proyecto-final-ml-preprocessing

June 8, 2023

1 PRÁCTICA FINAL MACHINE LEARNING. PARTE 1

Por Sofía Canela García-Arias, Berta Corriols Barrio & Víctor Panadero Gómez

[1]:
```
conda install -c conda-forge imbalanced-learn
```

```
Collecting package metadata (current_repodata.json): …working… done
Note: you may need to restart the kernel to use updated packages.



==> WARNING: A newer version of conda exists. <==
  current version: 23.3.1
  latest version: 23.5.0

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

     conda install conda=23.5.0



Solving environment: …working… done

## Package Plan ##

  environment location: C:\Users\nacho\anaconda3

  added / updated specs:
    - imbalanced-learn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
```

```
    ca-certificates-2023.5.7   |       h56e8100_0         145 KB  conda-forge
    certifi-2023.5.7           |     pyhd8ed1ab_0         149 KB  conda-forge
    imbalanced-learn-0.10.1    |     pyhd8ed1ab_0         131 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         425 KB

The following packages will be UPDATED:

  ca-certificates    pkgs/main::ca-certificates-2023.01.10~ --> conda-forge::ca-
certificates-2023.5.7-h56e8100_0
  certifi            pkgs/main/win-64::certifi-2022.12.7-p~ --> conda-
forge/noarch::certifi-2023.5.7-pyhd8ed1ab_0

The following packages will be SUPERSEDED by a higher-priority channel:

  imbalanced-learn   pkgs/main/win-64::imbalanced-learn-0.~ --> conda-
forge/noarch::imbalanced-learn-0.10.1-pyhd8ed1ab_0



Downloading and Extracting Packages

imbalanced-learn-0.1 | 131 KB    |            |   0%

ca-certificates-2023 | 145 KB    |            |   0%


certifi-2023.5.7     | 149 KB    |            |   0%


certifi-2023.5.7     | 149 KB    | #          |  11%

ca-certificates-2023 | 145 KB    | #1         |  11%

ca-certificates-2023 | 145 KB    | ##2        |  22%


certifi-2023.5.7     | 149 KB    | ########## | 100%


certifi-2023.5.7     | 149 KB    | ########## | 100%

ca-certificates-2023 | 145 KB    | ########## | 100%

ca-certificates-2023 | 145 KB    | ########## | 100%
imbalanced-learn-0.1 | 131 KB    | ########## | 100%
imbalanced-learn-0.1 | 131 KB    | ########## | 100%
```

```
Preparing transaction: …working… done
Verifying transaction: …working… done
Executing transaction: …working… done
```

[2]:
```
import pandas as pd
import pandas as pd
import numpy as np
import seaborn as sns    #visualización
import matplotlib.pyplot as plt   #visualización
import scipy
import math
%matplotlib inline
sns.set(color_codes=True)

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import RobustScaler
```

1.1 1. Motivación del problema y presentación de la base de datos

1.1.1 1.1. Motivación

La base de datos propuesta es una base de datos que aporta información acerca de las diferentes características presentes en los accidentes de tráfico en Canadá entre 1999 y 2014.

Es evidente que, cada año, los accidentes de tráfico tienen un peso no menor entre las causas de mortalidad en la población. El objetivo de este trabajo (atendiendo a las sugerencias del campus) será tratar de identificar qué factores definitorios del accidente pueden tener mayor importancia a la hora de determinar la existencia o no de fallecidos en el mismo, así como qué tipo de vehículos/conductores son más o menos propensos a tener accidentes.

En el primero de los casos, por ejemplo, nuestra variable objetivo sería C_SEV, que toma el valor 1 cuando hay al menos un/a fallecido/a y el valor 2 cuando no hay ninguno/a.

Así, utiizando diferentes técnicas de aprendizaje automático, intentaremos predecir si dado un accidente, habrá o no alguna víctima mortal.

Este tipo de información puede ser útil, no sólo desde el punto de vista de la prevención y la concienciación (a nivel institucional, por ejemplo), sino también desde el punto de vista de las compañías de seguros, que necesitan clasificar a sus clientes de manera que les asignen una u otra

poliza.

1.1.2 1.2. Presentación de la base de datos

El dataset que se usará a lo largo del trabajo es un dataset que incluye variables 100% categóricas que se utilizarán para caracterizar cada accidente.

A continuación, damos una definición de las variables:

COLLISION LEVEL DATA ELEMENTS

Atributo	descripción
C_YEAR	Year
C_MNTH	Month
C_WDAY	Day of week
C_HOUR	Collision hour
C_SEV	Collision severity
C_VEHS	Number of vehicles involved
C_CONF	Collision configuration
C_RCFG	Roadway configuration
C_WTHR	Weather condition
C_RSUR	Road surface
C_RALN	Road alignment
C_TRAF	Traffic control

VEHICLE LEVEL DATA ELEMENTS

Atributo	descripción
V_ID	Vehicle sequence number
V_TYPE	Vehicle type
V_YEAR	Vehicle model year

PERSON LEVEL DATA ELEMENTS

Atributo	descripción
P_ID	Person sequence number
P_SEX	Person sex
P_AGE	Person age
P_PSN	Person position
P_ISEV	Medical treatment required
P_SAFE	Safety device used
P_USER	Road user class

La codificación detallada de las variables que se acaban de enumerar se podrá encontrar en el archivo PDF adjunto en la entrega de la tarea. No obstante, para el preprocessing con el que se empezará a elaborar el proyecto, conviene tener en cuenta la siguiente codificación:

- **“Q”,“QQ”:** Choice is other than the preceding values.
- **“U”,“UU”,“UUUU”:** Unknown.
- **“X”,“XX”,“XXXX”:** Jurisdiction does not provide this data element.
- **“N”,“NN”,“NNNN”:** Data element is not applicable. e.g. “dummy” vehicle record created for the pedestrian.

Más adelante se explicará detalladamente la importancia de estas etiquetas a la hora de eliminar observaciones o imputar missing.

1.2 2. Preprocessing

1.2.1 2.1. Análisis descriptivo del dataset. Primera parte

[3]:
```
df = pd.read_csv("NCDB_1999_to_2014.csv")
df
```

```
C:\Users\nacho\AppData\Local\Temp\ipykernel_15464\211398921.py:1: DtypeWarning:
Columns (1,2,5,12) have mixed types. Specify dtype option on import or set
low_memory=False.
  df = pd.read_csv("NCDB_1999_to_2014.csv")
```

[3]:

	C_YEAR	C_MNTH	C_WDAY	C_HOUR	C_SEV	C_VEHS	C_CONF	C_RCFG	C_WTHR	\
0	1999	1	1	20	2	02	34	UU	1	
1	1999	1	1	20	2	02	34	UU	1	
2	1999	1	1	20	2	02	34	UU	1	
3	1999	1	1	08	2	01	01	UU	5	
4	1999	1	1	08	2	01	01	UU	5	
…	…	…	…	…	…	…	…	…	…	
5860400	2014	UU	U	UU	2	UU	UU	01	U	
5860401	2014	UU	U	23	2	01	06	05	1	
5860402	2014	UU	U	14	2	01	02	01	1	
5860403	2014	UU	U	22	1	01	06	01	2	
5860404	2014	UU	U	22	1	01	06	01	2	

	C_RSUR	…	V_ID	V_TYPE	V_YEAR	P_ID	P_SEX	P_AGE	P_PSN	P_ISEV	P_SAFE	\
0	5	…	01	06	1990	01	M	41	11	1	UU	
1	5	…	02	01	1987	01	M	19	11	1	UU	
2	5	…	02	01	1987	02	F	20	13	2	02	
3	3	…	01	01	1986	01	M	46	11	1	UU	
4	3	…	99	NN	NNNN	01	M	05	99	2	UU	
…	…	…	…	…	…	…	…	…	…	…	…	
5860400	U	…	13	07	UUUU	01	M	24	11	1	NN	
5860401	1	…	1	14	2006	01	M	29	96	2	09	
5860402	5	…	1	01	2006	01	F	UU	11	2	01	
5860403	4	…	1	22	UUUU	01	M	67	12	3	01	
5860404	4	…	1	22	UUUU	02	M	10	98	1	01	

	P_USER

```
0            1
1            1
2            2
3            1
4            3
…          …
5860400      1
5860401      5
5860402      1
5860403      U
5860404      U

[5860405 rows x 22 columns]
```

Tal y como se había adelantado antes, se puede ver a simple vista en la tabla anterior que disponemos de 22 columnas (22 variables) que nos ayudarán a caracterizar nuestras colisiones. Además, podemos ver también que disponemos de 5860405 observaciones sobre las que tendremos que trabajar.

Hacemos a continuación un .info() para ver que todas las variables de las que disponemos son, en efecto, las definidas anteriormente y ver su tipo:

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5860405 entries, 0 to 5860404
Data columns (total 22 columns):
 #   Column  Dtype
---  ------  -----
 0   C_YEAR  int64
 1   C_MNTH  object
 2   C_WDAY  object
 3   C_HOUR  object
 4   C_SEV   int64
 5   C_VEHS  object
 6   C_CONF  object
 7   C_RCFG  object
 8   C_WTHR  object
 9   C_RSUR  object
 10  C_RALN  object
 11  C_TRAF  object
 12  V_ID    object
 13  V_TYPE  object
 14  V_YEAR  object
 15  P_ID    object
 16  P_SEX   object
 17  P_AGE   object
 18  P_PSN   object
```

```
 19  P_ISEV  object
 20  P_SAFE  object
 21  P_USER  object
dtypes: int64(2), object(20)
memory usage: 983.6+ MB
```

Se puede comprobar que las variables son las mismas y se puede ver también que, a excepción de *C_SEV* (número de vehículos implicados) y *C_YEAR*, son todas variables categóricas.

No obstante, tenemos en cuenta (y se puede leer en el archivo PDF adjunto) que todas las variables están codificadas de acuerdo a la leyenda referenciada, de manera que toman valores enteros (a excepción de las entradas que tomen los valores de la codificación detallada en el apartado previo, como puede ser 'Q'). Sólo hay una variable que toma valores string, *P_SEX*, que hacer referencia al sexo de la persona y toma valores 'M' para *male* y 'F' para *female*.

1.2.2 2.2. Tratamiento de entradas missing, N's, X's, U's y Q's

Una vez visualizado el dataframe, vemos si hay presencia de datos nulos con el fin de ver si, posteriormente, tenemos que hacer una imputación de los mismos:

[5]:
```
missing = df.isnull().sum()
missing
```

[5]:
```
C_YEAR    0
C_MNTH    0
C_WDAY    0
C_HOUR    0
C_SEV     0
C_VEHS    3
C_CONF    0
C_RCFG    0
C_WTHR    0
C_RSUR    0
C_RALN    0
C_TRAF    0
V_ID      0
V_TYPE    0
V_YEAR    0
P_ID      0
P_SEX     0
P_AGE     0
P_PSN     0
P_ISEV    0
P_SAFE    0
P_USER    0
dtype: int64
```

[6]:
```
datos_nulos = df['C_VEHS'].isnull()
entradas_nulas = df[datos_nulos]
```

```
entradas_nulas
```

[6]:
```
         C_YEAR C_MNTH C_WDAY C_HOUR  C_SEV C_VEHS C_CONF C_RCFG C_WTHR  \
5400115    2013      7      3     16      2    NaN     QQ     01      1
5400116    2013      7      3     16      2    NaN     QQ     01      1
5500947    2013     10      6     15      2    NaN     QQ     UU      1

         C_RSUR  … V_ID V_TYPE V_YEAR P_ID P_SEX P_AGE P_PSN P_ISEV P_SAFE  \
5400115      1  …   99     NN   NNNN   01     M    19    99      2     NN
5400116      1  …   99     NN   NNNN   02     M    18    99      1     NN
5500947      Q  …   99     NN   NNNN   01     M    54    99      2     NN

         P_USER
5400115      3
5400116      3
5500947      3

[3 rows x 22 columns]
```

Se puede ver en las dos celdas previas que hay tres entradas nulas para tres de las observaciones en la variable *C_VEHS*. Podríamos tratar de imputar estas entradas pensando que son las únicas nulas, pero es aquí donde entra en juego la codificación que decíamos previamente en el apartado 1.2. que debería de ser tenida en cuenta, por lo que deberemos seguir varios pasos y tener en cuenta diferetes cosas:

- En **primer lugar**, las etiquetas 'X', 'XX', 'XXXX', 'U', 'UU' y 'UUUU' hacen referencia a entradas para las que no se conoce el dato de la variable en cuestión, por un motivo cualquiera. Podríamos decir que son etiquetas definidas de manera que hacen referencia a datos nulos (missing), por lo que las entradas que tomen estos valores serán tratados como tal.

En este caso, las variables que pueden tener entradas etiquetadas de esta forma son: todas a excepción de *C_YEAR*.

- En **segundo lugar**, las entradas que toman el valor 'N', 'NN' y 'NNNN' para alguna de las características, son entradas a las que no se les puede aplicar ninguna de las categorías de esa característica. Creemos que lo más lógico será eliminar estas observaciones, puesto que no aportan información relevante para algunas de las variables e imputar estos valores por la mediana sería hacerse trampas al solitario (de entrada sabemos que no son aplicables). Eso sí, deberemos comprobar que su eliminación no afecta en demasía a la dimensión del dataframe.

Tenemos también en cuenta que las variables que presentan valores 'N', 'NN' o 'NNNN' en algunas de sus entradas son: *V_TYPE*, *P_ID*, *P_SEX*, *P_AGE*, *P_PSN*, *P_ISEV* y *P_SAFE*

- Por **último**, las codificaciones 'Q', 'QQ', 'QQQQ' son usadas para hacer referencia a las veces en las que la respuesta a la variable en cuestión no se encuentra entre las opciones que se dan para esa variable. En vista a esto, creemos que lo más razonable sería también eliminar estas observaciones, pues imputarles la mediana sería distorsionar la información que realmente aportan. Otra opción posible, en caso de que se valore como muy drástica la eliminación de estas variables, será crear una nueva categoría que represente algo así como "otro" y que esté codificada con un nuevo valor numérico.

En este último caso, las variables que pueden tener entradas etiquetadas de esta forma son: *C_CONF*, *C_RCFG*, *C_WTHR*, *C_RSUR*, *C_RALN*, *C_TRAF*, *V_TYPE*, *P_PSN* y *P_SAFE*

2.2.1 Entradas X's y U's

[7]:
```
nulos = ["X", "XX", "XXXX", "U", "UU", "UUUU"]
#cargamos de nuevo el df en una copia (para mantener intacto el df original)␣
 ↪teniendo en cuenta que las entradas X's y U's
#representan valores nulos
df_c = pd.read_csv("NCDB_1999_to_2014.csv", na_values=nulos)
```

[8]:
```
missing = df_c.isnull().sum()
missing
```

[8]:
```
C_YEAR         0
C_MNTH       385
C_WDAY      1323
C_HOUR     59409
C_SEV          0
C_VEHS       544
C_CONF    179019
C_RCFG    504648
C_WTHR     87975
C_RSUR     78451
C_RALN    434710
C_TRAF    223483
V_ID         433
V_TYPE     29754
V_YEAR    324122
P_ID          16
P_SEX     234954
P_AGE     377140
P_PSN      57662
P_ISEV     80701
P_SAFE    615188
P_USER    175586
dtype: int64
```

Nótese que, haciendo esto, nada ha cambiado con respecto al dataframe anterior. Únicamente ocurre que ahora tenemos el df de manera que los valores nulos están bien cargados (Se mantienen los 3 que ya había y se añaden los nuevos).

2.2.2 Entradas N's Como ya se ha adelantado antes, en este apartado veremos si es factible hacer lo que nos pide nuestra intuición: eliminar las observaciones que tomen valores N en sus variables

[9]:
```
```

```
valores_N = ['N', 'NN', 'NNNN']
df_sinN = df_c.drop(df_c[df_c.apply(lambda x: x.isin(valores_N)).any(axis=1)].
  ↪index)
```

[10]:
```
df_sinN.shape
```

[10]:
```
(5116377, 22)
```

Se puede ver que hemos pasado de tener 5860405 observaciones a tener 5116377, es decir, hemos perdido un 12'69584% de las observaciones de la muestra:

[11]:
```
(5860405-5116377)/5860405
```

[11]:
```
0.12695846106199146
```

Este valor podría ser, en términos absolutos, un valor más que notable. Sin embargo, en términos relativos, teniendo en cuenta que estamos trabajando con una base de datos de cerca de 6 millones de observaciones, creemos que es un valor asumible.

Sin embargo, esta decisión puede ser aún desafortunada. En el primer apartado del proyecto, coméntabamos que, por ejemplo, si el objetivo es estimar si habrá fallecidos o no (atendiendo a los requirimientos del enunciado de la tarea), nuestra variable objetivo sería *C_SEV*. ¿Cómo de balanceada queda esta variable tras la eliminación de las "observaciones N"?:

[12]:
```
#Vemos el balanceo antes de eliminar las "observaciones N"
conteo = df_c['C_SEV'].value_counts()

#Vemos el balanceo después de eliminar las "observaciones N"
conteo_sinN = df_sinN['C_SEV'].value_counts()
```

[13]:
```
conteo[1]/df_c.shape[0], conteo_sinN[1]/df_sinN.shape[0]
```

[13]:
```
(0.0168304067722282, 0.01687619970146844)
```

El balanceo permanece prácticamente intacto, por lo que consideramos que la eliminación de las denominadas "observaciones N" no afecta al df en este sentido. Así, actualizamos nuestro df_c:

[14]:
```
df_c = df_sinN
```

2.2.3 Entradas Q's Lo primero que haremos será analizar qué supone para la dimensionalidad de nuestro dataframe la eliminación de estas variables:

[15]:
```
valores_Q = ['Q', 'QQ', 'QQQQ']
df_sinQ = df_c.drop(df_c[df_c.apply(lambda x: x.isin(valores_Q)).any(axis=1)].
  ↪index)
```

[16]:
```
df_sinQ.shape
```

[16]: (4487762, 22)

Se puede ver que hemos pasado de tener 5116377 observaciones a tener 4487762, es decir, hemos perdido un 12'69584% de las observaciones de la muestra:

[17]:
```
(5116377-4487762)/5116377
```

[17]: 0.122863307375512

Eliminar las "observacionesQ", supondría volver a eliminar otro 12'28% sobre el 12'7% previo... Esto supone eliminar en torno a un 23% de las observaciones originales (más de 1 millon de observaciones!!!!)... Aunque pueda llegar a ser asumible, preferimos no arriesgarnos a comprometer el rendimiento de nuestro modelo y conservar así las "observacionesQ". Eso sí, para que sea más sencilla la posterior imputación de missing, le daremos una etiqueta numérica a esta categoría.

Tras analizar el PDF en el que se expone la leyenda de las variables, decidimos que las entradas Q, QQ, QQQQ sean sustituidas por un 100, pasando así el valor 100 a representar esta categoría.

[18]:
```
df_c = df_c.replace(['Q', 'QQ', 'QQQQ'], '100')
```

[19]:
```
df_c
```

[19]:

	C_YEAR	C_MNTH	C_WDAY	C_HOUR	C_SEV	C_VEHS	C_CONF	C_RCFG	C_WTHR	\
0	1999	1.0	1.0	20.0	2	2.0	34	NaN	1	
1	1999	1.0	1.0	20.0	2	2.0	34	NaN	1	
2	1999	1.0	1.0	20.0	2	2.0	34	NaN	1	
3	1999	1.0	1.0	8.0	2	1.0	01	NaN	5	
5	1999	1.0	1.0	17.0	2	3.0	100	100	1	
…	…	…	…	…	…	…	…	…	…	
5860380	2014	12.0	7.0	19.0	2	1.0	03	01	1	
5860401	2014	NaN	NaN	23.0	2	1.0	06	05	1	
5860402	2014	NaN	NaN	14.0	2	1.0	02	01	1	
5860403	2014	NaN	NaN	22.0	1	1.0	06	01	2	
5860404	2014	NaN	NaN	22.0	1	1.0	06	01	2	

	C_RSUR	…	V_ID	V_TYPE	V_YEAR	P_ID	P_SEX	P_AGE	P_PSN	P_ISEV	P_SAFE	\
0	5	…	1.0	06	1990	01	M	41	11	1	NaN	
1	5	…	2.0	01	1987	01	M	19	11	1	NaN	
2	5	…	2.0	01	1987	02	F	20	13	2	02	
3	3	…	1.0	01	1986	01	M	46	11	1	NaN	
5	2	…	1.0	01	1984	01	M	28	11	1	NaN	
…	…	…	…	…	…	…	…	…	…	…	…	
5860380	3	…	1.0	01	2001	03	F	15	21	2	02	
5860401	1	…	1.0	14	2006	01	M	29	96	2	09	
5860402	5	…	1.0	01	2006	01	F	NaN	11	2	01	
5860403	4	…	1.0	22	NaN	01	M	67	12	3	01	
5860404	4	…	1.0	22	NaN	02	M	10	98	1	01	

```
         P_USER
0           1.0
1           1.0
2           2.0
3           1.0
5           1.0
…           …
5860380     2.0
5860401     5.0
5860402     1.0
5860403     NaN
5860404     NaN

[5116377 rows x 22 columns]
```

2.2.4 Imputación de missing En primer lugar, se aprecia que todas las variables (que son categóricas) están codificadas de manera que toman valores numéricos enteros (aunque a veces se expresan en formato float). La única variable que no está codificada de esta manera es *P_SEX*, que toma valores 'M' para indicar male y valores 'F' para indicar female. Al tener todas las variables codificadas con números, será más sencillo ajustar los modelos.

[20]:
```
df_c['P_SEX'] = df['P_SEX'].map({'M': 1, 'F': 0})
```

Antes de imputar missing, vemos un poco cuántos missing nos han quedado por variable tras la eliminación de las "observaciones n" y qué porcentaje sobre el total suponen. También se puede hacer lo mismo por observaciones. De esta manera, vemos cómo están distribuidos los missing, lo que nos puede ayudar a tomar decisiones acerca de su imputación o, en caso de que una observación presente missing para casi todas las variables, la eliminación de esa observación.

[21]:
```
# Por variables
conteo_col = df_c.isnull().sum()
porcentaje_col = (conteo_col / df_c.shape[0]) * 100

# Tabla resumen por columnas
tabla_col = pd.DataFrame({'Conteo Variables': conteo_col, 'Porcentaje':␣
 ↪porcentaje_col})
tabla_col
```

[21]:
```
         Conteo Variables  Porcentaje
C_YEAR                  0    0.000000
C_MNTH                177    0.003459
C_WDAY               1026    0.020053
C_HOUR              50057    0.978368
C_SEV                   0    0.000000
C_VEHS                 13    0.000254
C_CONF             156347    3.055815
C_RCFG             442980    8.658080
```

C_WTHR	79953	1.562688
C_RSUR	69816	1.364559
C_RALN	412828	8.068756
C_TRAF	177137	3.462157
V_ID	351	0.006860
V_TYPE	17325	0.338619
V_YEAR	239931	4.689471
P_ID	2	0.000039
P_SEX	151806	2.967060
P_AGE	262397	5.128570
P_PSN	56086	1.096205
P_ISEV	79849	1.560655
P_SAFE	548337	10.717291
P_USER	117370	2.294006

En el caso de la distribución de missing por variables, se puede observar que únicamente dos de las variables no incluyen missing: *C_YEAR* y *C_SEV* (nuestra variable objetivo). La existencia de missing en otras variables es prácticamente insignificante. Es el caso de *P_ID*, *C_VEHS* e incluso *C_MNTH*, *V_ID*, teniendo en cuenta que tenemos más de 5 millones de datos. Finalmente, encontramos otras variables que sí presentan un porcentaje mucho más elevado de missing, como es el caso de *P_SAFE* (10%).

A continuación, vemos la distribución por observaciones. Obviamente, no es interesante comentar todas las observaciones (ni siquiera muchas), pero sí comentaremos algunas de las que presentan mayor porcentaje de missing a lo largo de las variables:

[25]:
```
# Por filas
conteo_row = df_c.isnull().sum(axis=1).sort_values(ascending=False)
porcentaje_row = (conteo_row / df_c.shape[1]) * 100

# Tabla resumen por filas
tabla_row = pd.DataFrame({'Conteo por filas': conteo_row, 'Porcentaje':␣
 ↪porcentaje_row})
tabla_row.head(5)
```

[25]:

	Conteo por filas	Porcentaje
5245711	16	72.727273
5245712	16	72.727273
2626119	13	59.090909
170432	13	59.090909
96990	13	59.090909

Se puede observar que hay dos observaciones que presentan datos faltantes hasta en 16 de las 22 variables. A partir de ahí, las siguientes que más presentan lo hacen por una cantidad de 13 (59%). Es cierto que un 59% podría ser considerado bastante, pero lo que sí que está claro es que imputar missing en dos observaciones que tienen 16 missing de 20 valores, no parece tener muchísimo sentido.

Para no distorsionar, eliminamos estas dos observaciones.

[26]:
```
df_c = df_c.drop(5245711)
df_c = df_c.drop(5245712)
df_c.shape
```

[26]:
```
(5116375, 22)
```

A continuación, procedemos a imputar los datos missing por la moda:

[27]:
```
#Obtengo la moda de cada variable
moda_col = df_c.mode().iloc[0]

# Imputamos
for columna in df_c.columns:
    df_c[columna] = df_c[columna].fillna(moda_col[columna])
```

[28]:
```
df_c.isnull().sum()
```

[28]:
```
C_YEAR    0
C_MNTH    0
C_WDAY    0
C_HOUR    0
C_SEV     0
C_VEHS    0
C_CONF    0
C_RCFG    0
C_WTHR    0
C_RSUR    0
C_RALN    0
C_TRAF    0
V_ID      0
V_TYPE    0
V_YEAR    0
P_ID      0
P_SEX     0
P_AGE     0
P_PSN     0
P_ISEV    0
P_SAFE    0
P_USER    0
dtype: int64
```

Como se puede comprobar en la salida previa, ya no tenemos datos missing! Genial. **NOTA**: Hemos imputado por la moda porque, si lo hubiésemos hecho por la mediana, podríamos haber tenido algún problema para alguna de las variables en el sentido de que la imputación no fuese un entero que atendiese a la codificación original (**nótese** que podría provenir del promedio de dos etiquetas numéricas distintas en una situación límite con un número par de observaciones).

1.2.3 2.3. Análisis descriptivo. Segunda parte

2.3.1. Sobre la interacción entre variables en su conjunto. En este apartado, trataremos de estudiar la correlación presente entre las diferentes variables y la que hemos dicho que será nuestra variable objetivo, *C_SEV*. Para ello, empezaremos también a descartar alguna variable que consideremos que, independientemente de la correlación con la variable objetivo, podría no ser interesante para nuestro estudio.

NÓTESE: Un ejemplo, podría ser el caso de variables como la fecha del accidente *C_YEAR*. Obviamente, parece razonable pensar que a medida que pasan los años, la existencia de muertes en colisiones debería seguir una tendencia decreciente. Sin embargo, esto no nos aporta información porque lo que realmente hace que baje la mortalidad en los accidentes son **las mejoras** de las condiciones (de las carreteras y de los vehículos, por ejemplo), que obviamente se desarrollan con **el paso del tiempo**. Por esto decimos que la variable *C_YEAR* por sí misma, no sería realmente relevante.

[29]:
```
#recordamos cuáles eran nuestras variables
df_c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5116375 entries, 0 to 5860404
Data columns (total 22 columns):
 #   Column  Dtype
---  ------  -----
 0   C_YEAR  int64
 1   C_MNTH  float64
 2   C_WDAY  float64
 3   C_HOUR  float64
 4   C_SEV   int64
 5   C_VEHS  float64
 6   C_CONF  object
 7   C_RCFG  object
 8   C_WTHR  object
 9   C_RSUR  object
 10  C_RALN  object
 11  C_TRAF  object
 12  V_ID    float64
 13  V_TYPE  object
 14  V_YEAR  object
 15  P_ID    object
 16  P_SEX   float64
 17  P_AGE   object
 18  P_PSN   object
 19  P_ISEV  object
 20  P_SAFE  object
 21  P_USER  float64
dtypes: float64(7), int64(2), object(13)
memory usage: 897.8+ MB
```

En la salida anterior, y atendiendo a la definición y a la leyenda de las variables, se puede afirmar que todas las variables pueden ser, a priori, de interés para nuestro análisis, a excepción de *C_YEAR* y las variables identificativas: *V_ID* y *P_ID*.

Procedemos a la eliminación de estas variables para que no ensucien nuestra matriz de correlaciones y nuestra tabla de estadísticos posteriormente:

[30]:
```
elim = ['C_YEAR', 'V_ID', 'P_ID']
df_c = df_c.drop(elim, axis=1)
```

[31]:
```
df_c
```

[31]:

	C_MNTH	C_WDAY	C_HOUR	C_SEV	C_VEHS	C_CONF	C_RCFG	C_WTHR	C_RSUR	\
0	1.0	1.0	20.0	2	2.0	34	02	1	5	
1	1.0	1.0	20.0	2	2.0	34	02	1	5	
2	1.0	1.0	20.0	2	2.0	34	02	1	5	
3	1.0	1.0	8.0	2	1.0	01	02	5	3	
5	1.0	1.0	17.0	2	3.0	100	100	1	2	
…	…	…	…	…	…	…	…	…	…	
5860380	12.0	7.0	19.0	2	1.0	03	01	1	3	
5860401	8.0	5.0	23.0	2	1.0	06	05	1	1	
5860402	8.0	5.0	14.0	2	1.0	02	01	1	5	
5860403	8.0	5.0	22.0	1	1.0	06	01	2	4	
5860404	8.0	5.0	22.0	1	1.0	06	01	2	4	

	C_RALN	C_TRAF	V_TYPE	V_YEAR	P_SEX	P_AGE	P_PSN	P_ISEV	P_SAFE	P_USER
0	3	03	06	1990	1.0	41	11	1	02	1.0
1	3	03	01	1987	1.0	19	11	1	02	1.0
2	3	03	01	1987	0.0	20	13	2	02	2.0
3	6	18	01	1986	1.0	46	11	1	02	1.0
5	1	01	01	1984	1.0	28	11	1	02	1.0
…	…	…	…	…	…	…	…	…	…	…
5860380	4	18	01	2001	0.0	15	21	2	02	2.0
5860401	1	18	14	2006	1.0	29	96	2	09	5.0
5860402	4	18	01	2006	0.0	18	11	2	01	1.0
5860403	100	18	22	2000	1.0	67	12	3	01	1.0
5860404	100	18	22	2000	1.0	10	98	1	01	1.0

[5116375 rows x 19 columns]

Una vez hecho lo previo, miraremos la tabla de estadísticos descriptivos. ¿Deberíamos o tendría sentido estandarizar los datos para alguno de los algoritmos que se implementarán más adelante?

Convertimos todas nuestras variables a tipo entero (o float, da lo mismo) para poder sacar la tabla:

[32]:
```
df_c = df_c.astype(int)
```

[33]:
```
df_c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5116375 entries, 0 to 5860404
Data columns (total 19 columns):
 #   Column  Dtype
---  ------  -----
 0   C_MNTH  int32
 1   C_WDAY  int32
 2   C_HOUR  int32
 3   C_SEV   int32
 4   C_VEHS  int32
 5   C_CONF  int32
 6   C_RCFG  int32
 7   C_WTHR  int32
 8   C_RSUR  int32
 9   C_RALN  int32
 10  C_TRAF  int32
 11  V_TYPE  int32
 12  V_YEAR  int32
 13  P_SEX   int32
 14  P_AGE   int32
 15  P_PSN   int32
 16  P_ISEV  int32
 17  P_SAFE  int32
 18  P_USER  int32
dtypes: int32(19)
memory usage: 409.9 MB
```

También, antes de continuar, le prestaremos un poquito de atención, por primera vez, a nuestra variable objetivo. *C_SEV* es una variable dicotómica, que toma el valor '1' en caso de que el accidente haya provocado al menos un fallecimiento, y el valor '2' en caso contrario. Dicho esto, parece más natural que, al ser dicotómica, la variable tome los valores 0 y 1 (en el caso de reg. logística, por ejemplo, lo parece). Transformaremos la codificación de manera que los 2, pasen a ser 0. Los 1 se quedarán como están.

[34]:
```
df_c['C_SEV'] = df_c['C_SEV'].replace(2, 0)
df_c
```

[34]:

	C_MNTH	C_WDAY	C_HOUR	C_SEV	C_VEHS	C_CONF	C_RCFG	C_WTHR	\
0	1	1	20	0	2	34	2	1	
1	1	1	20	0	2	34	2	1	
2	1	1	20	0	2	34	2	1	
3	1	1	8	0	1	1	2	5	
5	1	1	17	0	3	100	100	1	
…	…	…	…	…	…	…	…	…	
5860380	12	7	19	0	1	3	1	1	
5860401	8	5	23	0	1	6	5	1	
5860402	8	5	14	0	1	2	1	1	
5860403	8	5	22	1	1	6	1	2	

```
5860404       8       5      22       1       1       6       1       2

         C_RSUR  C_RALN  C_TRAF  V_TYPE  V_YEAR  P_SEX  P_AGE  P_PSN  P_ISEV  \
0             5       3       3       6    1990      1     41     11       1
1             5       3       3       1    1987      1     19     11       1
2             5       3       3       1    1987      0     20     13       2
3             3       6      18       1    1986      1     46     11       1
5             2       1       1       1    1984      1     28     11       1
…           …       …       …       …    …       …      …      …
5860380       3       4      18       1    2001      0     15     21       2
5860401       1       1      18      14    2006      1     29     96       2
5860402       5       4      18       1    2006      0     18     11       2
5860403       4     100      18      22    2000      1     67     12       3
5860404       4     100      18      22    2000      1     10     98       1

         P_SAFE  P_USER
0             2       1
1             2       1
2             2       2
3             2       1
5             2       1
…           …       …
5860380       2       2
5860401       9       5
5860402       1       1
5860403       1       1
5860404       1       1

[5116375 rows x 19 columns]
```

Ahora, sí, sacamos la tabla de estadísticos descriptivos:

[35]:
```
df_c.describe()
```

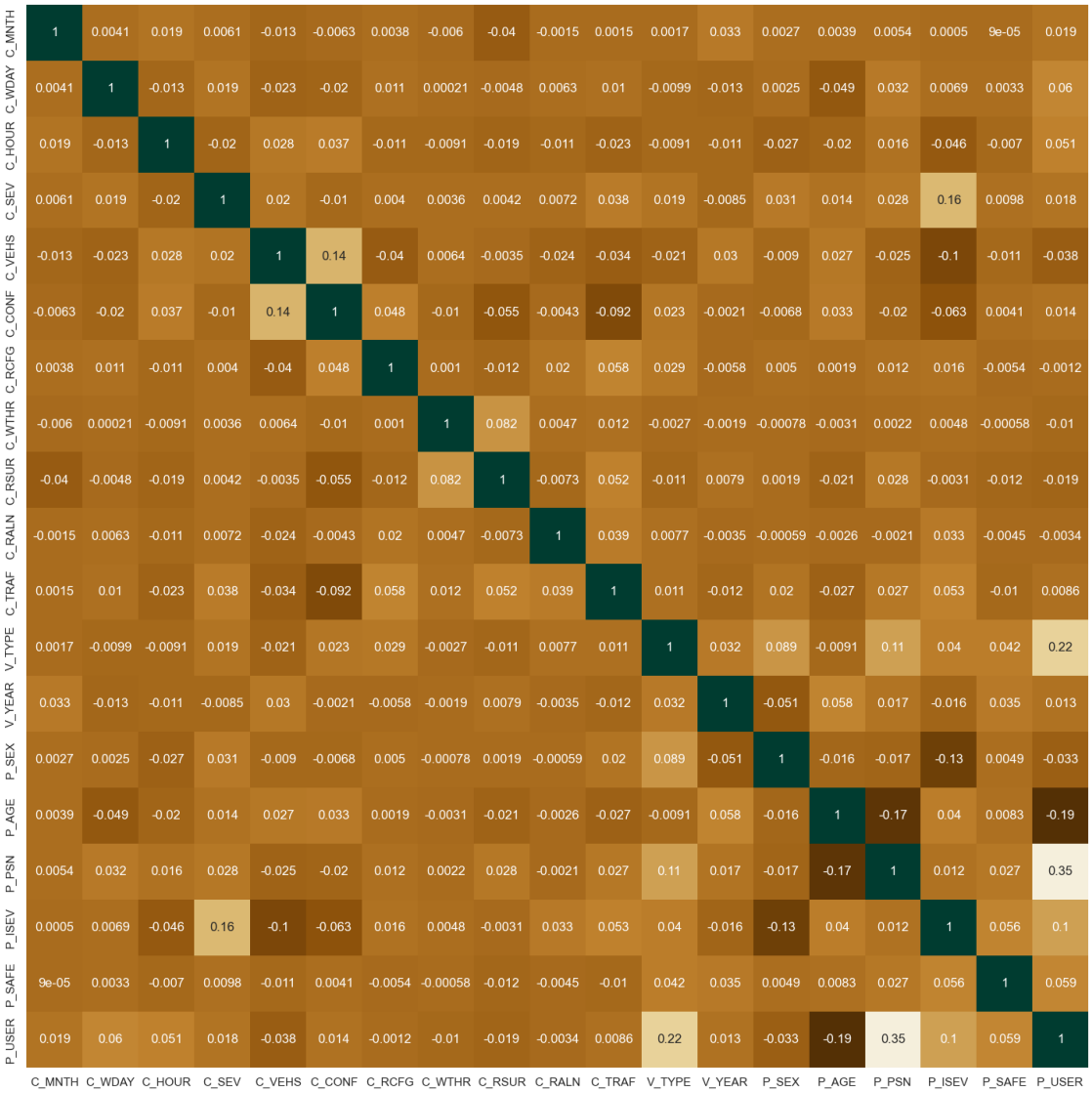
[35]:
```
              C_MNTH        C_WDAY        C_HOUR         C_SEV        C_VEHS  \
count   5.116375e+06  5.116375e+06  5.116375e+06  5.116375e+06  5.116375e+06
mean    6.705003e+00  4.022633e+00  1.370618e+01  1.687582e-02  2.067043e+00
std     3.445281e+00  1.937111e+00  5.182559e+00  1.288062e-01  1.211466e+00
min     1.000000e+00  1.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00
25%     4.000000e+00  2.000000e+00  1.000000e+01  0.000000e+00  2.000000e+00
50%     7.000000e+00  4.000000e+00  1.500000e+01  0.000000e+00  2.000000e+00
75%     1.000000e+01  6.000000e+00  1.700000e+01  0.000000e+00  2.000000e+00
max     1.200000e+01  7.000000e+00  2.300000e+01  1.000000e+00  7.700000e+01

              C_CONF        C_RCFG        C_WTHR        C_RSUR        C_RALN  \
count   5.116375e+06  5.116375e+06  5.116375e+06  5.116375e+06  5.116375e+06
mean    2.647397e+01  3.833888e+00  1.862653e+00  4.740066e+00  1.739233e+00
```

```
std    1.908507e+01  1.421172e+01  5.181282e+00  1.736540e+01  5.863605e+00
min    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
25%    2.100000e+01  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
50%    2.100000e+01  2.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
75%    3.500000e+01  2.000000e+00  2.000000e+00  2.000000e+00  1.000000e+00
max    1.000000e+02  1.000000e+02  1.000000e+02  1.000000e+02  1.000000e+02

              C_TRAF        V_TYPE        V_YEAR         P_SEX         P_AGE  \
count  5.116375e+06  5.116375e+06  5.116375e+06  5.116375e+06  5.116375e+06
mean   1.236341e+01  2.467674e+00  1.998525e+03  5.647416e-01  3.522479e+01
std    1.346777e+01  7.134091e+00  6.992480e+00  4.957909e-01  1.828744e+01
min    1.000000e+00  1.000000e+00  1.901000e+03  0.000000e+00  1.000000e+00
25%    1.000000e+00  1.000000e+00  1.994000e+03  0.000000e+00  2.000000e+01
50%    1.800000e+01  1.000000e+00  1.999000e+03  1.000000e+00  3.200000e+01
75%    1.800000e+01  1.000000e+00  2.003000e+03  1.000000e+00  4.800000e+01
max    1.000000e+02  1.000000e+02  2.015000e+03  1.000000e+00  9.900000e+01

              P_PSN        P_ISEV        P_SAFE        P_USER
count  5.116375e+06  5.116375e+06  5.116375e+06  5.116375e+06
mean   1.358750e+01  1.575275e+00  3.195875e+00  1.416534e+00
std    9.135078e+00  5.074589e-01  9.416804e+00  7.304997e-01
min    1.100000e+01  1.000000e+00  1.000000e+00  1.000000e+00
25%    1.100000e+01  1.000000e+00  2.000000e+00  1.000000e+00
50%    1.100000e+01  2.000000e+00  2.000000e+00  1.000000e+00
75%    1.300000e+01  2.000000e+00  2.000000e+00  2.000000e+00
max    1.000000e+02  3.000000e+00  1.000000e+02  5.000000e+00
```

Obviamente, información como la media no nos es de gran interés en este caso de variables categóricas etiquetadas/codificadas por enteros. Sin embargo, sí podemos intuir la distribución de alguna de las variables si atendemos a los cuartiles (más adelante, visualizaremos alguna que nos sea de interés). Si miramos los rangos en los que se mueven las variables, sí que notamos grandes diferencias. Esto es algo que ya se podía ver si se leía con detalle la leyenda de codificación. Al ser tan dispar el rango en el que se mueven alguna de las variables, se estandarizará en caso de ser necesario para el algoritmo que lo requiera.

A continuación, atenderemos a las correlaciones entre las diferentes variables:

[36]:
```
plt.figure(figsize=(18, 18))
corr = df_c.corr()
sns.heatmap(corr, cmap="BrBG", annot=True, square=True, cbar=False)
plt.show()
```



De un primer vistazo, podríamos sacar una primera afirmación: El total de las correlaciones son mayoritariamente muy bajas en valor absoluto. El signo de las mismas es bastante variado. Las correlaciones negativas, en valor absoluto no superan el 0.2, mientras que, por su parte, las positivas, son también bajas. Sólo encontramos, destacadamente por encima del 0.1, un 0.14, un 0.16, un 0.22 y un 0.35. (algún valor como 0.11 se puede encontrar también).

Ninguna de las correlaciones es superior siquiera al 0.5.

Si tratamos de explicar alguna de las correlaciones más altas:

- Encontramos un **0.14 de correlación entre** las variables ***C_VEHS* y *C_CONF***: Si miramos a la leyenda y a la definición de las variables, vemos que es razonable que la correlación no sea tan baja como la que hay entre la mayoría de variables (de hecho, podríamos pensar que podría ser más alta). *C_VEHS* nos da información sobre el número de vehículos involucrados en el accidente, mientras que *C_CONF* nos da información sobre la manera en la que

se produce un accidente: por ejemplo, si el accidente se da con un sólo vehículo, se pueden distinguir los casos en los que el vehículo colisiona con un objeto en movimiento, de los que el vehículo colisiona con un objeto que no se encuentre en movimiento, o simplemente de los casos en los que en el accidente no se produce colisión con ningún objeto externo (i.e. con 2 o más vehículos involucrados). Sí es cierto que, como para cada número de vehículos involucrados hay diferentes opciones de configuración, se puede perder correlación y, por ello, ser más baja de lo esperado.

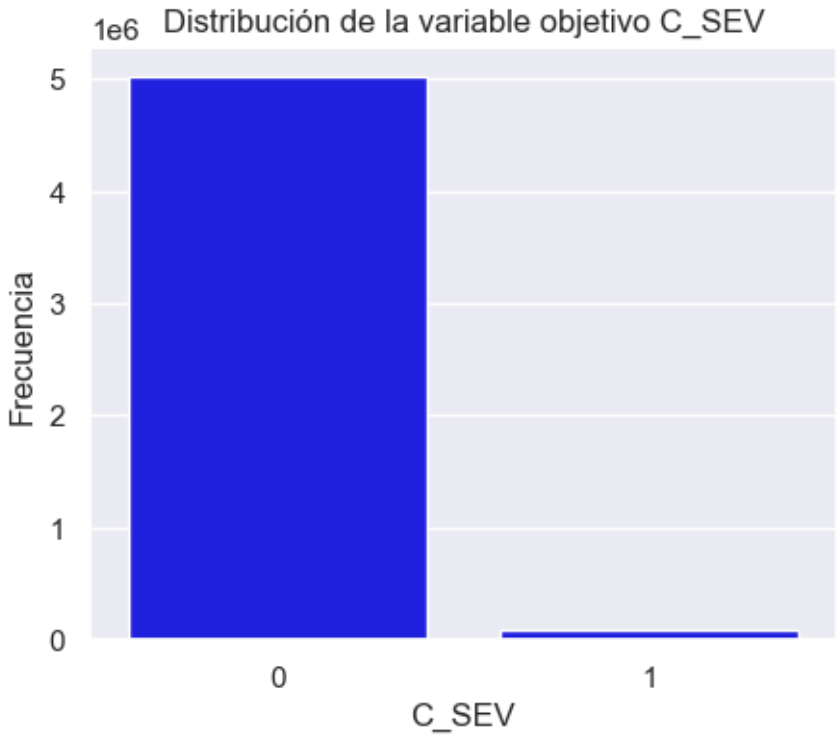
- En segundo lugar, existe un **0.16 de correlación entre *C_SEV***, nuestra variable objetivo, y ***P_ISEV***: Esta segunda, nos indica el tratamiento médico requerido tras el accidente y puede indicar lesión/daño, ausencia de lesión/daño y fallecimiento en el acto. Teniendo en cuenta que nuestra variable objetivo nos revela si las consecuencias finales del accidente son fallecimientos o no fallecimientos, es razonable que haya esa correlación.
- Por otro lado, hallamos una correlación del **0.22 entre *V_TYPE* y *P_USER***: La primera nos da información sobre el tipo de vehículo, mientras que la segunda, sobre el tipo de usuario de la vía (conductor de vehículo de motor, pasajero de vehículo de motor, peatón, ciclista…). Lógicamente, esto tiene bastante que ver con el tipo de vehículo implicado en el accidente.
- En el último caso de las **correlaciones positivas**, hablamos del **0.35 de correlación entre *P_PSN* y *P_USER***: La primera de las variables nos da información sobre la posición ocupada por la persona en el vehículo en el momento del accidente. Teniendo en cuenta que *P_USER* da información sobre lo indicado en el párrafo previo, podemos darle fácil explicación a la que es la mayor correlación en valor absoluto.
- Cambiando de tercio y hablando de las **correlaciones negativas**, podemos destacar, en primer lugar, el **-0.19 entre *P_USER* y *P_AGE***: En primer lugar, **¿por qué un signo negativo?** Pues bien, parece que el signo negativo puede deberse a la manera de codificar estas variables. *P_AGE* crece con la edad (nos muestra la edad del usuario de la vía implicado en el accidente). Por su parte, *P_USER* etiqueta con un 1 a los conductores de vehículos de motor, con un 2 a pasajeros, con un 3 a peatones… No necesariamente, pero la mayoría de los conductores son mayores que pasajeros o peatones (véase niños o gente que aún no está en edad de conducir). Obviamente, hay "peros" y esto debe ser cogido con pinzas, (veánse personas muy mayores que no conduzcan)…
- Una explicación similar se puede dar para el **-0.17 de correlación** que se da entre ***P_AGE* y *P_PSN***.
- Algo menos explicable puede parecer el **-0.13 entre *P_SEX* y *P_ISEV***.

Por último, y por señalar algo en lo que incidiremos más adelante, vemos que **todas** las correlaciones que presenta nuestra variable objetivo, *C_SEV*, con el resto de variables, son muy bajas. La más alta de ellas es la que ya hemos señalado con *P_ISEV*. Obviamente, aunque quizás podría interesarnos que fuesen más altas, esto no tiene que significar necesariamente que nuestras características no tengan ninguna incidencia sobre la variable objetivo.

De hecho, teniendo en cuenta que las variables son categóricas codificadas mediante números enteros, puede ocurrir perfectamente que la manera de codificarlas distorsione la verdadera correlación existente entre las diferentes variables.

2.3.2. Sobre alguna variable de interés En este apartado, empezaremos hablando de la variable que será más importante para nosotros: nuestra **variable objetivo**. En el subapartado previo, ya hemos transformado la codificación de la variable ***C_SEV***. A continuación, realizaremos un análisis descriptivo de su distribución y de su interacción con otras variables del df.

[37]:
```
#En primer lugar, graficamos la distribución de la variable C_SEV:
plt.figure(figsize=(5, 4))
sns.countplot(x='C_SEV', data=df_c, color ='blue')

plt.xlabel('C_SEV')
plt.ylabel('Frecuencia')
plt.title('Distribución de la variable objetivo C_SEV')

plt.show()
```



En la salida previa se puede visualizar a simple vista lo desbalanceada que está la base de datos en términos de nuestra variable objetivo. Esto es un **problema claro** que debemos de tratar para que nuestros modelos de predicción no estén sesgados hacia la clase mayoritaria.

Lo vemos numéricamente en la siguiente salida. En efecto, 5030032 del total de accidentes no terminan en fallecimiento, mientras que **sólo** 86343 sí lo hacen:

[38]:
```
df_c['C_SEV'].value_counts()
```

[38]:
```
0    5030032
1      86343
Name: C_SEV, dtype: int64
```

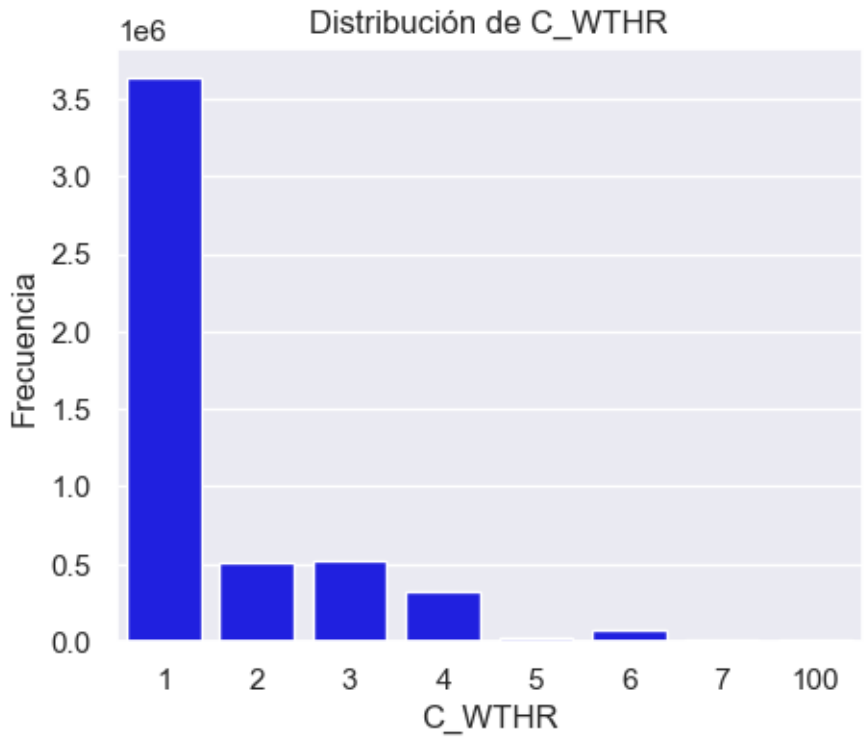
El balanceo de nuestra variable objetivo se realizará después de la selección de características. Entre tanto, vamos a continuar con el análisis sobre algunas de las variables para conocer también su distribución y ver si lo obtenido coincide con lo que podemos esperar de cada una de estas variables.

Comenzamos por ***C_WTHR***: Si acudimos a la leyenda, podemos ver que *C_WTHR* toma el valor 1 para indicar buen tiempo (claro y soleado), el valor '2', para nublado sin precipitación, el valor '3' para lluvioso… Mirando el diagrama, vemos que la mayor de las frecuencias absolutas en esta variable ocurre para el valor '1', es decir, la mayoría de los accidentes registrados en este dataframe ocurren en días de buen tiempo.

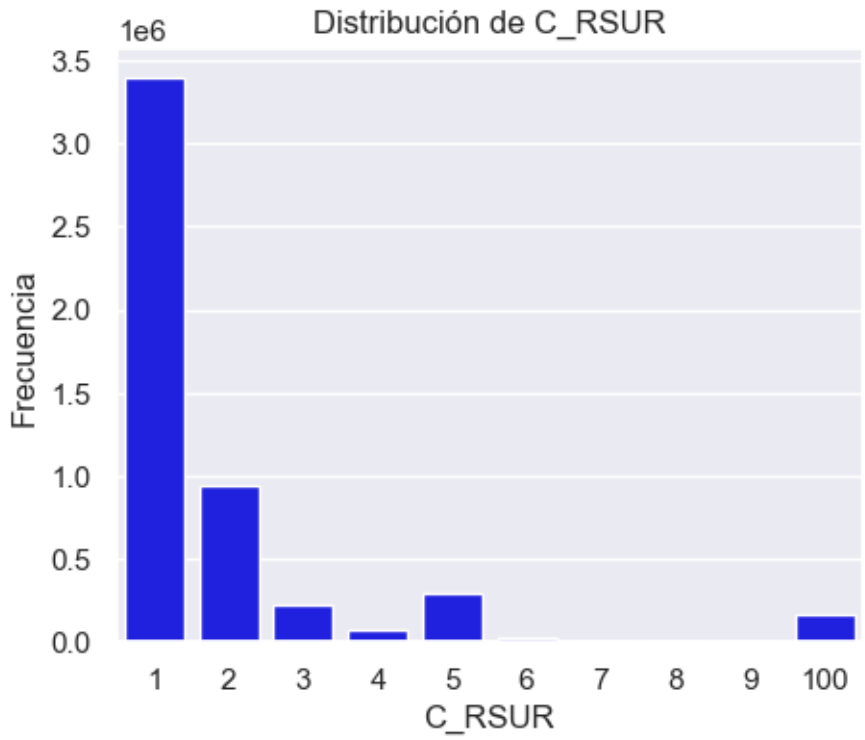
Si juntamos los valores '1' y '2', que denotan que no llovía en el momento del accidente, la diferencia que se da con los accidentes que ocurren con lluvía es aún más aplastante. Esto no contradice necesariamente la intuición de que con lluvia es más peligroso conducir: **NOTA** que podría significar simplemente que los momentos de lluvia son muchísimos menos que los momentos de no-precipitación.

Algo similar ocurre si hacemos el histograma para *C_RSUR*

[39]:
```
plt.figure(figsize=(5, 4))
sns.countplot(x='C_WTHR', data=df_c, color = 'blue')

plt.xlabel('C_WTHR')
plt.ylabel('Frecuencia')
plt.title('Distribución de C_WTHR')

plt.show()
```

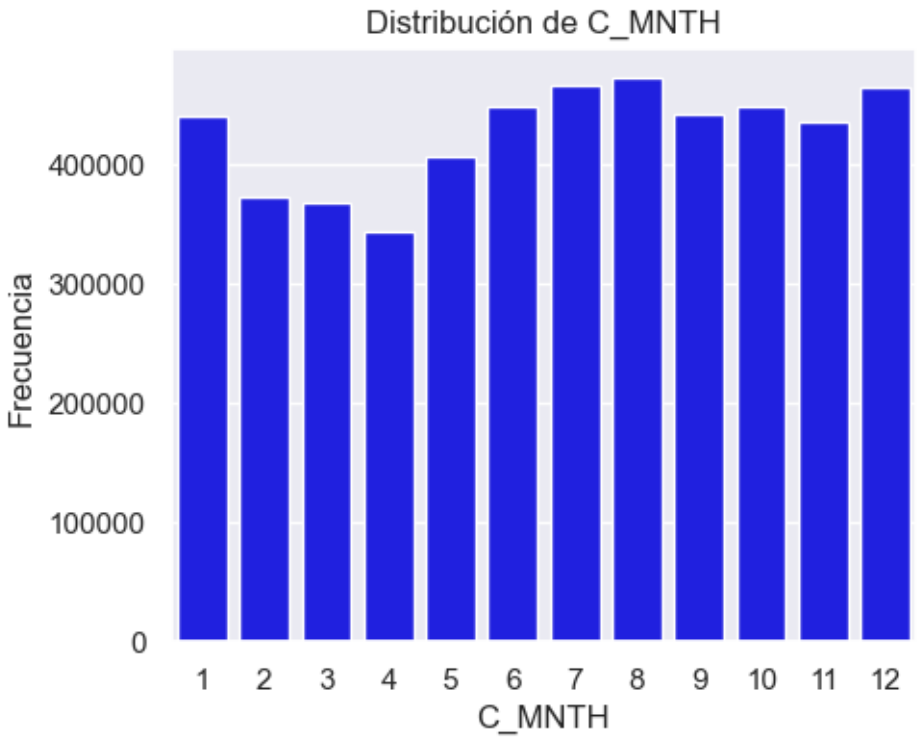


[40]:
```
plt.figure(figsize=(5, 4))
sns.countplot(x='C_RSUR', data=df_c, color = 'blue')

plt.xlabel('C_RSUR')
plt.ylabel('Frecuencia')
plt.title('Distribución de C_RSUR')

plt.show()
```



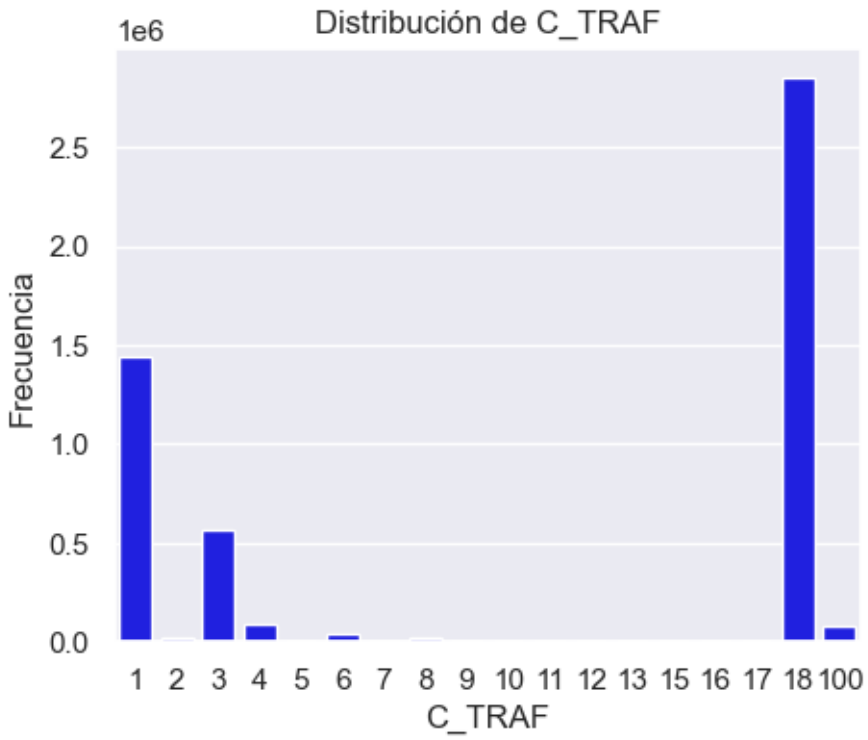
En el caso de ***C_MNTH***, obtenemos un resultado completamente previsible. Las mayores frecuencias absolutas de accidentes de tráfico se dan en los meses en los que estamos acostumbrados a oír hablar de operaciones salida:

[41]:
```
plt.figure(figsize=(5, 4))
sns.countplot(x='C_MNTH', data=df_c, color = 'blue')

plt.xlabel('C_MNTH')
plt.ylabel('Frecuencia')
plt.title('Distribución de C_MNTH')

plt.show()
```



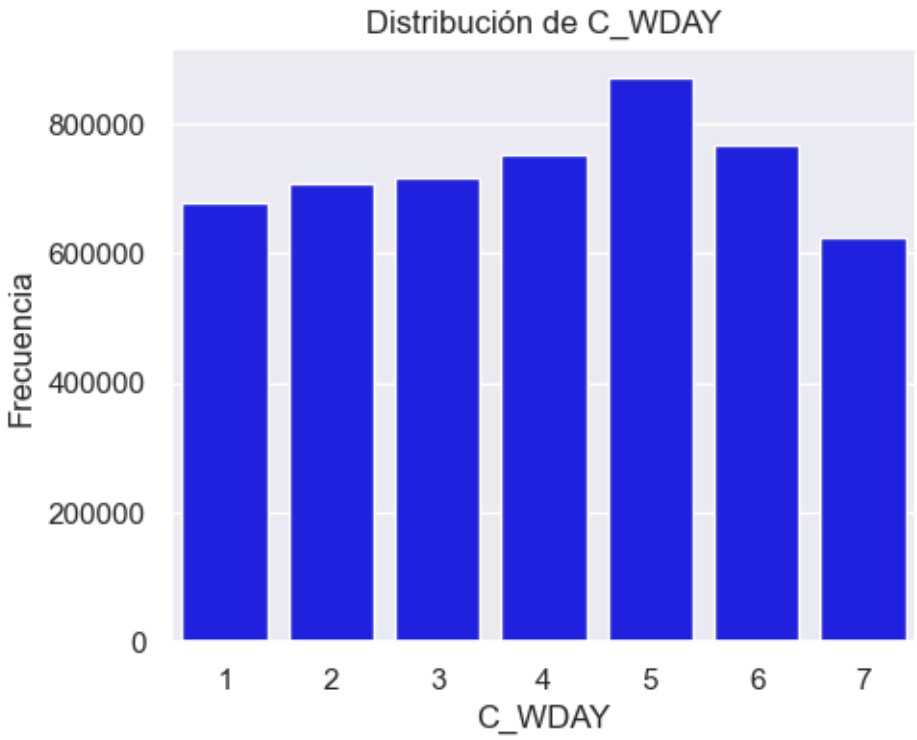
Variable *C_TRAF* - Destaca el número 18 con diferencia, valor que toma la variable cuando no hay control de tráfico presente, es decir, la mayoría de accidentes se producen bajo estas circunstancias: **es importante regular bien el tráfico**. - En segundo lugar encontramos el '1': el accidente se produce con todas las señales de tráfico opertivas - El tercer valor más significativo es '3': el accidente ha sido a causa de un conflicto en un stop

[42]:
```
plt.figure(figsize=(5, 4))
sns.countplot(x='C_TRAF', data=df_c, color = 'blue')

plt.xlabel('C_TRAF')
plt.ylabel('Frecuencia')
plt.title('Distribución de C_TRAF')

plt.show()
```



Atendiendo a la información derivada de la siguiente variable, ***C__WDAY***, podemos hacer el mismo comentario que hacíamos con *C_MNTH*. La mayor cantidad de accidentes ocurren a lo largo de los viernes y los sábados.

[43]:
```
plt.figure(figsize=(5, 4))
sns.countplot(x='C_WDAY', data=df_c, color = 'blue')

plt.xlabel('C_WDAY')
plt.ylabel('Frecuencia')
plt.title('Distribución de C_WDAY')

plt.show()
```



1.2.4 2.4. Transformación de variables

Ya hemos hecho alguna pequeña modificación en alguna variable conforme nos ha ido resultando necesario. Sin embargo, en ningún caso hemos alterado el significado de las diferentes codificaciones. Simplemente, hemos modificado la codificación, manteniendo eso sí las diferentes etiquetas.

En este apartado, estudiamos si tiene sentido o no la diferenciación a través de etiquetas de algunas condiciones del accidente para alguna variable.

Si atendemos a la codificación de algunas de las variables, podemos convenir una modificación de la misma en algunos casos.

Por ejemplo, si ponemos nuestra atención en ***C_WTHR***, la variable toma el valor '1' en caso de día soleado y '2' en caso de estar nublado, pero sin precipitación. Creemos que no resulta del todo interesante hacer esta distinción ya que, al menos por sí sola, sin tener otras cosas en cuenta, podría resultar no muy determinante.

[44]:
```
df_c['C_WTHR'] = df_c['C_WTHR'].replace(2, 1)
```

Así, de ahora en adelante, cuando no haya precipitación o alguna otra condición meteorológica "extraordinaria", esté claro o nublado, se asignará la etiqueta '1' en la variable *C_WTHR*.

Si recorremos el conjunto de variables, una agrupación similar se podría hacer en la variable ***P_SAFE***. En esta variable, hay diferentes etiquetas para identificar diferentes medidas de seguridad. Si bien creemos que no es conveniente mezclar algunas medidas de seguridad con otras

de naturaleza diferente (p.e. cinturón/casco por un lado y chaleco reflectante por otro), si consideramos que la etiqueta '2' y '9' podrían ser agrupadas.

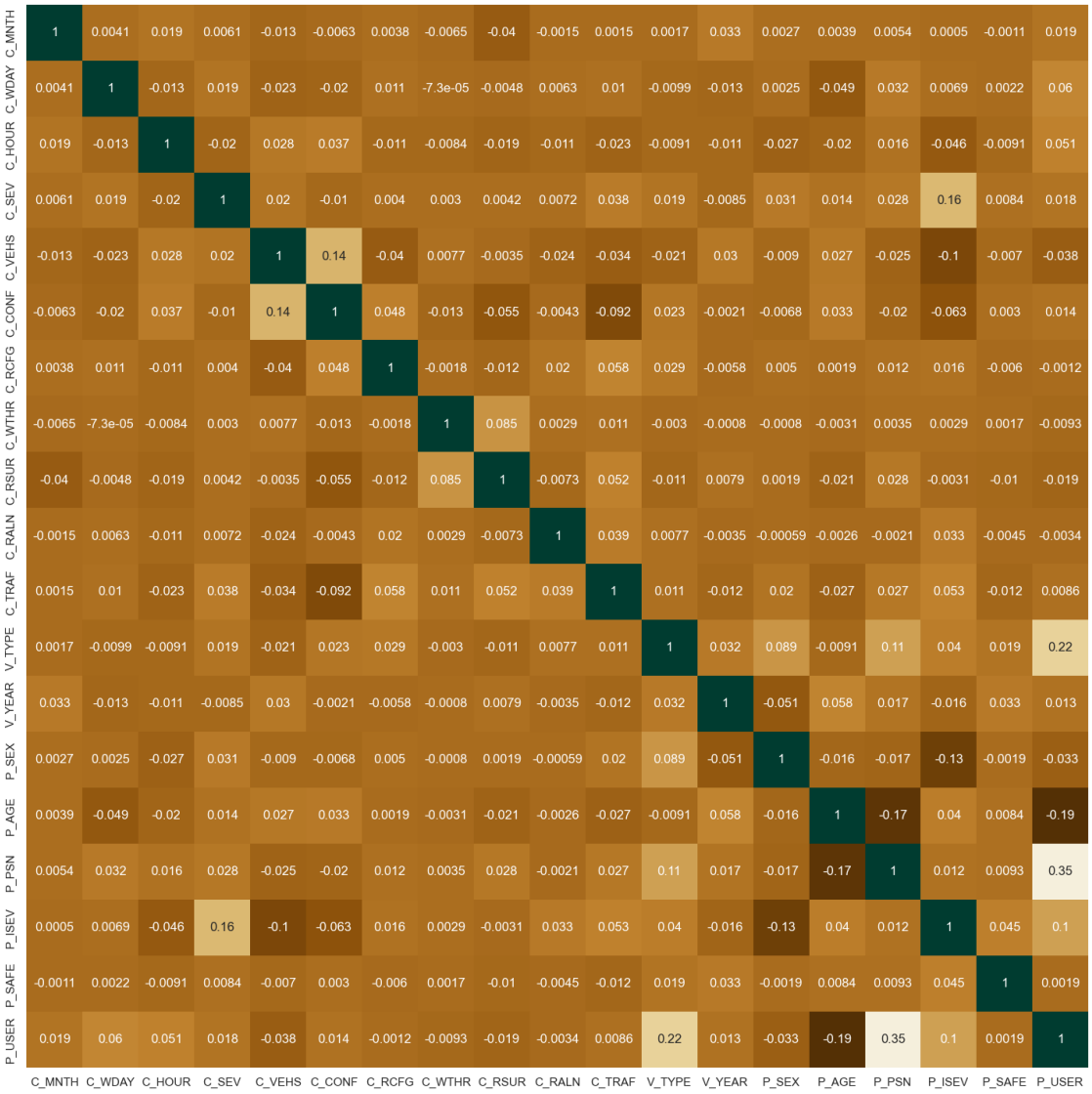
- La etiqueta '2' hace referencia a las siguientes situaciones: Safety device used or child restraint used.
- La etiqueta '9': Helmet worn.

Es evidente que son etiquetas análogas. La única diferencia sería que, en el primer caso, podemos referirnos a cinturones, airbags quizás en **turismos, camiones, autobuses...** mientras que, en el segundo caso, nos referimos simplemente al casco en **motos, bicis...**. En definitiva, la única diferencia entre estas dos etiquetas reside en el tipo de vehículo. Como esta información se aporta también a través de las variables *V_TYPE* y *P_USER*, podemos agrupar las etiquetas.

[45]:
```
df_c['P_SAFE'] = df_c['P_SAFE'].replace(9, 2)
```

Estas son las transformaciones que haremos en cuanto a la codificación de las variables. Las consecuencias en los respectivos histogramas de las variables, es evidente. **¿Y la estructura de correlaciones?¿Se mantiene?**

[46]:
```
plt.figure(figsize=(18, 18))
corr = df_c.corr()
sns.heatmap(corr, cmap="BrBG", annot=True, square=True, cbar=False)
plt.show()
```



Se puede apreciar que las correlaciones de las variables transformadas son ahora algo diferentes, pero que la diferencia es casi insignificante, por lo que podemos afirmar que se mantiene la estructura de correlaciones y que no ocurre nada raro al aplicarlas.

Nos hacemos ahora la siguiente pregunta: **¿Tiene sentido/se podría crear alguna variable extra a partir de las existentes?**

En nuestro caso y con variables categóricas, la creación de nuevas variables no es tan directa como lo sería en el caso de variables de naturaleza numérica/medible y continua. Sí es cierto que, a partir de las características descritas por diferentes variables, se pueden crear otras características que, por ejemplo, definan la intersección de ambas.

Lo que hay que tener en cuenta es que la intersección de ambas, en caso de haberla, ya la tendríamos, es decir, la nueva variable no nos va a aportar más información de la que ya tenemos. Sin embargo, es verdad que se podrían obtener mejoras en términos de correlaciones. No obstante, creemos

que las mejoras en correlaciones, visto el tipo de correlaciones que tenemos, no mejorarían mucho y que, por tanto, el esfuerzo de ponerse a combinar características y agruparlas en una nueva variable hasta dar con la combinación que nos reporte una verdadera mejora de correlación, puede que no merezca la pena.

1.2.5 2.5. Visualización y/o tratamiento de outliers

En este caso, al tratarse de variables categóricas, no tiene mucho sentido hablar de observaciones outliers en el sentido tradicional ya que, las etiquetas numéricas atienden a categorías y no a magnitudes continuas. Es cierto que es posible que existan valores inusuales en las variables categóricas, que podrían considerarse como observaciones atípicas en el contexto de la codificación esperada. Estos valores podrían indicar errores de codificación o datos incorrectos.

En nuestro caso, no podemos revisar la correcta codificación, por lo que la damos por buena.

1.2.6 2.6. Selección de características

¿Cómo podemos tomar la decisión de selección de características? Cuando miramos a las correlaciones de nuestra variable objetivo con el resto de variables, observamos que todas ellas son muy bajas. En este sentido, no podremos bajo este criterio otorgarle de manera robusta un orden de importancia a las mismas.

Vemos un método de ordenación de las variables por importancia de las mismas, para ello utilizaremos el algoritmo Random Forest, que funciona bien aunque las variables sean categóricas:

[47]:
```
# Seleccionamos las columnas de features y la variable objetivo
X = df_c.drop('C_SEV', axis=1)
y = df_c['C_SEV']
```

[48]:
```
#Spliteamos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)
```

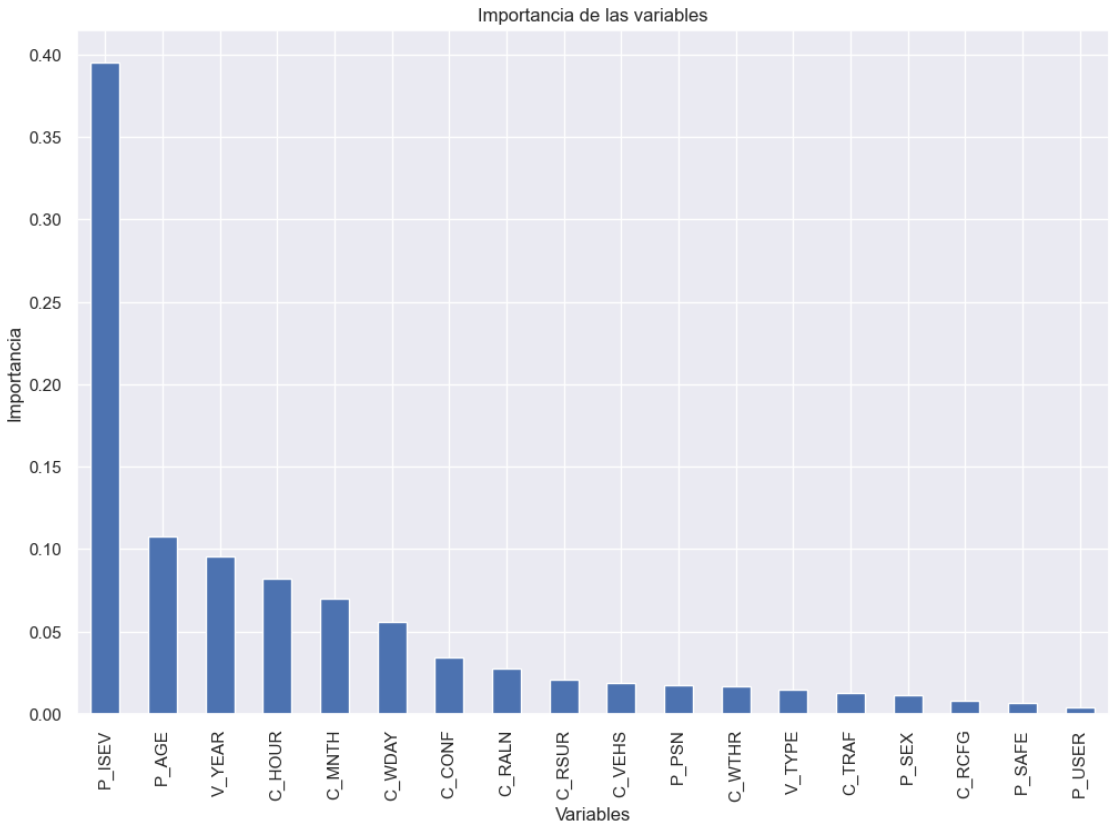
[49]:
```
#Escalamos
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

[50]:
```
# Creamos el modelo de Random Forest
regressor = RandomForestRegressor(n_estimators=25, random_state=0)
regressor.fit(X_train_scaled, y_train)
```

[50]:
```
RandomForestRegressor(n_estimators=25, random_state=0)
```

[51]:
```
# Creamos una serie con la importancia de cada feature
importances = pd.Series(regressor.feature_importances_, index=X.columns)

# Ordenamos las importancias de forma descendente
```

```
importances_sorted = importances.sort_values(ascending=False)

# Hacemos el gráfico
plt.figure(figsize=(12,8))
importances_sorted.plot(kind='bar')
plt.title('Importancia de las variables')
plt.xlabel('Variables')
plt.ylabel('Importancia')
plt.show()
```



En el gráfico previo, se puede observar como la variable *P_ISEV* parece ser la variable con más importancia. Si vamos a la leyenda de la base de datos, vemos que la codificación es la siguiente:

P_ISEV	Codificación
1	No injury
2	Injury
3	Fatality

¡¡Podemos observar que la variable *P_ISEV* es prácticamente la misma variable que la objetivo!! Por eso parece ser tan importante. Sin embargo, precisamente por esto, consideramos que no tiene

sentido incluirla entre las variables predictoras en nuestros modelos. Procedemos a eliminarla del dataset.

[52]:
```
df_c = df_c.drop('P_ISEV', axis=1)
```

Una vez eliminada esta variable, nos quedan 17 variables predictoras.**¿Podemos descartar alguna más para reducir un poco este número?** El criterio de importancia previo, nos sugiere que las variables de *P_SEX* en adelante (orden del gráfico), no son importantes.

Sin embargo, atendiendo a las definiciones de las variables, sí que consideramos que podrían llegar a ser importantes para intentar predecir nuestra variable objetivo. **¡¡¡¡NO HAY MÁS QUE VER *P_SAFE*!!!!**

Por ello, en principio, decidimos seguir trabajando con todas las variables restantes.

1.2.7 2.7. Balanceo de datos para *C_SEV*

Procedemos al balanceo de la variable por **undersampling**:

[53]:
```
#Separamos las características de la variable objetivo
X = df_c.drop('C_SEV', axis=1)
y = df_c['C_SEV']

# Aplicar el undersampling
undersampler = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = undersampler.fit_resample(X, y)

# Mostrar los datos balanceados
balanced_data = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.
 ↪Series(y_resampled, name='C_SEV')], axis=1)
df_bal = pd.DataFrame(balanced_data, columns=X.columns.tolist() + ['C_SEV'])
```

[54]:
```
df_bal
```

[54]:

	C_MNTH	C_WDAY	C_HOUR	C_VEHS	C_CONF	C_RCFG	C_WTHR	C_RSUR	\
0	4	5	16	3	21	2	5	5	
1	8	1	19	2	35	1	1	1	
2	3	5	7	2	21	1	3	2	
3	6	5	18	1	6	1	1	1	
4	9	4	9	1	3	100	3	2	
…	…	…	…	…	…	…	…	…	
172681	12	7	11	3	35	2	1	100	
172682	12	7	1	1	4	100	6	1	
172683	12	7	2	1	100	1	1	1	
172684	8	5	22	1	6	1	1	4	
172685	8	5	22	1	6	1	1	4	

	C_RALN	C_TRAF	V_TYPE	V_YEAR	P_SEX	P_AGE	P_PSN	P_SAFE	P_USER	\
0	1	8	1	1998	1	51	11	2	1	

```
1              1      18        1    2005       0     40      11        2        1
2              1      18        1    2006       0     53      11        2        1
3              1      18        1    2000       1      9      21        2        2
4              3      18        1    2001       0     25      11        2        1
…            …      …       …      …    …      …     …       …      …
172681         1       3        7    2014       0     35      11        2        1
172682         1      18        1    2011       0     30      11        2        1
172683         1      18        8    2013       1     37      11        2        1
172684       100      18       22    2000       1     67      12        1        1
172685       100      18       22    2000       1     10      98        1        1

        C_SEV
0           0
1           0
2           0
3           0
4           0
…         …
172681      1
172682      1
172683      1
172684      1
172685      1

[172686 rows x 18 columns]
```

[55]:
```
#comprobamos que el dataframe está ahora balanceado:
df_bal['C_SEV'].value_counts()
```

[55]:
```
0    86343
1    86343
Name: C_SEV, dtype: int64
```

Se puede comprobar en la salida anterior que, ahora sí, nuestra base de datos está balanceada correctamente. Podemos proceder al spliteo de nuestra muestra. Antes de ello, le cambiamos el nombre a nuestro dataframe.

[56]:
```
df_fin = df_bal
```

[57]:
```
df_fin.to_csv('preprocesado.csv', sep='\t')
```

POR MOTIVOS DE MEMORIA SE GUARDA EL PREPROCESSING HASTA ESTE PUNTO Y SE CONTINUA CON EL SPLIT Y EL ENTRENAMIENTO DE MODELOS EN OTRO SCRIPT

proyecto-final-ml-modelos

June 8, 2023

1 PRÁCTICA FINAL MACHINE LEARNING. PARTE 2

Por Sofía Canela García-Arias, Berta Corriols Barrio & Víctor Panadero Gómez

[74]:
```
import pandas as pd
import pandas as pd
import numpy as np
import seaborn as sns    #visualización
import matplotlib.pyplot as plt   #visualización
import scipy
import math
%matplotlib inline
sns.set(color_codes=True)

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import RobustScaler

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
 ↪recall_score
from sklearn.metrics import classification_report

import statsmodels.api as sm

from sklearn.metrics import precision_recall_curve, auc
```

[75]:
```
df_fin = pd.read_csv("preprocesado.csv", sep = '\t')
df_fin
```

[75]:

	Unnamed: 0	C_MNTH	C_WDAY	C_HOUR	C_VEHS	C_CONF	C_RCFG	C_WTHR	\
0	0	4	5	16	3	21	2	5	
1	1	8	1	19	2	35	1	1	
2	2	3	5	7	2	21	1	3	
3	3	6	5	18	1	6	1	1	

```
4                4        9        4        9        1        3      100        3
…              …      …      …      …      …      …      …      …
172681       172681       12        7       11        3       35        2        1
172682       172682       12        7        1        1        4      100        6
172683       172683       12        7        2        1      100        1        1
172684       172684        8        5       22        1        6        1        1
172685       172685        8        5       22        1        6        1        1

         C_RSUR  C_RALN  C_TRAF  V_TYPE  V_YEAR  P_SEX  P_AGE  P_PSN  P_SAFE  \
0             5       1       8       1    1998      1     51     11       2
1             1       1      18       1    2005      0     40     11       2
2             2       1      18       1    2006      0     53     11       2
3             1       1      18       1    2000      1      9     21       2
4             2       3      18       1    2001      0     25     11       2
…         …      …      …      …      …    …      …      …      …
172681      100       1       3       7    2014      0     35     11       2
172682        1       1      18       1    2011      0     30     11       2
172683        1       1      18       8    2013      1     37     11       2
172684        4     100      18      22    2000      1     67     12       1
172685        4     100      18      22    2000      1     10     98       1

         P_USER  C_SEV
0             1      0
1             1      0
2             1      0
3             2      0
4             1      0
…         …    …
172681        1      1
172682        1      1
172683        1      1
172684        1      1
172685        1      1

[172686 rows x 19 columns]
```

[76]:
```
#Eliminamos la primera columna
df_fin =df_fin.drop('Unnamed: 0', axis = 1)
```

1.1 3. Train-Test split

[77]:
```
X = df_fin.drop(['C_SEV'], axis = 1)
y = df_fin['C_SEV']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)
```

1.2 4. Escalado de los datos

En este apartado, se realizará un escalado de los datos por sí resultase necesario para la implementación de algún algoritmo. De esta manera, dispondremos de unos datos que no estarán escalados y de otros que sí lo están. Usaremos unos u otros en función de nuestras necesidaddes.

[78]:
```
robust_scaler = RobustScaler()
X_train_scaled = robust_scaler.fit_transform(X_train)
X_test_scaled = robust_scaler.fit_transform(X_test)

#Los guardamos en un data frame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X.columns.tolist())
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X.columns.tolist())
```

NOTAS: - Ahora que hemos escalado X_train, podremos implementar los modelos que requieran de escalado usando X_train_scaled. (En los que no lo requieran, realmente, también podemos). - Nótese que no hace falta reescalar y_train. Que la variable objetivo no esté escalada, no afecta al rendimiento de los modelos. - **¡Estamos reescalando variables categóricas con codificación entera!** Si finalmente usamos el reescalado de datos, deberemos tener cuidado con las interpretaciones posteriores, etc.

1.3 5. Modelos

En los sucesivos apartados iremos implementando modelos que nos permitan llevar a cabo **nuestro objetivo**: intentar predecir si diferentes accidentes son causa o no de fallecimiento. En otras palabras: tenemos un problema de **clasificación binaria**. Nuestra variable objetivo, como ya hemos detallado, es ***C_SEV*** y toma valores 0 y 1.

- **NOTA 1**: Este objetivo podría estar muy relacionado con las tareas de las **instituciones públicas**: campañas de concienciación para prevenir accidentes, pero sobre todo, o en última instancia, **para prevenir muertes**; (véanse campañas para el uso del cinturón de seguridad).
- **NOTA 2**: Por supuesto, se podrían plantear otros objetivos. Perseguimos el ya descrito porque se indica en la tarea que, como mínimo, se debe atacar este objetivo. Si da tiempo, se atacarán otros también.

Dicho esto: una vez implementados diferentes modelos, es importante, tal y como se indica en el enunciado de la práctica, comparar los mismos para la posterior selección del que consideremos que tiene mejor rendimiento. Para ello, es importante definir/escoger una métrica que consideremos adecuada y que nos permita evaluar los diferentes rendimientos de igual manera, para poder posteriormente comparar.

Así, la siguiente pregunta es: **¿Qué métrica podemos emplear?**

Pues bien, teniendo en cuenta que estamos ante un problema de clasificación binaria - que se puede ver como clasificación de positivos (hay fallecidos: '1') y negativos (no hay fallecidos: '0') - una buena métrica podría ser considerada el **RECALL**. Pero, **¿por qué esta y no otra?** Bien: ya hemos dicho que el problema que estamos abordando podría verse desde el punto de vista de las **instituciones públicas**. El objetivo final de éstas será **tratar de reducir los fallecimientos a causa de accidentes de tráfico. Por encima de cualquier cosa**. Es decir, es **importante predecir bien los accidentes que causan fallecimiento, aunque eso implique predecir**

alguno que no cause fallecimiento como uno que sí lo haga. Por esta razón, consideramos que el **recall es una buena opción**.

- El recall se centra en la capacidad del modelo para identificar correctamente los casos positivos (fallecimientos en este caso). Al maximizar el recall, se asegura que se capturan la mayoría de los casos positivos, lo que permite tomar acciones preventivas y asignar recursos adecuados para minimizar los fallecimientos. Dado que las instituciones públicas priorizan evitar la pérdida de vidas, el recall es la métrica más relevante para evaluar el desempeño del modelo en este caso.
- Se calcula como el cociente entre el número de verdaderos positivos (TP) y la suma de los verdaderos positivos y los falsos negativos (FN):

```
Recall = TP / (TP + FN)
```

1.3.1 5.1. KNN

El primer modelo que entrenaremos será **K-NearestNeighbours**. Antes de empezar, haremos algunas anotaciones:

1. Nuestra idea primera era poder optimizar **dos hiperparámetros**: el primero de ellos sería ***n***, que representa el número de primeros vecinos. El segundo sería ***p***, que representaría la distancia. Así, haciendo GridSearchCV, trataríamos de hallar los hiperparámetros óptimos que nos diesen el mejor modelo.
2. El número de primeros vecinos hemos dedicido limitarlo a un máximo de 15. Por su parte, para elegir la distancia, habíamos decidido buscar la ***p*** en la rejilla determinada por **np.linspace(1, 3, 21)**.
3. Una buena manera de buscar el mejor modelo hubiese sido hacer cv=LOO, es decir, implementar la **técnica del LeaveOneOut**: un k-fold en la que el tamaño de cada subconjunto es una observación. Dado el tamaño de nuestro set de entrenamiento, esto es **computacionalmente inviable (ADEMÄS DE QUE NOS DA MEMORY ERROR, CLARO)**. Tras está conclusión, pensamos en hacer **cv=3**, que se corresponde con una Validación Cruzada 3-fold. La conclusión es que sigue siendo **computacionalmente muy pesada**: nótese que habría que entrenar 3 modelos por cada iteración x 15 valores para la n x 21 valores para la p, lo que hace un total de 3x15x21 modelos a entrenar a una media de unos 2 minutos por modelo… Se va de las manos.

   Finalmente, por los **motivos computacionales** a los que estamos refiriéndonos, decidimos hacer una **validación cruzada simple** con la **búsqueda de un único hiperparámetro: el número de vecinos óptimo**. De esta manera, tenemos que entrenar tan sólo 15 modelos (que ya son, por otra parte). Si los resultados no son aceptables, se intentará computar una validación cruzada 3-fold para este mismo parámetro, aunque el coste computacional sea alto.

CÓDIGO PARA DOS HIPERPARÁMETROS CON LOO. (Para 3-fold sería el mismo poniendo cv=3 en GridSearch).

[33]:
```
#from sklearn.model_selection import LeaveOneOut

#knn = KNeighborsClassifier()
```

```
#ps = np.linspace(1, 3, 21)
#ns = list(range(1,16))
#h_parameters = {'p':ps, 'n_neighbors':ns}

#scoring = make_scorer(recall_score)
#loo = LeaveOneOut()

#cv = GridSearchCV(knn, h_parameters, cv=loo, n_jobs=-1, scoring=scoring)
#cv.fit(X_train_scaled_df, y_train)
```

Ahora sí, aplicamos **KNN** teniendo en cuenta lo detallado en los puntos previos. (Como ya hemos adelantado previamente, utilizaremos el recall como función score):

[6]:
```
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import GridSearchCV, ShuffleSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import make_scorer, recall_score


knn = KNeighborsClassifier()
ns = list(range(1,16))
h_parameters = {'n_neighbors':ns}
scoring = make_scorer(recall_score)
cv_simple = ShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

cv = GridSearchCV(knn, h_parameters, cv=cv_simple, scoring=scoring)
cv.fit(X_train_scaled_df, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
```

```
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
```

```
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
```

```
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[6]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=1, random_state=42, test_size=0.2,
train_size=None),
             estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15]},
             scoring=make_scorer(recall_score))
```

[7]:
```
#Sacamos el hiperparámetro n que maximiza el score:
cv.best_estimator_
```

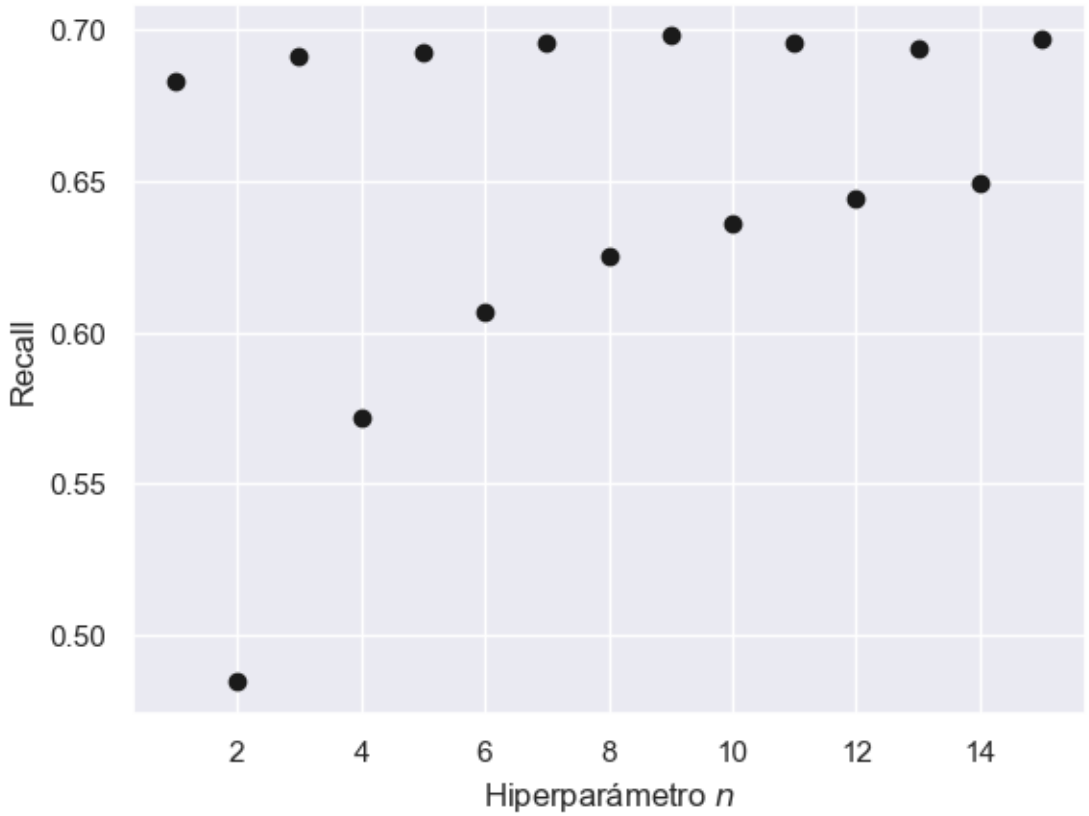
[7]:
```
KNeighborsClassifier(n_neighbors=9)
```

[8]:
```
#Sacamos los resultados de CV para entender la gráfica posterior:
cv.cv_results_
```

[8]:
```
{'mean_fit_time': array([0.04944229, 0.07815981, 0.07817006, 0.04686618,
0.06248689,
        0.03124571, 0.06248784, 0.04691267, 0.03123856, 0.04686713,
        0.04686999, 0.03124452, 0.03125978, 0.03125548, 0.03124809]),
 'std_fit_time': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.]),
 'mean_score_time': array([  60.46069884,   77.58832049,   79.67534137,
107.54092002,
         115.67363596,  120.74715209,  112.38401556, 1275.51437402,
          60.9321382 ,   61.32988191,   61.06554317,   62.65649581,
          65.96304989,   63.97722125,   65.57609415]),
 'std_score_time': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0.]),
 'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
```

```
  {'n_neighbors': 9},
  {'n_neighbors': 10},
  {'n_neighbors': 11},
  {'n_neighbors': 12},
  {'n_neighbors': 13},
  {'n_neighbors': 14},
  {'n_neighbors': 15}],
 'split0_test_score': array([0.68325693, 0.48446482, 0.69169759, 0.57156371,
0.69278906,
        0.60656334, 0.69613621, 0.62526377, 0.69810085, 0.63617842,
        0.69606345, 0.64410973, 0.69366223, 0.64920323, 0.69708215]),
 'mean_test_score': array([0.68325693, 0.48446482, 0.69169759, 0.57156371,
0.69278906,
        0.60656334, 0.69613621, 0.62526377, 0.69810085, 0.63617842,
        0.69606345, 0.64410973, 0.69366223, 0.64920323, 0.69708215]),
 'std_test_score': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0.]),
 'rank_test_score': array([ 8, 15,  7, 14,  6, 13,  3, 12,  1, 11,  4, 10,  5,
9,  2])}
```

[9]:
```
plt.plot(ns, cv.cv_results_['mean_test_score'], 'ok')
plt.xlabel('Hiperparámetro $n$')
plt.ylabel('Recall')
plt.grid(True)
```



En los outputs previos hemos obtenido, respectivamente por orden:

1. El hiperparámetro óptimo.
2. Un resumen del proceso de CV.
3. Una gráfica que ilustra el procedimiento de CV Simple. En ella, se puede ver que el mejor valor para nuestra scoring function se obtiene para el hiperparámetro n = Nº vecinos = 9.

Una vez ilustrado de manera gráfica lo que estamos haciendo en el ajuste del modelo, y teniendo en cuenta que el resultado de hacer CV nos dice que nuestro hiperparámtero óptimo es ***n=9***, vamos a ajustar el modelo con nuestro set de entrenamiento:

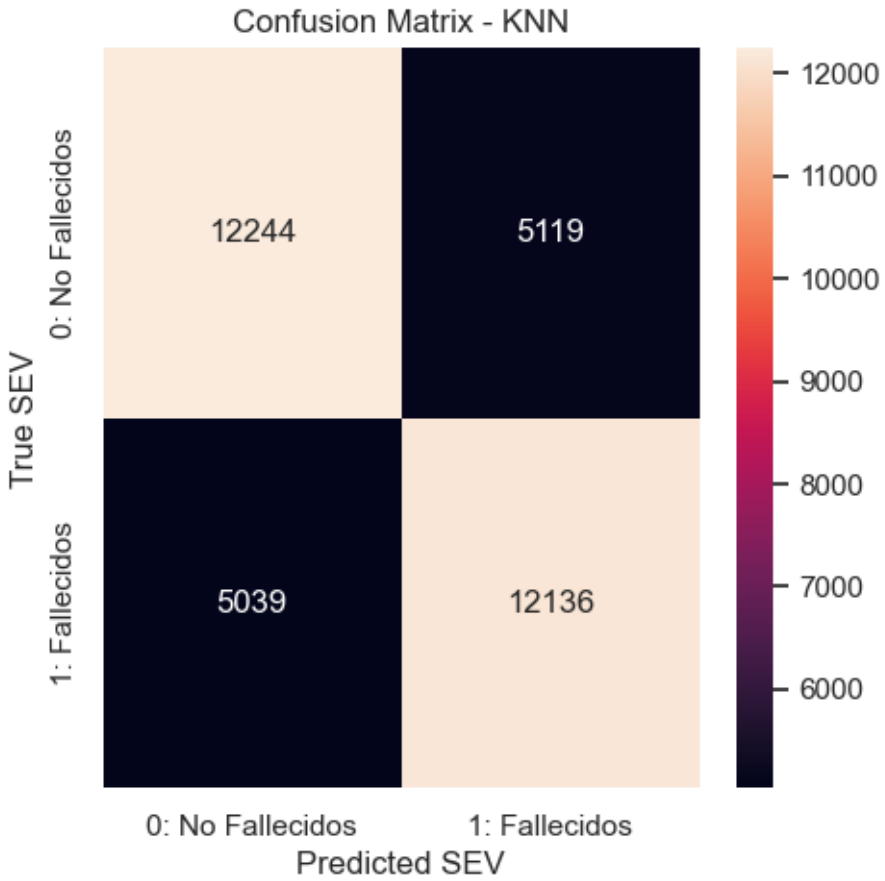
[10]:
```
#fit para train
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train_scaled_df, y_train)

#predict para test
y_pred = knn.predict(X_test_scaled_df)
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
```

```
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

A continuación, sacamos la matriz de confusión. Ya hemos dicho en varias ocasiones que la métrica de evaluación que más nos interesará y que utilizaremos para comparar modelos será el **recall para accidentes con fallecimientos ('1')**. No obstante, sacamos otras métricas para tener también otras referencias de cómo de bien funciona nuestro modelo (aunque no las vayamos a tener en cuenta a la hora de seleccionar el modelo óptimo).

[11]:
```
conf_matrix_KNN = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 5))
labels = ['0: No Fallecidos','1: Fallecidos']
sns.heatmap(conf_matrix_KNN, xticklabels=labels, yticklabels=labels,␣
 ↪annot=True, fmt="d");
plt.title("Confusion Matrix - KNN")
plt.ylabel('True SEV')
plt.xlabel('Predicted SEV')
plt.show()
print (classification_report(y_test, y_pred))
```



```
              precision    recall  f1-score   support

           0       0.71      0.71      0.71     17363
           1       0.70      0.71      0.70     17175

    accuracy                           0.71     34538
   macro avg       0.71      0.71      0.71     34538
weighted avg       0.71      0.71      0.71     34538
```

Para comenzar a comentar los resultados, destacamos **la métrica que nos interesa**: **OBTENEMOS UN RECALL PARA '1' DE 0.71.** Creemos que es un resultado aceptable teniendo en cuenta que no hemos podido hacer una búsqueda demasiado exhaustiva de hiperparámetro/s óptimo/s.

Además, si atendemos a otras métricas para hacernos una idea de como funcionaría nuestro modelo si nuestros objetivos fuesen otros, vemos que obtenemos exactamente el mismo valor, con una accuracy de 0.71 y una precisión de 0.71 también.

RECORDAMOS: - La **exactitud (accuracy)** es una medida de cuántas de las muestras totales se clasificaron correctamente, ya sean positivas o negativas. Es una métrica global que considera tanto los verdaderos positivos como los verdaderos negativos. Es **útil cuando todas las clases tienen una importancia similar** y no hay un desequilibrio significativo entre las clases. En este sentido y teniendo en cuenta lo que hemos explicado, ya sabemos que no será una métrica muy significativa para nosotros porque nuestras clases **no tienen la misma importancia**. Igualmente, vemos que no es mala.

```
Accuracy = (TP + TN) / (TP + TN + FP + FN)
```

A continuación, utilizamos el modelo de regresión logística:

1.3.2 5.2. Regresión logística

En segundo lugar, implementamos un modelo de regresión logística.

Comenzaremos ajustando el modelo con todas las variables y analizaremos posteriormente el resumen de salida:

[15]:
```
# import statsmodels.api as sm

X_m1 = sm.add_constant(X_train_scaled_df)
y_train_df = pd.DataFrame(y_train)
y_train_df = y_train_df.reset_index(drop=True)

logit1 = sm.Logit(y_train_df, X_m1)
results1 = logit1.fit()

results1.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.658440
         Iterations 6
```

[15]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                  C_SEV   No. Observations:               138148
Model:                          Logit   Df Residuals:                   138130
Method:                           MLE   Df Model:                           17
Date:                Wed, 07 Jun 2023   Pseudo R-squ.:                0.05007
Time:                        01:18:57   Log-Likelihood:                -90962.
converged:                       True   LL-Null:                       -95757.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0978      0.009     11.201      0.000       0.081       0.115
C_MNTH         0.0914      0.010      9.310      0.000       0.072       0.111
C_WDAY         0.2725      0.011     23.815      0.000       0.250       0.295
C_HOUR        -0.1853      0.009    -21.275      0.000      -0.202      -0.168
C_VEHS         0.0273      0.002     11.562      0.000       0.023       0.032
C_CONF        -0.0574      0.008     -7.334      0.000      -0.073      -0.042
C_RCFG     -2.07e-05      0.000     -0.056      0.955      -0.001       0.001
C_WTHR         0.0033      0.001      3.063      0.002       0.001       0.005
C_RSUR         0.0004      0.000      1.236      0.217      -0.000       0.001
C_RALN         0.0077      0.001      7.299      0.000       0.006       0.010
C_TRAF         0.4139      0.009     47.297      0.000       0.397       0.431
V_TYPE         0.0103      0.001     12.761      0.000       0.009       0.012
V_YEAR        -0.1059      0.007    -14.770      0.000      -0.120      -0.092
P_SEX          0.5030      0.012     43.183      0.000       0.480       0.526
P_AGE          0.2613      0.009     29.886      0.000       0.244       0.278
P_PSN          0.0293      0.001     24.766      0.000       0.027       0.032
P_SAFE         0.0056      0.001     10.513      0.000       0.005       0.007
P_USER         0.1445      0.008     18.056      0.000       0.129       0.160
==============================================================================
"""
```

Se puede ver en el output previo que tenemos **dos p-valores mayores que 0.01, 0.05 y 0.1**: son los correspondientes a las variables ***C_RCFG*** y ***C_RSUR***. Esto nos dice que no se puede rechazar la hipótesis nula de que estas variables no sean significativas. Procedemos a su eliminación:

[16]:
```
X_m2 = X_m1.drop(columns=['C_RCFG', 'C_RSUR'])
logit2 = sm.Logit(y_train_df, X_m2)
results2 = logit2.fit()

results2.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.658445
         Iterations 6
```

[16]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                  C_SEV   No. Observations:               138148
Model:                          Logit   Df Residuals:                   138132
Method:                           MLE   Df Model:                           15
Date:                Wed, 07 Jun 2023   Pseudo R-squ.:                0.05006
Time:                        01:19:04   Log-Likelihood:                -90963.
converged:                       True   LL-Null:                       -95757.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0995      0.009     11.599      0.000       0.083       0.116
C_MNTH         0.0909      0.010      9.273      0.000       0.072       0.110
C_WDAY         0.2722      0.011     23.796      0.000       0.250       0.295
C_HOUR        -0.1854      0.009    -21.289      0.000      -0.202      -0.168
C_VEHS         0.0274      0.002     11.606      0.000       0.023       0.032
C_CONF        -0.0577      0.008     -7.394      0.000      -0.073      -0.042
C_WTHR         0.0034      0.001      3.184      0.001       0.001       0.006
C_RALN         0.0077      0.001      7.294      0.000       0.006       0.010
C_TRAF         0.4144      0.009     47.489      0.000       0.397       0.431
V_TYPE         0.0103      0.001     12.759      0.000       0.009       0.012
V_YEAR        -0.1057      0.007    -14.752      0.000      -0.120      -0.092
P_SEX          0.5031      0.012     43.187      0.000       0.480       0.526
P_AGE          0.2611      0.009     29.874      0.000       0.244       0.278
P_PSN          0.0294      0.001     24.826      0.000       0.027       0.032
P_SAFE         0.0056      0.001     10.512      0.000       0.005       0.007
P_USER         0.1443      0.008     18.031      0.000       0.129       0.160
==============================================================================
"""
```

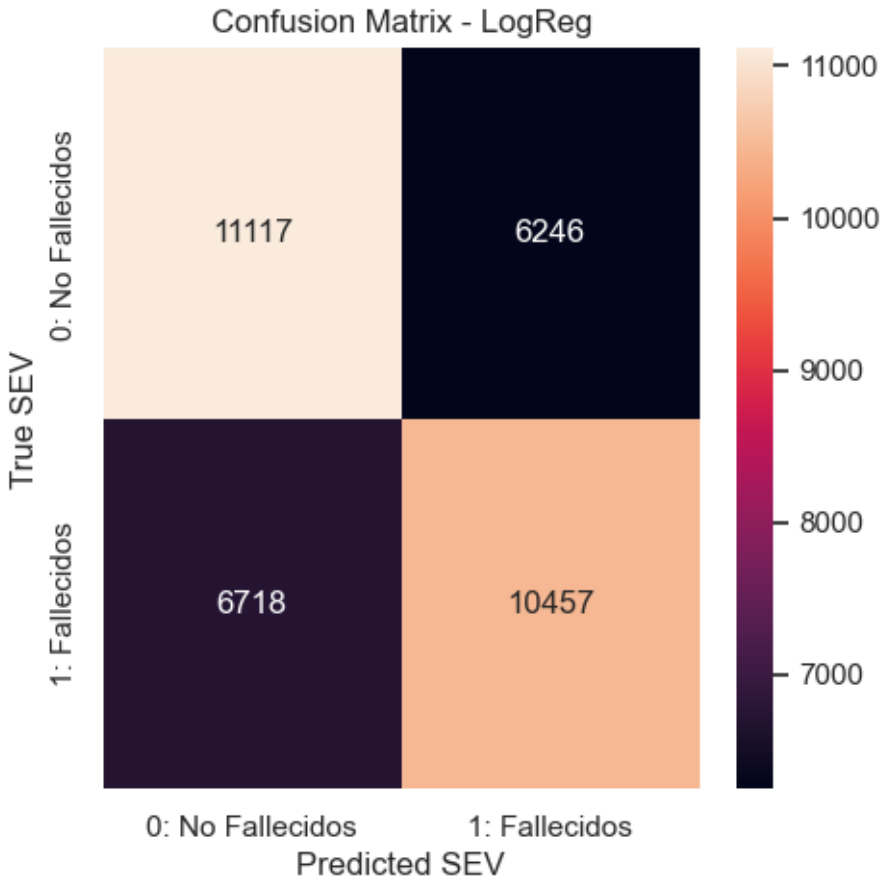
Ahora ya tenemos todas las variables significativas al 1%. Además, el valor de Log-Likelihood apenas cambia de un modelo a otro, lo cual podría decirse bueno, en el sentido de que el funcionamiento de nuestro modelo no ha cambiado (no se ha empeorado nuestro modelo con respecto al previo).

Por otro lado, sí que notamos que tenemos una R^2 ajustada bastante baja en ambos modelos. Igualmente, utilizamos este último modelo ajustado para ver cómo de bien predice sobre X_test:

[17]:
```
X_test_scaled_m2= X_test_scaled_df.drop(columns=['C_RCFG', 'C_RSUR'])
X_test_scaled_m2 = sm.add_constant(X_test_scaled_m2)

pred_test2 = results2.predict(X_test_scaled_m2) #probabilidades
```

```
pred_test2 = pred_test2.apply(lambda x: 1 if x>0.5 else 0)
```

[18]:
```
conf_matrix_LogReg = confusion_matrix(y_test, pred_test2)
plt.figure(figsize=(5, 5))
labels = ['0: No Fallecidos','1: Fallecidos']
sns.heatmap(conf_matrix_LogReg, xticklabels=labels, yticklabels=labels,␣
 ↪annot=True, fmt="d");
plt.title("Confusion Matrix - LogReg")
plt.ylabel('True SEV')
plt.xlabel('Predicted SEV')
plt.show()
print (classification_report(y_test, pred_test2))
```




```
              precision    recall  f1-score   support

           0       0.62      0.64      0.63     17363
           1       0.63      0.61      0.62     17175

    accuracy                           0.62     34538
```

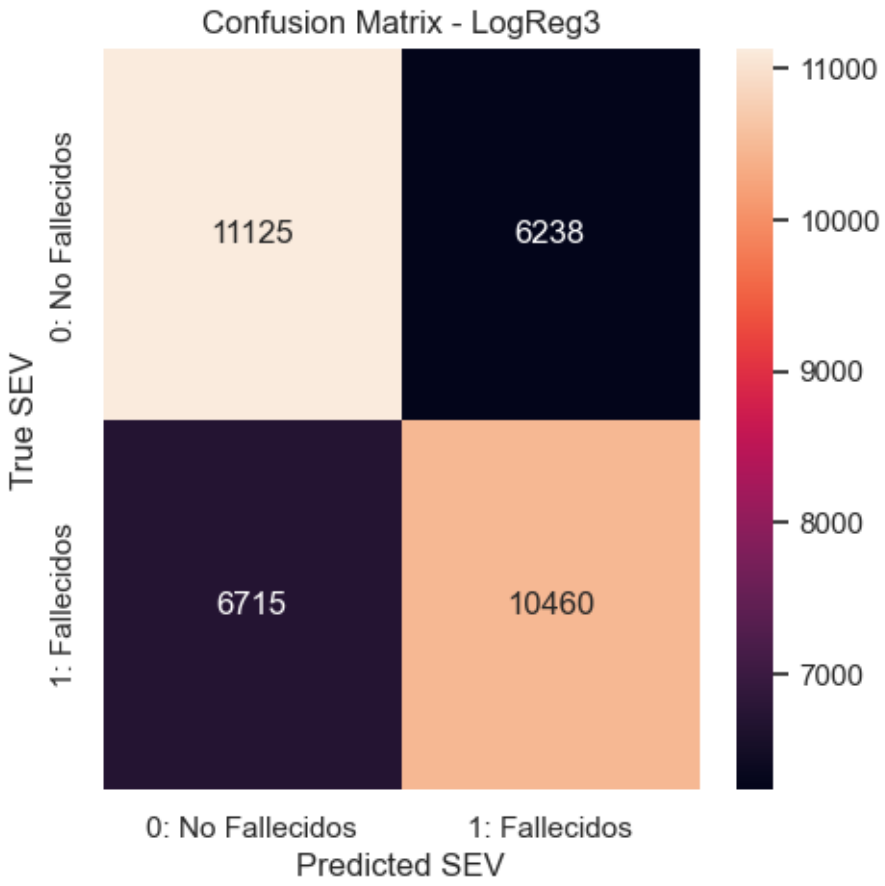
```
   macro avg       0.62      0.62      0.62     34538
weighted avg       0.62      0.62      0.62     34538
```

Se puede ver cómo nuestra métrica de referencia ha bajado 10 centésimas. De hecho, aunque las tenemos menos en cuenta, el resto de métricas también se han visto afectadas con la implementación de este modelo. De momento, no parece que vaya a ser el modelo seleccionado.

Como los resultados en el caso de este modelo no han sido satisfactorios, probamos a ajustar el modelo de otra manera por si el motivo fuese alguna toma de decisión errónea a la hora de la eliminación de alguna de las dos variables descartadas:

[19]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train_scaled, y_train)

y_pred = classifier.predict(X_test_scaled)
```

[20]:
```
conf_matrix_LogReg3 = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 5))
labels = ['0: No Fallecidos','1: Fallecidos']
sns.heatmap(conf_matrix_LogReg3, xticklabels=labels, yticklabels=labels,␣
 ↪annot=True, fmt="d");
plt.title("Confusion Matrix - LogReg3")
plt.ylabel('True SEV')
plt.xlabel('Predicted SEV')
plt.show()
print (classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.62	0.64	0.63	17363
1	0.63	0.61	0.62	17175
accuracy			0.62	34538
macro avg	0.63	0.62	0.62	34538
weighted avg	0.63	0.62	0.62	34538

Se puede comprobar que los resultados en términos de las métricas de evaluación, pero, en especial, en términos de recall, no mejoran. Son prácticamente identicos.

1.3.3 5.3. Árbol de decisión

En lo siguiente, se ajustará un modelo de Árbol de decisión. Lo primero que se hará será hacer una búsqueda CV análoga a la que hicimos en el caso de KNN. En esta ocasión, el hiperparámetro a optimizar será la máxima profundidad de nuestro árbol:

[79]:
```
from sklearn.tree import DecisionTreeClassifier

arbol = DecisionTreeClassifier()

#nuestro hiperparámetro a optimizar será depth. Lo acotamos en un rango de␣
 ↪valores entre 1 y 10 para que no se nos vaya de las
#manos
ds = list(range(1,11))
h_parameters = {'max_depth':ds}
cv_simple = ShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
scoring = make_scorer(recall_score)

cv = GridSearchCV(arbol, h_parameters, cv=cv_simple, n_jobs=-1, scoring=scoring)

cv.fit(X_train, y_train)
```

[79]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=1, random_state=42, test_size=0.2,
train_size=None),
             estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             scoring=make_scorer(recall_score))
```

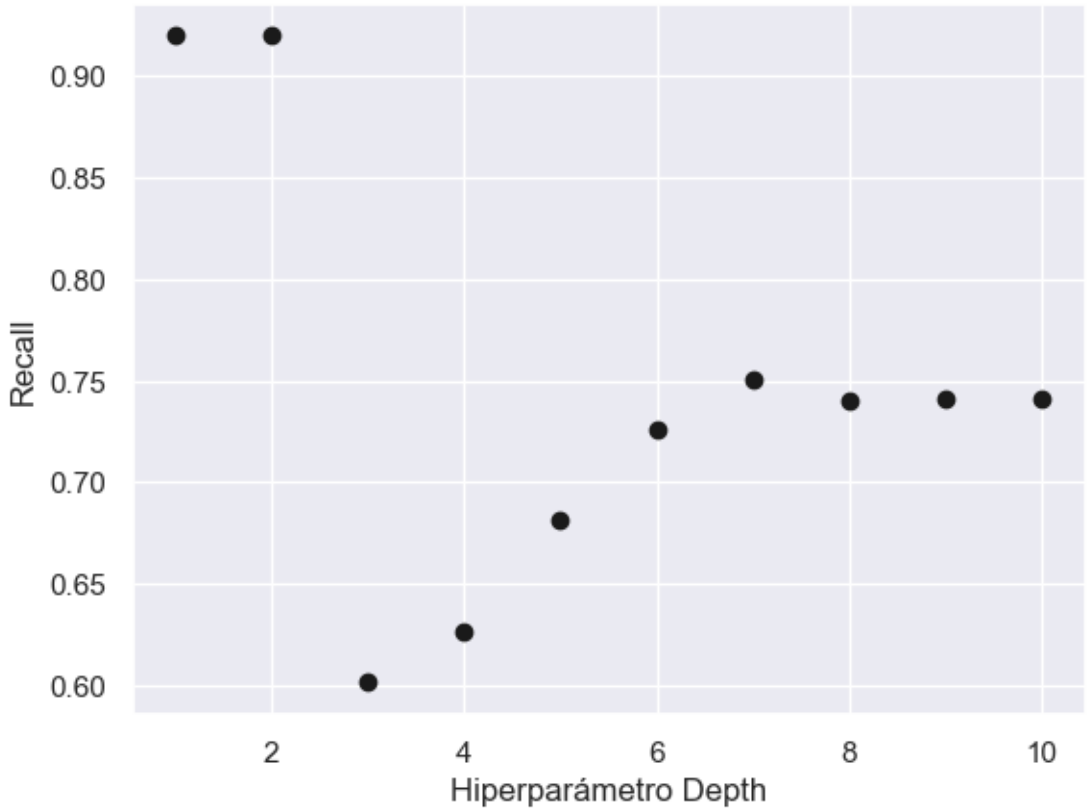
Una vez finalizada la búsqueda, obtenemos que nuestro hiperparámetro óptimo de profundidad es el siguiente:

[80]:
```
cv.best_estimator_
```

[80]:
```
DecisionTreeClassifier(max_depth=1)
```

Lo ilustramos de igual manera que lo hacíamos en KNN:

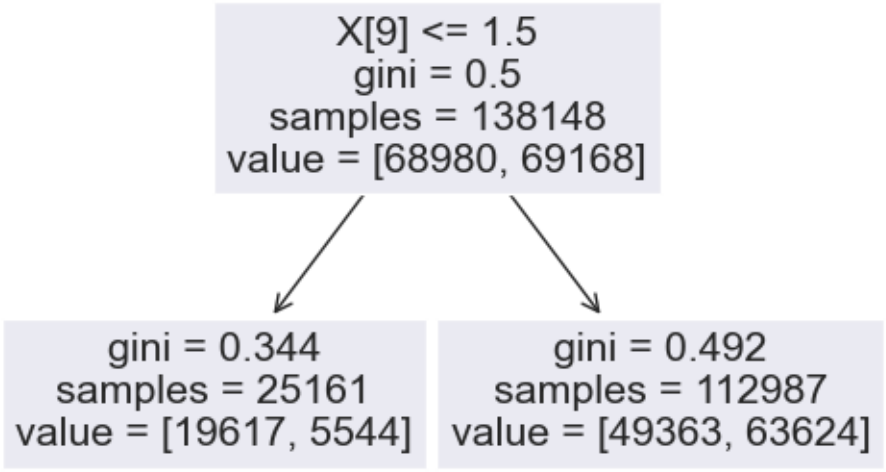
[81]:
```
plt.plot(ds, cv.cv_results_['mean_test_score'], 'ok')
plt.xlabel('Hiperparámetro Depth')
plt.ylabel('Recall')
plt.grid(True)
```



[83]:
```
arbol = DecisionTreeClassifier(max_depth=1)
arbol.fit(X_train, y_train)
```

[83]:
```
DecisionTreeClassifier(max_depth=1)
```

[84]:
```
from sklearn.tree import plot_tree

#ploteamos el árbol
plt.figure(figsize=(6, 4))
plot_tree(arbol)
plt.show()
```

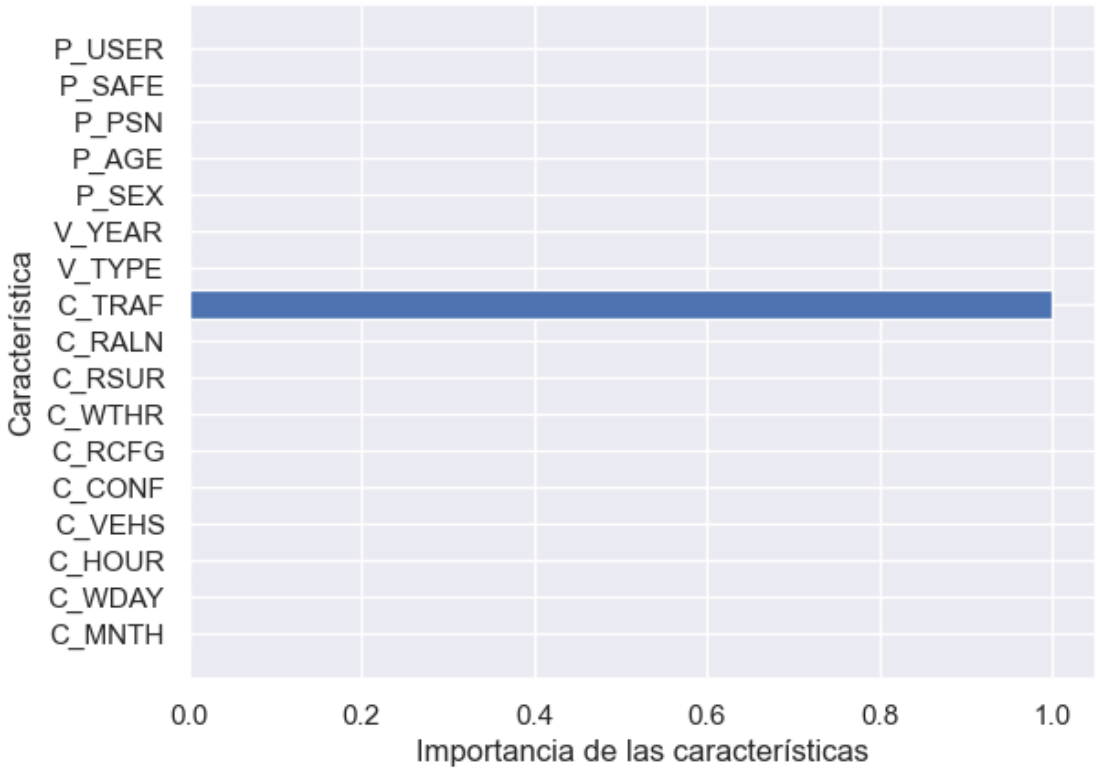


Como se puede observar en la salida previa, con este método ganamos mucho en interpretabilidad. El método que se emplea es muy sencillo: en cada nodo, se escoge uno o varios puntos críticos relacionados con alguna de las variables y, a partir de los mismos, se divide la muestra atendiendo a esos puntos. Es por tanto un método bastante ilustrativo en el que, además, se pueden identificar las variables importantes que realmente se han utilizado para clasificar.

En este caso, tras el resultado de CV, obteníamos que la profundidas máxima había de ser 1. Por tanto, el único nodo que tenemos es el inicial y, en consecuencia, sólo se usa una de las variables para clasificar. (X[9], que se corresponde con ***C_TRAF***).

En la siguiente salida, hemos querido ilustrar como, en efecto, esta es la única variable relevante para la clasificación cuando usamos este modelo.

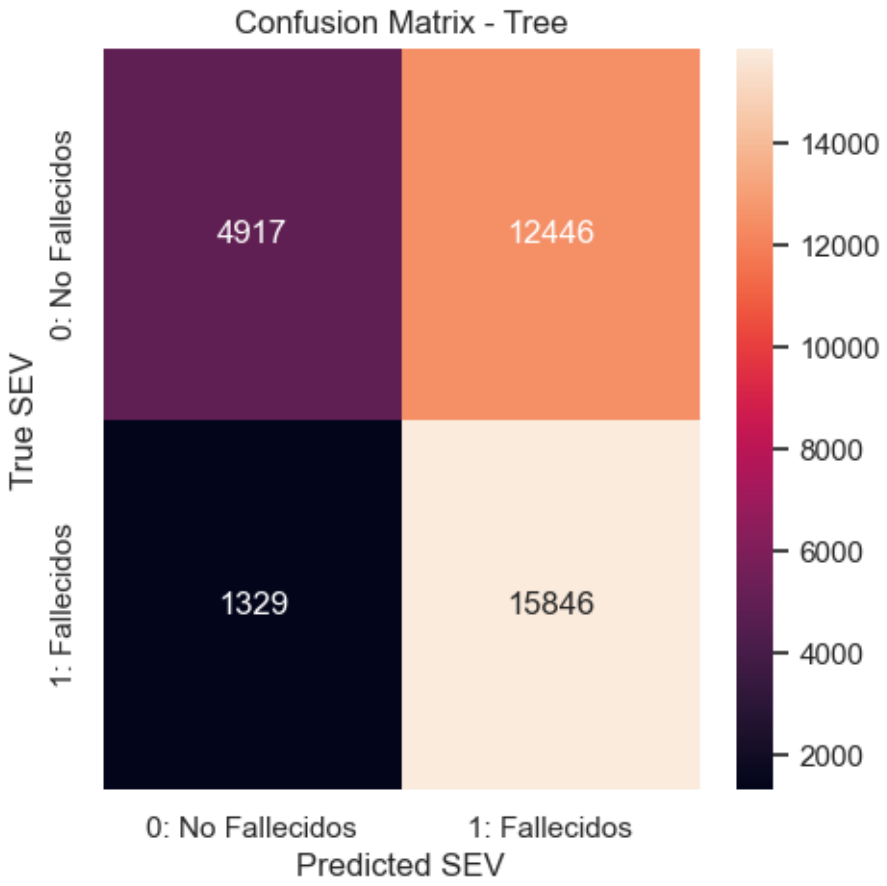
[85]:
```
caract = X_train_scaled_df.shape[1]

plt.barh(range(caract), arbol.feature_importances_) #barras horizontales

plt.yticks(np.arange(caract), X_train_scaled_df.columns)
plt.xlabel('Importancia de las características')
plt.ylabel('Característica')
plt.show()
```



A continuación, utilizamos el modelo para predecir sobre test:

[86]:
```
y_pred = arbol.predict(X_test)
```

[87]:
```
conf_matrix_Tree = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 5))
labels = ['0: No Fallecidos','1: Fallecidos']
sns.heatmap(conf_matrix_Tree, xticklabels=labels, yticklabels=labels,␣
 ↪annot=True, fmt="d");
plt.title("Confusion Matrix - Tree")
plt.ylabel('True SEV')
plt.xlabel('Predicted SEV')
plt.show()
print (classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.79	0.28	0.42	17363
1	0.56	0.92	0.70	17175
accuracy			0.60	34538
macro avg	0.67	0.60	0.56	34538
weighted avg	0.67	0.60	0.56	34538

En este caso, comentamos bastantes cosas: - En primer lugar, si atendemos al recall para '1', vemos que obtenemos un resultado muy muy bueno: 0.92, es decir, **somos capaces de identificar el 92% de los verdaderos accidentes con fallecimiento**. Teniendo en cuenta que lo que podríamos identificar como nuestro principal objetivo, es esto, es un dato muy positivo.

- Ahora bien, el trade of de obtener este resultado, no es ni mucho menos barato. El pago que estamos haciendo para lograr esta cifra es meter 12446 observaciones de falsos positivos. En este sentido, estamos incurriendo en un coste alto. La autoridad competente habría de valorar si le compensa incurrir en este coste o no. Claro, este suceso, disminuye el recall para '0' al 0.28: **sólo somos capaces de identificar correctamente el 28% de los accidentes que**

no causan muertes (de los negativos).

- **Otras consecuencias**: De nuevo, aunque según lo expuesto previamente no tendría por qué ser de especial relevancia para nosotros, la alta cantidad de falsos positivos obtenidos como resultado del trade of ante minimizar los falsos negativos, hace que la precisión al estimar los accidentes con fallecimiento (precision '1') caiga al 0.56. **NOTA**: esta cifra se calcula haciendo: Precision1 = TP/(TP+FP)

  Así, sólo el 56% de predicciones que dicen que el accidente causa fallecidos es cierto.

- Por su parte, la accuracy también se ve disminuida.

1.3.4 5.4. Random Forest

Para elegir nuestro mejor modelo Random Forest, haremos de nuevo una búsqueda CV. En esta ocasión, los hiperparámetros que optimizaremos serán:

- **Máxima Profundidad** de cada árbol del bosque.
- **Criterio**: determina la métrica utilizada para evaluar la calidad de una división en los árboles. Eligiendo entre "gini" y "entropy" se puede explorar qué métrica funciona mejor para el conjunto de datos.
- **N_estimators**: que determina el número de árboles en el bosque

Hay otros hiperparámetros que podrían haber sido también optimizados a traves de CV. Entre ellos, podemos encontrar: **min_samples_split**, que determina el número mínimo de observaciones necesarias para dividir un nodo interno en dos nodos hijos; **min_samples_leaf**, que especifica el número mínimo de observaciones necesarias en un nodo hoja…

Sin embargo, la optimización de algunos de estos hiperparámetros, sumada a la de los ya escogido, hacía que el número de modelos entrenados en la búsqueda nos condujese a un coste computacional que, quizás, no nos merezca la pena.

Además, al igual que en el resto de búsquedas realizadas hasta ahora, utilizamos CV-simple y no k-fold por motivos computacionales.

NOTA: OBVIAMENTE, SI UTILIZÁSEMOS K-FOLD, HARÍAMOS UNA BÚSQUEDA MÁS EXHAUSTIVA Y, PROBABLEMENTE, OBTENDRÍAMOS UNOS RESULTADOS ALGO MEJORES

[95]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state = 42)

ds = list(range(1,11))
criterion = ['gini','entropy']
n_s = [100, 200, 300]
h_parameters = {'max_depth': ds, 'criterion': criterion, 'n_estimators': n_s}

cv_simple = ShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
scoring = make_scorer(recall_score)

cv = GridSearchCV(rf, h_parameters, cv=cv_simple, n_jobs=-1, scoring=scoring)
```

```
cv.fit(X_train_scaled, y_train)
```

[95]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=1, random_state=42, test_size=0.2,
train_size=None),
             estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [100, 200, 300]},
             scoring=make_scorer(recall_score))
```

Una vez finalizada la búsqueda, sacamos los hiperparámetros óptimos obtenidos:

[96]:
```
cv.best_estimator_
```

[96]:
```
RandomForestClassifier(criterion='entropy', max_depth=2, n_estimators=200,
                       random_state=42)
```

[97]:
```
cv.cv_results_
```

[97]:
```
{'mean_fit_time': array([  9.00077486,  15.92099166,  25.74305964,  13.72214866,
         24.27702832,  35.82894278,  16.09693217,  31.41905594,
         48.12198305,  18.32891417,  36.32205391,  53.65559649,
         20.68004942,  44.58233953,  67.53939152,  24.48426294,
         50.7048316 ,  79.25309706,  27.45616937,  61.09754467,
         83.07116485,  29.81155205,  62.7268424 ,  98.49646187,
         38.91051626,  72.22869086, 101.42557311,  37.02539134,
         67.73937392, 102.97208214,   7.20984864,  15.29566002,
         24.85594654,  12.27189064,  23.85549855,  32.92649055,
         16.41849732,  30.74532127,  45.43522096,  19.50750923,
         39.03127122,  57.73897791,  21.00043011,  47.08337903,
         65.18140531,  25.48857808,  53.48433471,  73.81813693,
         28.68978715,  54.74820447,  85.41872907,  32.61863565,
         63.5825243 ,  85.22580481,  34.51361823,  63.23326111,
         76.45607042,  35.80251837,  53.88699174,  64.13097453]),
 'std_fit_time': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 'mean_score_time': array([0.42923117, 0.74074912, 1.42809987, 0.42937422,
1.01976609,
        1.38066292, 0.40097809, 1.04622674, 1.58466983, 0.65957689,
        1.19282961, 1.71055889, 0.50252175, 1.56949997, 1.98002267,
        0.84698033, 1.3334322 , 1.99289346, 0.52327967, 2.51967382,
        1.90849137, 0.86777639, 1.60028338, 3.01366234, 0.92201447,
        1.79512453, 2.65311265, 0.98692298, 1.97605658, 2.9194212 ,
        0.46478415, 0.88675189, 1.08731174, 0.40791106, 1.30267072,
```

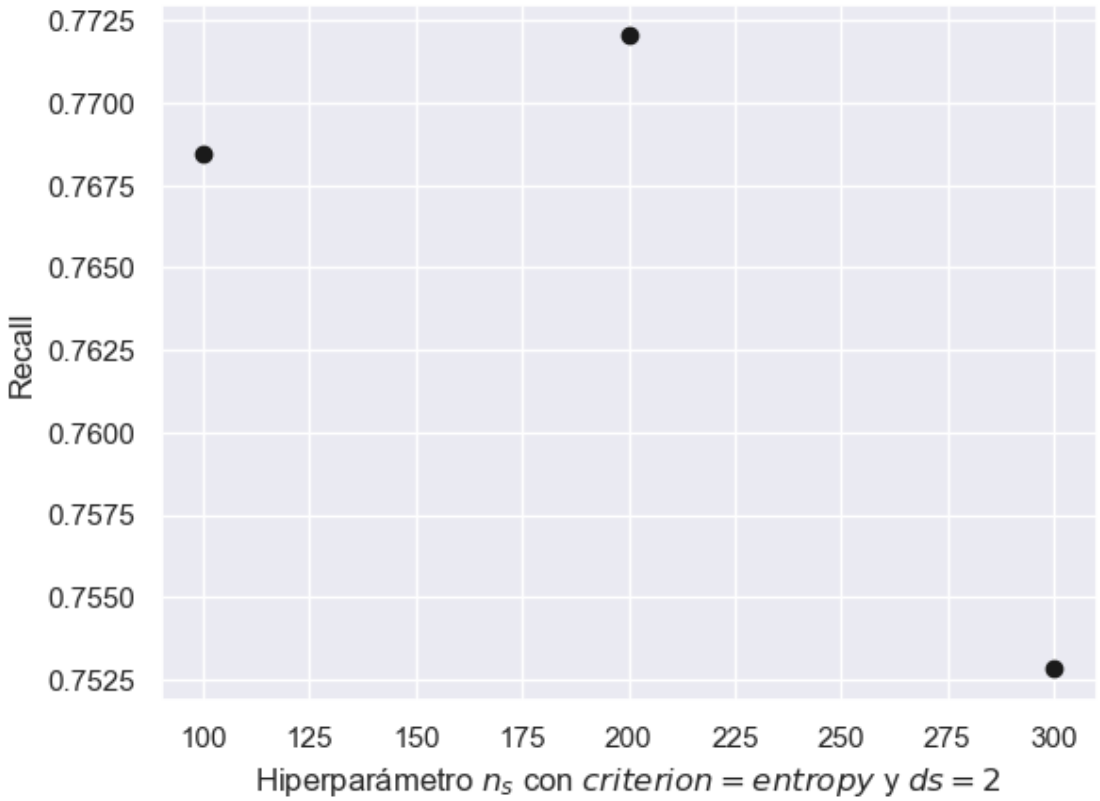
```
        1.47583294, 0.58061838, 1.09868312, 1.61638999, 0.59667444,
        1.24633265, 1.8047204 , 0.64358377, 1.27100492, 1.85339189,
        0.70596528, 1.67931151, 1.97701597, 0.76853991, 1.47446895,
        2.26208448, 0.54938889, 1.69060159, 2.02317214, 1.17720962,
        1.58407331, 0.9097898 , 1.26963425, 1.47436118, 1.05264997]),
 'std_score_time': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 'param_criterion': masked_array(data=['gini', 'gini', 'gini', 'gini', 'gini',
'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_depth': masked_array(data=[1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5,
5, 6, 6, 6,
                    7, 7, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10, 1, 1, 1, 2, 2,
                    2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8,
                    8, 9, 9, 9, 10, 10, 10],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
```

```
 'param_n_estimators': masked_array(data=[100, 200, 300, 100, 200, 300, 100,
200, 300, 100, 200,
                   300, 100, 200, 300, 100, 200, 300, 100, 200, 300, 100,
                   200, 300, 100, 200, 300, 100, 200, 300, 100, 200, 300,
                   100, 200, 300, 100, 200, 300, 100, 200, 300, 100, 200,
                   300, 100, 200, 300, 100, 200, 300, 100, 200, 300, 100,
                   200, 300, 100, 200, 300],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'criterion': 'gini', 'max_depth': 1, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 1, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 1, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 2, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 2, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 2, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 3, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 3, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 3, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 4, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 4, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 4, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 5, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 5, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 5, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 6, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 6, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 6, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 7, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 7, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 7, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 8, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 8, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 8, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 9, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 9, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 9, 'n_estimators': 300},
  {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 100},
  {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 200},
  {'criterion': 'gini', 'max_depth': 10, 'n_estimators': 300},
```

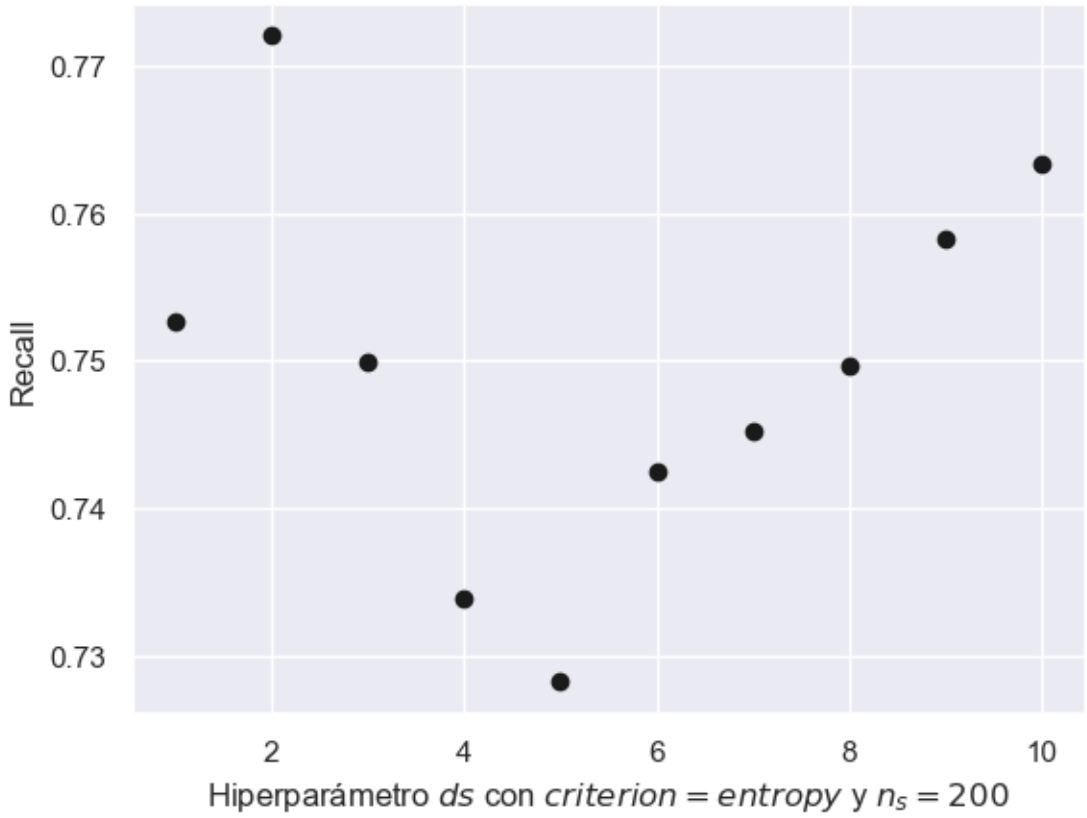
```
  {'criterion': 'entropy', 'max_depth': 1, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 1, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 1, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 2, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 2, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 2, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 4, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 4, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 4, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 5, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 5, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 5, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 8, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 8, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 8, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 9, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 9, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 9, 'n_estimators': 300},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 100},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 200},
  {'criterion': 'entropy', 'max_depth': 10, 'n_estimators': 300}],
 'split0_test_score': array([0.73833952, 0.74794441, 0.74474278, 0.7544932 ,
0.75209197,
       0.7502001 , 0.75791312, 0.74416066, 0.74859929, 0.72997162,
       0.73499236, 0.73622935, 0.72822528, 0.72938951, 0.73084479,
       0.73819399, 0.74044968, 0.73892163, 0.74961799, 0.74830823,
       0.74954522, 0.74845376, 0.75311067, 0.7542749 , 0.757331  ,
       0.75784036, 0.75871353, 0.76271556, 0.76111475, 0.76257004,
       0.74416066, 0.75260132, 0.75187368, 0.76846395, 0.7720294 ,
       0.75281962, 0.75158262, 0.74998181, 0.75180092, 0.73099032,
       0.73382813, 0.73615659, 0.72560576, 0.72822528, 0.72837081,
       0.73943098, 0.74248708, 0.74030415, 0.74110456, 0.74525213,
       0.74867205, 0.74656189, 0.74969075, 0.74990904, 0.75660336,
       0.75820418, 0.75922288, 0.76562614, 0.76337044, 0.76322491]),
 'mean_test_score': array([0.73833952, 0.74794441, 0.74474278, 0.7544932 ,
0.75209197,
       0.7502001 , 0.75791312, 0.74416066, 0.74859929, 0.72997162,
       0.73499236, 0.73622935, 0.72822528, 0.72938951, 0.73084479,
```

```
        0.73819399, 0.74044968, 0.73892163, 0.74961799, 0.74830823,
        0.74954522, 0.74845376, 0.75311067, 0.7542749 , 0.757331  ,
        0.75784036, 0.75871353, 0.76271556, 0.76111475, 0.76257004,
        0.74416066, 0.75260132, 0.75187368, 0.76846395, 0.7720294 ,
        0.75281962, 0.75158262, 0.74998181, 0.75180092, 0.73099032,
        0.73382813, 0.73615659, 0.72560576, 0.72822528, 0.72837081,
        0.73943098, 0.74248708, 0.74030415, 0.74110456, 0.74525213,
        0.74867205, 0.74656189, 0.74969075, 0.74990904, 0.75660336,
        0.75820418, 0.75922288, 0.76562614, 0.76337044, 0.76322491]),
 'std_test_score': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 'rank_test_score': array([47, 35, 38, 16, 21, 25, 12, 39, 32, 55, 51, 49, 58,
56, 54, 48, 43,
        46, 29, 34, 30, 33, 18, 17, 14, 13, 10,  6,  8,  7, 39, 20, 22,  2,
         1, 19, 24, 26, 23, 53, 52, 50, 60, 58, 57, 45, 41, 44, 42, 37, 31,
        36, 28, 27, 15, 11,  9,  3,  4,  5])}
```

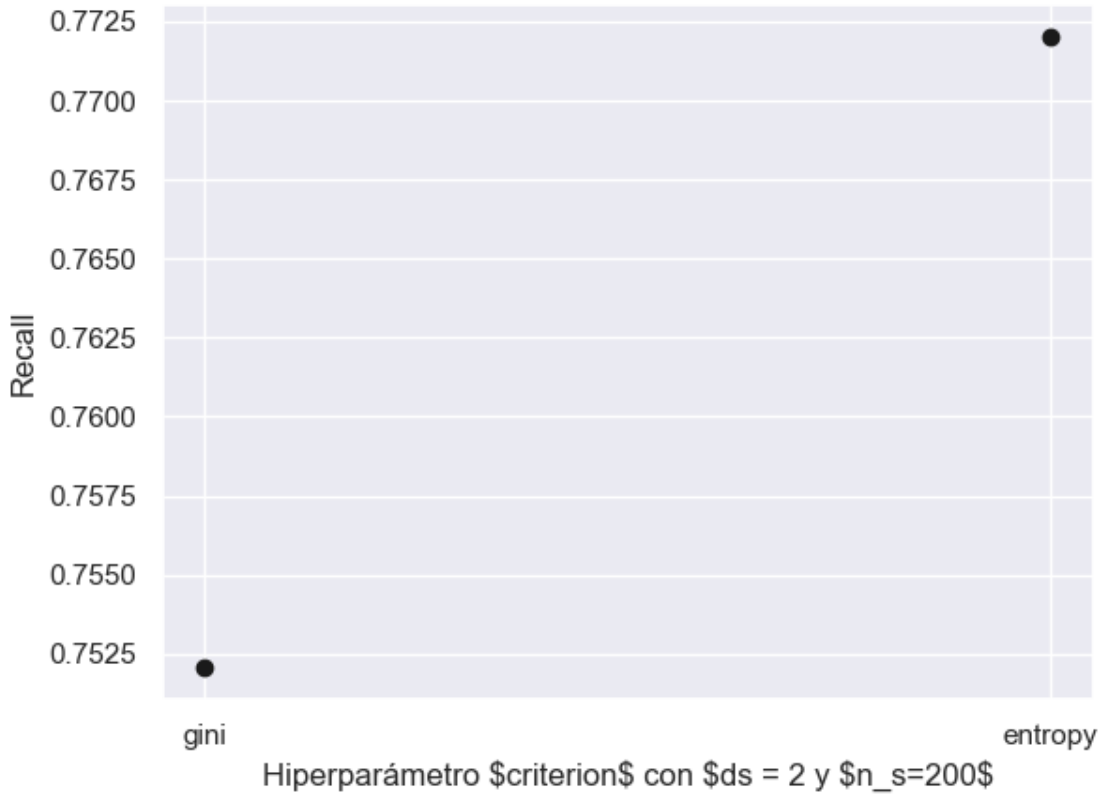
[98]:
```
plt.plot(n_s, cv.cv_results_['mean_test_score'][33:36], 'ok')
plt.xlabel('Hiperparámetro $n_s$ con $criterion = entropy$ y $ds = 2$')
plt.ylabel('Recall')
plt.grid(True)
```



[101]:
```
# el 31: porque empiezo a gráficar a partir de la posición trigésimoseungda␣
 ↪(incluida)
# len(...) porque lo hago hasta el final del array
# 3 porque lo hago dando saltos de 3 en 3
plt.plot(ds, cv.cv_results_['mean_test_score'][list(range(31, len(cv.
 ↪cv_results_['mean_test_score']), 3))], 'ok')
plt.xlabel('Hiperparámetro $ds$ con $criterion = entropy$ y $n_s=200$')
plt.ylabel('Recall')
plt.grid(True)
```



[102]:
```
plt.plot(criterion, cv.cv_results_['mean_test_score'][list(range(4, len(cv.
↪cv_results_['mean_test_score']), 30))], 'ok')
plt.xlabel('Hiperparámetro $criterion$ con $ds = 2 y $n_s=200$')
plt.ylabel('Recall')
plt.grid(True)
```



En las 5 salidas previas, hemos obtenido por orden, respectivamente:

1. Los hiperparámetros óptimos.
2. Un resumen del proceso de CV.
3. 3 gráficas que ilustra el punto 2 (el procedimiento CV Simple). En ellas, se puede ver cuál es cada hiperparámetro óptimo fijando los otros dos en su óptimo.

Una vez ilustrado de manera gráfica lo que estamos haciendo en el ajuste del modelo, vamos a ajustar el modelo con nuestro set de entrenamiento:

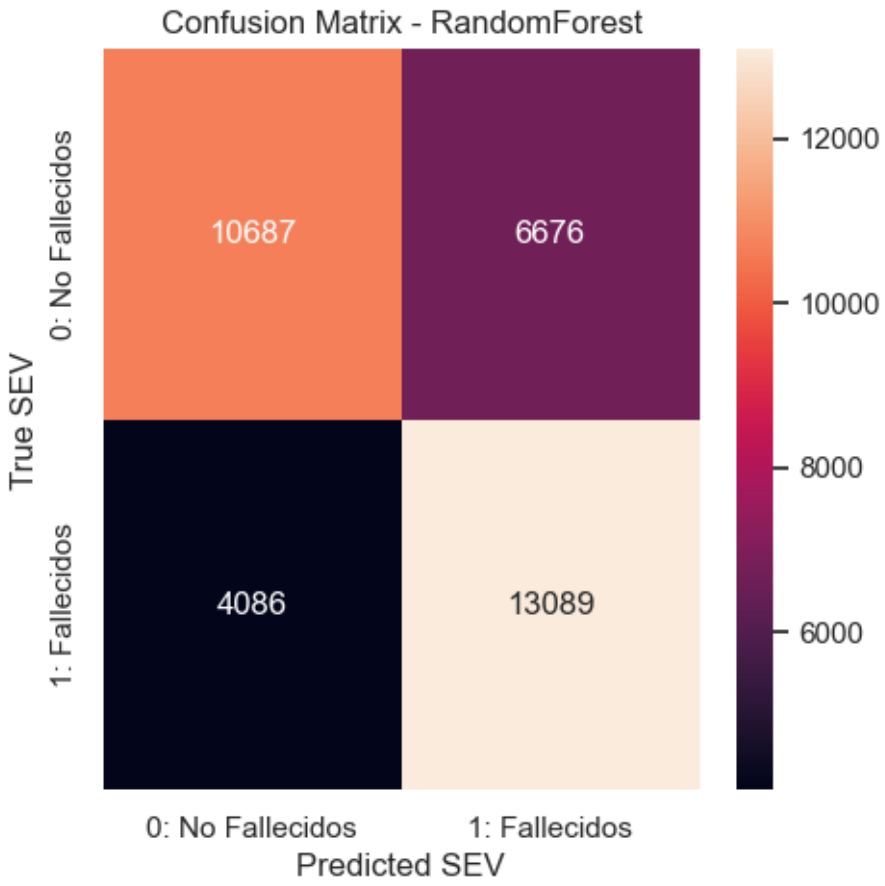
[104]:
```
rf = RandomForestClassifier(random_state = 42, n_estimators = 200, max_depth =␣
↪2, criterion = 'entropy')
rf.fit(X_train_scaled, y_train)
```

[104]:
```
RandomForestClassifier(criterion='entropy', max_depth=2, n_estimators=200,
                       random_state=42)
```

A continuación, utilizamos el modelo para predecir:

[105]:
```
y_pred = rf.predict(X_test_scaled)
```

[106]:
```
conf_matrix_rf = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 5))
labels = ['0: No Fallecidos','1: Fallecidos']
sns.heatmap(conf_matrix_rf, xticklabels=labels, yticklabels=labels, annot=True,␣
 ↪fmt="d");
plt.title("Confusion Matrix - RandomForest")
plt.ylabel('True SEV')
plt.xlabel('Predicted SEV')
plt.show()
print (classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.72	0.62	0.67	17363
1	0.66	0.76	0.71	17175
accuracy			0.69	34538
macro avg	0.69	0.69	0.69	34538
weighted avg	0.69	0.69	0.69	34538

En esta ocasión, obtenemos unos resultados de evaluación bastante más equilibrados que con el modelo de árboles de decisión. El recall para '1' es mayor que en el caso de KNN, alcanzando el 0.76, aunque es menor que en el caso de árboles de decisión. Sin embargo, que sea menor, no es a cambio de nada. Conseguimos elevar el recall para '0' de 0.28 a 0.62.

Además, la precisión es ahora también más equilibrada para ambos '0' y '1'. Por último, obtenemos una accuracy muy cercana a la de KNN y mayor a la de árbol.

A continuación, intentaremos aplicar aprendizaje supervisado a nuestro problema de clasificación.

Desde un primer momento, descartamos las redes convolucionales, puesto las redes convolucionales son especialmente efectivas cuando se trabaja con datos estructurados en forma de imágenes o datos con una estructura espacial similar, como señales de audio o series temporales. Están especialmente diseñadas para aprovechar las características locales y las relaciones espaciales en los datos. No es nuestro caso y de ahí la decisión.

Así, una opción que puede ser buena para clasificación es la del Perceptrón-Multicapa:

1.3.5 5.5. Perceptrón Multicapa

En este caso, hay también muchísimos hiperparámetros que podrían ser optimizados para encontrar el mejor modelo. Nosotros elegiremos optimizar 2:

1. **Tamaño de las capas ocultas**: para determinar el número de neuronas en cada capa.
2. **Función de activación**: para determinar la función que se aplica a la salida de cada neurona.

Algún otro hiperparámetro que es también importante es la **tasa de aprendizaje**. En nuestro caso, hemos decidido no entrar a su optimización puesto que recorrer una rejilla para determinar la tasa óptima elevaría bastante nuestro coste computacional, que con los ordenadores que tenemos ya es alto.

[107]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier()

tamaño_capas_ocultas = [(50,), (100,), (50, 50), (100, 100)]
f_act = ['reluc', 'logistic']
h_parameters = {'hidden_layer_sizes': tamaño_capas_ocultas, 'activation': f_act}

cv_simple = ShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
scoring = make_scorer(recall_score)

cv = GridSearchCV(mlp, h_parameters, cv=cv_simple, scoring = scoring)
cv.fit(X_train_scaled, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:692:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\model_selection\_validation.py:372: FitFailedWarning:
4 fits failed out of a total of 8.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
4 fits failed with the following error:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\model_selection\_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py", line 752, in fit
    return self._fit(X, y, incremental=False)
  File "C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py", line 384, in _fit
    self._validate_hyperparameters()
  File "C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py", line 493, in
_validate_hyperparameters
    raise ValueError(
ValueError: The activation 'reluc' is not supported. Supported activations are
['identity', 'logistic', 'relu', 'softmax', 'tanh'].

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\ProgramData\Anaconda3\lib\site-
```

```
packages\sklearn\model_selection\_search.py:969: UserWarning: One or more of the
test scores are non-finite: [       nan        nan        nan        nan
0.7595867  0.77457615
 0.74015863 0.78971113]
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[107]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=1, random_state=42, test_size=0.2,
train_size=None),
             estimator=MLPClassifier(),
             param_grid={'activation': ['reluc', 'logistic'],
                         'hidden_layer_sizes': [(50,), (100,), (50, 50),
                                                (100, 100)]},
             scoring=make_scorer(recall_score))
```

Sacamos los hiperparámetros óptimos:

[108]:
```
cv.best_estimator_
```

[108]:
```
MLPClassifier(activation='logistic', hidden_layer_sizes=(100, 100))
```

Entrenamos el modelo de acuerdo a esos hiperparámetros obtenidos:

[110]:
```
mlp = MLPClassifier(hidden_layer_sizes = (100,100), activation = 'logistic')
mlp.fit(X_train_scaled, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

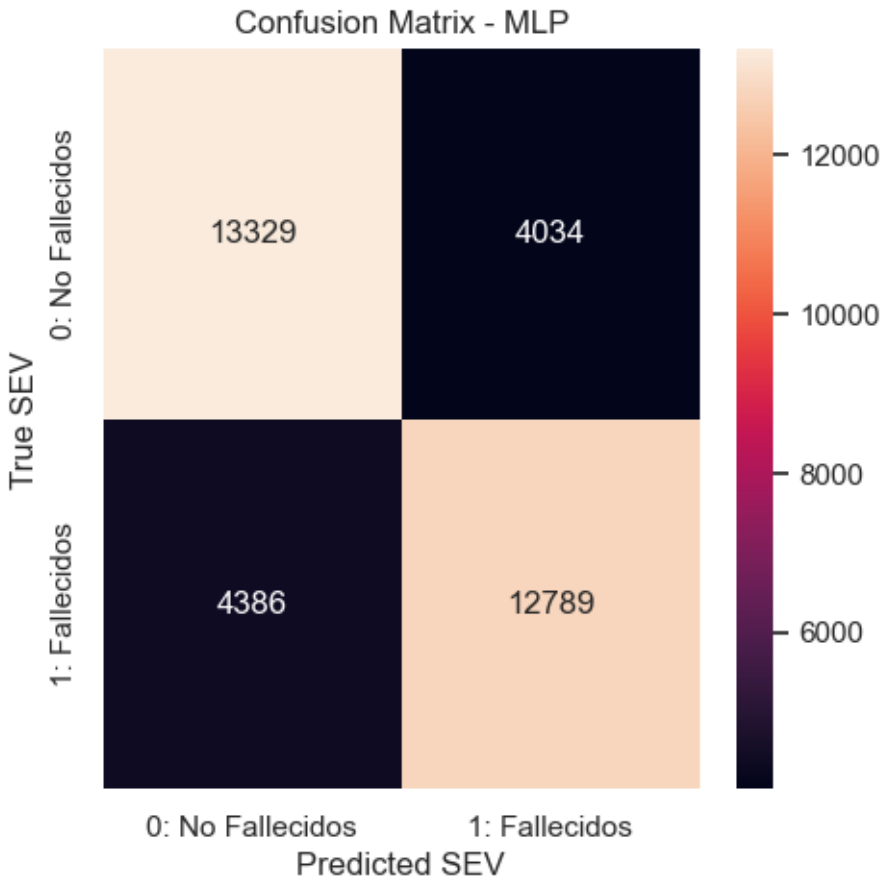
[110]:
```
MLPClassifier(activation='logistic', hidden_layer_sizes=(100, 100))
```

Predecimos sobre test:

[111]:
```
y_pred = mlp.predict(X_test_scaled)
```

[112]:
```
conf_matrix_mlp = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 5))
labels = ['0: No Fallecidos','1: Fallecidos']
sns.heatmap(conf_matrix_mlp, xticklabels=labels, yticklabels=labels,␣
 ↪annot=True, fmt="d");
plt.title("Confusion Matrix - MLP")
plt.ylabel('True SEV')
```

```
plt.xlabel('Predicted SEV')
plt.show()
print (classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.75	0.77	0.76	17363
1	0.76	0.74	0.75	17175
accuracy			0.76	34538
macro avg	0.76	0.76	0.76	34538
weighted avg	0.76	0.76	0.76	34538

En este caso, se obtienen unos resultados también equilibrados. Todas las métricas de evaluación son mejor que las obtenidas para KNN. El recall en este caso es algo peor que el obtenido para Random-Forest: 0.74 ahora frente a 0.76 antes, aunque el resto de métricas son mejores. En definitiva: resultados algo mejores que KNN y muy similares a los de Random Forest, con 2 centésimas menos de scoring para el recall '1', (que es el que nos interesa).

1.4 6. Comparación y selección del modelo óptimo

En primer lugar, haremos una tabla comparativa entre los diferentes recall de tipo 1 obtenidos:

MODELO	RECALL
K-NN	0.71
Regresión Logística	0.61
Árbol de Decisión	0.92
Random Forest	0.76
Perceptrón Multicapa	0.74

Atendiendo a la tabla previa y teniendo en cuenta única y exclusivamente el **Recall de tipo 1**, recomendaríamos a la institución pública correspondiente que se trabajase con el modelo de Árboles de Decisión, ya que es el que mayor porcentaje de observaciones con fallecimiento es capaz de corregir correctamente.

No obstante, y como ya se ha comentado antes, este buen resultado de recall no es gratis, sino que conlleva un deterioro de resultados en el resto de métricas que evalúan el modelo. En este caso, la prioridad por minimizar los falsos negativos, lleva a un elevado número de falsos positivos. Este coste se ve reflejado en el resto de métricas, como ya se dijo en el correspondiente apartado, y podría no compensarle a la institución correspondiente. Sería, por tanto, nuestra obligación exponer la existencia de este coste y que fuesen ellos los que decidiesen o no asumirlo.

En caso de que decidiesen no asumirlo, hemos visto que los dos siguientes modelos que mejor funcionan son Random Forest y Perceptrón Multicapa. El primero tiene un recall de tipo 1 ligeramente superior al segundo que, por su parte, demuestra superioridad en el resto de métricas. En función de cómo valorase el cliente ese 0.02 de diferencia, se acabaría usando uno u otro. Por nuestra parte, recomendaríamos usar el MLP, ya que ese 0.02 de diferencia es muy pequeño en relación a la diferencia en el resto de métricas:

MODELO	Precision de tipo 1	Precisión de tipo 0	Recall de tipo 0	Accuracy
Random Forest	0.66	0.72	0.62	0.69
Perceptrón Multicapa	0.76	0.75	0.77	0.76

En cualquier caso, creemos que la decisión dependería de la disposición a asumir costes del cliente como medio para cumplir su objetivo final: evitar fallecimientos en accidentes.

Por otro lado, es importante tener en cuenta que los costes de más en los que se incurran, no tienen por qué caer en saco roto, sino que podrían servir para solucionar otros problemas

Por último, nos gustaría hacer un **comentario acerca de los resultados obtenidos**: Creemos que teniendo más capacidad computacional y haciendo búsquedas CV más exhaustivas (optimizando más hiperparámetros, haciendo k-fold, etc.) podríamos haber obtenido resultados algo mejores (tampoco sabemos si mucho mejores, porque seguro que también se pueden mejorar aspectos del preprocessing). Sin embargo, el margen de mejora creemos que no merecía la pena, además de que, en algunos casos, estábamos obteniendo algún *Memory Error*. No obstante, creemos que, pese a no ser excelentísimos resultados en términos de métricas de evaluación, sí son aceptables y, en algún caso y para alguna métrica, incluso buenos.