Modelo *logit* binomial.

Clasificando a las empresas eólicas.

Vamos a considerar una muestra de 100 empresas de producción eléctrica mediante tecnología eólica. Estas empresas han sido valoradas por un grupo de expertos en el sector, a fin de aconsejar o desaconsejar el invertir en ellas. Nuestro objetivo es estimar un modelo ***logit* binomial** que nos permita automatizar esta clasificación de empresas (entre las que son de inversión aconsejable y las que no), en función de una serie de variables económico-financieras. Estas variables son: la rentabilidad económica (RENECO), la rentabilidad financiera (RENFIN), la liquidez (LIQUIDEZ), el margen de beneficio (MARGEN), el grado de solvencia (SOLVENCIA) y el apalancamiento (APALANCA). La variable cualitativa, factor o atributo que recoge la opinión del panel de expertos es VALORA, y puede tomar dos categorías: "aconseja" (invertir) y "desaconseja" (invertir).

Preparando Datos.

Abriremos R-Studio y crearemos nuestro **proyecto** siguiendo la instrucción File → New Project. Luego nos preguntará si crea el proyecto en una nueva carpeta o en una ya existente. Vamos a crearlo, por ejemplo, en el disco extraíble D, carpeta R, subcarpeta "logit", que ya existe. Nos saldrá una ventana para buscar la carpeta y, cuando la encontremos, pulsamos Open y Create Project.

Vamos a ir a la carpeta del proyecto y vamos a guardar en ella los dos archivos de esta práctica: un archivo de Microsoft® Excel® llamado "eolica_100_log.xlsx" y un *script* denominado "logit_eolica.R". Si abrimos el archivo de Microsoft® Excel®, comprobaremos que se compone de 4 hojas. La primera muestra el criterio de búsqueda de casos en la base de datos

Sabi®; la segunda recoge la descripción de las variables consideradas, y la tercera (hoja "Datos") guarda los datos que debemos importar desde R-Studio. Estos datos se corresponden con diferentes variables económico-financieras de las empresas productoras de electricidad mediante generación eólica. La cuarta hoja, "Simula", se utilizará para llevar a cabo la última fase de la práctica.

Luego vamos a cerrar el archivo de Microsoft® Excel® y volveremos a R-Studio. Vamos a abrir nuestro script "logit_eolica.R" con File → Open File… Este script contiene el programa que vamos a ir ejecutando en la práctica.

La primera línea / instrucción en el script es:

```
rm(list = ls())
```

La instrucción tiene como objeto limpiar el *Environment* de objetos de anteriores sesiones de trabajo. Para importar los datos, ejecutaremos el código:

```
# DATOS

library(readxl)
datos <- read_excel("eolica_100_log.xlsx", sheet = "Datos")
```

Podemos observar cómo, en el *Environment,* ya aparece un objeto. Este objeto es una estructura de datos tipo *data frame*, se llama "datos" y contiene 8 columnas, una por cada una de las variables almacenadas en el archivo de Microsoft® Excel®.

R ha considerado la primera columna (NOMBRE) como una variable de tipo cualitativo. En realidad, no es una variable, sino el nombre de los casos (empresas). Para evitar que R tome los nombres de los individuos como una variable, podemos redefinir nuestro *data frame* diciéndole que tome esa primera columna como los nombres de los individuos o casos (filas):

```
datos <- data.frame(datos, row.names = 1)
```

En la línea anterior hemos asignado al *data frame* "datos" sus propios datos; pero indicando que la primera columna no es una variable; sino el nombre de los individuos, casos o filas. Vemos que ya no aparece NOMBRE como variable, y que en el *Environment* ya aparece el *data frame* "datos" con 100 observaciones (casos), pero con 7 variables (una menos).

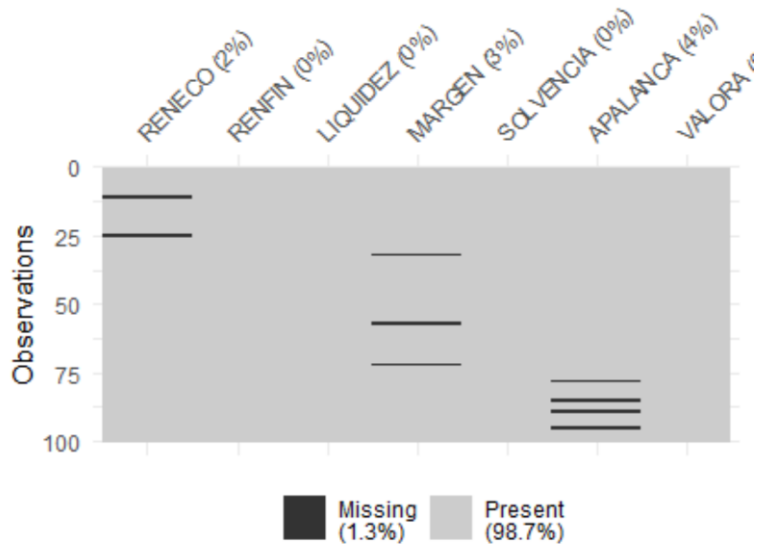
Vamos a utilizar todas las variables del *data frame* "datos", luego **no es necesario** generar otro *data frame* con las variables que entran en el análisis, mediante la función select() del paquete dplyr.

El siguiente paso será localizar los posibles ***missing values***, ya que para realizar el análisis es necesario que todos los casos posean dato en todas las variables del estudio. Para tener una idea general, se puede utilizar la función vis_miss() del paquete visdat, que nos localizará gráficamente los *missing values* de las diferentes variables, y calculará el porcentaje de casos que supone, con respecto al total de observaciones:

```
# Identificando missing values.

library(visdat)
vis_miss(datos)
```

El resultado del código anterior es el siguiente gráfico:



Del gráfico anterior se desprende que existen varios *missing values*. Para localizarlos, podemos filtrar nuestro *data frame* con las herramientas de dplyr:

```
library(dplyr)
datos %>% filter(is.na(RENECO) | is.na(RENFIN) | is.na(LIQUIDEZ) |
                 is.na(MARGEN) | is.na(SOLVENCIA) | is.na(APALANCA) |
                 is.na(VALORA)) %>%
          select(RENECO, RENFIN, LIQUIDEZ, MARGEN, SOLVENCIA, APALANCA,
VALORA)
```

Los casos detectados con *missing values* son:

	RENECO	RENFIN	LIQUIDEZ	MARGEN	SOLVENCIA	APALANCA	VALORA
Viesgo Renovables SL.	NA	3.200	0.272	11.818	65.883	13.330	DESACONSEJA
Sargon Energias SLU	NA	26.900	0.188	-615.625	-12.811	-879.289	ACONSEJA
Parc Eolic Sant Antoni SL	1.361	9.746	0.839	NA	13.964	595.281	DESACONSEJA
Eolica La Brujula SA	7.295	14.174	1.643	NA	51.474	85.954	DESACONSEJA
WPD Parque Eolico Navillas SL.	-0.416	-14.302	0.014	NA	2.912	3127.288	DESACONSEJA
Elecdey Palencia S.L.	3.965	83.297	1.683	22.016	4.760	NA	ACONSEJA
Sistemes Energetics Conesa I S.L.	3.688	-115.485	1.248	16.649	-3.193	NA	DESACONSEJA
Inversiones Fotovoltaicas Mallorquinas SL.	1.248	38.585	0.968	11.661	3.235	NA	DESACONSEJA
El Paramo Parque Eolico SL	3.416	-287.974	0.708	16.267	-1.186	NA	DESACONSEJA

Ante la existencia de *missing values*, se puede actuar de varios modos. Por ejemplo, **se puede intentar obtener por otro canal de información el valor** que no está disponible, **o recurrir a alguna estimación**. En caso de que esto sea difícil, se puede optar, simplemente, por **eliminar** los casos. En nuestro ejemplo, supondremos que hemos optado por esta última vía:

```
datos <- datos %>% filter(! is.na(RENECO) & ! is.na(RENFIN) & !
is.na(LIQUIDEZ) &
                          ! is.na(MARGEN) & ! is.na(SOLVENCIA) & !
is.na(APALANCA) &
                          ! is.na(VALORA)) %>%
                  select(RENECO, RENFIN, LIQUIDEZ, MARGEN, SOLVENCIA,
APALANCA, VALORA)
```

Podemos verificar en el *Environment* que el *data frame* "datos" ha pasado a tener 91 casos.

Nota: sería conveniente, como es habitual, una detección y tratamiento de los *outliers*, ya que su inclusión en la muestra puede derivar en una pérdida en la calidad de los resultados. Al tratarse de una toma de contacto con la técnica, se omite este paso.

Además, conviene **declarar** la variable categórica VALORA, que es nuestra variable dependiente, **como factor**, ya que, de no hacerlo, no se podrá estimar el modelo *logit*, como se verá a continuación. Para ello, basta hacer:

```
# La variable dependiente, categorica, ha de ser declarada factor

datos$VALORA <- as.factor(datos$VALORA)
```

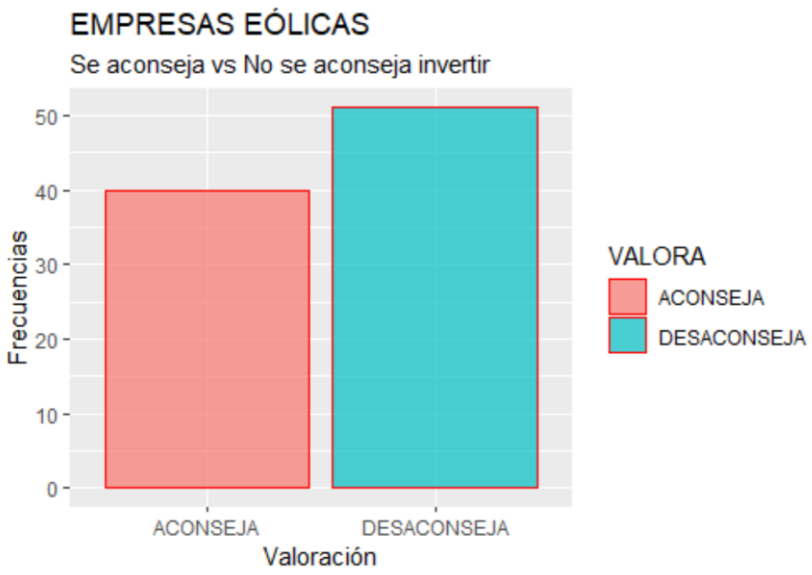
Conviene construir, por otro lado, un gráfico que informe sobre la cantidad de casos que toman cada opción en el factor VALORA, para lo cual utilizaremos la función ggplot() del paquete ggplot2:

```
library (ggplot2)

ggplot(data = datos, map = aes(x = VALORA, fill = VALORA)) +
  geom_bar(colour = "red", alpha = 0.7) +
```

```
  ggtitle("EMPRESAS EÓLICAS", subtitle = "Se aconseja vs No se aconseja
invertir")+
  xlab("Valoración") +
  ylab("Frecuencias")
```

El resultado muestra cómo existe un mayor número de empresas en las que se desaconseja invertir:



Por otro lado, cuando se estiman modelos *logit*, es corriente **dividir la muestra en dos submuestras generadas aleatoriamente**: una para **estimar** el modelo (con en torno al 80% de los casos), y otra para **validar** el modelo (con entorno el 20% de casos restante). Para ello, podrá utilizarse el código:

```
# Dividiendo la muestra en muestra de estimación y muestra de validación

set.seed(1000)
n_valida <- sample(nrow(datos), nrow(datos)*0.2)
valida <- datos[n_valida,]
estima <- datos[-n_valida,]
```

Con el código anterior, se ha divido el *data frame* "datos" en dos: Un *data frame* llamado **"estima"**, con los casos que se utilizarán en la estimación, y un *data frame* denominado **"valida"**, que servirá para hacer una prueba de bondad del modelo estimado. La función para elegir aleatoriamente las filas (casos) del *data frame* original es sample(), con dos argumentos: el que le dice el número de filas (de casos) que hay en total, y el número de estas filas (casos) que ha de seleccionar. La función set.seed() se utiliza para que, en caso de repetir la ejecución del código, que no cambien los resultados (filas o casos elegidos aleatoriamente).

Especificación y estimación del modelo.

Vamos a estimar un modelo *logit* inicial, en la que la variable dependiente es el factor dicotómico VALORA, y las variables explicativas son la rentabilidad económica (RENECO), la rentabilidad financiera (RENFIN), la liquidez (LIQUIDEZ), el margen de beneficio (MARGEN), el grado de solvencia (SOLVENCIA) y el apalancamiento (APALANCA). La estimación se guardará en un objeto de R al que llamaremos "modelo.inicial". El modelo nos aportará, una vez estimado, la probabilidad de que VALORA tome valor 1 (es decir, que los expertos desaconsejen invertir en la empresa en cuestión). En realidad, por defecto R codifica las categorías del factor que actúa como variable dependiente dicotómica, asignando **valor 0 a la primera categoría según el orden alfabético ("aconseja"), y valor 1 a la segunda ("desaconseja")**.

La función que permite estimar el modelo *logit* binomial es glm(). Hemos de recordar que, para alimentar al modelo, no se utilizarán todos los casos, sino **solo los del *data frame* "estima"**. Para especificar el tipo de modelo (*logit* binomial), habrá que indicar el argumento: **family=binomial(link="logit"):**

```
modelo.inicial <- glm(formula = VALORA ~ RENECO + RENFIN + LIQUIDEZ +
MARGEN + SOLVENCIA + APALANCA,
                      data=estima, family=binomial(link="logit"))
summary (modelo.inicial)
```

Tras ejecutar el código anterior, se nos ofrecen los siguientes resultados:

```
Call:
glm(formula = VALORA ~ RENECO + RENFIN + LIQUIDEZ + MARGEN +
    SOLVENCIA + APALANCA, family = binomial(link = "logit"),
    data = estima)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.98550  -0.33372   0.01025   0.38626   2.83457

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  6.613351   1.776083   3.724 0.000196 ***
RENECO      -0.109825   0.128347  -0.856 0.392170
RENFIN      -0.101306   0.040934  -2.475 0.013329 *
LIQUIDEZ    -1.868930   0.696148  -2.685 0.007260 **
MARGEN       0.003887   0.006225   0.624 0.532394
SOLVENCIA   -0.037261   0.021133  -1.763 0.077879 .
APALANCA     0.001484   0.001176   1.262 0.207086
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 100.087  on 72  degrees of freedom
Residual deviance:  41.127  on 66  degrees of freedom
AIC: 55.127

Number of Fisher Scoring iterations: 9
```

La columna "Estimate" recoge los parámetros estimados en el modelo *logit*, es decir, los parámetros $\hat{\beta}$ en la expresión:

$$\hat{p}(y = 1/x_1, x_2, \ldots, x_k) = \frac{1}{1 + e^{-(X_1\hat{\beta}_1 + X_2\hat{\beta}_2 + \cdots + X_k\hat{\beta}_k)}}$$

"Intercept" es el término independiente $(\hat{\beta}_1)$. La columna "Pr(>|z|)" recoge los **p-valores** asociados al contraste de significación estadística de cada parámetro. Es común considerar estadísticamente significativos los parámetros con un **p-valor asociado menor a 0.05**. En nuestro caso, los parámetros estadísticamente significativos son: el término independiente, y los parámetros asociados las variables RENFIN y LIQUIDEZ. Ambas serían las variables que más influyen en la probabilidad de que el panel de expertos "desaconseje" (invertir). En concreto, al ser números negativos, en sentido inverso: a mayor RENFIN y LIQUIDEZ, menor es la probabilidad de que la variable VALORA tome valor 0 ("desaconseja").

Una información importante que se nos muestra es el "**AIC**" (*Criterio de Información de Akaike*). Esta medida de bondad, relacionada con la función de verosimilitud de la muestra, toma un valor menor (incluso negativo) cuanto mejor representa la realidad el modelo estimado. Es utilizado para comparar estimaciones alternativas.

Precisamente, el siguiente paso será intentar mejorar la especificación del modelo inicial. Para ello, se utilizará la función step(), en su versión "backward". Este algoritmo elimina de la especificación variables explicativas, siempre y cuando esto implique una **disminución del valor de "AIC"** (mejor bondad). Este procedimiento de mejora premia las especificaciones sencillas, atendiendo al *"Principio de Parsimonia".* El código es:

```
modelo.final <- step(modelo.inicial, direction = "backward", trace = 1,
k = 2)
summary (modelo.final)
```

El primer argumento es el nombre del modelo a “mejorar”. El argumento “direction” determina la versión del algoritmo. El argumento “trace” indica si se quiere que se visualice en pantalla el proceso de mejora del modelo (1) o no (0), y el argumento “k =” se refiere al número mínimo de variables que se quiere que incorpore la especificación del modelo. El modelo final, resultado del proceso de mejora, se guarda asignándolo al objeto “modelo.final”. Al hacer el summary() de esta estimación, se obtiene:

```
Call:
glm(formula = VALORA ~ RENFIN + LIQUIDEZ + SOLVENCIA + APALANCA,
    family = binomial(link = "logit"), data = estima)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.01272  -0.30341   0.00606   0.38876   2.85150

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  6.7729906  1.7923954   3.779 0.000158 ***
RENFIN      -0.1202871  0.0355406  -3.384 0.000713 ***
LIQUIDEZ    -1.8663149  0.6971728  -2.677 0.007429 **
SOLVENCIA   -0.0470824  0.0184200  -2.556 0.010587 *
APALANCA     0.0019177  0.0009812   1.954 0.050660 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 100.087  on 72  degrees of freedom
Residual deviance:  41.978  on 68  degrees of freedom
AIC: 51.978

Number of Fisher Scoring iterations: 9
```

Puede observarse que el modelo ha pasado a contener solo 4 variables explicativas (descartando RENECO y MARGEN). Por otro lado, el valor de “AIC” ha disminuido, lo que implica una mayor bondad del modelo final con respecto al inicial.

Precisamente, vamos a pasar a medir dicha bondad del ajuste (capacidad del modelo para representar la realidad). Lo llevaremos a cabo mediante dos vías: el cálculo del *coeficiente de determinación de Nagelkerke*, y mediante la construcción de la *matriz de confusión*.

Validación del modelo.

El ***coeficiente de determinación de Nagelkerke*** es una adaptación del coeficiente de determinación lineal (R^2) de los modelos de regresión estándar. El R^2 toma valores entre 0 y 1, e indica la proporción de varianza de la variable dependiente recogida por el modelo estimado. A mayor R^2, mejor bondad. Para calcular el *coeficiente de determinación de Nagelkerke* será necesario utilizar la función NagelkerkeR2() del paquete fmsb:

```
#BONDAD DEL AJUSTE

#R2 de Nagelkerke

library (fmsb)
NagelkerkeR2(modelo.final)
```

El resultado obtenido es:

```
$N
[1] 73

$R2
[1] 0.7355975
```

Es decir, el modelo posee una bondad razonablemente aceptable, con un R^2 de 0,74.

La **matriz de confusión** es una tabla de doble entrada en la que, por columnas, se muestra el número de casos (observados) que realmente toman una u otra categoría del factor ("aconseja" o 0, y "desaconseja o 1"); mientras que por filas se cuentan los casos clasificados en una u otra categoría por el modelo. Así, cuanto mayor sea el número de casos aparecidos en la diagonal principal de la matriz, mejor será la capacidad de predicción (y por tanto, la bondad) del modelo (casos bien clasificados).

Es importante tener en cuenta que en la construcción de la matriz se utilizan solo los **casos de la muestra que no han sido empleados a la hora de estimar** el modelo; es decir, en nuestro caso, los pertenecientes al *data frame* **"valida"**.

Antes de construir la matriz, hemos de proceder a guardar la clasificación de los casos (predicción) hecha por el modelo, a fin de poder contrastar los resultados con los datos reales. Para ello, se utiliza la función predict(). Al utilizar el argumento "type= response", esta función nos devuelve, para

cada caso de los que integran el *data frame* "valida", la probabilidad que otorga el modelo a que el factor VALORA tome categoría 1 ("desaconseja"). Esta probabilidad se redondea, con la función round(), a 0 decimales; de modo que si la probabilidad predicha es igual o mayor que 0,5, la predicción se convierte en 1 ("desaconseja"); mientras que si es menor de 0,5, al redondear se convierte en 0 ("aconseja"). Esos valores dicotómicos son los que se asignan al objeto "prediccion" (que es un vector). Hay que tener en cuenta que en el argumento "newdata= " hay que indicar el *data frame* "valida":

```
prediccion <- round(predict(modelo.final, newdata= valida, type=
"response"), 0)
```

Para crear la tabla, asignaremos a un nombre de objeto, por ejemplo "tb", el resultado de aplicar la función table(). Este objeto ya no es un *data frame*, sino un **objeto "tabla"**.

El primer argumento indica que, por filas, estarán las frecuencias de las categorías predichas (los unos y los ceros, es decir, "aconseja" y "desaconseja", respectivamente). El segundo argumento se refiere a las frecuencias de cada categoría observadas en la submuestra "valida". Para que se puedan contabilizar, hay que considerar el factor VALORA como variable métrica (números enteros), por lo que se utiliza la función as.numeric(). Por último, dado que as.numeric() codifica por defecto las categorías dotándolas de un número entero, comenzando desde 1 (o sea, en este caso, 1 y 2), habrá que restarle una unidad (para que sean 0 y 1, como las previsiones):

```
tb<-table(prediccion, as.numeric(valida$VALORA)-1)
tb
```

El resultado es el siguiente:

```
prediccion  0  1
         0  8  0
         1  0 10
```

En la tabla se muestra cómo en el *data frame* "valida" hay 8 casos en los que VALORA toma realmente valor 0 ("aconseja, primera columna"), mientras que hay 10 casos en los que toma valor 1 ("desaconseja, segunda columna). Mirando las filas, vemos que el modelo ha acertado en todos los casos con su clasificación, por lo que, en las celdas de la tabla de la diagonal principal están todas las frecuencias, y en el resto se muestran ceros.

La apariencia de la tabla se puede mejorar notablemente. Para ello, activaremos las librerías knitr y kableExtra, e indicaremos que la tabla se mostrará en formato “html”:

```
library(knitr)
library(kableExtra)
knitr.table.format = "html"
```

Luego el código para especificar la visualización de la tabla, será:

```
addmargins(tb) %>%
  kable(caption="Matriz de confusión",
        col.names = c("Aconseja (0)", "Desaconseja (1)", "Sum")) %>%
  kable_styling(full_width = F, bootstrap_options = "striped",
"bordered", "condensed", position = "center", font_size = 12) %>%
  add_header_above(c("PREDICCIÓN"= 1, "OBSERVADO"=3), bold=T, line=T)
%>%
  row_spec(0, bold= T, align = "c") %>%
  row_spec(1:3, bold= F, align = "c") %>%
  column_spec(1, bold = T)
```

La función addmargins() añade la suma de las filas y columnas de la tabla. Con la función kable(), del paquete knitr, se añade el título de la tabla, y el nombre de las columnas. Las funciones kable_styling(), add_header_above(), row_spec() y column_spec() pertenecen al paquete kableExtra, y permiten personalizar en mayor grado la apariencia de la tabla. El resultado es el siguiente:

Matriz de confusión

PREDICCIÓN	OBSERVADO		
	Aconseja (0)	Desaconseja (1)	Sum
0	8	0	8
1	0	10	10
Sum	8	10	18

Finalmente, el porcentaje de aciertos en la predicción del modelo se puede calcular fácilmente mediante el código:

```
round((sum(diag(tb))/sum(tb)*100),2) #porcentaje de aciertos
```

Al ejecutar la línea anterior, el porcentaje de aciertos es, lógicamente, el 100%:

```
[1] 100
```

Utilización del modelo: simulación.

Como última etapa, y una vez hemos validado el modelo, comprobando que es capaz de representar de modo aceptable la realidad, vamos a emplearlo en una aplicación. En concreto, haremos un ejercicio de simulación. Para ello, propondremos un escenario (valor de las variables explicativas), y obtendremos, según tales valores, la clasificación que del caso del escenario hace el modelo.

El escenario está en la hoja "Simula" del archivo de Microsoft® Excel® llamado "eolica_100_log.xlsx", y es el siguiente:

NOMBRE	RENECO	RENFIN	LIQUIDEZ	MARGEN	SOLVENCIA	APALANCA
ESCENARIO A	6,00	9,00	1,02	50,20	50,20	91,96

Como puede observarse, el escenario propone un valor para cada una de las variables explicativas. Es importante tener en cuenta que, para poder hacer la simulación, bastaría con que solo tuviera valor en las variables RENFIN, LIQUIDEZ, SOLVENCIA y APALANCA, porque son las que al final entraron en la especificación del modelo final (mejorado).

El escenario será importado a R como si se tratara de cualquier otro *data frame*. De hecho, podríamos haber propuesto más escenarios alternativos, como filas del *data frame*. Lo llamaremos, por ejemplo, "escenario". El código será el habitual:

```
escenario <- read_excel("eolica_100_log.xlsx", sheet = "Simula")
escenario <- data.frame(escenario, row.names = 1)
summary (escenario)
```

Para realizar la simulación, haremos uso de la función predict(), exactamente como hicimos para construir la *tabla de confusión*. El resultado de la simulación lo almacenaremos en el vector "simulación":

```
simulacion <- round(predict(modelo.final, newdata = escenario,
type="response"), 0)
simulacion
```

Vemos que el resultado de la simulación para el escenario propuesto es 1, es decir, se "desaconseja" invertir en un caso con esos valores en las variables explicativas.

Como cierre, construiremos una tabla para mostrar, de un modo más elegante, el resultado de la simulación. En primer lugar, construiremos el *data frame* "resultado" a partir de la unión del *data frame* "escenario" y el vector "simulación", con la función cb(). Luego, aplicaremos las funcionalidades de los paquetes knitr y kableExtra:

```
resultado <- cbind(simula, simulacion)
resultado %>%
            kable(caption = "Simulación / Previsión Modelo LOGIT
Inversiones", digits = c(0, 2, 2, 2, 2, 2, 0),
                  col.names = c("Escenario", "R. Económica", "R.
Financiera", "Liquidez", "Margen", "Solvencia", "Apalancamiento",
"Aconseja (0) / Desaconseja (1)")) %>%
            kable_styling(full_width = F, bootstrap_options =
"striped", "bordered", "condensed", position = "center", font_size = 11)
%>%
            row_spec(0, bold= T, align = "c") %>%
            row_spec(1:(nrow(resultado)), bold= F, align = "c")
```

El resultado de ejecutar el código anterior será:

Simulación / Previsión Modelo LOGIT Inversiones

Escenario	R. Económica	R. Financiera	Liquidez	Margen	Solvencia	Apalancamiento	Aconseja (0) / Desaconseja (1)
ESCENARIO A	6	9	1.02	50.2	50.2	92	1

Updated: 27/02/2023