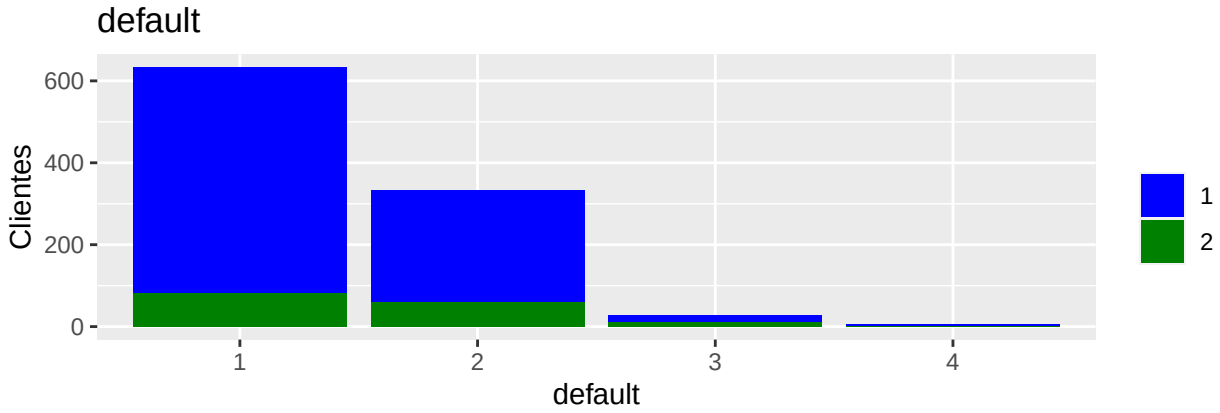



```
# Ahora representamos el tercer lote.
plotby_age<-ggplot(df_original,aes(age,fill=dependents))+geom_bar() +labs(x="age", y="Clientes")+ guide
plotby_foreign_worker<-ggplot(df_original,aes(foreign_worker,fill=dependents))+geom_bar() +labs(x="fore
grid.arrange(plotby_age,plotby_foreign_worker,ncol=1)
```






```
# Ahora representamos la última tanda de gráficas
plotby_job<-ggplot(df_original,aes(job,fill=dependents))+geom_bar() +labs(x="job", y="Clientes")+ guides
plotby_default<-ggplot(df_original,aes(existing_credits, fill=dependents))+geom_bar() +labs(x="default"
grid.arrange(plotby_job,plotby_default,ncol=1)
```



Ahora se calculan las tablas de contingencia asociadas:

```
# Vamos a hacer uso de un bucle para poder calcular todo más rápido
for (col in cols[-which(cols == 'dependents')]) {
  tabla_contingencia <- table(df_original[[col]], df_original$dependents)

  # Imprimimos la tabla de contingencia
  cat("Tabla de contingencia entre", col, "y dependents:\n")
  print(tabla_contingencia)
  cat("\n")

  # Ahora imprimimos la tabla de contingencia pero en proporciones
  cat("Tabla de contingencia (%) entre", col, "y dependents:\n")
  print(prop.table(tabla_contingencia, margin = 1))
  cat('-----------------------------------------------------\n')
}
```

```
## Tabla de contingencia entre checking_balance y dependents:
##
##                 1   2
##   < 0 DM      222  52
##   > 200 DM     54   9
##   1 - 200 DM 238  31
##   unknown     331  63
##
## Tabla de contingencia (%) entre checking_balance y dependents:
##
##                       1         2
```

```
##   < 0 DM     0.8102190 0.1897810
##   > 200 DM   0.8571429 0.1428571
##   1 - 200 DM 0.8847584 0.1152416
##   unknown    0.8401015 0.1598985
## ---------------------------------------------------
## Tabla de contingencia entre months_loan_duration y dependents:
##
##        1   2
##   4    0   6
##   5    1   0
##   6   57  18
##   7    5   0
##   8    6   1
##   9   41   8
##   10  21   7
##   11   7   2
##   12 155  24
##   13   4   0
##   14   4   0
##   15  58   6
##   16   2   0
##   18  98  15
##   20   7   1
##   21  27   3
##   22   2   0
##   24 158  26
##   26   1   0
##   27   8   5
##   28   2   1
##   30  36   4
##   33   3   0
##   36  70  13
##   39   5   0
##   40   1   0
##   42   9   2
##   45   5   0
##   47   1   0
##   48  37  11
##   54   1   1
##   60  12   1
##   72   1   0
##
## Tabla de contingencia (%) entre months_loan_duration y dependents:
##
##               1          2
##   4  0.00000000 1.00000000
##   5  1.00000000 0.00000000
##   6  0.76000000 0.24000000
##   7  1.00000000 0.00000000
##   8  0.85714286 0.14285714
##   9  0.83673469 0.16326531
##   10 0.75000000 0.25000000
##   11 0.77777778 0.22222222
##   12 0.86592179 0.13407821
```

```
##   13 1.00000000 0.00000000
##   14 1.00000000 0.00000000
##   15 0.90625000 0.09375000
##   16 1.00000000 0.00000000
##   18 0.86725664 0.13274336
##   20 0.87500000 0.12500000
##   21 0.90000000 0.10000000
##   22 1.00000000 0.00000000
##   24 0.85869565 0.14130435
##   26 1.00000000 0.00000000
##   27 0.61538462 0.38461538
##   28 0.66666667 0.33333333
##   30 0.90000000 0.10000000
##   33 1.00000000 0.00000000
##   36 0.84337349 0.15662651
##   39 1.00000000 0.00000000
##   40 1.00000000 0.00000000
##   42 0.81818182 0.18181818
##   45 1.00000000 0.00000000
##   47 1.00000000 0.00000000
##   48 0.77083333 0.22916667
##   54 0.50000000 0.50000000
##   60 0.92307692 0.07692308
##   72 1.00000000 0.00000000
## ----------------------------------------------------
## Tabla de contingencia entre credit_history y dependents:
##
##                            1   2
##   critical               244  49
##   delayed                 70  18
##   fully repaid            33   7
##   fully repaid this bank  36  13
##   repaid                 462  68
##
## Tabla de contingencia (%) entre credit_history y dependents:
##
##                                    1         2
##   critical               0.8327645 0.1672355
##   delayed                0.7954545 0.2045455
##   fully repaid           0.8250000 0.1750000
##   fully repaid this bank 0.7346939 0.2653061
##   repaid                 0.8716981 0.1283019
## ----------------------------------------------------
## Tabla de contingencia entre personal_status y dependents:
##
##                   1   2
##   divorced male  47   3
##   female        296  14
##   married male   90   2
##   single male   412 136
##
## Tabla de contingencia (%) entre personal_status y dependents:
##
##                           1          2
```

```
##   divorced male 0.94000000 0.06000000
##   female        0.95483871 0.04516129
##   married male  0.97826087 0.02173913
##   single male   0.75182482 0.24817518
## ----------------------------------------------------
## Tabla de contingencia entre other_debtors y dependents:
##
##                  1   2
##   co-applicant  37   4
##   guarantor     41  11
##   none         767 140
##
## Tabla de contingencia (%) entre other_debtors y dependents:
##
##                          1          2
##   co-applicant 0.90243902 0.09756098
##   guarantor    0.78846154 0.21153846
##   none         0.84564498 0.15435502
## ----------------------------------------------------
## Tabla de contingencia entre property y dependents:
##
##                              1   2
##   building society savings 197  35
##   other                    293  39
##   real estate              235  47
##   unknown/none             120  34
##
## Tabla de contingencia (%) entre property y dependents:
##
##                                    1         2
##   building society savings 0.8491379 0.1508621
##   other                    0.8825301 0.1174699
##   real estate              0.8333333 0.1666667
##   unknown/none             0.7792208 0.2207792
## ----------------------------------------------------
## Tabla de contingencia entre existing_credits y dependents:
##
##       1   2
##   1 550  83
##   2 273  60
##   3  18  10
##   4   4   2
##
## Tabla de contingencia (%) entre existing_credits y dependents:
##
##             1         2
##   1 0.8688784 0.1311216
##   2 0.8198198 0.1801802
##   3 0.6428571 0.3571429
##   4 0.6666667 0.3333333
## ----------------------------------------------------
## Tabla de contingencia entre age y dependents:
##
##       1  2
```

```
##    19  2  0
##    20 13  1
##    21 13  1
##    22 27  0
##    23 46  2
##    24 42  2
##    25 41  0
##    26 44  6
##    27 51  0
##    28 41  2
##    29 35  2
##    30 36  4
##    31 31  7
##    32 23 11
##    33 29  4
##    34 22 10
##    35 36  4
##    36 31  8
##    37 21  8
##    38 17  7
##    39 13  8
##    40 17  8
##    41 10  7
##    42 15  7
##    43 10  7
##    44 11  6
##    45  9  6
##    46 14  4
##    47 12  5
##    48  8  4
##    49 11  3
##    50 12  0
##    51  5  3
##    52  9  0
##    53  5  2
##    54  9  1
##    55  6  2
##    56  3  0
##    57  9  0
##    58  5  0
##    59  3  0
##    60  6  0
##    61  7  0
##    62  2  0
##    63  7  1
##    64  5  0
##    65  5  0
##    66  5  0
##    67  3  0
##    68  3  0
##    70  1  0
##    74  2  2
##    75  2  0
##
```

```
## Tabla de contingencia (%) entre age y dependents:
##
##               1          2
##   19 1.00000000 0.00000000
##   20 0.92857143 0.07142857
##   21 0.92857143 0.07142857
##   22 1.00000000 0.00000000
##   23 0.95833333 0.04166667
##   24 0.95454545 0.04545455
##   25 1.00000000 0.00000000
##   26 0.88000000 0.12000000
##   27 1.00000000 0.00000000
##   28 0.95348837 0.04651163
##   29 0.94594595 0.05405405
##   30 0.90000000 0.10000000
##   31 0.81578947 0.18421053
##   32 0.67647059 0.32352941
##   33 0.87878788 0.12121212
##   34 0.68750000 0.31250000
##   35 0.90000000 0.10000000
##   36 0.79487179 0.20512821
##   37 0.72413793 0.27586207
##   38 0.70833333 0.29166667
##   39 0.61904762 0.38095238
##   40 0.68000000 0.32000000
##   41 0.58823529 0.41176471
##   42 0.68181818 0.31818182
##   43 0.58823529 0.41176471
##   44 0.64705882 0.35294118
##   45 0.60000000 0.40000000
##   46 0.77777778 0.22222222
##   47 0.70588235 0.29411765
##   48 0.66666667 0.33333333
##   49 0.78571429 0.21428571
##   50 1.00000000 0.00000000
##   51 0.62500000 0.37500000
##   52 1.00000000 0.00000000
##   53 0.71428571 0.28571429
##   54 0.90000000 0.10000000
##   55 0.75000000 0.25000000
##   56 1.00000000 0.00000000
##   57 1.00000000 0.00000000
##   58 1.00000000 0.00000000
##   59 1.00000000 0.00000000
##   60 1.00000000 0.00000000
##   61 1.00000000 0.00000000
##   62 1.00000000 0.00000000
##   63 0.87500000 0.12500000
##   64 1.00000000 0.00000000
##   65 1.00000000 0.00000000
##   66 1.00000000 0.00000000
##   67 1.00000000 0.00000000
##   68 1.00000000 0.00000000
##   70 1.00000000 0.00000000
```

```
##   74 0.50000000 0.50000000
##   75 1.00000000 0.00000000
## ---------------------------------------------------
## Tabla de contingencia entre foreign_worker y dependents:
##
##         1   2
##   no   26  11
##   yes 819 144
##
## Tabla de contingencia (%) entre foreign_worker y dependents:
##
##               1         2
##   no  0.7027027 0.2972973
##   yes 0.8504673 0.1495327
## ---------------------------------------------------
## Tabla de contingencia entre job y dependents:
##
##                              1   2
##   mangement self-employed 127  21
##   skilled employee        551  79
##   unemployed non-resident  19   3
##   unskilled resident      148  52
##
## Tabla de contingencia (%) entre job y dependents:
##
##                                    1         2
##   mangement self-employed 0.8581081 0.1418919
##   skilled employee        0.8746032 0.1253968
##   unemployed non-resident 0.8636364 0.1363636
##   unskilled resident      0.7400000 0.2600000
## ---------------------------------------------------
## Tabla de contingencia entre default y dependents:
##
##       1   2
##   1 591 109
##   2 254  46
##
## Tabla de contingencia (%) entre default y dependents:
##
##             1         2
##   1 0.8442857 0.1557143
##   2 0.8466667 0.1533333
## ---------------------------------------------------
## Tabla de contingencia entre housing y dependents:
##
##              1   2
##   for free  78  30
##   own      607 106
##   rent     160  19
##
## Tabla de contingencia (%) entre housing y dependents:
##
##                    1         2
##   for free 0.7222222 0.2777778
```

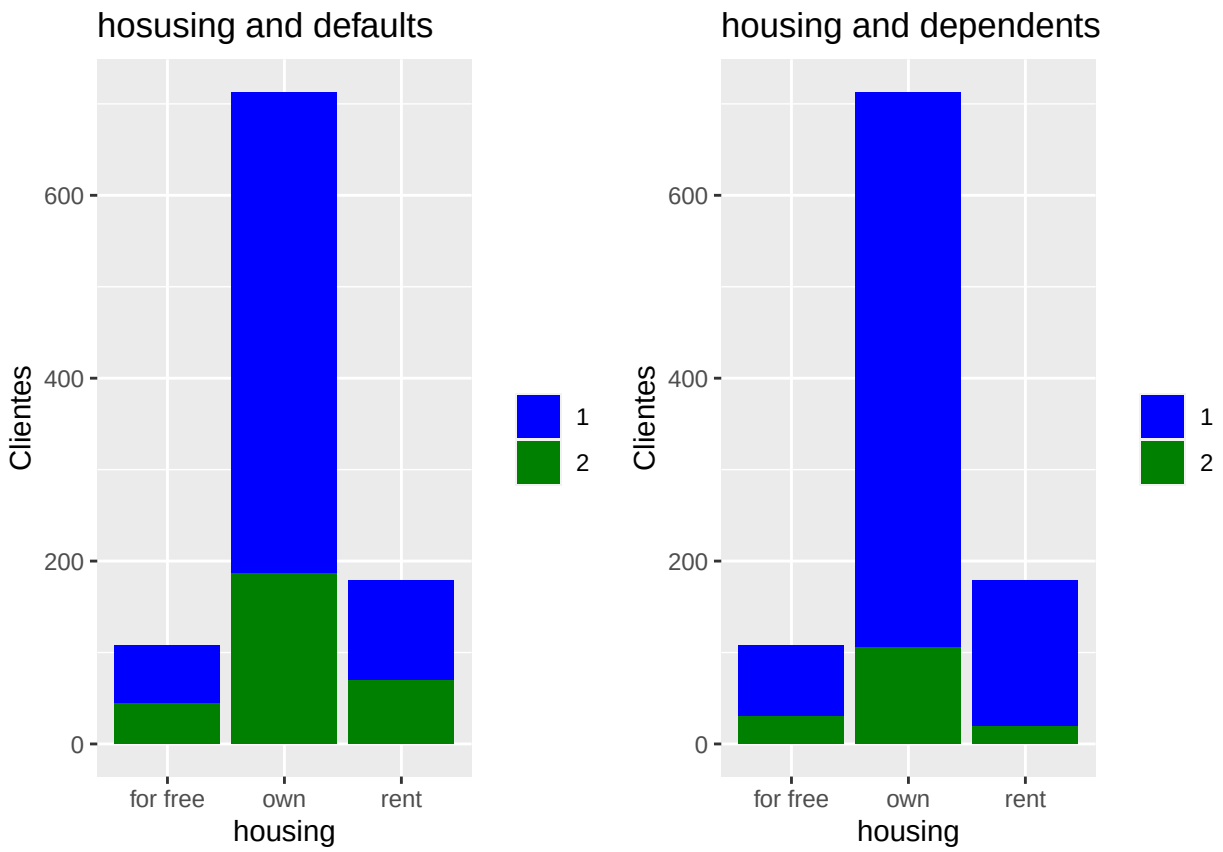
```
##   own      0.8513324 0.1486676
##   rent     0.8938547 0.1061453
## ----------------------------------------------------
## Tabla de contingencia entre purpose y dependents:
##
##                          1   2
##   business              82  15
##   car (new)            182  52
##   car (used)            81  22
##   domestic appliances   12   0
##   education             39  11
##   furniture            165  16
##   others                10   2
##   radio/tv             250  30
##   repairs               17   5
##   retraining             7   2
##
## Tabla de contingencia (%) entre purpose y dependents:
##
##                                1          2
##   business             0.84536082 0.15463918
##   car (new)            0.77777778 0.22222222
##   car (used)           0.78640777 0.21359223
##   domestic appliances  1.00000000 0.00000000
##   education            0.78000000 0.22000000
##   furniture            0.91160221 0.08839779
##   others               0.83333333 0.16666667
##   radio/tv             0.89285714 0.10714286
##   repairs              0.77272727 0.22727273
##   retraining           0.77777778 0.22222222
## ----------------------------------------------------
```

Para la relación entre la variable `dependents` vemos como a lo largo de todas las gráficas se mantiene un alto grado de propocionalidad, aunque para la gráfica `age` vemos un comportamiento un tanto interesante. En dicha gráfica se observa como para una edad intermedia, el número de personas dependientes aumenta. Esto quiere decir que los clientes que tienen una sola persona dependiente, suelen ser o muy jóvenes, o muy mayores, mientras que los clientes que tienen a dos personas dependientes, suelen ser de mediana edad y son los que más ocurrencias acumulan. Estudiando los resultados analíticos de las tabalas de contingencia, vemos como a partir de la gráfica `foreign_worker` hasta la gráfica de `default`, el mínimo porcentaje de clientes con una sola persona dependiente es del 70'27% frente al 29'73% de clientes con dos personas dependientes.

Podemos concluir esta relación de variables, diciendo que en este caso, la relación entre las personas dependientes y el resto de variables solo ha resultado dar información en el campo de la cantidad de meses que los clientes suelen tomarse para pagar un crédito.

Habiendo revisado de nuevo el conjunto de variables del juego de datos, el análisis que se acaba de realizar se va a repetir para la variable `housing`. Véase el siguiente *chunk* y más abajo el resultado que este arroja:

```
# Primero relacionamos housing con default
plotby_housing1<-ggplot(df_original,aes(housing,fill=default))+geom_bar() +labs(x="housing", y="Clientes

# Ahora relacionamos housing con dependents
plotby_housing2<-ggplot(df_original,aes(housing,fill=dependents))+geom_bar() +labs(x="housing", y="Clien

# Ahora representamos las dos gráficas
grid.arrange(plotby_housing1,plotby_housing2,ncol=2)
```



Viendo los resultados de arriba, no se observa ningún comportamiento extraño. Simplemente se puede apreciar como en la mayoría de los casos, dominan los clientes en estado de *default* (i.e., '`default=1`) con casas propias así como en los clientes con alquiler o que no pagan por su domicilio. No obstante, proporcionalmente hablando se observa como los clientes que no pagan por su alojamiento y aquellos que están alquilados, son menos propensos a acabar en una situación de *default*, de hecho, estudiando la tabla de contingencia de arriba asociada al par `housing-dependents`, vemos como proporcionalmente, los clientes con `housing=rent` son los que más en cuanto solo tener una única persona dependiente de ellos. Por otra parte en la gráfica de la derecha, puede observarse una diferencia en cuanto a proporciones, yq que aunque predominen los clientes con una sola persona dependiente, aquellos clientes que están de alquiler son menos propensos a tener dos personas dependientes, algo que tiene mucho sentido en la vida real.

Ahora vamos a profundizar en los tests estadísticos de significancia, para ello se va a instalar la librería "DescTools" y se van a aplicar `Phi(·)` y `CramerV(·)` a las tablas de contingencia, para poder estudiar la asociación estadística:

```
# instalamos el paquete necesario
if(!require(DescTools)){
install.packages('DescTools', repos='http://cran.us.r-project.org')
library(DescTools)
}
```

Ahora que ya se ha cargado el paquete, primero se utiliza para las tablas de contingencia relativas a la relación entre el subconjunto de variables seleccionadas, y la variable `default`

```
# Vamos a hacer uso de un bucle para poder crear todo más rápido
for (col in cols[-which(cols == 'default')]) {
  tabla_contingencia <- table(df_original[[col]], df_original$default)

  # Imprimimos la tabla de contingencia
```

```
  cat("Valores de Phi, de la relación entre", col, "y default: ",
      Phi(tabla_contingencia), '\n')

  # Ahora imprimimos la tabla de contingencia pero en proporciones
  cat("Valores de la V de Cramér, de la relación entre", col, "y default: ",
      CramerV(tabla_contingencia), "\n")
  cat('-----------------------------------------------------\n')
}
```

```
## Valores de Phi, de la relación entre checking_balance y default:  0.3517399
## Valores de la V de Cramér, de la relación entre checking_balance y default:  0.3517399
## -----------------------------------------------------
## Valores de Phi, de la relación entre months_loan_duration y default:  0.2808682
## Valores de la V de Cramér, de la relación entre months_loan_duration y default:  0.2808682
## -----------------------------------------------------
## Valores de Phi, de la relación entre credit_history y default:  0.2483775
## Valores de la V de Cramér, de la relación entre credit_history y default:  0.2483775
## -----------------------------------------------------
## Valores de Phi, de la relación entre personal_status y default:  0.09800619
## Valores de la V de Cramér, de la relación entre personal_status y default:  0.09800619
## -----------------------------------------------------
## Valores de Phi, de la relación entre other_debtors y default:  0.08151912
## Valores de la V de Cramér, de la relación entre other_debtors y default:  0.08151912
## -----------------------------------------------------
## Valores de Phi, de la relación entre dependents y default:  0.003014853
## Valores de la V de Cramér, de la relación entre dependents y default:  0.003014853
## -----------------------------------------------------
## Valores de Phi, de la relación entre property y default:  0.1540115
## Valores de la V de Cramér, de la relación entre property y default:  0.1540115
## -----------------------------------------------------
## Valores de Phi, de la relación entre existing_credits y default:  0.05168364
## Valores de la V de Cramér, de la relación entre existing_credits y default:  0.05168364
## -----------------------------------------------------
## Valores de Phi, de la relación entre age y default:  0.2397444
## Valores de la V de Cramér, de la relación entre age y default:  0.2397444
## -----------------------------------------------------
## Valores de Phi, de la relación entre foreign_worker y default:  0.0820795
## Valores de la V de Cramér, de la relación entre foreign_worker y default:  0.0820795
## -----------------------------------------------------
## Valores de Phi, de la relación entre job y default:  0.04341838
## Valores de la V de Cramér, de la relación entre job y default:  0.04341838
## -----------------------------------------------------
## Valores de Phi, de la relación entre housing y default:  0.1349068
## Valores de la V de Cramér, de la relación entre housing y default:  0.1349068
## -----------------------------------------------------
## Valores de Phi, de la relación entre purpose y default:  0.1826375
## Valores de la V de Cramér, de la relación entre purpose y default:  0.1826375
## -----------------------------------------------------
```

Como se ha visto en Cramer WIKI y en Phi WIKI, las dos funciones `CramerV(·)` y `Phi(·)` devuelven un valor $x : x \in [0, 1]$, donde 0 expresa una asociatividad nula entre las dos variables, y 1 la asociatividad máxima. Cabe destacar, como la función `CramerV(·)` se centra en la asociatividad de valores nominales. Además, en el ejemplo de esta PEC, se menciona que los valores obtenidos con estas dos funciones y comprendidos entre 0.1 y 0.3 indican una baja asociatividad, aquellos que se encuentren en el intervalo de 0.3-0.5 entonces

representarán una asociatividad media.

Observando los resultados de arriba, se puede ver claramente, como las dos variables que más grado de asociatividad comparten son: `checking_balance-default` con un 0'35174, seguido por el par `months_loan_duration-default` con un grado de asociatividad del 0.2808682, que a su vez le sigue el par `credit_history-default` con un 0'2483775.

En definitiva, las variables que más relación tienen con `default` son aquellas relacionadas con el dinero que tienen los clientes en la cuenta, seguido de la cantidad de meses que tienen para pagar el préstamo, y su historial crediticio. A estas variables se les prestará especial atención a la hora de aplicar el modelo, ya que la variable `default` será la variable que desearemos clasificar después de haber creado nuestro árbol de decisión, a fin de poder clasificar a los clientes que caerán en una situación de *default* (i.e., `default=1`) y a aquellos que no (`default=2`)

Ahora hacemos lo mismo pero para la relación entre todas las variables seleccionadas, y la variable `dependents`

```
# Vamos a hacer uso de un bucle para poder crear todo más rápido
for (col in cols[-which(cols == 'dependents')]) {
  tabla_contingencia <- table(df_original[[col]], df_original$dependents)

  # Imprimimos la tabla de contingencia
  cat("Valores de Phi, de la relación entre", col, "y dependents: ",
      Phi(tabla_contingencia), '\n')

  # Ahora imprimimos la tabla de contingencia pero en proporciones
  cat("Valores de la V de Cramér, de la relación entre", col, "y dependents: ",
      CramerV(tabla_contingencia), "\n")
  cat('-----------------------------------------------------\n')
}
```

```
## Valores de Phi, de la relación entre checking_balance y dependents:  0.07694403
## Valores de la V de Cramér, de la relación entre checking_balance y dependents:  0.07694403
## -----------------------------------------------------
## Valores de Phi, de la relación entre months_loan_duration y dependents:  0.2465693
## Valores de la V de Cramér, de la relación entre months_loan_duration y dependents:  0.2465693
## -----------------------------------------------------
## Valores de Phi, de la relación entre credit_history y dependents:  0.09768714
## Valores de la V de Cramér, de la relación entre credit_history y dependents:  0.09768714
## -----------------------------------------------------
## Valores de Phi, de la relación entre personal_status y dependents:  0.2842505
## Valores de la V de Cramér, de la relación entre personal_status y dependents:  0.2842505
## -----------------------------------------------------
## Valores de Phi, de la relación entre other_debtors y dependents:  0.04800818
## Valores de la V de Cramér, de la relación entre other_debtors y dependents:  0.04800818
## -----------------------------------------------------
## Valores de Phi, de la relación entre property y dependents:  0.09476967
## Valores de la V de Cramér, de la relación entre property y dependents:  0.09476967
## -----------------------------------------------------
## Valores de Phi, de la relación entre existing_credits y dependents:  0.120665
## Valores de la V de Cramér, de la relación entre existing_credits y dependents:  0.120665
## -----------------------------------------------------
## Valores de Phi, de la relación entre age y dependents:  0.3731319
## Valores de la V de Cramér, de la relación entre age y dependents:  0.3731319
## -----------------------------------------------------
## Valores de Phi, de la relación entre foreign_worker y dependents:  0.07707085
```

```
## Valores de la V de Cramér, de la relación entre foreign_worker y dependents:  0.07707085
## ----------------------------------------------------
## Valores de Phi, de la relación entre job y dependents:  0.1459557
## Valores de la V de Cramér, de la relación entre job y dependents:  0.1459557
## ----------------------------------------------------
## Valores de Phi, de la relación entre default y dependents:  0.003014853
## Valores de la V de Cramér, de la relación entre default y dependents:  0.003014853
## ----------------------------------------------------
## Valores de Phi, de la relación entre housing y dependents:  0.1261363
## Valores de la V de Cramér, de la relación entre housing y dependents:  0.1261363
## ----------------------------------------------------
## Valores de Phi, de la relación entre purpose y dependents:  0.1637498
## Valores de la V de Cramér, de la relación entre purpose y dependents:  0.1637498
## ----------------------------------------------------
```

Como podemos ver por los resultados obtenidos arriba, la mayor asociatividad estadística que se aprecia, es para el par de variables `age-dependents`, que cuenta con una grado de asociatividad de 0'3731319. Curiosamente, (o no tanto) este grado de asociatividad justifica la explicación de 4 *chunks* anteriores, relacionada con la proporcionalidad entre el aumento de número de personas dependientes, según aumenta la edad del cliente. Viendo el resultado podríamos decir que estas dos variables guardan una asociatividad estadística media, incluso más que la obtenida para el par anterior `checking_balance-default`, ya que 0'3731 > 0'35174. Al par `age-dependents` le sigue el par `personal_status-dependents`, con una asociatividad del 0'2842505, al que también le sigue el par `months_loan_duration-dependents` con una asociatividad de 0'2465693.

En definitiva, la variable de personas dependientes, está muy relacionada con la edad de los clientes, con el estado personal (mujer, hombre casado/divorciado, etc) y con el número de meses para pagar un crédito.

Como ya hemos mencionado antes, el objetivo de esta PEC es la de diseñar un árbol de decisión con el objetivo de estudiar y clasificar que clientes del juego de datos, caerán en una situación de impago o similares, a partir del resto de variables. Por ello la variable que se va a escoger para la clasificación será `default`.

De manera similar al ejemplo que se muestra al inicio de esta PEC, se va a visualizar el principio y el final del juego de datos, véase el siguiente *chunk* de codigo.

```
# visualizamos los primeros registros
head(df_original)
```

```
##   checking_balance months_loan_duration credit_history   purpose amount
## 1           < 0 DM                    6       critical  radio/tv   1169
## 2       1 - 200 DM                   48         repaid  radio/tv   5951
## 3          unknown                   12       critical education   2096
## 4           < 0 DM                   42         repaid furniture   7882
## 5           < 0 DM                   24        delayed car (new)   4870
## 6          unknown                   36         repaid education   9055
##   savings_balance employment_length installment_rate personal_status
## 1         unknown           > 7 yrs                4     single male
## 2        < 100 DM         1 - 4 yrs                2          female
## 3        < 100 DM         4 - 7 yrs                2     single male
## 4        < 100 DM         4 - 7 yrs                2     single male
## 5        < 100 DM         1 - 4 yrs                3     single male
## 6         unknown         1 - 4 yrs                2     single male
##   other_debtors residence_history                property age installment_plan
## 1          none                 4             real estate  67            none
## 2          none                 2             real estate  22            none
## 3          none                 3             real estate  49            none
## 4     guarantor                 4 building society savings  45            none
## 5          none                 4             unknown/none  53            none
```

```
## 6          none                 4             unknown/none  35             none
##    housing existing_credits default dependents telephone foreign_worker
## 1      own                2       1          1       yes            yes
## 2      own                1       2          1      none            yes
## 3      own                1       1          2      none            yes
## 4 for free                1       1          2      none            yes
## 5 for free                2       2          2      none            yes
## 6 for free                1       1          2       yes            yes
##                  job
## 1   skilled employee
## 2   skilled employee
## 3 unskilled resident
## 4   skilled employee
## 5   skilled employee
## 6 unskilled resident
```

```
#Visualizamos los últimos registros
tail(df_original)
```

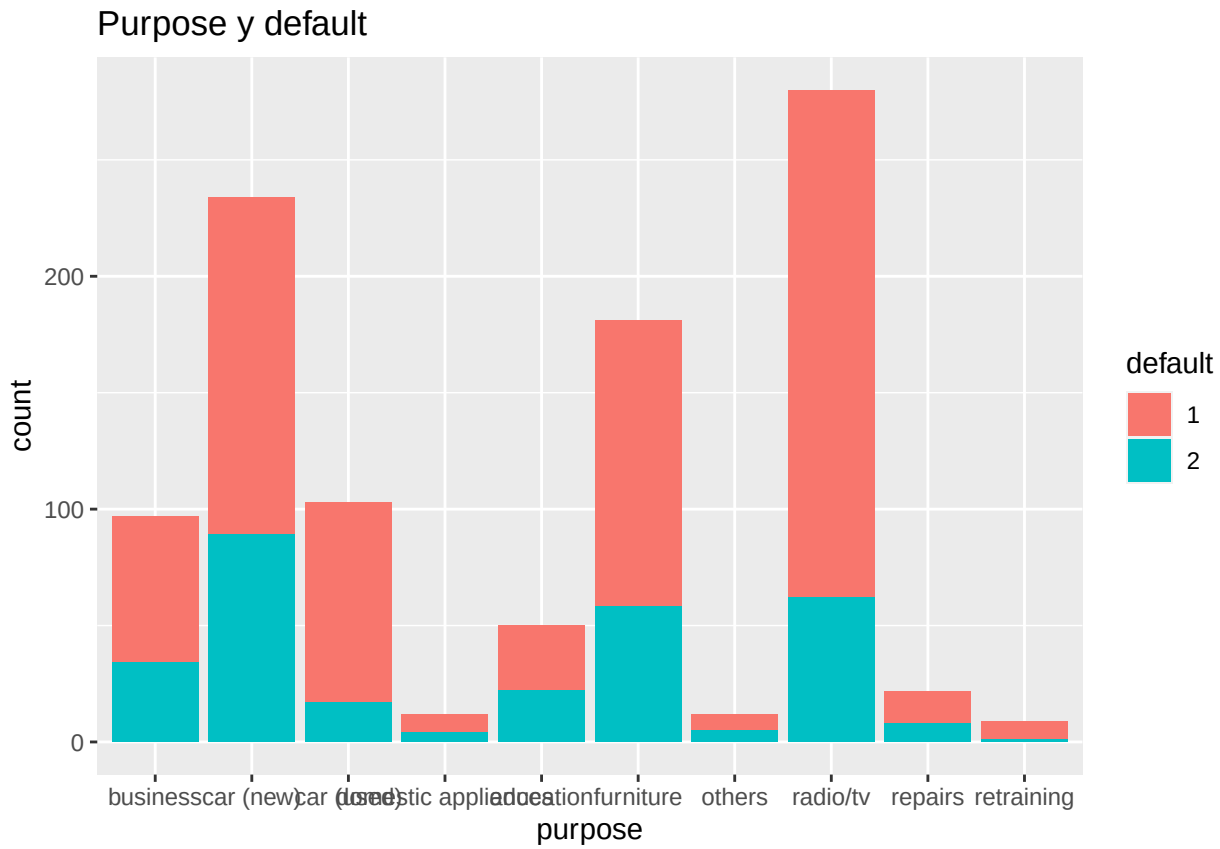
```
##      checking_balance months_loan_duration credit_history    purpose amount
## 995           unknown                   12         repaid  car (new)   2390
## 996           unknown                   12         repaid  furniture   1736
## 997            < 0 DM                   30         repaid car (used)   3857
## 998           unknown                   12         repaid   radio/tv    804
## 999            < 0 DM                   45         repaid   radio/tv   1845
## 1000       1 - 200 DM                   45       critical car (used)   4576
##      savings_balance employment_length installment_rate personal_status
## 995          unknown           > 7 yrs                4     single male
## 996         < 100 DM         4 - 7 yrs                3          female
## 997         < 100 DM         1 - 4 yrs                4   divorced male
## 998         < 100 DM           > 7 yrs                4     single male
## 999         < 100 DM         1 - 4 yrs                4     single male
## 1000    101 - 500 DM        unemployed                3     single male
##      other_debtors residence_history                property age
## 995           none                 3                   other  50
## 996           none                 4              real estate  31
## 997           none                 4 building society savings  40
## 998           none                 4                   other  38
## 999           none                 4             unknown/none  23
## 1000          none                 4                   other  27
##      installment_plan  housing existing_credits default dependents telephone
## 995              none      own                1       1          1       yes
## 996              none      own                1       1          1      none
## 997              none      own                1       1          1       yes
## 998              none      own                1       1          1      none
## 999              none for free                1       2          1       yes
## 1000             none      own                1       1          1      none
##      foreign_worker                    job
## 995             yes        skilled employee
## 996             yes      unskilled resident
## 997             yes mangement self-employed
## 998             yes        skilled employee
## 999             yes        skilled employee
## 1000            yes        skilled employee
```

```r
# vemos de nuevo cuantos niveles de valores hay en cada variable
str(df_original)
```

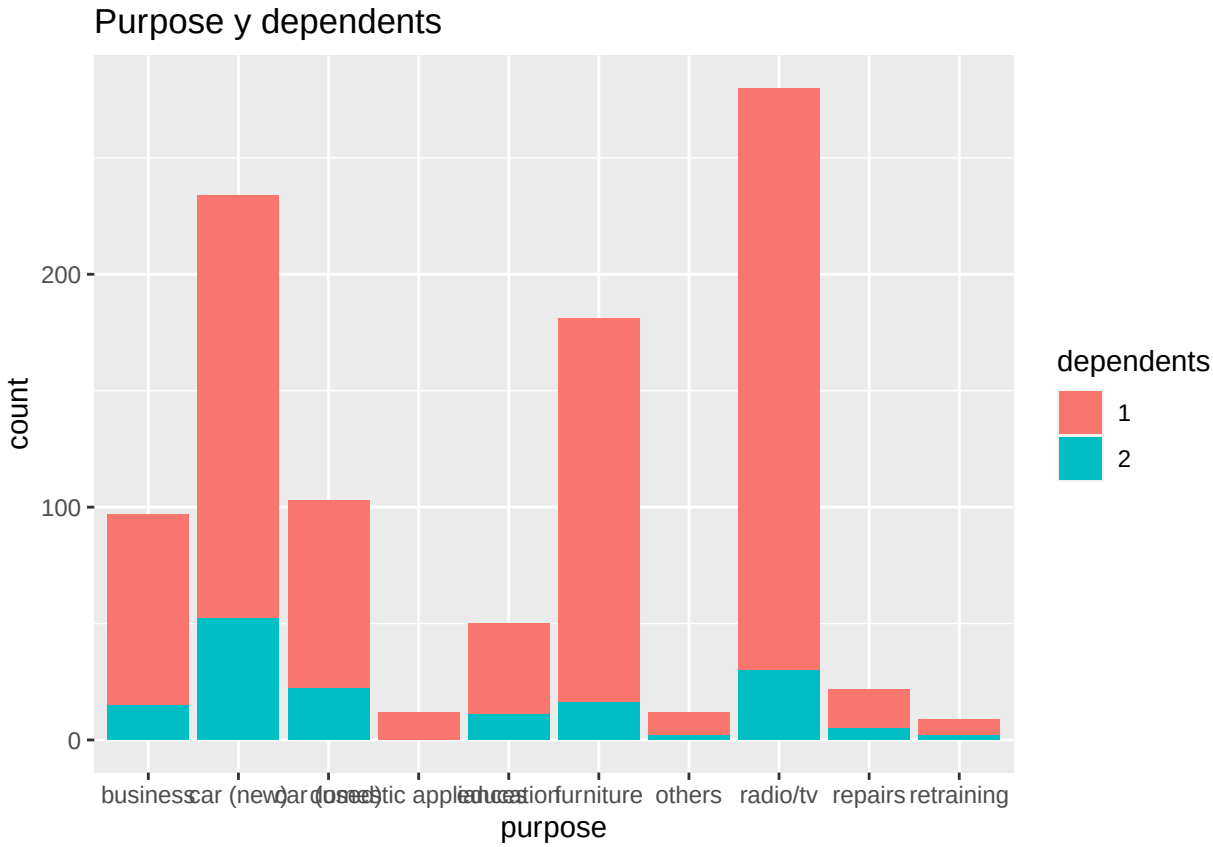
```
## 'data.frame':    1000 obs. of  21 variables:
##  $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4 1 1 4 4 3 4 3 ...
##  $ months_loan_duration: Factor w/ 33 levels "4","5","6","7",..: 3 30 9 27 18 24 18 24 9 22 ...
##  $ credit_history      : Factor w/ 5 levels "critical","delayed",..: 1 5 1 5 2 5 5 5 5 1 ...
##  $ purpose             : Factor w/ 10 levels "business","car (new)",..: 8 8 5 6 2 5 6 3 8 2 ...
##  $ amount              : Factor w/ 921 levels "250","276","338",..: 143 771 391 849 735 870 534 814 5
##  $ savings_balance     : Factor w/ 5 levels "< 100 DM","> 1000 DM",..: 5 1 1 1 1 5 4 1 2 1 ...
##  $ employment_length   : Factor w/ 5 levels "> 7 yrs","0 - 1 yrs",..: 1 3 4 4 3 3 1 3 4 5 ...
##  $ installment_rate    : Factor w/ 4 levels "1","2","3","4": 4 2 2 2 3 2 3 2 2 4 ...
##  $ personal_status     : Factor w/ 4 levels "divorced male",..: 4 2 4 4 4 4 4 4 1 3 ...
##  $ other_debtors       : Factor w/ 3 levels "co-applicant",..: 3 3 3 2 3 3 3 3 3 3 ...
##  $ residence_history   : Factor w/ 4 levels "1","2","3","4": 4 2 3 4 4 4 4 2 4 2 ...
##  $ property            : Factor w/ 4 levels "building society savings",..: 3 3 3 1 4 4 1 2 3 2 ...
##  $ age                 : Factor w/ 53 levels "19","20","21",..: 49 4 31 27 35 17 35 17 43 10 ...
##  $ installment_plan    : Factor w/ 3 levels "bank","none",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ housing             : Factor w/ 3 levels "for free","own",..: 2 2 2 1 1 1 2 3 2 2 ...
##  $ existing_credits    : Factor w/ 4 levels "1","2","3","4": 2 1 1 1 2 1 1 1 1 2 ...
##  $ default             : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...
##  $ dependents          : Factor w/ 2 levels "1","2": 1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone           : Factor w/ 2 levels "none","yes": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreign_worker      : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ job                 : Factor w/ 4 levels "mangement self-employed",..: 2 2 4 2 2 4 2 1 4 1 ...
```

Acabo de ver como la variable `purpose` también puede ser interesante, por eso se va a analizar a continuación e individualmente para analizar si puede ser una variable determinante en la posterior de clasificación. Véase a continuación el siguiente *chunk* de código:

```r
# Primero relacionamos purpose con default
ggplot(data = df_original, aes(x=purpose,fill=default))+geom_bar()+ggtitle("Purpose y default")+labs(x="
```




```
# Ahora relacionamos purpose con dependents
ggplot(data = df_original, aes(x=purpose,fill=dependents))+geom_bar()+ggtitle("Purpose y dependents")+la
```



Por los resultados obtenidos, vemos como las asociatividades estadísticas, tanto para el par `purpose-default` como para el par `purpose-dependents`, se observan las siguientes asociatividades 0'1826375 y 0'1637498 respectivamente. Haciendo atención a los resultados gráficos, podemos ver como para la gráfica 'Purpose y default' vemos como mayoritariamente predominan los clientes con `default=1` no obstante, para los préstamos de coches nuevos, educación, reparaciones u otros, también es muy probable que el cliente no acabe en situación de *default*, para estos purposes se obtienen los siguientes porcentajes de *default=1*, 61'96%, 56%, 63'63% y 58'33%. Por último, para la gráfica 'Purpose y dependents' no se observa ningun comportamiento extraño, solamente decir que como a lo largo de esta práctica se ha visto, predominan los clientes en situación de *default*.

Ahora que ya se han relacionado una gran parte de las variables, y ya se han identificado aquellas que pueden resultar determinantes para la clasificación, se procede a la preparación de los datos para el modelo. Esto es importante, sobretodo cuando se acometa la evaluación del árbol de decisión diseñado. En esta etapa de preparación de los datos, tendremos que separar el juego de datos en un conjunto destinado al entrenamiento, y en otro destinado al *test* del modelo (esto es algo que ya hemos visto en temas anteriores, y es algo muy importante ya que hay que hacerlo bien para evitar que el modelo clasificatorio se desconcierte, en caso de que este vea un dato a la hora del *test* que previamente no ha visto en la etapa de entrenamiento)

Como ya hemos estudiado en teoría, lo más apropiado es emplear un conjunto de datos distinto al que se va a usar para desarrollar el árbol de decisión, es decir, un conjunto que no sea el de entrenamiento. Sabemos que no existe una proporción predefinida en relación con el número relativo de elementos en cada subconjunto, pero la proporción más comúnmente adoptada suele ser de 2/3 para el conjunto de entrenamiento y de 1/3 para el conjunto de prueba (exactamente lo que hemos visto en teoría).

Como hemos mencionado antes, la variable que determinará la tarea de clasificación, será `default`, ahora vamos a comenzar con la división en el siguiente *chunk* de código

```
head(df_original)
```

```
##   checking_balance months_loan_duration credit_history   purpose amount
## 1           < 0 DM                    6       critical  radio/tv   1169
## 2       1 - 200 DM                   48         repaid  radio/tv   5951
## 3          unknown                   12       critical education   2096
## 4           < 0 DM                   42         repaid furniture   7882
## 5           < 0 DM                   24        delayed car (new)   4870
## 6          unknown                   36         repaid education   9055
##   savings_balance employment_length installment_rate personal_status
## 1         unknown           > 7 yrs                4     single male
## 2        < 100 DM         1 - 4 yrs                2          female
## 3        < 100 DM         4 - 7 yrs                2     single male
## 4        < 100 DM         4 - 7 yrs                2     single male
## 5        < 100 DM         1 - 4 yrs                3     single male
## 6         unknown         1 - 4 yrs                2     single male
##   other_debtors residence_history                property age installment_plan
## 1          none                 4             real estate  67             none
## 2          none                 2             real estate  22             none
## 3          none                 3             real estate  49             none
## 4     guarantor                 4 building society savings  45             none
## 5          none                 4            unknown/none  53             none
## 6          none                 4            unknown/none  35             none
##    housing existing_credits default dependents telephone foreign_worker
## 1      own                2       1          1       yes            yes
## 2      own                1       2          1      none            yes
## 3      own                1       1          2      none            yes
## 4 for free                1       1          2      none            yes
## 5 for free                2       2          2      none            yes
## 6 for free                1       1          2       yes            yes
##                  job
## 1   skilled employee
## 2   skilled employee
## 3 unskilled resident
## 4   skilled employee
## 5   skilled employee
## 6 unskilled resident
```

```r
set.seed(666)
y <- df_original[,17] # default está en la columna 17

# hacemos la selección de columnas para no coger la etiqueta col(17)
rest_cols = c(1:16, 18:21)
rest_cols2 = c(1:4)
cols_omit = c(17)

# x <- df_original[, setdiff(rest_cols, cols_omit)]
x <- df_original[, setdiff(rest_cols, cols_omit)]
```

Ahora que ya tenemos los conjuntos para el entrenamiento y validación (i.e., *test*) vamos a definir de manera dinámica la manera de separar en función de un parámetro, a fin de poder definir un parámetro que controla el *split* de forma dinámica.

```r
split_prop <- 3
indexes = sample(1:nrow(df_original), size=floor(((split_prop-1)/split_prop)*nrow(df_original)))
trainx<-x[indexes,]
trainy<-y[indexes]
```

```
testx<-x[-indexes,]
testy<-y[-indexes]
```

En el código de arriba, primero se está determinando el factor con el que se va a dividir el conjunto, luego generamos un conjunto aleatorio de índices que usaremos a fin de dividir el juego de datos original, en un subconjunto destinado al entrenamiento del modelo, y en otro destinado a su validación. El tamaño de este conjunto de índices viene dado por el factor especificado en la variable `split_prop`. Conocidos los índices, podemos generar los conjuntos de *train* y de *test*, y esto es lo que se hace en las siguientes lineas. La variable `trainx` contiene el conjunto de datos destinado al entrenamiento del modelo, formado a partir de la selección con los índices generados antes, de las filas del *dataframe* `x`. Luego, en la variable `trainy` se guardan los datos etiquetados, necesarios para el entrenamiento. Ya por último, en las variables `testx` y `testy` se hace exactamente lo mismo que en las variables `trainx` y `trainy` respectivamente, a diferencia de que ahora, la selección de las filas en las variables `testx` y `testy` se realiza especificando un "-" delante, indicando la selección de las filas que no están en el conjunto de datos del entrenamiento.

Ya hemos extraído de manera estocástica los casos, por ello es imprescindible comprobar que todos los subconjuntos de datos que se han creado no contienen ningún error. Por esta razón, primero se va a comprobar que la proporción de clientes en situación de *default* es constante en los dos nuevos conjuntos.

```
summary(trainx)
```

```
##    checking_balance months_loan_duration                credit_history
##  < 0 DM    :182     24     :122          critical              :190
##  > 200 DM  : 42     12     :115          delayed               : 60
##  1 - 200 DM:189     18     : 85          fully repaid          : 26
##  unknown   :253     36     : 60          fully repaid this bank: 33
##                     6      : 43          repaid                :357
##                     15     : 38
##                     (Other):203
##        purpose        amount         savings_balance  employment_length
##  radio/tv  :181   1478   :  3   < 100 DM     :406    > 7 yrs   :162
##  car (new) :155   433    :  2   > 1000 DM    : 32    0 - 1 yrs :110
##  furniture :128   609    :  2   101 - 500 DM : 65    1 - 4 yrs :230
##  car (used): 68   932    :  2   501 - 1000 DM: 37    4 - 7 yrs :120
##  business  : 66   1082   :  2   unknown      :126    unemployed: 44
##  education : 32   1126   :  2
##  (Other)   : 36   (Other):653
##  installment_rate      personal_status      other_debtors residence_history
##  1: 99            divorced male: 34    co-applicant: 33   1: 83
##  2:151            female       :207    guarantor   : 31   2:203
##  3:102            married male : 56    none        :602   3: 96
##  4:314            single male  :369                       4:284
##
##
##
##                      property        age      installment_plan     housing
##  building society savings:148   26     : 34   bank  : 86       for free: 73
##  other                  :220   27     : 34   none  :551       own     :475
##  real estate            :194   28     : 34   stores: 29       rent    :118
##  unknown/none           :104   23     : 32
##                                30     : 30
##                                25     : 28
##                                (Other):474
##  existing_credits dependents telephone  foreign_worker
##  1:420            1:569      none:394   no : 22
```

```
##  2:225            2: 97      yes :272   yes:644
##  3: 16
##  4:  5
##
##
##
##                       job
##  mangement self-employed:103
##  skilled employee       :428
##  unemployed non-resident: 16
##  unskilled resident     :119
##
##
##
```

```
summary(trainy)
```

```
##   1   2
## 466 200
```

```
summary(testx)
```

```
##    checking_balance months_loan_duration               credit_history
##  < 0 DM    : 92     12     :64           critical              :103
##  > 200 DM  : 21     24     :62           delayed               : 28
##  1 - 200 DM: 80     6      :32           fully repaid          : 14
##  unknown   :141     18     :28           fully repaid this bank: 16
##                     15     :26           repaid                :173
##                     36     :23
##                     (Other):99
##        purpose       amount         savings_balance  employment_length
##  radio/tv  :99   1154   :  2   < 100 DM     :197    > 7 yrs   : 91
##  car (new) :79   250    :  1   > 1000 DM    : 16    0 - 1 yrs : 62
##  furniture :53   339    :  1   101 - 500 DM : 38    1 - 4 yrs :109
##  car (used):35   362    :  1   501 - 1000 DM: 26    4 - 7 yrs : 54
##  business  :31   385    :  1   unknown      : 57    unemployed: 18
##  education :18   426    :  1
##  (Other)   :19   (Other):327
##  installment_rate      personal_status      other_debtors residence_history
##  1: 37            divorced male: 16    co-applicant:  8   1: 47
##  2: 80            female       :103    guarantor   : 21   2:105
##  3: 55            married male : 36    none        :305   3: 53
##  4:162            single male  :179                       4:129
##
##
##
##                      property        age      installment_plan     housing
##  building society savings: 84   24     : 20   bank  : 53       for free: 35
##  other                    :112   27     : 17   none  :263       own     :238
##  real estate              : 88   29     : 17   stores: 18       rent    : 61
##  unknown/none             : 50   23     : 16
##                                  26     : 16
##                                  35     : 14
##                                  (Other):234
##  existing_credits dependents telephone  foreign_worker
```

```
##  1:213            1:276      none:202   no : 15
##  2:108            2: 58      yes :132   yes:319
##  3: 12
##  4:  1
##
##
##
##                        job
##  mangement self-employed: 45
##  skilled employee       :202
##  unemployed non-resident:  6
##  unskilled resident     : 81
##
##
##
```

```
summary(testy)
```

```
##   1   2
## 234 100
```

Estudiando los resultados de arriba podemos afirmar, que los conjuntos que se han creado son correctos ya que, en en el caso de `trainy` vemos como sumando los clientes en situación de *default* (i.e., 466) con aquellos que no lo están (i.e., 200) se obtienen 666 clientes, que se corresponden con el 66'6% del total de clientes (i.e., 666/1000). Además, analizando la composición de esta cifra, vemos como los 466 clientes en situación de *default* constituyen el 69'969% del total de dicho subconjunto (i.e., 466/666), mientras que los otros clientes con `default=2` constituyen el 30'030% restante. Esta dinámica se repite en el conjunto `testy`, pues los clientes en situación de *default* (234) constituyen el 70'06% del total del subconjunto `testy` (i.e., 234/334) mientras que el resto de clientes consitutyen el 29'94% (i.e., 100/334).

Finalmente vemos como el conjunto `trainy` refleja el 66'66% de los datos totales dentro del juego de datos, mientras que el conjunto de datos destinado al *test* del modelo está compuesto por el 33'4%, cumpliendo con la proporción de división mencionada antes y vista en teoría.

Realizar un primer árbol de decisión. Puedes decidir utilizar todas las variables o, de forma justificada, quitar alguna para el ajuste del modelo

Como se ha podido ver en el anterior ejercicio, los datos ya están limpios, ya se han destacado las variables interesantes, y hemos creado los conjuntos de datos tanto para el entrenamiento del modelo como para su correcta validación, cumpliendo con los estándares vistos en teoría.

Es por lo anterior, que ya estamos en condiciones de construir el modelo que nos va a permitir predecir/clasificar los clientes que caerán o no en situación de *default* y consecuentemente, saber quiénes cumplirán (potencialmente) con el pago de un crédito.

Como se ha podido estudiar en teoría, los árboles de decisión juegan un papel muy importante en el campo del aprendizaje automático, no solo por su potencia sino también por su versatilidad e intuitividad. Estos permiten al programador conocer y determinar los aspectos específicos de un árbol. Los árboles de decicisón son uno de los modelos supervisados de clasificación que se usan más en problemas de minería de datos, principalmente por su alta capacidad explicativa debido a que es muy fácil de interpretar. Como hemos estudiado, estas estructuras pueden implementarse tanto en problemas supervisados de clasificación como en problemas supervisados de regresión. Recordamos como los problemas supervisados, eran aquellos que basan su tarea (clasificar, predecir, etc) en resultados y datos ya conocidos, i.e., datos "etiquetados", mientras que los problemas no supervisados, no precisan de datos etiquetados, no obstante se requieren de un mayor número de expertos en la materia, para asegurarse que los resultados obtenidos son correctos.

La idea principal que cimienta el concepto de los árboles de decisión es la división del espacio de datos de

entrada que acometen, a fin de crear regiones separadas, asegurando que todas las muestras en una región pertenezcan a la misma clase. En caso de que una región contenga muestras de clases diferentes, se divide en regiones más pequeñas utilizando el mismo criterio. Este proceso continúa hasta que todas las regiones contienen solo muestras de una clase. Un árbol de decisión se considera completo o puro si es factible construir un árbol que cumpla con esta condición.

Teniendo los datos preparados y los concepts afianzados, procedemos con la construcción del árbol que vamos a implementar. Es aquí donde comienza la etapa de creación del modelo. Antes de construir el modelo, cabe destacar, que para la creación del árbol de decisión hemos tomado todos los datos del juego original (i.e., todas las variables, las 21) en caso de observar un funcionamiento no deseado, se suprimirán las variables menos trascendentales, pero tendremos que volver a generar los conjuntos de `train` y `test`. Véase el siguiente *chunk* de código:

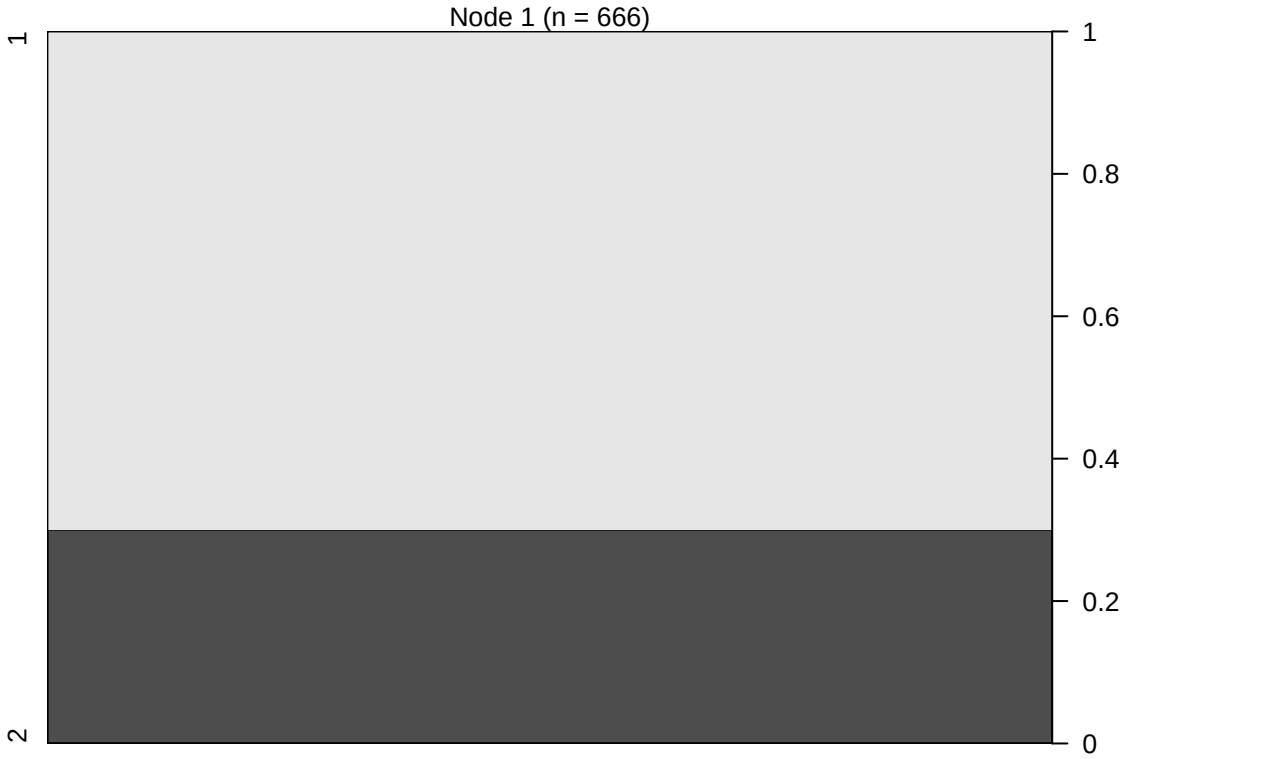
```
trainy <- as.factor(trainy)
model <- C50::C5.0(trainx, trainy,rules=TRUE )
summary(model)
```

```
##
## Call:
## C5.0.default(x = trainx, y = trainy, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Feb  4 22:52:43 2024
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 666 cases (21 attributes) from undefined.data
##
## Rules:
##
## Default class: 1
##
##
## Evaluation on training data (666 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       0  200(30.0%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     466          (a): class 1
##     200          (b): class 2
##
##
## Time: 0.0 secs
```

Como se puede observar, no obtenemos buenos resultados. Para empezar, el algoritmo no nos imprime las reglas que el árbol debería de seguir a la hora de clasificar, por lo tanto hay algo que no funciona bien. Esto se puede deber a la gran cantidad de variables que el árbol está teniendo en cuenta. Observando el resultadp de la clasificación que el algoritmo ha llevado a cabo, podemos ver como el árbol ha clasificado solo el 70% correctamente, pero el otro 30% no aparece como tal, solamente se nos indica que han habido 200 errores.

Para tener más información al respecto, se va representar el árbol de decisión que se ha creado, véase el siguiente *chunk* de código:

```
model <- C50::C5.0(trainx, trainy)
plot(model,gp = gpar(fontsize = 9.5))
```



Como puede comprobarse, no se ha conseguido ningún árbol como tal, sino un nodo. Por teória sabemos que los árboles de decisión están compuestos por nodos terminales, que representan regiones etiquetadas de acuerdo a una clase, y nodos internos que representan condiciones que permiten decidir a que subregión va cada elemento que llega a dicho nodo. Observando el "árbol" que hemos obtenido arriba, nos damos cuenta de que al haber un nodo terminal, esto podría deberse al hecho de que algoritmo, al tener tantas variables, ha generalizado, y de golpe ha determinado que todos excepto un 20% (que no muestra) se tratan de clientes en situación de `default`.

Como es obvio, esto es un error, por lo tanto, se van a seleccionar las variables que tienen un mayor grado de asociatividad estadística, y que se destacaron en el anterior ejercicio. Las variables para el siguiente árbol son las siguientes:

- `default`
- `checking_balance`
- `months_loan_duration`
- `credit_history`
- `purpose`
- `age`

Y opcionalmente, seleccionaremos la variable `purpose` después de obtener el árbol, habiendo contemplado solo esas variables.

Ahora volvemos a seleccionar las variables deseadas del conjunto original de datos, y repetimos el último proceso del ejercicion anterior, véase el siguiente *chunk*

```
head(df_original)
```

```
##   checking_balance months_loan_duration credit_history   purpose amount
## 1           < 0 DM                    6       critical  radio/tv   1169
## 2       1 - 200 DM                   48         repaid  radio/tv   5951
## 3          unknown                   12       critical education   2096
## 4           < 0 DM                   42         repaid furniture   7882
## 5           < 0 DM                   24        delayed car (new)   4870
## 6          unknown                   36         repaid education   9055
##   savings_balance employment_length installment_rate personal_status
## 1         unknown           > 7 yrs                4     single male
## 2        < 100 DM         1 - 4 yrs                2          female
## 3        < 100 DM         4 - 7 yrs                2     single male
## 4        < 100 DM         4 - 7 yrs                2     single male
## 5        < 100 DM         1 - 4 yrs                3     single male
## 6         unknown         1 - 4 yrs                2     single male
##   other_debtors residence_history                property age installment_plan
## 1          none                 4             real estate  67             none
## 2          none                 2             real estate  22             none
## 3          none                 3             real estate  49             none
## 4     guarantor                 4 building society savings  45             none
## 5          none                 4            unknown/none  53             none
## 6          none                 4            unknown/none  35             none
##    housing existing_credits default dependents telephone foreign_worker
## 1      own                2       1          1       yes            yes
## 2      own                1       2          1      none            yes
## 3      own                1       1          2      none            yes
## 4 for free                1       1          2      none            yes
## 5 for free                2       2          2      none            yes
## 6 for free                1       1          2       yes            yes
##                  job
## 1   skilled employee
## 2   skilled employee
## 3 unskilled resident
## 4   skilled employee
## 5   skilled employee
## 6 unskilled resident
```

```r
set.seed(666)
# creamos un data frame nuevo que contenga solo las columnas que queremos:
selec_cols = c('checking_balance', 'months_loan_duration', 'credit_history',
               'purpose', 'default')

# rest_cols2 = c(1:4)
# cols_omit = c(17)
df_original_sub <- df_original[, selec_cols]
head(df_original_sub)
```

```
##   checking_balance months_loan_duration credit_history   purpose default
## 1           < 0 DM                    6       critical  radio/tv       1
## 2       1 - 200 DM                   48         repaid  radio/tv       2
## 3          unknown                   12       critical education       1
## 4           < 0 DM                   42         repaid furniture       1
## 5           < 0 DM                   24        delayed car (new)       2
## 6          unknown                   36         repaid education       1
```

```
# ahora separamos el resto de variables de la etiqueta (variable a clasificar)
y <- df_original_sub[,5] # seleccionamos la columna de default
x <- df_original_sub[,1:4]
head(x)
```

```
##   checking_balance months_loan_duration credit_history   purpose
## 1           < 0 DM                    6       critical  radio/tv
## 2       1 - 200 DM                   48         repaid  radio/tv
## 3          unknown                   12       critical education
## 4           < 0 DM                   42         repaid furniture
## 5           < 0 DM                   24        delayed car (new)
## 6          unknown                   36         repaid education
```

```
head(y)
```

```
## [1] 1 2 1 1 2 1
## Levels: 1 2
```

Ahora que ya hemos creado el nuevo conjunto de datos, procedemos a dividir y crear los nuevos conjuntos:

```
split_prop <- 3
indexes = sample(1:nrow(df_original_sub), size=floor(((split_prop-1)/split_prop)*nrow(df_original_sub)))
trainx<-x[indexes,]
trainy<-y[indexes]
testx<-x[-indexes,]
testy<-y[-indexes]
```

Ahora comprobamos los conjuntos, tal y como hicimos la primera vez, véase el siguiente *chunk* de código:

```
summary(trainx)
```

```
##    checking_balance months_loan_duration                credit_history
##  < 0 DM    :182     24     :122          critical              :190
##  > 200 DM  : 42     12     :115          delayed               : 60
##  1 - 200 DM:189     18     : 85          fully repaid          : 26
##  unknown   :253     36     : 60          fully repaid this bank: 33
##                     6      : 43          repaid                :357
##                     15     : 38
##                     (Other):203
##        purpose
##  radio/tv  :181
##  car (new) :155
##  furniture :128
##  car (used): 68
##  business  : 66
##  education : 32
##  (Other)   : 36
```

```
summary(trainy)
```

```
##   1   2
## 466 200
```

```
summary(testx)
```

```
##    checking_balance months_loan_duration                credit_history
##  < 0 DM    : 92     12     :64           critical              :103
##  > 200 DM  : 21     24     :62           delayed               : 28
```

```
##  1 - 200 DM: 80     6      :32          fully repaid          : 14
##  unknown   :141     18     :28          fully repaid this bank: 16
##                     15     :26          repaid                :173
##                     36     :23
##                     (Other):99
##        purpose
##  radio/tv  :99
##  car (new) :79
##  furniture :53
##  car (used):35
##  business  :31
##  education :18
##  (Other)   :19
```

```
summary(testy)
```

```
##   1   2
## 234 100
```

Nada extraño en los nuevos conjuntos, además los valores (proporciones) que se han obtenido de `default` son los mismos que en el primer intento, por lo tanto confirmamos que estos conjuntos de datos nuevos cumplen con los requisitos.

Ahora si, llevamos a cabo la creación del modelo. Mirése el siguiente *chunk*.

```
trainy <- as.factor(trainy)
model <- C50::C5.0(trainx, trainy,rules=TRUE )
summary(model)
```

```
##
## Call:
## C5.0.default(x = trainx, y = trainy, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Feb  4 22:52:43 2024
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 666 cases (5 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (253/36, lift 1.2)
##  checking_balance = unknown
##  ->  class 1  [0.855]
##
## Rule 2: (250/47, lift 1.2)
##  credit_history in {critical, delayed}
##  ->  class 1  [0.810]
##
## Rule 3: (533/133, lift 1.1)
##  months_loan_duration in {4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
##                           18, 20, 21, 22, 24, 26, 27, 33, 39, 40, 42, 47,
##                           54}
##  ->  class 1  [0.750]
```

```
##
## Rule 4: (43/12, lift 2.4)
##  checking_balance in {< 0 DM, 1 - 200 DM}
##  months_loan_duration in {28, 30, 36, 45, 48, 60}
##  credit_history = repaid
##  ->  class 2  [0.711]
##
## Rule 5: (52/18, lift 2.2)
##  checking_balance in {< 0 DM, > 200 DM, 1 - 200 DM}
##  credit_history in {fully repaid, fully repaid this bank}
##  ->  class 2  [0.648]
##
## Default class: 1
##
##
## Evaluation on training data (666 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       5  181(27.2%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     446    20    (a): class 1
##     161    39    (b): class 2
##
##
##  Attribute usage:
##
##   86.49% months_loan_duration
##   52.25% checking_balance
##   51.80% credit_history
##
##
## Time: 0.0 secs
```

Gracias a `summary(model)` podemos visualizar el número de errores así como el porcentaje de casos mal clasificados en el subconjunto de entrenamiento. Viendo los resultados de arriba, podemos comprobar como el árbol clasifica de manera incorrecta 181 clientes, de los 666 casos totales, lo que se corresponde con un tasa de error del 27,2% (a mi parecer, un error un poco alto)

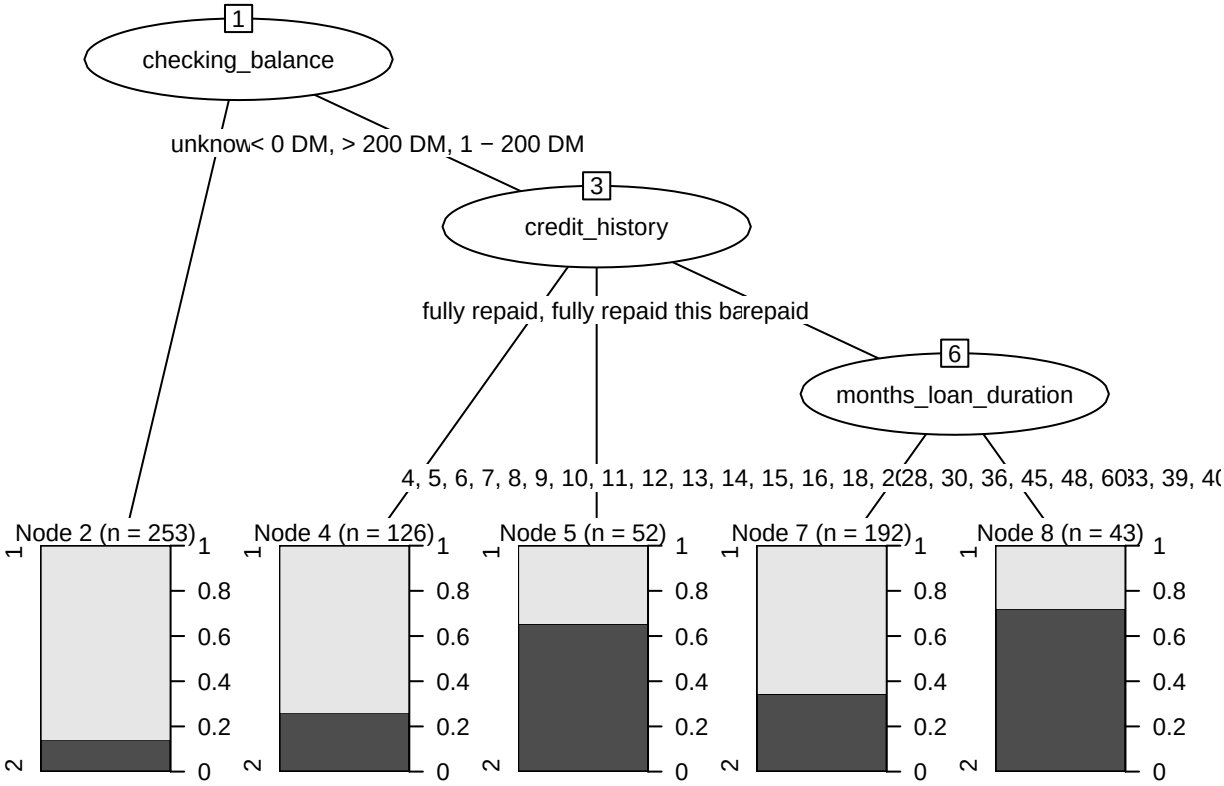
Respecto a las reglas de decisión, al haber especifocado: `rules = TRUE` las podemos visualizar arriba. Las reglas de decisión son 5, y son las siguientes:

- **checking_balance** = "unknown" → `default=1`. Validez: 85'5%
- **credit_history** = {"critical", "delayed"} → `default=1`. Validez: 81%
- **months_loan_duration** = {4, … , 16, 18, 20, 21, 22, 24, 26, 27, 33, 39, 40, 42, 47, 54} → `default=1`. Validez: 75%
- **checking_balance** = {"< 0 DM", "1 - 200 DM"}, **months_loan_duration** = {28, 30, 36, 45, 48, 60}, **credit_history** = "reapaid" → `default=2`. Validez: 71'1%
- **checking_balance** = {"< 0 DM", "> 200 DM", "1 - 200 DM"}, **credit_history** = {"fully repaid", "fully repaid this bank"} → `default=2`. Validez: 64'8%

(en el siguiente apartado se interpretan los resultados, esta un poco más abajo)

Ahora comprobamos el árbol que se ha creado:

```
modelo <- C50::C5.0(trainx, trainy)
plot(modelo,gp = gpar(fontsize = 8.5))
```



Al principio, la variable `age` estaba incluida en el nuevo juego de datos, pero a la hora de ejecutar el modelo, no arrojaba ninguna regla y de hecho obteníamos el mismo resultado que en el primer intento, esto puede deberse al tipo de variable, aunque dudo un poco de que sea por esto ya que al principio de esta PEC todas las variables se convirtieron en modo `factor`.

Con el árbol obtenido, realiza una breve explicación de las reglas obtenidas así como de todos los puntos que te parezcan interesantes. Un elemento a considerar es, por ejemplo, cuantas observaciones caen dentro de cada regla

Observando el árbol de tres hojas, que se ha obtenido arriba, podemos ver las tres variables que se han utilizado para contstruirlo: `checking_balance`, `credit_history` y `months_loan_duration` (la variable `purpose` no se ha usado)

En el resultado gráfico y en las reglas obtenidas con el flag `rules=TRUE` podemos ver que si el `checking_balance = unknown` entonces hay un 15% de probabilidad de que el cliente no caiga en *default* y por lo tanto un 85% de que caiga en *default*. Luego, si los clientes tienen que $checking\_balance \in (\backslash < \ 0\,DM'', \backslash > \ 200\,DM'', \backslash 1 - 200\,DM'')$ entonces se decide entre tres opciones de `credit_history`, en el caso de los clientes con $credit\_history \in (\backslash critical'', \backslash delayed'')$ entonces hay una probabilidad del 81% de que el cliente caiga en `default`, mientras que los clientes con $credit\_history \in (\backslash fully\ repaid'', \backslash fully\ repaid\ this\ bank'')$ tienen una mayor probabilidad de no caer en *default* (64'8%) que de acabar cayendo, y por último, aquellos clientes con $credit\_history = repaid$ generarán otro nodo intermedio. En este último nodo intermedio, hay dos nodos terminales, aquellos clientes cuyos meses de pago están en este intervalo: (4, … , 16, 18, 20, 21, 22, 24, 26, 27, 33, 39, 40, 42, 47,

54) clasificados con un 75% de `default=1`, y por último, están los clientes con los siguientes meses: (28, 30, 36, 45, 48, 60) con un 71% de `default=2`.

Es pertinente destacar que si bien en las tres primeras reglas, el árbol solo realiza una observación por regla, en la cuarta regla se llevan a cabo 3 observacioens (`checking_balance`, `months_loan_duration`, `credit_history`), pero en la última regla (la quinta) se llevan a cabo dos observaciones: `checking_balance` y `credit_history`. Vemos como los clientes con `credit_history=repaid` pasan por dos observaciones más: `checking_balance` y `months_loan_duration`. De manera similar a lo anterior, pasa con los clientes con `credit_history={"fully repaid", "fully repaid this bank"}` ya que pasan por una observación de `checking_balance`. Esto es curioso pero lógico, ya que las reglas que mas observaciones tienen son la penúltima y la última, que justamente se correpsonden con los clientes que tienen menos riesgo de caer en estado de *default* (i.e., %(default=2)>%(default=1)), esto tiene sentido, porque como ya se vió al inicio de esta PEC, los clientes con `default=1` predominan con un 70%, frente al 30% de los clientes que tienen `default=2`, por lo tanto, para encontrar clientes con `default=2` se tienen que cumplir con más características específicas.

Una vez tengas un modelo válido, procede a realizar un análisis de la bondad de ajuste sobre el conjunto de test y matriz de confusión. ¿Te parece un modelo suficientemente bueno como para utilizarlo? Justifica tu respuesta considerando todos los posibles tipos de error

Ya hemos analizado los resultados del entrenamiento, ahora vamos a acometer la validación del modelo, para ello se ha implementado la siguiente linea de código:

```
#predicted_probs <- predict(model, titanic_test, type = "response")
predicted_modelo <- predict(modelo, testx, type="class" )
print(predicted_modelo)
```

```
##   [1] 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1
##  [75] 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1
## [112] 1 2 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 2 1 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
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
## [260] 1 1 1 1 1 1 2 1 1 1 1 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
## [297] 1 1 1 1 1 2 1 1 1 2 2 1 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 2 1 1
## [334] 1
## Levels: 1 2
```

```
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_modelo == testy) / length(predicte
```

```
## [1] "La precisión del árbol es: 72.7545 %"
```

Como se puede observar, la precisión del árbol es del 72'7545 %, una precisión aceptable, pero no demasiado antes, habría que buscar opciones que mejoren este resultado. No obstante, antes de investigar a cerca de variaciones (que se realizará más adelante) y aprovechando que estamos resolviendo un problema de clasificación binaria, es decir, un problema con dos clases: *('cliente en situación de default (default = 1)', 'clientes que NO están en situación de default (default = 2)')*, podemos analiar la calidad de predicción con una matriz de confusión, que nos va a mostrar los diferentes tipos de errores que se han cometido en la tarea de clasificación, llevada a cabo por el árbol de decisión. Véase a continuación el siguiente *chunk* de código:

```
mat_conf<-table(testy,Predicted = predicted_modelo)
mat_conf
```

```
##      Predicted
## testy   1   2
```

```
##      1 216  18
##      2  73  27
```

Para corroborar y confirmar que efectivamente, la precisión del árbol que hemos diseñado es del 72'7545 %, vamos a calcular la cifra a paritr de la matriz de confusión que hemos calculado arriba. Véase a continuación el siguiente *chunk* que implemetna este cálculo.

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

```
## [1] "El % de registros correctamente clasificados es: 72.7545 %"
```

Como podemos ver por el resultado de arriba efectivamente la precisión del modelo es del 72'7545 %.

Ahora que tenemos el resultado confirmado, vamos a interpretar los resultados de la matriz de confusión. Tal y como hemos visto en teoría (`PID_00284573`), una matriz de confusión propia de un problema de clasificación binario, contiene 4 valores asociados a los siguientes 4 conceptos:

- `TN`: en inglés, *True Negative*. Es una muestra negativa que el sistema ha predicho como negativa.
- `FP`: en inglés, *False Positive*. Es una muestra negativa que el sistema ha predicho como positiva.
- `FN`: en inglés, *False Negative*. Es una muestra positiva que el sistema ha predicho como negativa.
- `TP`: en inglés, *True Positive*. Es una muestra positiva que el sistema ha predicho como positiva.

Teniendo estos conceptos claros, y fijándonos en la matriz de confusión que hemos obtenido, se tiene la siguiente forma

```
cat("--------|--------\n\n")
```

```
## --------|--------
```

```
cat("   TP   |   FN   \n\n")
```

```
##    TP   |   FN
```

```
cat("--------|--------\n\n")
```

```
## --------|--------
```

```
cat("   FP   |   TN   \n\n")
```

```
##    FP   |   TN
```

```
cat("--------|--------")
```

```
## --------|--------
```

Por lo tanto, teniendo esto en cuenta, estudiamos los valores:

- `TP`= 216
- `FN`= 18
- `FP`= 73
- `TN`= 27

En el caso del los `TP` podemos ver como 216 casos positivos que efectivamente son positivos, se han clasificado correctamente. Por lo tanto, de 234 clientes, se han clasificado correctamente 216, i.e., se han clasificado correctamente, el 92,3077 % de los clientes en estado de *default*. Como se han clasificado correctamente 216/234, hay 18 clientes que han sido clasificados erróneamente, i.e., clientes que en la realidad cumplen con `default = 1` pero que nuestro árbol de clasificación ha etiquetado como `default = 2`.

Luego, de los 100 clientes que hay en el subconjunto de datos para el *test* del modelo, solo 27 clientes han sido clasificados correctamente (`TN`), es decir, clientes que no están en estado de *default*, i.e., (clientes con `default = 2`) y que han sido etiquetados como tal. Como hay 100 clientes en el juego de datos del `test`, los `TNs` solo consituyen el 27 % del total, significando esto que para el caso de los clientes con `default = 2` solo

se acierta un 27 %, y se clasifica erróneamente un 73 % de las veces (FP) Estas cifras no son muy buenas. Ya que parece ser que el árbol de decicisón es capaz de clasificar correctamente aquellas muestras/clientes que están en estado de `default`, pero no a los clientes que no lo están. De alguna manera, este era un resultado que me podía esperar, ya que como se vió al principio de esta práctica, la mayoría de los clientes de este *dataset* (el oirignal) son clientes con `default = 1`, puesto que son 700 clientes (70%) frente al 30% restante con `default = 2`. Por teoría, los problemas binarios que tienen un número de muestras muy superior de una clase, en comparación al de la otra clase, se les conoce como "Problemas binarios no equilibrados" y el tratamiento que se les ha de dar para poder evaluarlos correctamente, es exactamente el que se acaba de estudiar.

Finalizando este anásis, y teniendo en cuenta el anterior párrafo, podríamos decir que el modelo que se ha construido no muy bueno, porque aunque ronda el 73 % de precisión, tiene un error muy grave, que es el de la poca precisión para clasificar los clientes de la clase minoritaria. Por lo que toda la precisión que tiene el modelo, es gracias a la clasificación de los clientes pertenecientes a la clase mayoritaria. Además, como hemos visto en teoría, los modelos bien construidos, son aquellos que tienen valores grandes en la digaonal principal, y valores cercanos al 0 en el resto de posiciones de la matriz de confusión. En nuestro caso, el primer elemento de la diagonal principal es grande, pero el segundo ya no lo es, porque el elemento a la izquierda de este último es grande.

No obstante, para reafirmar la tesitura/problemática expuesta arriba, antes de probar con otro árbol de decisión se va a representar la curva ROC, a partir de los resultados expuestos arriba. Antes de construir esta gráfica, hay recordar, cual es la información que la curva ROC arroja.

Como sabemos por teoría, las curvas ROC resultan ser una herramienta muy efectiva y rápida a la hora de validar un modelo de clasificación supervisado y binario. Estas curvas representan la tasa de verdaderos positivos, i.e., la sensibilidad, en función de la tasa de falsos positivos (FP) (1 - Especificidad) para varios umbrales de clasificación.

- **Sensibilidad**: esto es lo que se mide en el eje y de la gráfica, y se calcula de la siguiente manera: $S = \frac{TP}{TP+FN}$, i.e., los verdaderos positivos entre el total de muestras.
- **Especificidad**: esta métrica se ve reflejada en el eje x, y cuantifica la proporción de muestras negativas, i.e., $FP + TN$ que son clasificadas como negativas, i.e., TN. Esta métrica puede calcularse de la siguiente manera: $E = \frac{TN}{FP+TN}$

Los indicios que nos permiten saber si estamos de un buen o mal modelo, es la cantidad de área debajo de la curva. Cuanto más se acerque la curva ROC a una linea diagonal, peor modelo será, mientras que cuanto más a la izquierda esté la curva, el modelo será de mejor calidad.

Teniendo claro esto, se procede a construir la curva ROC del árbol clasificatorio, véase el siguiente *chunk* de código:

```
# primero instalamos el paquete necesario
# install.packages("pROC")

# cargamos la librería
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following object is masked from 'package:colorspace':
##
##     coords

## The following object is masked from 'package:Metrics':
##
##     auc
```

```
## The following object is masked from 'package:gmodels':
##
##     ci
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

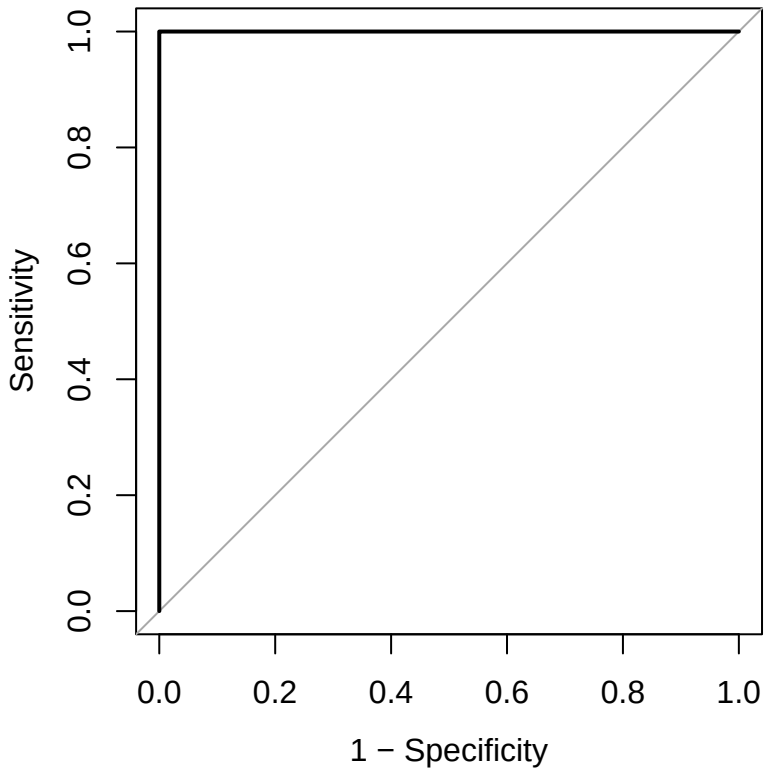
```
# Creamos un objeto ROC a partir de los resultados de la matriz de confusión
cat(' Esto es mat_conf[, 1]', mat_conf[, 1])
```

```
##  Esto es mat_conf[, 1] 216 73
```

```
par(pty = 's') # par aquitar el rellenos de los laterales
roc(mat_conf[, 1], mat_conf[, 2], plot=TRUE, legacy.axes=TRUE)
```

```
## Setting levels: control = 73, case = 216
```

```
## Setting direction: controls > cases
```




```
##
## Call:
## roc.default(response = mat_conf[, 1], predictor = mat_conf[,     2], plot = TRUE, legacy.axes = TRUE
##
## Data: mat_conf[, 2] in 1 controls (mat_conf[, 1] 73) > 1 cases (mat_conf[, 1] 216).
## Area under the curve: 1
```

El comando `legacy.axes=TRUE` lo he utilizado para poder invertir el eje x, para que en vez de que represente la especificidad, grafique: $1-Especificidad$. Sorprendentemente, observamos un resultaddo bastante bueno, que contrastas bastante con el error que hemos obtenido. Por lo tanto se opta a continuación por implementar otro código que construya la curva ROC.

```r
# primero instalamos el paquete necesario
# install.packages("pROC")

# cargamos la librería
library(pROC)

# Calcular la curva ROC
predicted_modelo <- predict(modelo, testx, type="class" )

str(predicted_modelo)
```

```
##  Factor w/ 2 levels "1","2": 1 1 1 2 1 1 1 2 1 1 ...
```

```r
num_predicted_modelo <- as.numeric(levels(predicted_modelo))[predicted_modelo]
print(length(num_predicted_modelo))
```

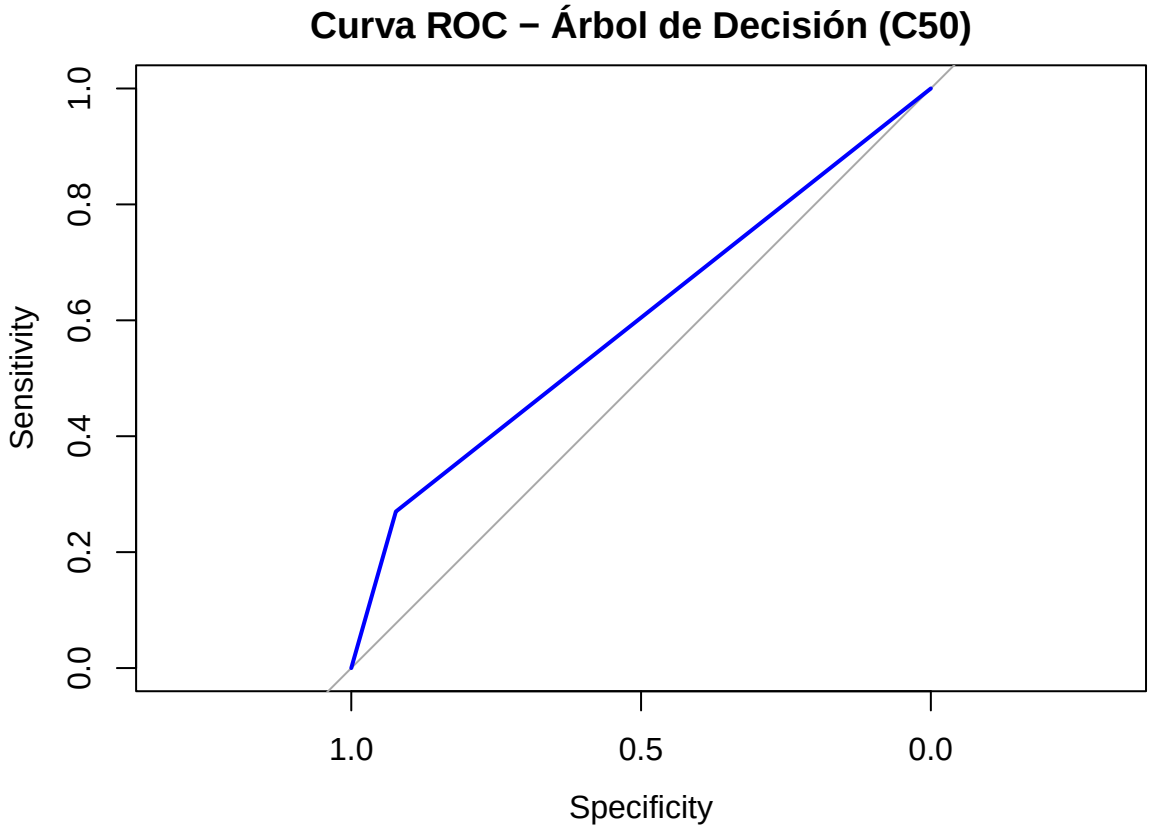
```
## [1] 334
```

```r
curva_roc <- roc(testy, num_predicted_modelo) #size(num_predict_modelo) = 668
```

```
## Setting levels: control = 1, case = 2
```

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

```r
# Dibujar la curva ROC
plot(curva_roc, main = "Curva ROC - Árbol de Decisión (C50)", col = "blue", lwd = 2)
```




```r
# Calcular el área bajo la curva (AUC)
auc_value <- auc(curva_roc)
cat("Área bajo la curva (AUC):", auc_value, "\n")
```

```
## Área bajo la curva (AUC): 0.5965385
```

Como se puede ver, esto tiene un poco más de sentido, teniendo en cuenta el problema mencionado antes, y por ello, no parece que la curva ROC sea muy buena, pues como se mencionó en párrafos anteriores, cuanto más se parezca el la curva a una linea diagonal, peor. No obstante, el area bajo la curva es de 0.6, un resultado aceptable, pero bastante mejorable.

Ahora, antes de probar otro modelo, vamos a obtener más información acerca del modelo. Véase a continuación el siguiente *chunk* de código:

```
if(!require(gmodels)){
  install.packages('gmodels', repos='http://cran.us.r-project.org')
  library(gmodels)
}

CrossTable(testy, predicted_modelo,prop.chisq  = FALSE, prop.c = FALSE, prop.r =FALSE,dnn = c('Reality'
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  334
##
##
##              | Prediction
##      Reality |         1 |         2 | Row Total |
## -------------|-----------|-----------|-----------|
##            1 |       216 |        18 |       234 |
##              |     0.647 |     0.054 |           |
## -------------|-----------|-----------|-----------|
##            2 |        73 |        27 |       100 |
##              |     0.219 |     0.081 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       289 |        45 |       334 |
## -------------|-----------|-----------|-----------|
##
##
```

Como podemos observar, no se obtiene información adicional que no se haya discutido antes, pues los decimales que aparecen debajo se corresponden con la porción/porcentaje de registros sobre el total, que cada conjunto de datos aglutina (trainx, trainy, testx, testy), y todos ellos suman 1.001 (0.647 + 0.054 + 0.219 + 0.081)

```
print(0.647 + 0.054 + 0.219 + 0.081)
```

```
## [1] 1.001
```

Otro aspecto a tener en cuenta, es el uso de las variables en el árbol de decisión, i.e., el peso de las variables que se han usado para construir el árbol. Esto es un aspecto muy importante, ya que junto con las variables que se han identidicado en el análisis del *dataset* realizado al principio de esta práctica, este análisis nos puede dar pistas sobre que variables cobran más o menos protagonismo. Sabiendo esto, podremos identificar las combinaciones de variables que faciliten y mejoren la tarea de clasificación del árbol de decisión. Véase

esta información, en el resultado arrojado por el siguiente *chunk* de código. Pero antes hay que aclarar los dos comandos que se han usado principalmente, y que son los siguientes:

- **C50::C5imp(modelo2, metric = "usage")**: esta linea de código, nos va a devolver un vector con las cantidades correspondientes a las veces que cada variable se ha utilizado para construir el árbol de decisión, esto nos va a permitir ver, las variables que más protagonismo tienen.
- **C50::C5imp(model, metric = "splits")**: esta linea de código, permite al programador, saber la importancia de cada una de las variables en un árbol de decisión. En este caso `"splits"` se refiere al número de divisiones realizadas en el árbol de decisión. Por lo tanto esta función nos permitirá obtener a modo de vector, la cantidad de veces que una variable se ha utilizado para realizar divisiones en el árbol de decisión.

Esta información analítica la podemos gráficar, véase el siguiente *chunk* de código:

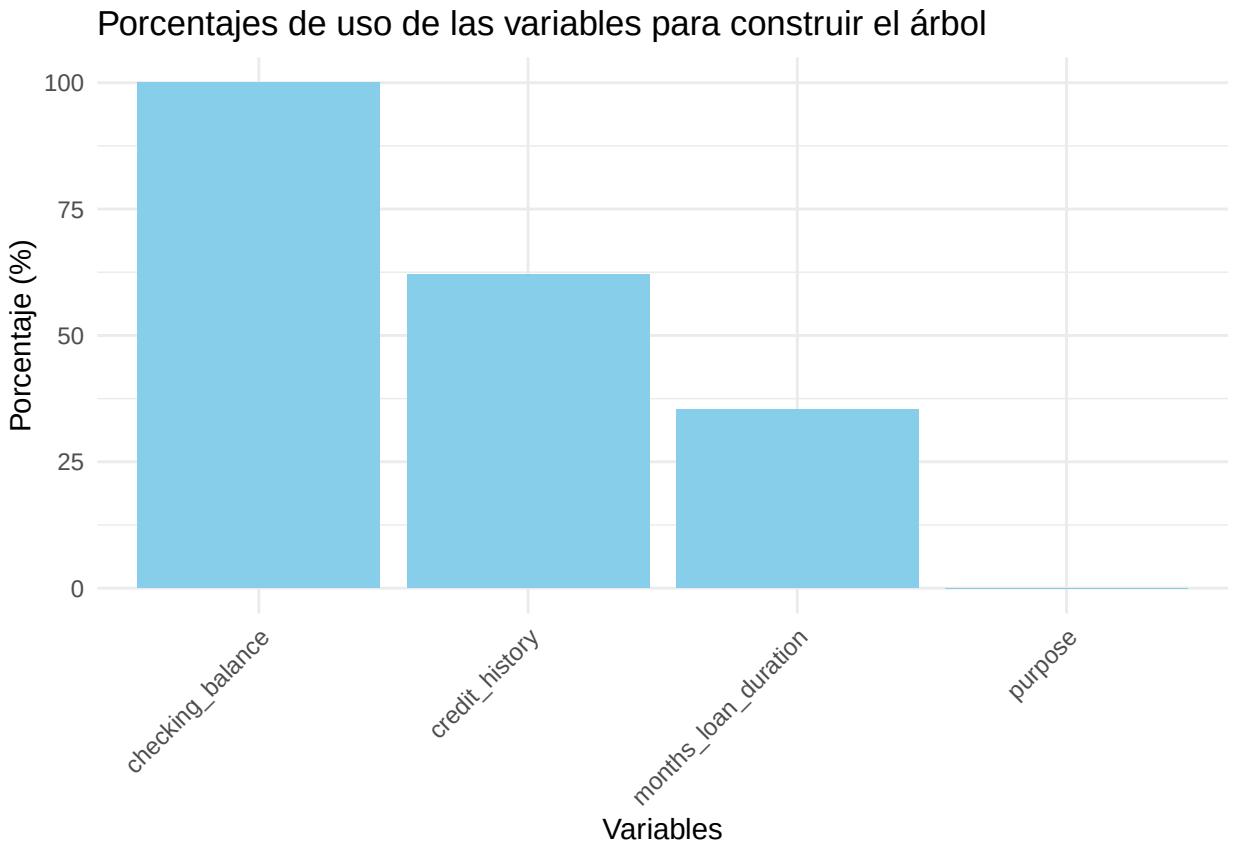
```
library(C50)
library(ggplot2)
model <- C5.0(trainx, trainy)

# Extrae las reglas del árbol
importanciaVariables = C5imp(model, metric = "usage")
importancia_splits <- C50::C5imp(model, metric = "splits")
print(importanciaVariables)
```

```
##                      Overall
## checking_balance      100.00
## credit_history         62.01
## months_loan_duration   35.29
## purpose                 0.00
```

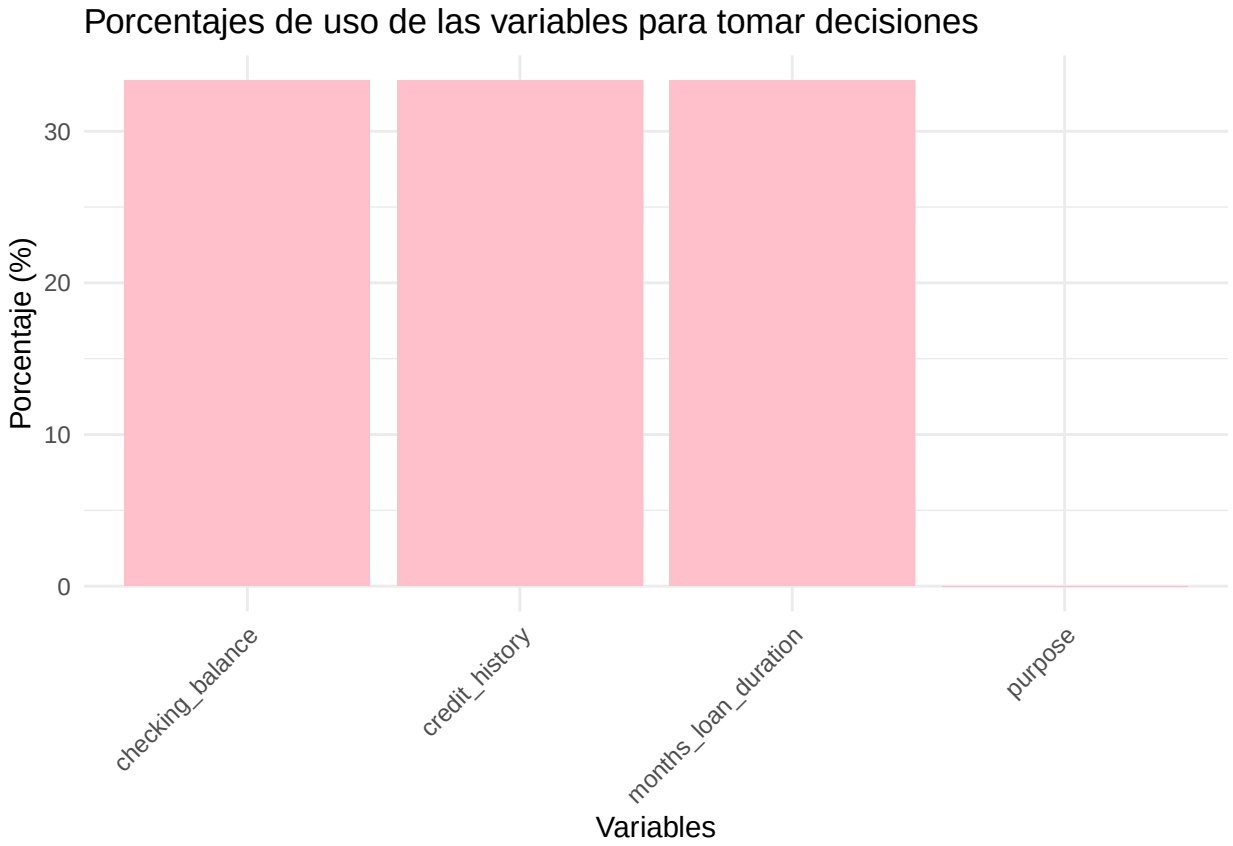
```
print(importancia_splits)
```

```
##                       Overall
## checking_balance     33.33333
## credit_history       33.33333
## months_loan_duration 33.33333
## purpose               0.00000
```

```
# Muestra la importancia de las variables
# Convertimos las importancias a formato dataframe
df_variables <- data.frame(variable = names(importanciaVariables), importancia = importanciaVariables, 
df_splits <- data.frame(variable = names(importancia_splits), importancia = importancia_splits, tipo = 

df_variables$indices = c('checking_balance', 'credit_history', 'months_loan_duration', 'purpose')

# definimos la información que va a tener cada eje
indice = df_variables$indices # los indices para las dos gráficas (eje x)
Overall = df_variables$Overall # (% uso/variable de construcción árbol)
divisiones = df_splits$Overall # (% uso/variable de divisiones árbol)

# Uso de cada atributo para construir el árbol (gráfica)
ggplot(df_variables, aes(x = indice, y = Overall)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Porcentajes de uso de las variables para construir el árbol",
       x = "Variables",
       y = "Porcentaje (%)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# Uso de cada atributo para la toma de decisiones en el árbol (gráfica)
ggplot(df_splits, aes(x = indice, y = divisiones)) +
  geom_bar(stat = "identity", fill = "pink") +
  labs(title = "Porcentajes de uso de las variables para tomar decisiones",
       x = "Variables",
       y = "Porcentaje (%)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



En los resultados de arriba, podemos ver como la variable con más peso e influencia en la construcción del modelo es el la de `checking_balance` con un 100 % de importancia, seguida por `credit_history` con una importancia del 62.01 %, a su vez seguida por el 35.29 % de uso de la variable `months_loan_duration`. Sorprendentemente, la variable `purpose` no se ha usado en ningun momento, lo que incita a eliminarla en el próximo árbol de decisión. Comparando estos pesos con los del modelo del ejemplo de esta PEC, observamos como nuestras variables no tienen pesos igual de altos, algo que nos indica que deberíamos probar con otro árbol de distinta combinación de variables, no osbtante, en el ejemplo de la PEC solo hay 3 varibales, mientras que nosotros tenemos 21, por ello, aun no habremos dado con la combinación de variables ideal para construir el árbol.

En la última gráfica de arriba (rosa) vemos el porcentaje de uso de cada variable, utilizado para tomar las decisiones en el árbol, es decir, el porcentaje de uso de cada una de las variables para dividir el árbol en distintos nodos. Claramente, vemos como la variable `purpose` se ha usado un 0 %, esto es lógico, puesto que si esta no se ha usado para construir el árbol, dificilmente va a poder ser usada para determinar las decisiones que el árbol ha de reflejar. Las 3 variables restantes, comparten porcentaje de uso; 33.33333 %, un resultado curioso, aunque esperable. Esto se debe a que, a pesar de que estas tres variables no hayan tenido el mismo peso para construir el árbol, pueden tener porcentajes más o menos altos a la hora de dividirlo, sobretodo en un problema de clasificación binario y desequilibrado, pues hay muchos más clientes de una clase que de otra, por lo tanto, para que los clientes de la clase minoritaria sean clasificados correctamente, estos tienen que cumplir con requisitos más específicos, que suponen el uso y la contemplación de muchas variables a la vez para que se pueda dar dicha clasificación, por lo tanto es por esto que la variable `months_loan_duration` tiene la misma importancia en cuanto a toma de decisiones (en la estructura del árbol) que la variable `checking_balance` o que la variable `credit_history`.

Ya hemos analizado el segundo modelo de árbol implementado (el primero no era válido), ahora, los pasos a seguir van a ser los siguientes:

- **Construir otro modelo pero con otra combinación de variables**
- **Implementar el algoritmo Random Forest**

- **En el siguiente ejercicio se implementarán variaciones de los árboles válidos, que se consigan implementar al final de este ejercicio con la librería C50**

Ahora se va a construir otro árbol de decisión con la librería `C50`, pero lo único que se va a variar, va a ser el `split` realizado en el `test` del modelo. En este caso se ha especificado un `split=2`.*

```
head(df_original)
```

```
##   checking_balance months_loan_duration credit_history   purpose amount
## 1           < 0 DM                    6       critical  radio/tv   1169
## 2       1 - 200 DM                   48         repaid  radio/tv   5951
## 3          unknown                   12       critical education   2096
## 4           < 0 DM                   42         repaid furniture   7882
## 5           < 0 DM                   24        delayed car (new)   4870
## 6          unknown                   36         repaid education   9055
##   savings_balance employment_length installment_rate personal_status
## 1         unknown           > 7 yrs                4     single male
## 2        < 100 DM         1 - 4 yrs                2          female
## 3        < 100 DM         4 - 7 yrs                2     single male
## 4        < 100 DM         4 - 7 yrs                2     single male
## 5        < 100 DM         1 - 4 yrs                3     single male
## 6         unknown         1 - 4 yrs                2     single male
##   other_debtors residence_history                property age installment_plan
## 1          none                 4             real estate  67             none
## 2          none                 2             real estate  22             none
## 3          none                 3             real estate  49             none
## 4     guarantor                 4 building society savings  45             none
## 5          none                 4             unknown/none  53             none
## 6          none                 4             unknown/none  35             none
##    housing existing_credits default dependents telephone foreign_worker
## 1      own                2       1          1       yes            yes
## 2      own                1       2          1      none            yes
## 3      own                1       1          2      none            yes
## 4 for free                1       1          2      none            yes
## 5 for free                2       2          2      none            yes
## 6 for free                1       1          2       yes            yes
##                  job
## 1   skilled employee
## 2   skilled employee
## 3 unskilled resident
## 4   skilled employee
## 5   skilled employee
## 6 unskilled resident
```

```
set.seed(666)
# creamos un data frame nuevo que contenga solo las columnas que queremos:
selec_cols2 = c('checking_balance', 'months_loan_duration', 'credit_history',
                'purpose', 'default')

df_original_sub2 <- df_original[, selec_cols2]
head(df_original_sub2)
```

```
##   checking_balance months_loan_duration credit_history   purpose default
## 1           < 0 DM                    6       critical  radio/tv       1
## 2       1 - 200 DM                   48         repaid  radio/tv       2
## 3          unknown                   12       critical education       1
```

```
## 4           < 0 DM                   42         repaid furniture       1
## 5           < 0 DM                   24        delayed car (new)       2
## 6          unknown                   36         repaid education       1
```

```
# ahora separamos el resto de variables de la etiqueta (variable a clasificar)
y2 <- df_original_sub2[,5] # seleccionamos la columna de default
x2 <- df_original_sub2[,1:4] # seleccionamos el resto

# visualizamos
head(x2)
```

```
##   checking_balance months_loan_duration credit_history   purpose
## 1           < 0 DM                    6       critical  radio/tv
## 2       1 - 200 DM                   48         repaid  radio/tv
## 3          unknown                   12       critical education
## 4           < 0 DM                   42         repaid furniture
## 5           < 0 DM                   24        delayed car (new)
## 6          unknown                   36         repaid education
```

```
head(y2)
```

```
## [1] 1 2 1 1 2 1
## Levels: 1 2
```

```
split_prop2 <- 2
indexes2 = sample(1:nrow(df_original_sub2), size=floor(((split_prop2-1)/split_prop2)*nrow(df_original_su
trainx2<-x2[indexes2,]
trainy2<-y2[indexes2]
testx2<-x2[-indexes2,]
testy2<-y2[-indexes2]
```

Ahora comprobamos los conjuntos, tal y como hemos hecho ya varias veces, véase el siguiente *chunk* de código:

```
summary(trainx2)
```

```
##    checking_balance months_loan_duration               credit_history
##  < 0 DM    :136     24     : 90          critical              :137
##  > 200 DM  : 28     12     : 82          delayed               : 51
##  1 - 200 DM:138     18     : 65          fully repaid          : 22
##  unknown   :198     36     : 47          fully repaid this bank: 26
##                     6      : 35          repaid                :264
##                     48     : 32
##                     (Other):149
##        purpose
##  radio/tv  :136
##  car (new) :108
##  furniture : 98
##  car (used): 55
##  business  : 53
##  education : 25
##  (Other)   : 25
```

```
summary(trainy2)
```

```
##   1   2
## 349 151
```

```
summary(testx2)
```

```
##    checking_balance months_loan_duration                credit_history
##  < 0 DM    :138     12     : 97          critical              :156
##  > 200 DM  : 35     24     : 94          delayed               : 37
##  1 - 200 DM:131     18     : 48          fully repaid          : 18
##  unknown   :196     6      : 40          fully repaid this bank: 23
##                     15     : 40          repaid                :266
##                     36     : 36
##                     (Other):145
##        purpose
##  radio/tv  :144
##  car (new) :126
##  furniture : 83
##  car (used): 48
##  business  : 44
##  education : 25
##  (Other)   : 30
```

```
summary(testy2)
```

```
##   1   2
## 351 149
```

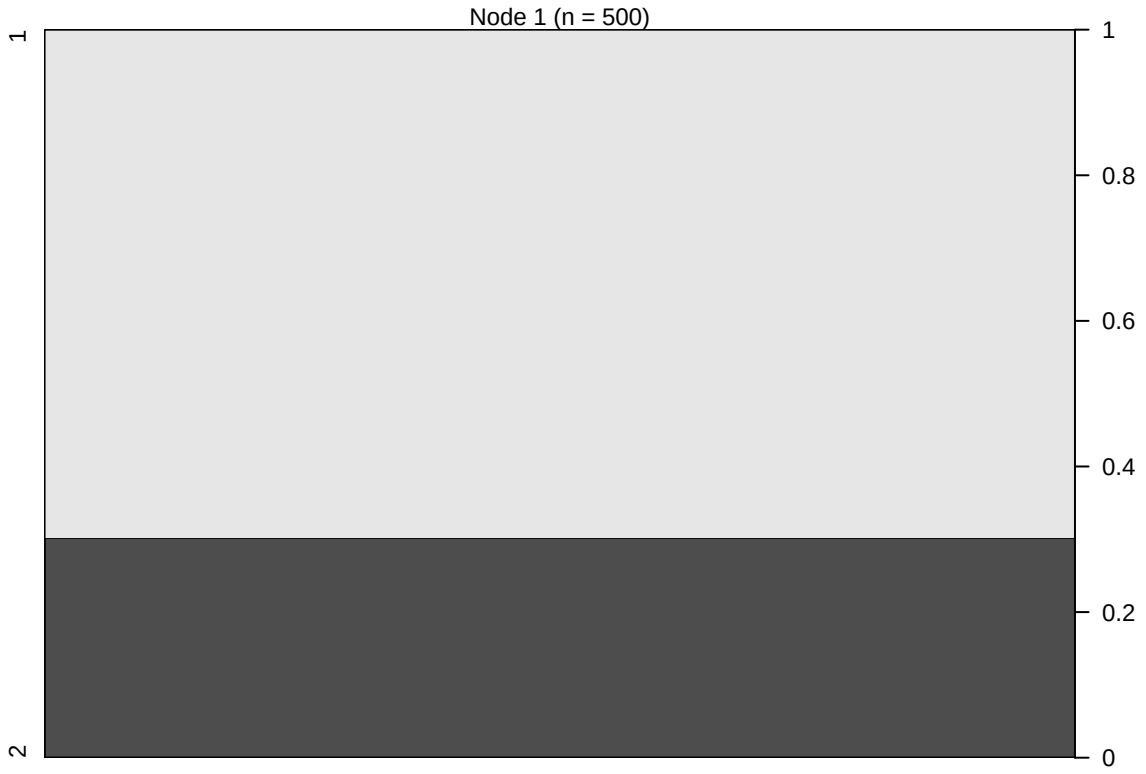
Ahora si, llevamos a cabo la creación del modelo. Mirése el siguiente *chunk*.

```
library(ggplot2)
trainy2 <- as.factor(trainy2)
modelo2 <- C50::C5.0(trainx2, trainy2,rules=TRUE )
summary(modelo2)
```

```
##
## Call:
## C5.0.default(x = trainx2, y = trainy2, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Feb  4 22:52:44 2024
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 500 cases (5 attributes) from undefined.data
##
## Rules:
##
## Default class: 1
##
##
## Evaluation on training data (500 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       0  151(30.2%)   <<
##
```

```
##
##     (a)   (b)    <-classified as
##    ----  ----
##     349          (a): class 1
##     151          (b): class 2
##
##
## Time: 0.0 secs
```

```
modelo2 <- C50::C5.0(trainx2, trainy2)
plot(modelo2,gp = gpar(fontsize = 8.5))
```



Como podemos observar no es árbol válido, no obstante calculamos su precisión.

```
#predicted_probs <- predict(model, titanic_test, type = "response")
predicted_modelo2 <- predict(modelo2, testx2, type="class" )
print(predicted_modelo2)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [445] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## Levels: 1 2
```

```
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_modelo2 == testy) / length(predicte
```

```
## Warning in `==.default`(predicted_modelo2, testy): longer object length is not
## a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
## [1] "La precisión del árbol es: 71.0000 %"
```

Como se puede observar, la precisión del árbol ha bajado hasta un 71.0000 % respecto al primer árbol. Pero lo importante no es esto, sino las proporciones de los conjuntos de `train` y de `test` ya que al haber especificado `split=2` estamos teniendo el mismo número de registros en el `train` como en el `test` por lo tanto el modelo se está entrenando con menos datos, lo que se traduce en una menor precisión y en un árbol de decisión no válido. Ahora vamos a probar con `split=4`

Nótese como estamos implementado de nuevo todo el código, para evitar errores a la hora de sobre-escribir variables, cuando se quiera generar el archivo html.

```
set.seed(666)
# creamos un data frame nuevo que contenga solo las columnas que queremos:
selec_cols3 = c('checking_balance', 'months_loan_duration', 'credit_history',
                'purpose', 'default')

df_original_sub3 <- df_original[, selec_cols3]
head(df_original_sub3)
```

```
##   checking_balance months_loan_duration credit_history   purpose default
## 1           < 0 DM                    6       critical  radio/tv       1
## 2       1 - 200 DM                   48         repaid  radio/tv       2
## 3          unknown                   12       critical education       1
## 4           < 0 DM                   42         repaid furniture       1
## 5           < 0 DM                   24        delayed car (new)       2
## 6          unknown                   36         repaid education       1
```

```
# ahora separamos el resto de variables de la etiqueta (variable a clasificar)
y3 <- df_original_sub3[,5] # seleccionamos la columna de default
x3<- df_original_sub3[,1:4] # seleccionamos el resto

# visualizamos
head(x3)
```

```
##   checking_balance months_loan_duration credit_history   purpose
## 1           < 0 DM                    6       critical  radio/tv
## 2       1 - 200 DM                   48         repaid  radio/tv
## 3          unknown                   12       critical education
## 4           < 0 DM                   42         repaid furniture
## 5           < 0 DM                   24        delayed car (new)
## 6          unknown                   36         repaid education
```

```
head(y3)
```

```
## [1] 1 2 1 1 2 1
## Levels: 1 2
```

Ahora, creamos de nuevo los conjuntos de datos:

```
split_prop3 <- 4
indexes3 = sample(1:nrow(df_original_sub3), size=floor(((split_prop3-1)/split_prop3)*nrow(df_original_su
trainx3<-x3[indexes3,]
trainy3<-y3[indexes3]
testx3<-x3[-indexes3,]
testy3<-y3[-indexes3]
```

Ahora comprobamos los conjuntos, tal y como hemos hecho ya varias veces, véase el siguiente *chunk* de código:

```
summary(trainx3)
```

```
##    checking_balance months_loan_duration               credit_history
##  < 0 DM    :204     12     :133          critical              :217
##  > 200 DM  : 46     24     :132          delayed               : 67
##  1 - 200 DM:207     18     : 96          fully repaid          : 27
##  unknown   :293     36     : 66          fully repaid this bank: 34
##                     6      : 48          repaid                :405
##                     15     : 44
##                     (Other):231
##        purpose
##  radio/tv  :215
##  car (new) :173
##  furniture :141
##  car (used): 74
##  business  : 70
##  education : 36
##  (Other)   : 41
```

```
summary(trainy3)
```

```
##   1   2
## 529 221
```

```
summary(testx3)
```

```
##    checking_balance months_loan_duration               credit_history
##  < 0 DM    : 70     24     :52           critical              : 76
##  > 200 DM  : 17     12     :46           delayed               : 21
##  1 - 200 DM: 62     6      :27           fully repaid          : 13
##  unknown   :101     15     :20           fully repaid this bank: 15
##                     18     :17           repaid                :125
##                     36     :17
##                     (Other):71
##        purpose
##  radio/tv  :65
##  car (new) :61
##  furniture :40
##  car (used):29
##  business  :27
##  education :14
##  (Other)   :14
```

```
summary(testy3)
```

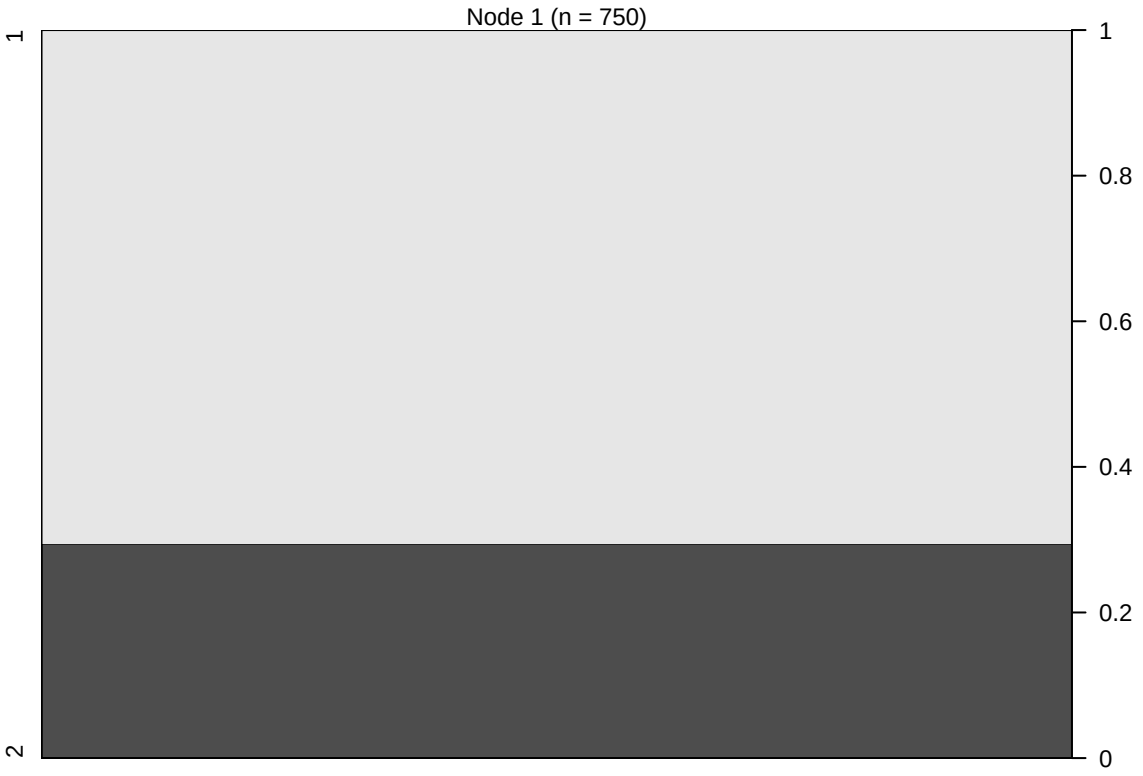
```
##   1   2
## 171  79
```

Ahora implementamos el modelo del nuevo árbl con `split=4`:

```
set.seed(666)
trainy3 <- as.factor(trainy3)
modelo3 <- C50::C5.0(trainx3, trainy3,rules=TRUE )
summary(modelo3)
```

```
##
## Call:
## C5.0.default(x = trainx3, y = trainy3, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Feb  4 22:52:44 2024
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 750 cases (5 attributes) from undefined.data
##
## Rules:
##
## Default class: 1
##
##
## Evaluation on training data (750 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       0  221(29.5%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     529          (a): class 1
##     221          (b): class 2
##
##
## Time: 0.0 secs
```

```
modelo3 <- C50::C5.0(trainx3, trainy3)
plot(modelo3,gp = gpar(fontsize = 8.5))
```



Como podemos observar, no obtenemos un árbol válido, ahora calculamos la precisión del nuevo modelo:

```
predicted_modelo3 <- predict(modelo3, testx3, type="class" )
print(predicted_modelo3)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## Levels: 1 2
```

```
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_modelo3 == testy3) / length(predic
```

```
## [1] "La precisión del árbol es: 68.4000 %"
```

Vemos como se obtiene una precisión del 68.4000 %, pero esto nos da igual, ya que el árbol que hemos obtenido no es válido. No obstante, si para este nuevo árbol, cambiamos la semilla aleatoria, a `set.seed(667)` , obtenemos un árbol de decisión válido, pero solo formado por dos variables: `checking_balance` y `credit history` con una precisión del 71,200 %, pero da igual porque la construcción del árbol estaba ignorando el uso del resto de variables, por lo tanto no tenía mucho sentido, ya que el porcentaje de uso de las variables restantes sería de 0 % así como su protagonismo a la hora de determinar las decisiones que el árbol ha de tomar. Además en el resumen de cada uno de los conjuntos creados, como hay muchos registros para el entrenamiento del modelo, y muy pocos para su validación por lo que esta sería una de las razones principales por las cuales se obtendría un modelo con una precisión más baja. Puesto que no ha podido ser validado correctamente.

Como se puede observar, los resultados son muy sensibles y volátiles, a la hora de cambiar la semilla aleatoria, por lo que, para mantener un orden, se va a continuar con la semilla `set.seed(666)` ya que es para esta semilla, para la cual se han llevado a cabo todos los análisis de los árboles implementados. En caso de no

encontrar nada alfinal del ejercicio, se recurrirá a la modificación de la semilla aleatoria, como última opción.

Hemos podido ver como el valor del split, que nos devuelve resultados razonables, es para `split_prop=3` ya que está asegura los porcentajes ideales, tanto del entrenamiento como del test, por ello a continuación se va a intentar implementar otro árbol, pero con un conjunto de variables diferente pero con el mismo `split`. Primero vamos a introducir menos variables, por lo tanto, para este árbol se ha tenido en cuenta el siguiente conjunto de valores (no se cogerán y se probarán varias combinaciones)

- **checking_balance**
- **credit_history**
- **personal_status**
- **other_debtors**
- **existing_credits**
- **default** (variable clasificadora)

Véase la implementación del modelo:

```
set.seed(666)
# creamos un data frame nuevo que contenga solo las columnas que queremos:
selec_cols4 = c('checking_balance', 'credit_history', 'personal_status',
                'default')

df_original_sub4 <- df_original[, selec_cols4]
head(df_original_sub4)
```

```
##   checking_balance credit_history personal_status default
## 1           < 0 DM       critical     single male       1
## 2       1 - 200 DM         repaid          female       2
## 3          unknown       critical     single male       1
## 4           < 0 DM         repaid     single male       1
## 5           < 0 DM        delayed     single male       2
## 6          unknown         repaid     single male       1
```

```
# ahora separamos el resto de variables de la etiqueta (variable a clasificar)
y4 <- df_original_sub4[,4] # seleccionamos la columna de default
x4<- df_original_sub4[,1:3] # seleccionamos el resto

# visualizamos
head(x4)
```

```
##   checking_balance credit_history personal_status
## 1           < 0 DM       critical     single male
## 2       1 - 200 DM         repaid          female
## 3          unknown       critical     single male
## 4           < 0 DM         repaid     single male
## 5           < 0 DM        delayed     single male
## 6          unknown         repaid     single male
```

```
head(y4)
```

```
## [1] 1 2 1 1 2 1
## Levels: 1 2
```

```
set.seed(666)
split_prop4 <- 3
indexes4 = sample(1:nrow(df_original_sub4), size=floor(((split_prop4-1)/split_prop4)*nrow(df_original_su
trainx4<-x4[indexes4,]
trainy4<-y4[indexes4]
```

```
testx4<-x4[-indexes4,]
testy4<-y4[-indexes4]

summary(trainx4)
```

```
##    checking_balance               credit_history      personal_status
##  < 0 DM    :182     critical              :190    divorced male: 34
##  > 200 DM  : 42     delayed               : 60    female       :207
##  1 - 200 DM:189     fully repaid          : 26    married male : 56
##  unknown   :253     fully repaid this bank: 33    single male  :369
##                     repaid                :357
```

```
summary(trainy4)
```

```
##   1   2
## 466 200
```

```
summary(testx4)
```

```
##    checking_balance               credit_history      personal_status
##  < 0 DM    : 92     critical              :103    divorced male: 16
##  > 200 DM  : 21     delayed               : 28    female       :103
##  1 - 200 DM: 80     fully repaid          : 14    married male : 36
##  unknown   :141     fully repaid this bank: 16    single male  :179
##                     repaid                :173
```

```
summary(testy4)
```

```
##   1   2
## 234 100
```

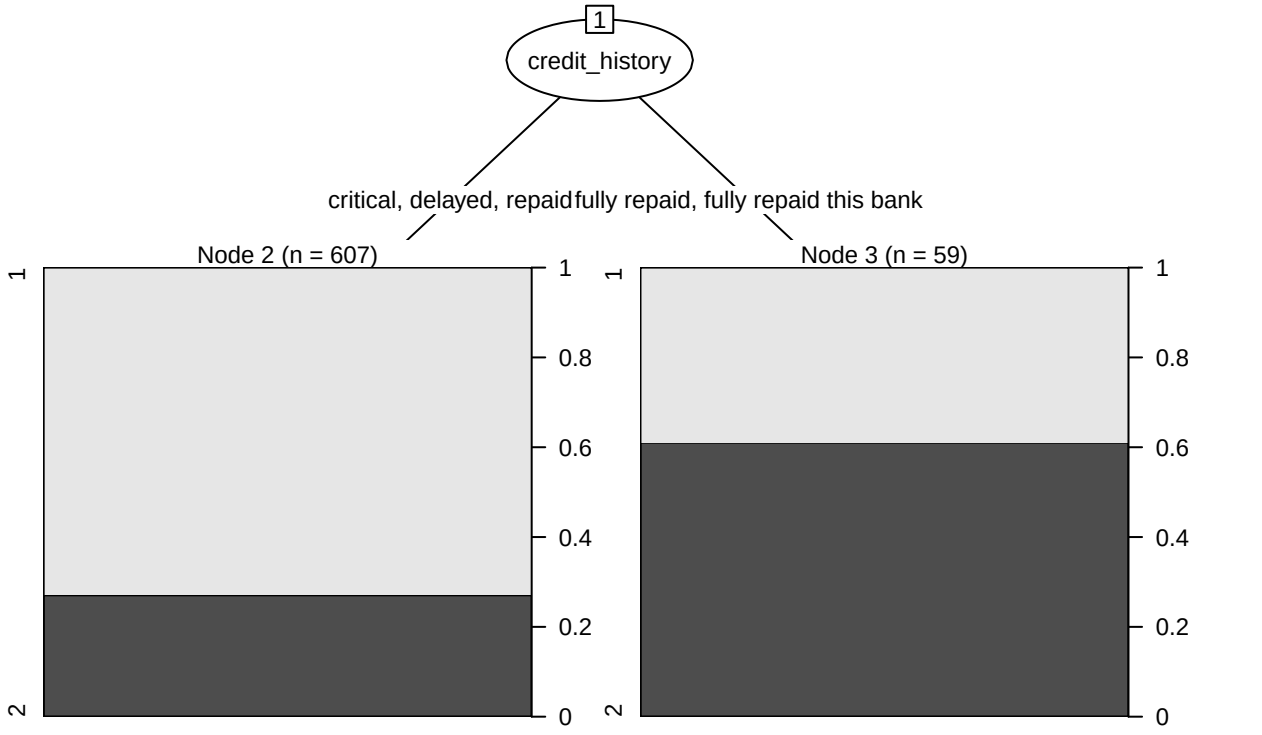
Los *datasets* están correctamente divididos. Ahora implementamos el modelo

```
set.seed(666)
trainy4 <- as.factor(trainy4)
modelo4 <- C50::C5.0(trainx4, trainy4, rules=TRUE )
summary(modelo4)
```

```
##
## Call:
## C5.0.default(x = trainx4, y = trainy4, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Feb  4 22:52:45 2024
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 666 cases (4 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (607/164, lift 1.0)
##  credit_history in {critical, delayed, repaid}
##  ->  class 1  [0.729]
##
## Rule 2: (59/23, lift 2.0)
```

```
##  credit_history in {fully repaid, fully repaid this bank}
##  ->  class 2  [0.607]
##
## Default class: 1
##
##
## Evaluation on training data (666 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       2  187(28.1%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     443    23    (a): class 1
##     164    36    (b): class 2
##
##
##  Attribute usage:
##
##  100.00% credit_history
##
##
## Time: 0.0 secs
```

```
modelo4 <- C50::C5.0(trainx4, trainy4)
plot(modelo4,gp = gpar(fontsize = 8.5))
```



Ahora tenemos solo un nodo, correspondiente a: `credit_history`, por lo tanto se intuye que el resto de variables, no tienen ningun peso ni en la construcción ni en la división del árbol de decisión, por lo tanto vamos a calcular la precisión del árbol y su matriz de confusión. Como la precisión ha bajado un punto porcentual, el error de clasificación ha subido un punto porcentual, llegando al 28'1 % de error.

```
set.seed(666)
predicted_modelo4<- predict(modelo4, testx4, type="class" )
print(predicted_modelo4)
```

```
##   [1] 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 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 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1
## [334] 1
## Levels: 1 2
```

```
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_modelo4 == testy4) / length(predic
```

```
## [1] "La precisión del árbol es: 71.2575 %"
```

La precisión del árbol ha bajado un punto porcentual, pero nos da igual porque el árbol está clasificando solo en base a una variable, y esto no es algo bueno, porque está ignorando el resto de datos, que son necesarios para poder clasificar a los clientes.

Por curiosidad, vamos a calcular la matriz de confusión y su curva ROC.

```
# cargamos la librería para construit la curva ROC
library(pROC)

# calculamos la matriz de confusión
mat_conf4<-table(testy4,Predicted = predicted_modelo4)
mat_conf4
```

```
##       Predicted
## testy4   1   2
##      1 221  13
##      2  83  17
```

```
# Calcular la curva ROC
predicted_modelo4 <- predict(modelo4, testx4, type="class" )

str(predicted_modelo4)
```

```
##  Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 2 1 1 ...
```

```
num_predicted_modelo4 <- as.numeric(levels(predicted_modelo4))[predicted_modelo4]
print(length(num_predicted_modelo4))
```

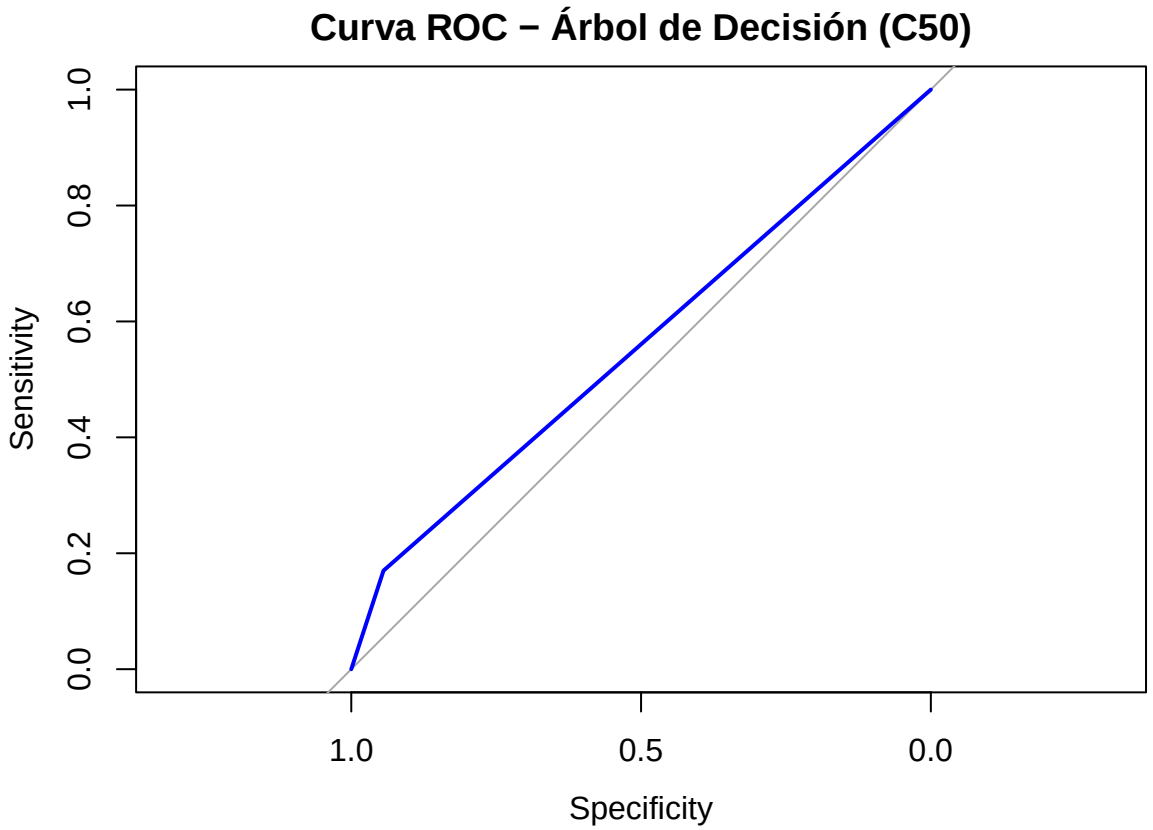
```
## [1] 334
```

```
curva_roc4 <- roc(testy4, num_predicted_modelo4) #size(num_predict_modelo) = 668
```

```
## Setting levels: control = 1, case = 2
```

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

```r
# Dibujar la curva ROC
plot(curva_roc4, main = "Curva ROC - Árbol de Decisión (C50)", col = "blue", lwd = 2)
```




```r
# Calcular el área bajo la curva (AUC)
auc_value4 <- auc(curva_roc4)
cat("Área bajo la curva (AUC):", auc_value4, "\n")
```

```
## Área bajo la curva (AUC): 0.5572222
```

Como se puede observar, se observan resultados muy parecidos al primer árbol implementado, pero con menos precisión y un 1% más de errores (véase la matriz de confusión)

Como hemos podido ver, no hemos conseguido mejorar los resultados, por lo tanto se va a probar con otras variables del conjunto mencionado antes (fruto del análisis inicial del juego de datos) Vamos a probar a cambiar la combinación de las variables y la semilla aleatoria, hasta mejorar la precisión del primer árbol válido que era de: 72'7545 %

```r
# importamos la librería necesaria
library(C50)
library(ggplot2)
library(gplots)
```

```
## Registered S3 method overwritten by 'gplots':
##   method         from
##   reorder.factor gdata
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:gdata':
```

```
##
##     reorder.factor

## The following object is masked from 'package:DescTools':
##
##     reorder.factor

## The following object is masked from 'package:stats':
##
##     lowess
```

```
# especificamos el número de la semilla aletaoria.
num_seed = 1878
set.seed(num_seed)

# creamos un data frame nuevo que contenga solo las columnas que queremos:
selec_cols5 = c('checking_balance', 'credit_history','default')
df_original_sub5 <- df_original[, selec_cols5]
head(df_original_sub5)
```

```
##   checking_balance credit_history default
## 1           < 0 DM       critical       1
## 2       1 - 200 DM         repaid       2
## 3          unknown       critical       1
## 4           < 0 DM         repaid       1
## 5           < 0 DM        delayed       2
## 6          unknown         repaid       1
```

```
# ahora separamos el resto de variables de la etiqueta (variable a clasificar)
y5 <- df_original_sub5[,length(selec_cols5)] # seleccionamos la columna de default
x5<- df_original_sub5[,1:length(selec_cols5)-1] # seleccionamos el resto

# visualizamos
head(x5)
```

```
##   checking_balance credit_history
## 1           < 0 DM       critical
## 2       1 - 200 DM         repaid
## 3          unknown       critical
## 4           < 0 DM         repaid
## 5           < 0 DM        delayed
## 6          unknown         repaid
```

```
head(y5)
```

```
## [1] 1 2 1 1 2 1
## Levels: 1 2
```

```
split_prop5 <- 3
indexes5 = sample(1:nrow(df_original_sub5), size=floor(((split_prop5-1)/split_prop5)*nrow(df_original_su
trainx5<-x5[indexes5,]
trainy5<-y5[indexes5]
testx5<-x5[-indexes5,]
testy5<-y5[-indexes5]

summary(trainx5)
```

```
##    checking_balance               credit_history
```

```
##  < 0 DM    :177     critical              :210
##  > 200 DM  : 37     delayed               : 55
##  1 - 200 DM:177     fully repaid          : 26
##  unknown   :275     fully repaid this bank: 31
##                     repaid                :344
```

```
summary(trainy5)
```

```
##   1   2
## 470 196
```

```
summary(testx5)
```

```
##    checking_balance               credit_history
##  < 0 DM    : 97     critical              : 83
##  > 200 DM  : 26     delayed               : 33
##  1 - 200 DM: 92     fully repaid          : 14
##  unknown   :119     fully repaid this bank: 18
##                     repaid                :186
```

```
summary(testy5)
```

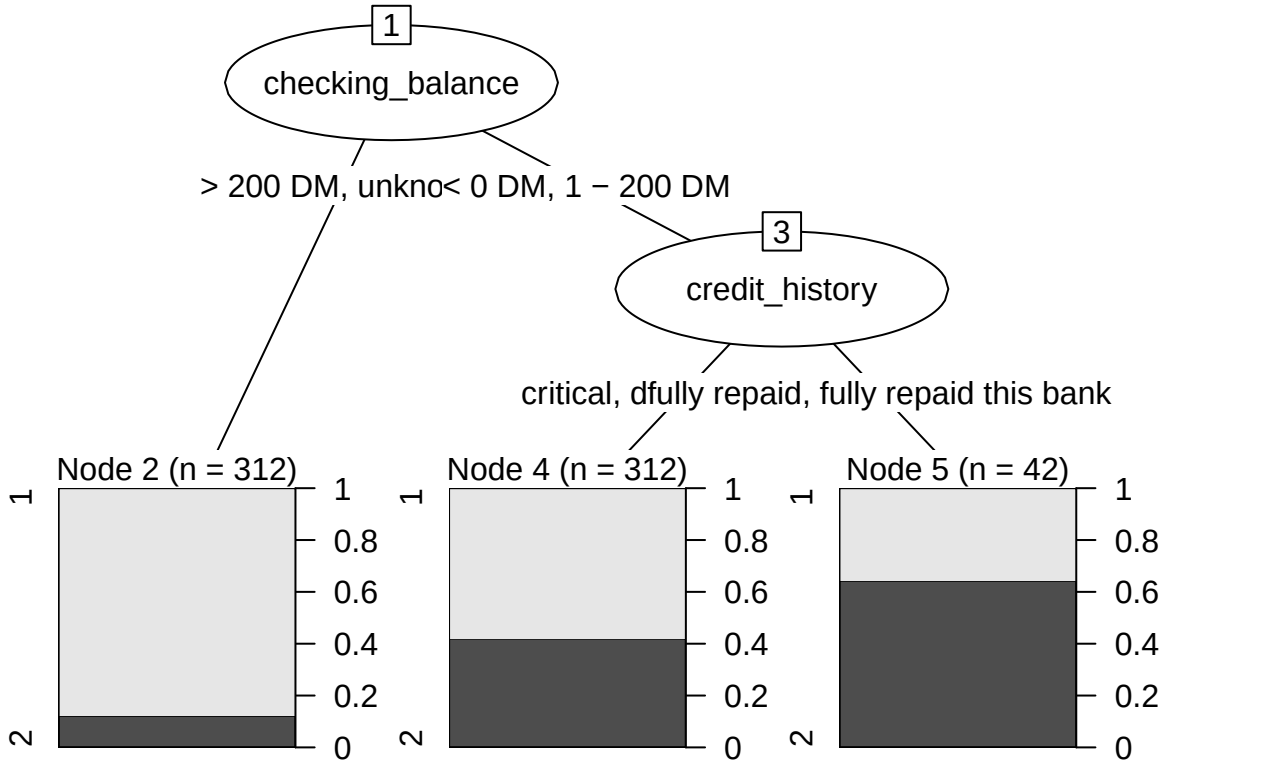
```
##   1   2
## 230 104
```

```
# desplegamos el modelo:
trainy5 <- as.factor(trainy5)
modelo5 <- C50::C5.0(trainx5, trainy5,rules=TRUE )
summary(modelo5)
```

```
##
## Call:
## C5.0.default(x = trainx5, y = trainy5, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Feb  4 22:52:45 2024
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 666 cases (3 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (312/38, lift 1.2)
##  checking_balance in {> 200 DM, unknown}
##  ->  class 1  [0.876]
##
## Rule 2: (609/167, lift 1.0)
##  credit_history in {critical, delayed, repaid}
##  ->  class 1  [0.725]
##
## Rule 3: (42/15, lift 2.2)
##  checking_balance in {< 0 DM, 1 - 200 DM}
##  credit_history in {fully repaid, fully repaid this bank}
##  ->  class 2  [0.636]
##
```

```
## Default class: 1
##
##
## Evaluation on training data (666 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       3  184(27.6%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     455    15    (a): class 1
##     169    27    (b): class 2
##
##
##  Attribute usage:
##
##   97.75% credit_history
##   53.15% checking_balance
##
##
## Time: 0.0 secs
```

```
modelo5 <- C50::C5.0(trainx5, trainy5)
plot(modelo5)
```



```
#determinamos la precisión
predicted_modelo5<- predict(modelo5, testx5, type="class" )
print(predicted_modelo5)
```

```
##   [1] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1
##  [75] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2
## [112] 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 2 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1
## [334] 1
## Levels: 1 2
```

```
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_modelo5 == testy5) / length(predic
```

```
## [1] "La precisión del árbol es: 73.0539 %"
```

```
# calculamos matriz de confusión y curva ROC
mat_conf5<-table(testy5,Predicted = predicted_modelo5)
mat_conf5
```

```
##       Predicted
## testy5   1   2
##      1 224   6
##      2  84  20
```

```
# Calcular la curva ROC
predicted_modelo5 <- predict(modelo5, testx5, type="class" )
str(predicted_modelo5)
```

```
##  Factor w/ 2 levels "1","2": 1 1 1 1 2 1 1 1 1 1 ...
```

```
num_predicted_modelo5 <- as.numeric(levels(predicted_modelo5))[predicted_modelo5]
print(length(num_predicted_modelo5))
```

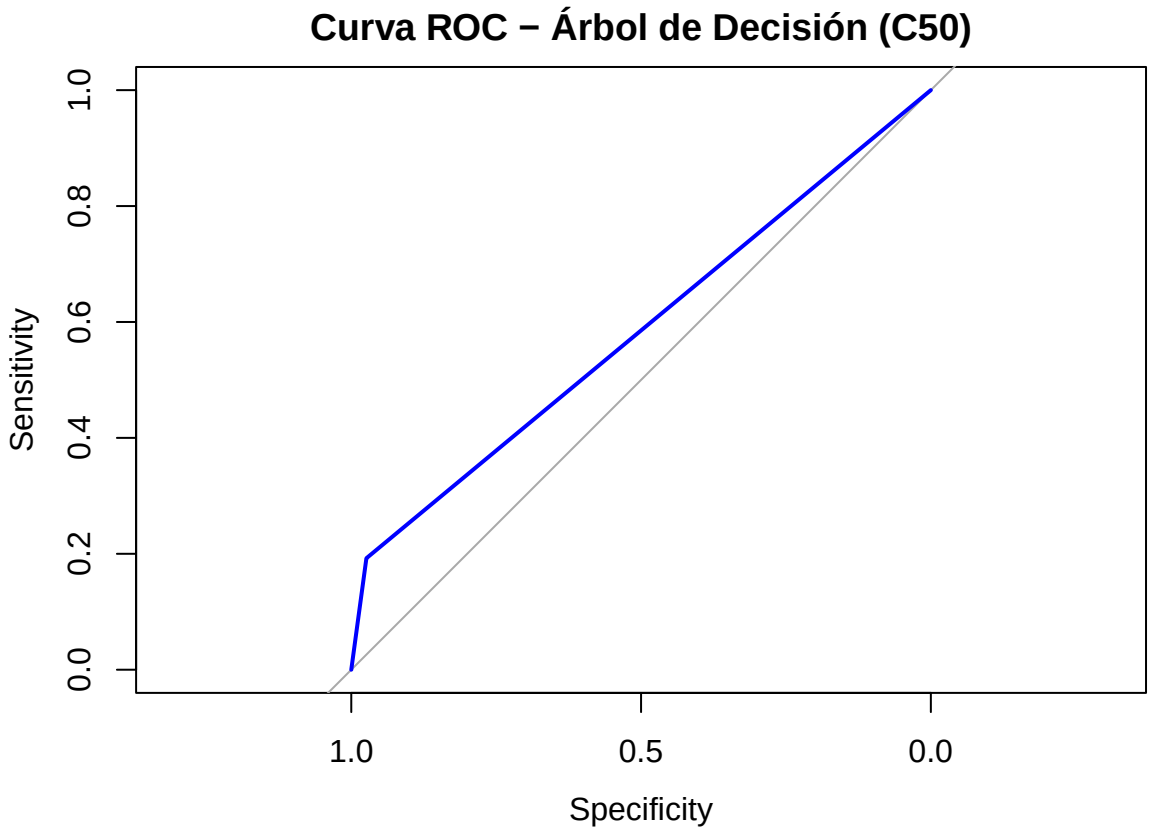
```
## [1] 334
```

```
curva_roc5 <- roc(testy5, num_predicted_modelo5) #size(num_predict_modelo) = 668
```

```
## Setting levels: control = 1, case = 2
```

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

```
# Dibujar la curva ROC
plot(curva_roc5, main = "Curva ROC - Árbol de Decisión (C50)", col = "blue", lwd = 2)
```




```
# Calcular el área bajo la curva (AUC)
auc_value5 <- auc(curva_roc5)
cat("Área bajo la curva (AUC):", auc_value5, "\n")
```

```
## Área bajo la curva (AUC): 0.5831104
```

Como podemos observar, hemos conseguido mejorar la marca del modelo implementado (73'0539 % > 72'7545 %), y además con una variable menos, esto es importante, pues como hemos visto en teoría, es mejor menos variables que representen más información correcta, y mejor, que muchas variables que presenten la misma calidad de información o el mismo resultado. En este caso hemos sido capaces de mejorar la precisión, disminuyendo en uno, el número de variables, y por lo tanto esto es un punto positivo.

En cuanto a las reglas creadas del árbol, hemos obtenido solo 3 reglas, (2 menos en comparación al primer árbol), esto tiene sentido porque tenemos 2 variables menos. No obstante, el número de reglas en un árbol de decisión no está estrictamente vinculado al número de variables. Podemos observar que en las dos primeras reglas, se realiza una sola observación individualmente, la primera realiza una observación en la variable `checking_balance`, y la segunda realiza una observación en la variable `credit_history`, en ambas reglas prima la clase mayoritaria. Por último, la última regla realiza una observación en las dos variables anteriores y en caso de que estas vdos ariables cumplan con los valores que muestra el modelo, el árbol clasificará a los clientes que cumplan con estos valores, asignándoles `defaul=2` con una probabilidad del 63'6 %

El resultado que se observa arriba, es gracias a una semilla aleatoria `set.seed=1878` y gracias a las variables `'checking_balance', 'credit_history' y 'default'` . Con estos parámetros, obtenemos una precisión del 73.0539 %", aunque viendo los resultados de la matriz de confusión, los `FP` parecen haber aumentado. Respecto al primer arbol, vemos como se clasifica aun mejor a los clientes con `default=1` mientras que los clientes con `default=2` se clasifican aún peor ya que: $TN_{1º árbol} > TN_{2º árbol} $, y este último tipo de clientes son los que queremos clasificar mejor, pero este modelo no lo consigue a pesar de alcanzar mejor precisión.

Cabe destacar que se han probado muchas combinaciones de variables, no solo aquellas que se han identificado

como interesantes en el análisis realizado al inicio de esta PEC. Además, se ha probado también para muchos valores de la semilla aleatoria y no ha habido suerte, ya que la precisión no aumentaba, y los árboles que obteníamos eran o muy simples o muy complejos, y habiéndolos estudiado, y viendo las precisiones obtenidas, se podía ver claramente como los árboles más complejos tenían precisiones más bajas. Esto puede deberse al problema de incluir variables que no aportan mucha información al modelo y que por lo tanto no están tan correladas entre sí. Por ello, el mejor modelo que se ha obtenido en este caso, solo incluye dos variables/atributos, que se de hecho se corresponden con las variables que guardan más relación con la variable clasificatoria; `default`, estudiadas en el análisis al principio de este proyecto.

Finalmente, analizando la curva ROC, vemos como tiene una forma parecida a la del primer árbol válido, pero en este caso parece ser un poco más ancha (respecto a la diagonal gris), pues se ha mejorado un poco la precisión.

Ahora vamos a analizar la importancia de cada una de las tres variables en el modelo, así como el peso de cada una de ellas en cuanto a construcción del árbol como la división del mismo en nodos decisivos. Véase el siguiente *chunk* de código.

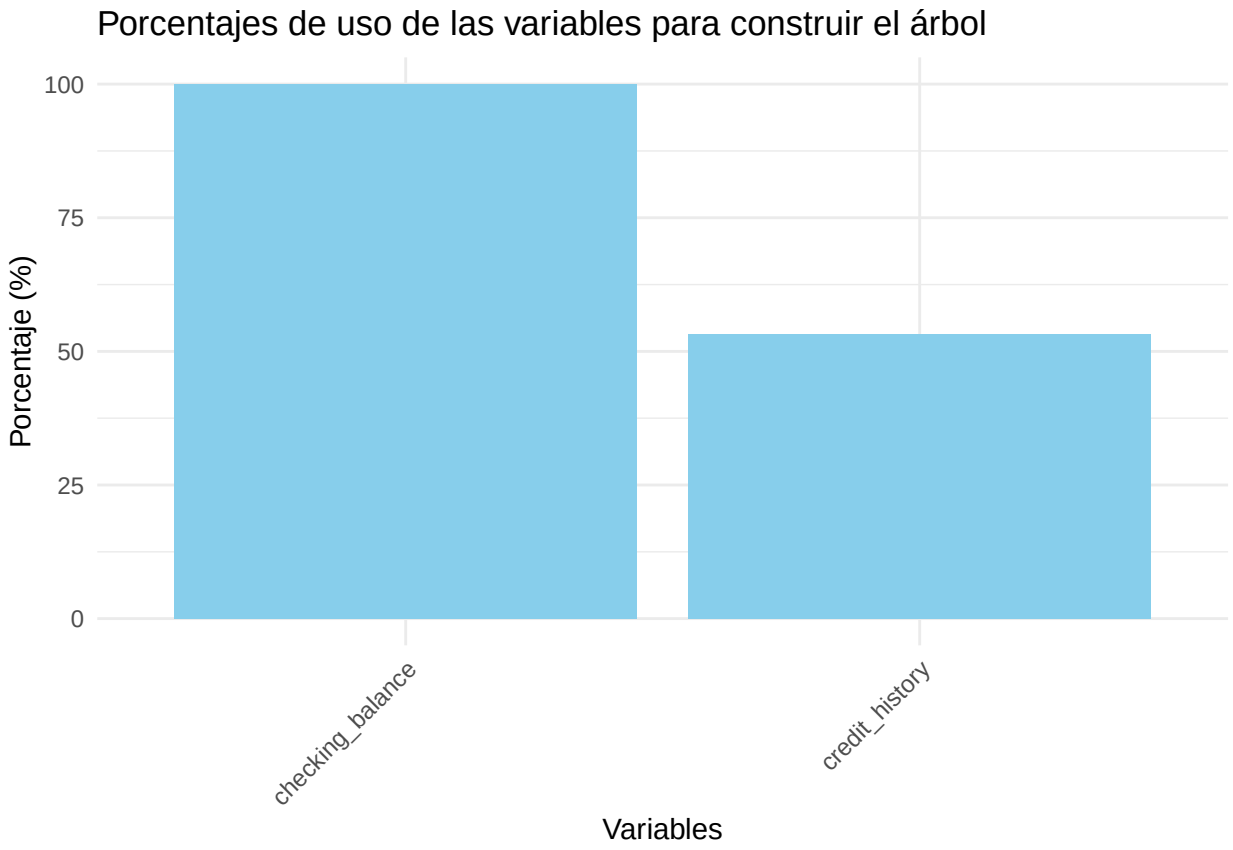
```
# Importamos la librería necesaria
library(C50)
library(ggplot2)

# Extrae las reglas del árbol
importanciaVariables5 = C5imp(modelo5, metric = "usage")
importancia_splits5 <- C50::C5imp(modelo5, metric = "splits")
print(importanciaVariables5)
```

```
##                   Overall
## checking_balance  100.00
## credit_history     53.15
```

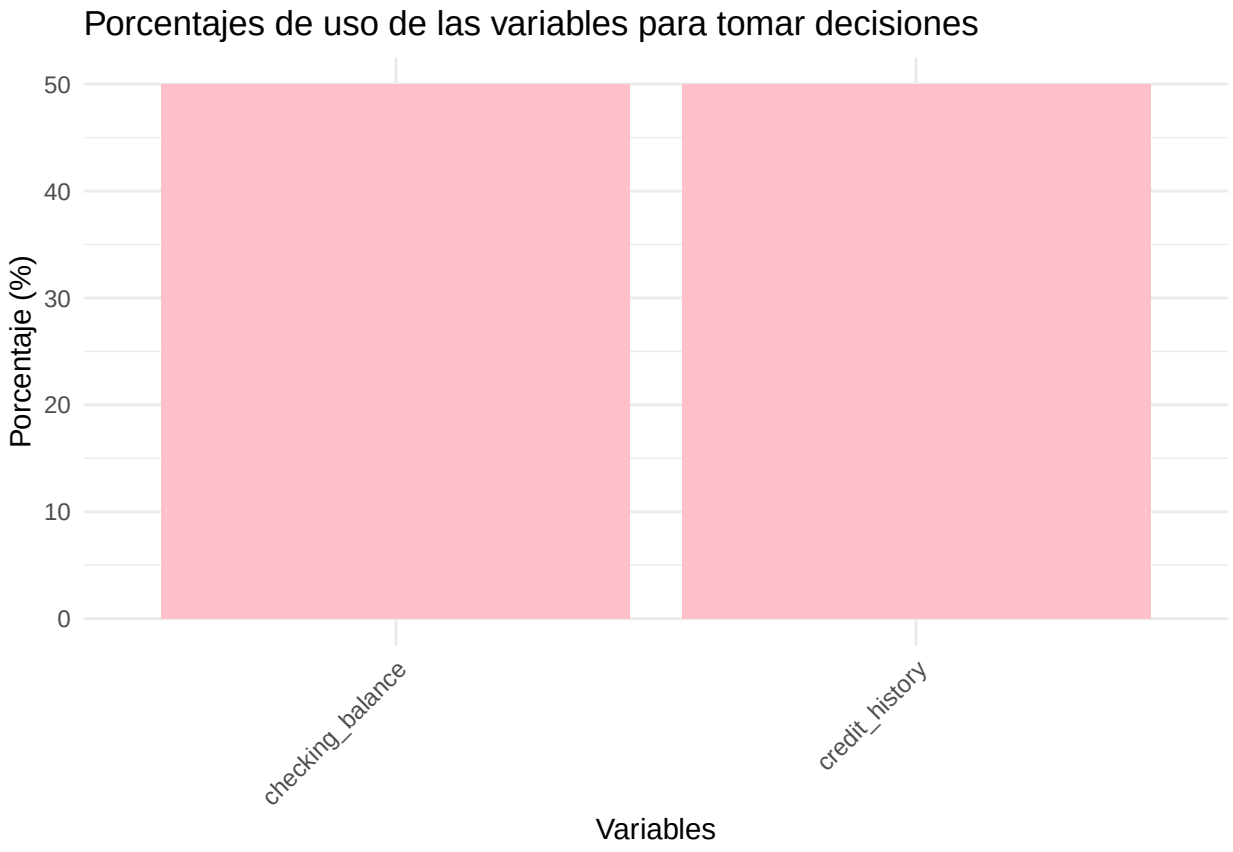
```
print(importancia_splits5)
```

```
##                   Overall
## checking_balance       50
## credit_history         50
```

```
# Muestra la importancia de las variables
# Convertimos las importancias a formato dataframe
df_variables5 <- data.frame(variable = names(importanciaVariables5), importancia = importanciaVariables5
df_splits5 <- data.frame(variable = names(importancia_splits5), importancia = importancia_splits5, tipo

df_variables5$indices = c('checking_balance', 'credit_history')

# definimos la información que va a tener cada eje
indice5 = df_variables5$indices # los indices para las dos gráficas (eje x)
Overall5 = df_variables5$Overall # (% uso/variable de construcción árbol)
divisiones5 = df_splits5$Overall # (% uso/variable de divisiones árbol)

# Uso de cada atributo para construir el árbol (gráfica)
ggplot(df_variables5, aes(x = indice5, y = Overall5)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Porcentajes de uso de las variables para construir el árbol",
       x = "Variables",
       y = "Porcentaje (%)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# Uso de cada atributo para la toma de decisiones en el árbol (gráfica)
ggplot(df_splits5, aes(x = indice5, y = divisiones5)) +
  geom_bar(stat = "identity", fill = "pink") +
  labs(title = "Porcentajes de uso de las variables para tomar decisiones",
       x = "Variables",
       y = "Porcentaje (%)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Como podemos observar, la variable `checking_balance` se ha usado con una tasa del 100 % para construir el árbol, mientras que la variable `credit_history` solo se ha usado un 52 %. En cuanto a división del árbol, las dos variables están igualadas, esto tiene sentido porque son las dos únicas variables presentes en el árbol y las dos desembocan en el mismo número de nodos, la variable `checking_balance` tiene dos caminos, un nodo terminal,y un nodo intermedio que da pie a la variable `credit_history`, a su vez, esta variable genera dos nodos terminales.

Hemos consguido mejorar la precisión del árbol, pero no hemos conseguido mejorar el error de clasificación en los clientes pertenecientes a la clase minoritaria, es por esto, que se va a buscar otra aproximación.

Para esta tarea, implementaremos un algoritmo llamado: *Random Forest*. Según IBM en RANDOM FOREST, *"El random forest, es un algoritmo de aprendizaje automático de uso común, que combina los resultados de varios árboles de decisión para llegar a un único resultado, es un algoritmo fácil de usar y flexible, tanto es así, que su adopción se ha extendido notablemente. Este algoritmo puede implementarse tanto en problemas de clasificación como de regresión."*

Este algoritmo, además, nos va a permitir realizar un análisis correspondiente a la interpretación de las variables en las predicciones llevadas a cabo por el modelo clasificador. Para poder utilizarlo, primero tenemos que cargar la librería:

```
if(!require(randomForest)){
  install.packages('randomForest',repos='http://cran.us.r-project.org')
  library(randomForest)
}

if(!require(iml)){
  install.packages('iml', repos='http://cran.us.r-project.org')
  library(iml)
}
```

Con las librerías cargadas, lanzamos un *random forest*, véase el siguiente *chunk* de código:

```
# train.data <- as.data.frame(cbind(trainx,trainy))
# colnames(train.data)[4] <- "default"
# rf <- randomForest(default ~ ., data = train.data, ntree = 50)
#
# # representamos la importancia de cada variable para las predicciones del random
# # forest que se ha lanzado.
# xx <- train.data[which(names(train.data) != "default")]
# predictor <- Predictor$new(rf, data = xx, y = train.data$default)
# imp <- FeatureImp$new(predictor, loss = "ce")
# plot(imp)
```

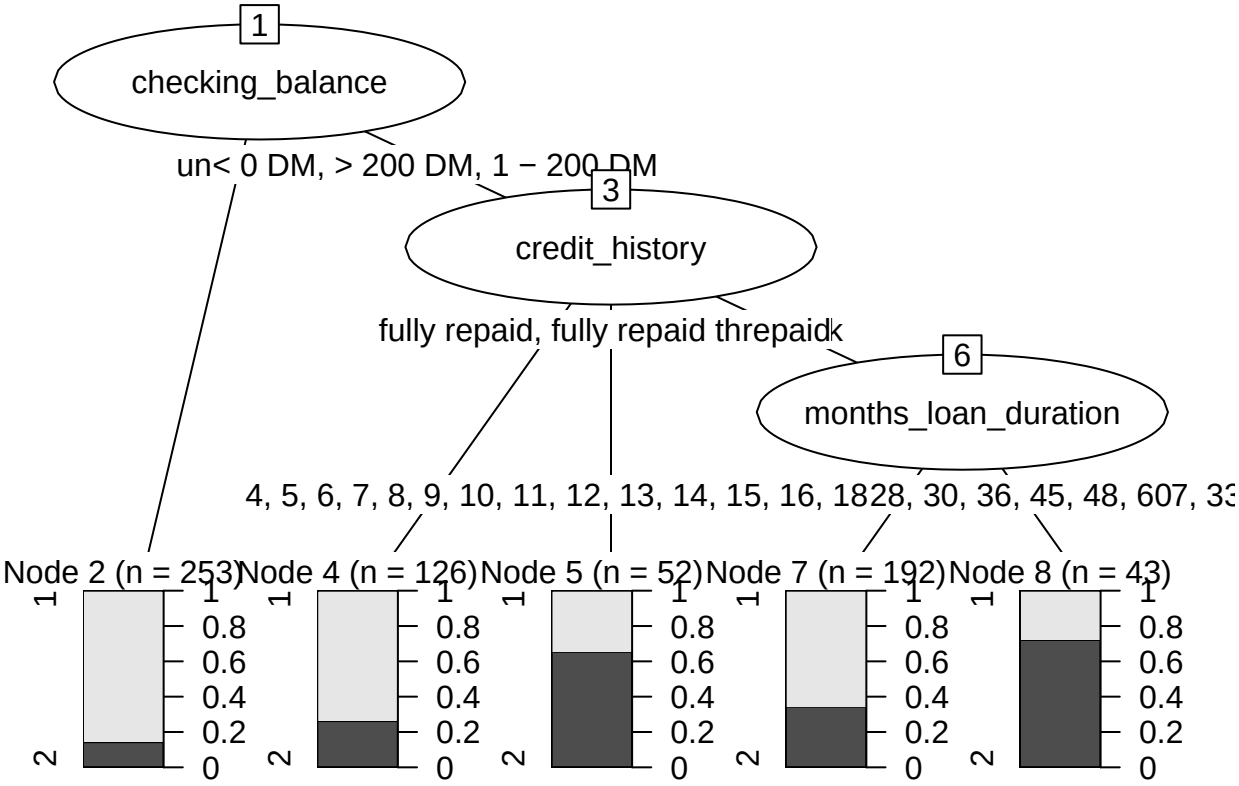
No se ha podido implementar el *random forest* ya que me daba el siguiente error: `Error in Ops.factor(actual, predicted) : level sets of factors are different` , he estado comprobando el tamaño de las variables y de los parámetros, pero todo estaba en orden. He buscado en foros y en StackOverflow y no he encontrado solución.

Con un enfoque parecido a los puntos anteriores y considerando las mismas variables, enriquece el ejercicio mediante el ajuste de modelos de árbol de decisión complementarios. ¿Es el nuevo enfoque mejor que el original? Justifica la respuesta

El árbol de decisión presentado en el anterior apartado es un modelo de clasificación válido, y aceptable (aunque muy mejorable), no obstante, se ha identificado un problema que habría que intentar solventar, y este es, disminuir el error de clasificación en clientes con `default=2` (minoría), es decir, aumentar la precisión a la hora de clasificar a los clientes que nos son positivos. Esto es algo de especial importancia, sobretodo, cuando hay dinero de por medio. En teoría, ya hemos estudiado varias veces el ejemplo de la detección de una enfermedad muy grave o mortal, y se llegó a la conclusión de que aunque fuese muy molesto tener que recetar a una paciente sano, unos medicamentos para poliar una enfermedad que no tiene, porque el modelo encargado de ello no acometió correctamente su tarea, es importante resaltar que este error es tolerable. No obstante, lo que no es tolerable, es que un paciente con una enfermedad grave/mortal pase desapercibido, y no se le diagnostique correctamente. Teniendo en cuenta esto, uno puede ver las similitudes con este proyecto, pues ocurre algo parecido, ya que, aunque puede ser molesto que un cliente sea clasificado como "cliente en situación de default", a pesar de no serlo o de no reunir las características para acabar siéndolo, esto accarrea menos consecuencias que la de no catalogar a un cliente en situación de *default* cuando en verdad si que lo está, ya que se le podría conceder un préstamo a una persona que reune todas las características para no acabar pagándolo o haciéndolo a destiempo, suponiendo pérdidas para la entidad financiera.

A fin de mejorar nuestro modelo, se va probar a modificar la variable `trials` que es un parámetro que se puede especificar en la función `C50::C5.0(trainX, trainy)`, y que especifica el número de iteraciones de refuerzo. Este parámetro se utiliza para determinar el número de iteraciones máximo, que el algoritmo de construcción del árbol de decisión debería realizar. Cada una de las iteraciones se corresponde con un intento de mejorar el árbol. Por lo tanto, primero lo vamos a establecer a 3, i.e., `trials=3`, véase el siguiente *chunk* de código.

```
modelo_mod1 <- C50::C5.0(trainx, trainy, trials = 3)
plot(modelo_mod1)
```



A grandes rasgos, el árbol no ha cambiado mucho, por eso, a continuación se va a repetir el mismo proceso que se ha llevado a cabo antes, para determinar si la precisión del modelo ha aumentado o no. Para ello, se van a calcular las nuevas predicciones asociadas a esta nueva arquitectura de árbol, véase el siguiente *chunk* de código.

```
predicted_modelo_mod1 <- predict( modelo_mod1, testx, type="class" )
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_modelo_mod1 == testy) / length(pred
```

```
## [1] "La precisión del árbol es: 72.7545 %"
```

Podemos ver como la precisión del árbol no ha aumentado, de hecho es la misma que la anterior, por ello, hemos probado con diferentes valores de `trials` pero para todos los valores distintos a 3 que se han probado ($trials \in (1, 2, ..., 100)/\{3\}$), todos han devuelto el mismo error: `Error in FUN(X[[i]], ...) : Variable match was not found.`

Vamos a comprobar si la matriz de confusión ha cambiado (no debería, porque la precisión es la misma)

```
mat_conf_mod1<-table(testy,Predicted=predicted_modelo_mod1)
mat_conf_mod1
```

```
##      Predicted
## testy   1   2
##     1 216  18
##     2  73  27
```

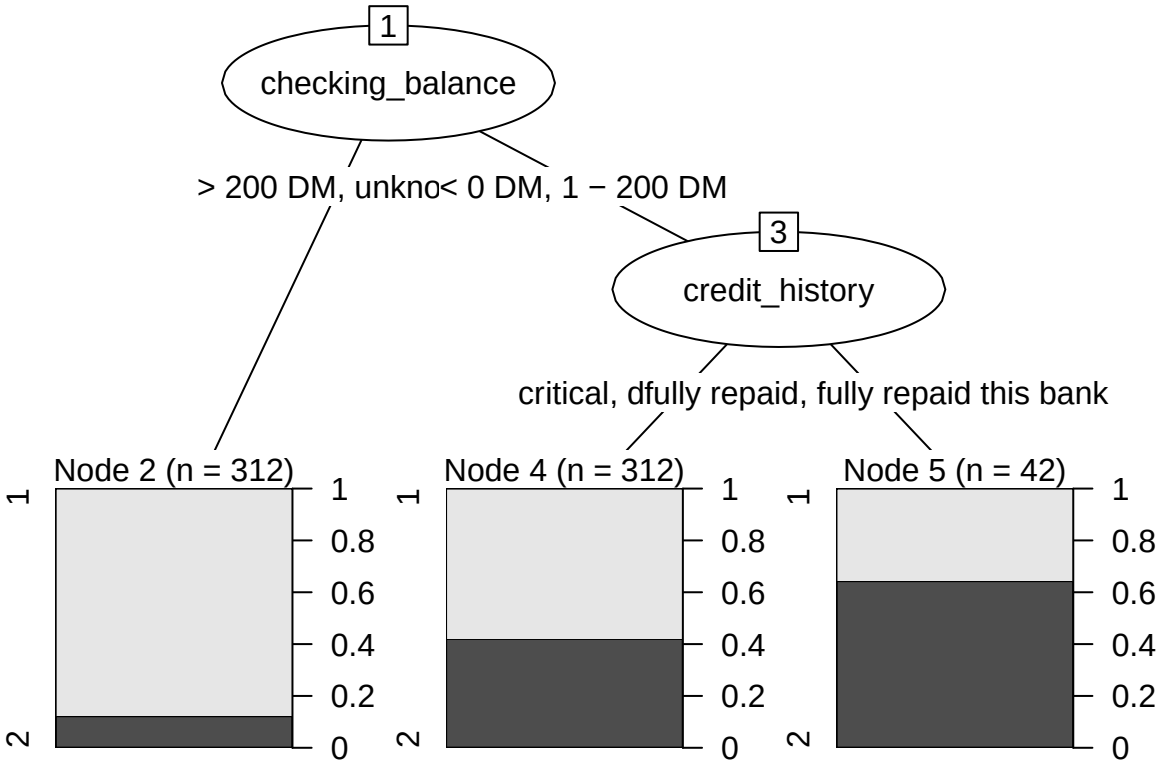
```
# calculamos de la segunda forma, la precisión del modelo.
porcentaje_correcto_mod1<-100 * sum(diag(mat_conf_mod1)) / sum(mat_conf_mod1)
print(sprintf("El %% de registros correctamente clasificados es: %.4f %%",porcentaje_correcto_mod1))
```

```
## [1] "El % de registros correctamente clasificados es: 72.7545 %"
```

Efectivamente no ha cambiado.

Ahora que hemos intentado implementar una variante del primer árbol que hemos diseñado (aquel con precisión: 72'745 %), ahora se va a repetir la tarea pero para el último árbol del anterior ejercicio (aquel con precisión 73'0539 %) En este caso, incialmente también empezaremos con 3 iteraciones de mejora, i.e., `trials=3`

```
set.seed(num_seed)
modelo_mod5 <- C50::C5.0(trainx5, trainy5, trials = 3)
plot(modelo_mod5)
```



A grandes rasgos, el árbol no ha cambiado mucho, por eso, a continuación se va a repetir el mismo proceso que se ha llevado a cabo antes, para determinar si la precisión del modelo ha aumentado o no. Para ello, se van a calcular las nuevas predicciones asociadas a esta nueva arquitectura de árbol, véase el siguiente *chunk* de código.

```
predicted_modelo_mod5 <- predict(modelo_mod5, testx5, type="class")
print(sprintf("La precisión del árbol es: %.4f %%",100*sum(predicted_modelo_mod5 == testy) / length(pred
```

```
## [1] "La precisión del árbol es: 66.4671 %"
```

Como bien puede verse por el resultado de arriba, la precisión ha bajado a 66'4671 %, que respecto a la precisión del árbol original, supone un empeoramiento del 6.5868 %. Pero esto resulta un poco raro, por lo tanto, como se ha hecho para la modificación del primer árbol al principio de este ejercicio, se va a calcular la matriz de confusión para ver si han cambiado mucho las métricas que determinan la calidad de la clasificación:

```
mat_conf_mod5<-table(testy5,Predicted=predicted_modelo_mod5)
mat_conf_mod5
```

```
##       Predicted
## testy5   1   2
##      1 224   6
##      2  82  22
```

```r
# calculamos de la segunda forma, la precisión del modelo.
porcentaje_correcto_mod5<-100 * sum(diag(mat_conf_mod5)) / sum(mat_conf_mod5)
print(sprintf("El %% de registros correctamente clasificados es: %.4f %%",porcentaje_correcto_mod5))
```

```
## [1] "El % de registros correctamente clasificados es: 73.6527 %"
```

Como podemos ver arriba, el % de registros correctamente clasificados es: 73'6527 %, por lo que la precisión ha subido respecto al árbol original (73'0539 %) esta mejora se traduce en un incremento del 0.6 % de la precisión. Pero, ¿donde ha habido una mejora? pues justo en los `TN`, que es justo donde queríamos que se produjese la mejora. Mientras que en el primer modelo del árbol obteníamos `TN=20` en este caso obtenemos `TN=22`, por lo que, los `FP` pasan de `FP=84` a `FP=82`, aunque esto no sea una mejora sustancial, si que arroja luz al diseño del árbol, dando esperanzas a que puede haber otros caminos o algoritmos que mejoren el modelo. No obstante el modelo sigue siendo mejorable.

Por lo tanto teniendo estos dos modelos válidos en la mano, diríamos que el segundo es mucho mejor (aunque no lo veamos reflejado en las métricas) por varias razones.

- **Mayor precisión**, obtenemos una mejora del 0.6 %
- **Menos variables**, hemos conseguido mejorar la precisión y los falsos negativos a partir de menos datos.
- **Mejora de los FPs**, aunque no ha sido una mejora sustancial, se ha conseguido mejorar la marca del primer modelo. Y por ello, hay que tenerlo en cuenta.

Por lo tanto, personalmente diría, que el segundo árbol (con precisión 73'6527 %) si que podría ser un modelo utilizable, pero habría que arreglar el problema de los FPs para hacerlo más robusto. Además aumentando los `trials` hemos conseguido alcanzar dicha mejora, por lo tanto es mejor este árbol (modificado) que el original (con precisión = 73'0539 %))

Haz un resumen de las principales conclusiones de todos los análisis y modelos realizados

Esta ha sido una de las prácticas mas completas que he realizado, ya que hemos partido de un *dataset* que no ofrecía ninguna explicación referente a las variables o al contexto de los registros. Curiosamente, he podido comprobar que el juego de datos que se ha procesado en esta PEC, se corresponde con un *dataset* relacionado con los créditos y la banca, al igual que el *dataset* que escogí en la PAC1, y aunque no sean iguales, hay algunos campos que se repiten, pero tengo que decir que este *dataset* tiene más atributos que el que usé yo (creo recordar)

Como se puede comprobar, en los primeros ejercicios de este documento HTML, se encuentra el análisis exhaustivo del juego de datos. Primeramente, hemos tenido que realizar una investigación para conocer en detalle el significado de muchas variables, gracias a esta labor hemos podido descubrir el significado de la variable `default` y gracias a ello nos hemos dado cuenta de que esta era la variable idónea para clasificar. Seguidamente, se han seguido las pautas ya estudiadas en teoría, acerca de la epxloración del juego de datos así como su limpieza y preparación, de cara a la implementación de modelos supervisados o no supervisados, de distintos tipos, ya sean problemas de clasificación o regresión, binarios o de multiclase. Hemos podido comprobar, como el juego de datos no tenía ningún registro vacío, ni tampoco valores nulos, y adicionalmente se ha comprobado si exístian valores repetidos, ya que en la PAC1, esto era un problema, ya no tanto en la misma práctica, sino de cara a la PAC2 que es donde se aplicarán los modelos deseados.

Tras revisar el estado del juego de datos, y confirmar que estaba todo limpio y en orden, se ha procedido a la extracción de conocimiento a partir de los datos. Para ello, se han relacionado variables tanto gráficamente, como mediante tablas de contingencia. Gracias a estas tablas de contingencia, se ha podido descubrir, como solo los clientes que habían pagado al completo, y los que lo habían pagado al completo también al banco, i.e., `fully repaid this bank` eran los únicos donde predominaba `default=2`. Con estas relaciones, se ha podido constatar rambién la fuerte correlación entre el sueldo de los clientes y su vairable `default`, ya que se ha demostrado como los clientes con mayor sueldo, son más propensos a acabar en `default`. En

definitiva, gracias a este análisis pudimos descubrir la gran importancia de las variables `credit_history` y `checking_balance`.

Teniendo el conjunto de variables claro, primero se apostó por un árbol de decisión muy grande (muchas variables) luego a la hora de deplegarlo, se pudo ver como no obteníamos un árbol valido ni tampoco reglas. Rebajando el número de variables conseguimos obtener el primer árbol válido, con una precisión del 72'745 %, pero con un problema de clasificación en las clases minoritarias, no obstante, aceptable. Luego hemos realizado variaciones del mismo, pero no hemos conseguido subir su precisión. Por ello, a mitad de camino se han intentado modelar otros árboles con diferentes combinaciones de variables, pero como se ha podido ver, los árboles solo arrojaban resultados aceptables, cuando estaban presentes las variables `checking_balance` y `credit_history`, es por esto que en el último caso se ha optado por implementar un árbol solo con esas dos variables y la variable clasificadora (`default`) y se han obtenido mejores resultados tanto en la precisión como en la clasificación de los FP, no obstante es un modelo muy mejorable. Por último, se ha intentando implementar el algoritmo 'random_forest" pero no ha sido posible por errores de variables, que no he podido solucionar.

Cabe destacar que se han probado muchas combinaciones de variables, no solo aquellas que se han identificado como interesantes en el análisis realizado al inicio de esta PEC. Además, se ha probado también para muchos valores de la semilla aleatoria y no ha habido suerte, ya que la precisión no aumentaba, y los árboles que obteníamos eran o muy simples o muy complejos, y habiéndolos estudiado, y viendo las precisiones obtenidas, se podía ver claramente como los árboles más complejos tenían precisiones más bajas. Esto puede deberse al problema de incluir variables que no aportan mucha información al modelo y que por lo tanto no están tan correladas entre sí. Por ello, el mejor modelo que se ha obtenido en este caso, solo incluye dos variables/atributos, que se de hecho se corresponden con las variables que guardan más relación con la variable clasificatoria; `default`, estudiadas en el análisis al principio de este proyecto.

En definitiva, en esta práctica hemos sido capaces de realizar las tareas fundamentales del ciclo de vida de un proyecto de minería de datos, donde además hemos puesto en práctica la aplicación de modelos supervisados de clasificación binaria como son los árboles de decisión. Además, hemos implementado varios diseños de árboles, que hemos sabido evaluar a partir de sus matrices de confusión y curvas ROC. Finalmente solo dos árboles han sido resultado ser válidos, pero el último ha resultado ser el más preciso.

Rúbrica

- (Obligatorio) Se debe realizar un breve informe (Html) donde se respondan a las preguntas concretas, mostrando en primer lugar el código utilizado, luego los resultados y posteriormente los comentarios que se consideren pertinentes para cada apartado.

- 10% Hay un estudio sobre los datos de los que se parte, las variables que componen los datos. Los datos son preparados correctamente.
- 10% Se realiza un análisis descriptivo univariante (o análisis de relevancia) de algunas variables una vez se han tratado vs el target a nivel gráfico, comentando las que aparentemente son más interesantes. Análogamente se realiza un análisis de correlaciones.
- 20% Se aplica un árbol de decisión de forma correcta y se obtiene una estimación del error, mostrando gráficamente el árbol obtenido. La visualización debe ser comprensible y adecuada al problema a resolver.
- 15% Se explican las reglas que se obtienen en términos concretos del problema a resolver.
- 10% Se usa el modelo para predecir con muestras no usadas en el entrenamiento (holdout) y se obtiene una estimación del error. En base a la matriz de confusión, se comentan los tipos de errores y se valora de forma adecuada la capacidad predictiva del algoritmo.
- 10% Se prueba otro modelo de árbol o variantes diferentes del C50 y se comparan los resultados obtenidos, valorando si son mejores. Se recomienda experimentar usando un Random Forest (librería

de R *randomForest* (https://cran.r-project.org/web/packages/randomForest/randomForest.pdf)).

- 10% Con los resultados obtenidos anteriormente, se presentan unas conclusiones contextualizadas donde se expone un resumen de los diferentes modelos utilizados (al menos 3) así como el conocimiento adquirido tras el trabajo realizado y los descubrimientos más importantes realizados en el conjunto de datos.
- 10% Utiliza métricas de explicabilidad como las comentadas en el ejemplo para obtener conclusiones de los datos.
- 5% Se presenta el código y es fácilmente reproducible.