Práctica Final Machine Learning

Ángela Cristina de Luna López
Marina Tévar Molina

Índice

BASE DE DATOS

En esta práctica abordaremos un problema de clasificación de detección de siniestros.
- Como variable objetivo tenemos una variable binaria que toma valores:

'1': percances graves (al menos una muerte)
'2': percances leves (sin muertes)
en un accidente de tráfico.

Realizaremos distintos modelos de clasificación para facilitar a la aseguradora la detección de posibles siniestros según características del conductor, medioambientales, del vehículo etc.
Además de previamente preparar la base de datos para ello.

	0	1	2	3	4
C_YEAR	1999	1999	1999	1999	1999
C_MNTH	01	01	01	01	01
C_WDAY	1	1	1	1	1
C_HOUR	20	20	20	08	08
C_SEV	2	2	2	2	2
C_VEHS	02	02	02	01	01
C_CONF	34	34	34	01	01
C_RCFG	UU	UU	UU	UU	UU
C_WTHR	1	1	1	5	5
C_RSUR	5	5	5	3	3
C_RALN	3	3	3	6	6
C_TRAF	03	03	03	18	18
V_ID	01	02	02	01	99
V_TYPE	06	01	01	01	NN
V_YEAR	1990	1987	1987	1986	NNNN
P_ID	01	01	02	01	01
P_SEX	M	M	F	M	M
P_AGE	41	19	20	46	05
P_PSN	11	11	13	11	99
P_ISEV	1	1	2	1	2
P_SAFE	UU	UU	02	UU	UU
P_USER	1	1	2	1	3

Tratamiento de valores faltantes

```
car.isnull().sum()
```

```
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
```

No hay demasiados, pero es una información falsa ya que haciendo uso de la leyenda que se nos proporciona junto al dataset vemos que muchos de estos missings están codificados de otra forma (U, UU, UUUU, Q, QQ, QQQQ, N, NN, NNNN, X, XX, XXXX), vemos cuántas veces aparece cada uno y los cambiaremos por valores NaN.

```
elementos = ['U','UU','UUUU','Q','QQ','QQQQ','N','NN','NNNN','X','XX','XXXX']
conteo = car.values.flatten().tolist().count(elementos[0]), car.values.flatten().tolist().count(elementos[1]), car.v
print(conteo)
```

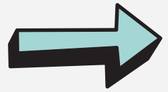
```
(1093700, 2047678, 324122, 213832, 609752, 0, 305497, 843004, 260256, 0, 0, 0)
```

```
car = car.replace(['U','UU','UUUU','Q','QQ','N','NN','NNNN'], np.nan)
```

Observamos la cantidad de missings que hay por variable:

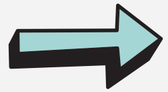
```
car.isnull().sum()
```

```
C_YEAR          0
C_MNTH        385
C_WDAY       1323
C_HOUR      59409
C_SEV           0
C_VEHS        544
C_CONF     463999
C_RCFG     648946
C_WTHR     102988
C_RSUR     248668
C_RALN     463312
C_TRAF     305501
V_ID          433
V_TYPE     283111
V_YEAR     584378
P_ID        10992
P_SEX      249740
P_AGE      395156
P_PSN       97677
P_ISEV     371412
P_SAFE    1234284
P_USER     175586
dtype: int64
```



Y vamos rellenando cada uno de estos valores con la moda a través del siguiente bucle:

```
l=list(car.columns.values)
for i in range(22):
    car[l[i]].fillna(car[l[i]].mode()[0], inplace=True)
```



```
car.isnull().sum()
```

```
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
```

Estudio de elementos duplicados

Comprobamos el número de duplicados que tenemos en la totalidad de los datos y los eliminamos:

```
duplicate_rows = car[car.duplicated()]
print("numero de duplicados: ", duplicate_rows.shape)
```

```
numero de duplicados:  (6691, 22)
```

```
car=car.drop_duplicates()
```

Por último vemos que se han eliminado correctamente:

```
duplicate_rows = car[car.duplicated()]
print("numero de duplicados: ", duplicate_rows.shape)
```

```
numero de duplicados:  (0, 22)
```

Conversión de tipos

El conjunto de datos está formado en su mayoría por variables de tipo object, y para poder trabajar bien con ellas las convertiremos a tipo float. Antes de hacer esta conversión, para la variable P_SEX haremos una codificación para asignar la F a un 1 y la M a un 2 mediante el uso del Label Encoder

```
from sklearn.preprocessing import LabelEncoder

P_SEX=car['P_SEX'].values
enc = LabelEncoder()
label_encoder = enc.fit(P_SEX)
P_SEX = label_encoder.transform(P_SEX) + 1
label_dict = {1: "F", 2:"M"}

car['P_SEX']=P_SEX

P_SEX
```

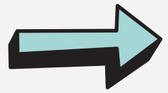
```
array([2, 2, 1, ..., 1, 2, 2])
```

Y ahora convertimos el resto de variables:

```
car[['C_YEAR','C_MNTH','C_WDAY','C_HOUR','C_SEV','C_VEHS','C_CONF','C_RCFG','C_WTHR','C_RSUR','C_RALN','C_TRAF',
     'V_ID','V_TYPE','V_YEAR','P_ID','P_SEX','P_AGE','P_PSN','P_ISEV','P_SAFE','P_USER']] = car[['C_YEAR','C_MNTH',
     'C_WDAY','C_HOUR','C_SEV','C_VEHS','C_CONF','C_RCFG','C_WTHR','C_RSUR','C_RALN','C_TRAF','V_ID','V_TYPE',
     'V_YEAR','P_ID','P_SEX','P_AGE','P_PSN','P_ISEV','P_SAFE','P_USER']].astype(float)
```

Outliers

Creamos una función que nos elimine los outliers, de tal forma que vaya recorriendo las filas detectando la cantidad de outliers que tiene y si se considera que hay más de 2 eliminar directamente la observación.
Primero miramos la longitud del data set para comparar cómo varía su dimesión una vez eliminados los outliers.



```
def filtrar_outlier_tukey(x):
    q1 = np.percentile(x, 25)
    q3 = np.percentile(x, 75)
    iqr = q3 - q1
    print("[q1=%f, q3=%f, iqr=%f]" % (q1, q3, iqr))

    floor = q1 - 1.5*iqr
    ceiling = q3 + 1.5*iqr
    print("[floor=%f, ceiling=%f]" % (floor, ceiling))

    outlier_indices = list(x.index[(x < floor)|(x > ceiling)])
    outlier_values = list(x[outlier_indices])

    return outlier_indices, outlier_values
```

```
for i in range(22):
    outlier_indices, outlier_values = filtrar_outlier_tukey(car[l[i]])
for i in range(22):
    contador=0
    borrar=0
    j=0
    while j<=(i+1)*21:
        if outlier_indices[j] in range(i*22, 22+(i)*22):
            contador=contador+1
            j=j+1
        else:
            j=j+1
        if contador>=2:
            if borrar==0:
                car.drop([i], axis=0, inplace=True)
            borrar=1
car
```

```
len(car)
```

```
5853714
```

	C_YEAR	C_MNTH	C_WDAY	C_HOUR	C_SEV	C_VEHS	C_CONF	C_RCFG	C_WTHR	C_RSUR	...	V_ID	V_TYPE	V_YEAR	P_ID	P_SEX	P_AGE
0	1999.0	1.0	1.0	20.0	2.0	2.0	34.0	2.0	1.0	5.0	...	1.0	6.0	1990.0	1.0	2.0	41.0
1	1999.0	1.0	1.0	20.0	2.0	2.0	34.0	2.0	1.0	5.0	...	2.0	1.0	1987.0	1.0	2.0	19.0
2	1999.0	1.0	1.0	20.0	2.0	2.0	34.0	2.0	1.0	5.0	...	2.0	1.0	1987.0	2.0	1.0	20.0
3	1999.0	1.0	1.0	8.0	2.0	1.0	1.0	2.0	5.0	3.0	...	1.0	1.0	1986.0	1.0	2.0	46.0
4	1999.0	1.0	1.0	8.0	2.0	1.0	1.0	2.0	5.0	3.0	...	99.0	1.0	2000.0	1.0	2.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5860400	2014.0	8.0	5.0	16.0	2.0	2.0	21.0	1.0	1.0	1.0	...	13.0	7.0	2000.0	1.0	2.0	24.0
5860401	2014.0	8.0	5.0	23.0	2.0	1.0	6.0	5.0	1.0	1.0	...	1.0	14.0	2006.0	1.0	2.0	29.0
5860402	2014.0	8.0	5.0	14.0	2.0	1.0	2.0	1.0	1.0	5.0	...	1.0	1.0	2006.0	1.0	1.0	18.0
5860403	2014.0	8.0	5.0	22.0	1.0	1.0	6.0	1.0	2.0	4.0	...	1.0	22.0	2000.0	1.0	2.0	67.0
5860404	2014.0	8.0	5.0	22.0	1.0	1.0	6.0	1.0	2.0	4.0	...	1.0	22.0	2000.0	2.0	2.0	10.0

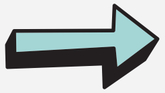
5853714 rows × 22 columns

Observamos que no tenemos más de dos outliers por fila por tanto decidimos no eliminarlos porque podríamos perder información importante.

Balanceo de datos

```
car['C_SEV'].value_counts()
```

```
2.0    5755101
1.0      98613
Name: C_SEV, dtype: int64
```



Vemos que en el caso de la variable objetivo tenemos una gran descompensación entre el valor 1 y 2

Hacemos uso de las técnicas de balanceo para ver cómo se comportan los datos con cada una de ellas.

En principio el análisis gráfico lo haremos con la técnica de undersampling ya que consideramos que para este caso puede dar resultados más realistas y es computacionalmente más eficiente.
La técnica de oversampling es inviable ya que tenemos 5 millones de datos.

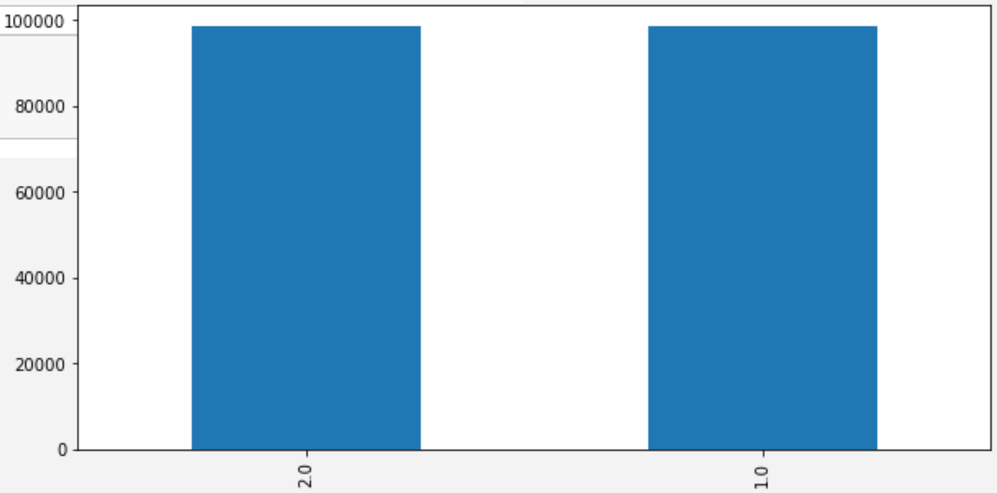
Para el caso de undersampling utilizaremos la técnica de submuestreo aleatorio para abordar el desequilibrio de clases en un conjunto de datos.

```
#Undersampling
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state = 0)
X_rus, y_rus = rus.fit_resample(car[['C_YEAR', 'C_MNTH', 'C_WDAY', 'C_HOUR', 'C_SEV', 'C_VEHS',
       'C_CONF', 'C_RCFG', 'C_WTHR', 'C_RSUR', 'C_RALN', 'C_TRAF', 'V_ID', 'V_TYPE', 'V_YEAR',
       'P_ID', 'P_SEX', 'P_AGE', 'P_PSN', 'P_ISEV', 'P_SAFE', 'P_USER']], car[['C_SEV']])
print('Filas totales:', len(rus.sample_indices_))
```

```
Filas totales: 197226
```

```
car_undersampled = pd.DataFrame(X_rus)
car_undersampled['C_SEV']=y_rus
```

```
car_undersampled['C_SEV'].value_counts().plot.bar(figsize=(10,5))
plt.show()
```



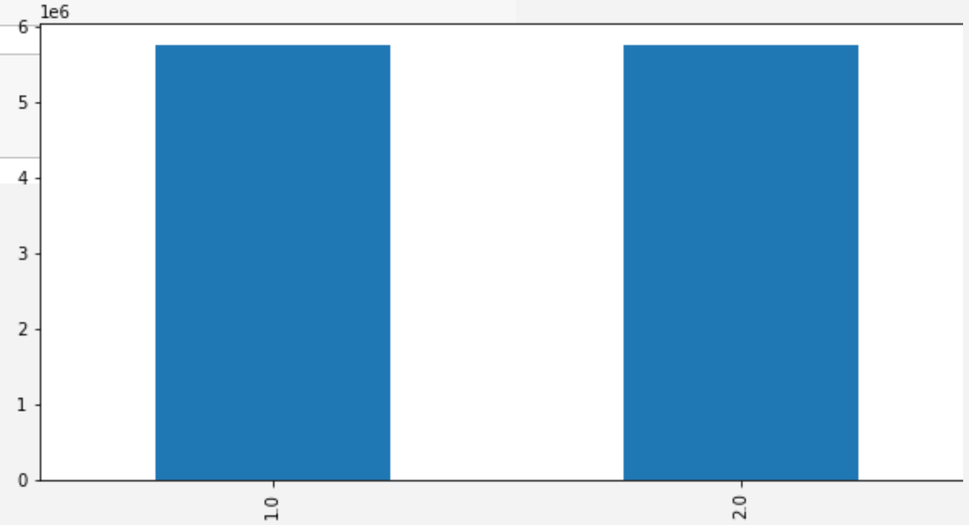
Para el caso de oversampling utilizaremos la técnica SMOTE que aborda este desequilibrio generando ejemplos sintéticos de la clase minoritaria para equilibrar la proporción de las clases.

```
#Oversampling
from imblearn.over_sampling import SMOTE

ros = SMOTE(random_state = 0)
X_res, y_res = ros.fit_resample(car[['C_YEAR', 'C_MNTH', 'C_WDAY', 'C_HOUR', 'C_SEV', 'C_VEHS',
       'C_CONF', 'C_RCFG', 'C_WTHR', 'C_RSUR', 'C_RALN', 'C_TRAF', 'V_ID', 'V_TYPE', 'V_YEAR',
       'P_ID', 'P_SEX', 'P_AGE', 'P_PSN', 'P_ISEV', 'P_SAFE', 'P_USER']], car[['C_SEV']])
print('Filas totales:' , len(y_res))
```

```
Filas totales: 11510202
```

```
car_oversampled = pd.DataFrame(X_res)
car_oversampled['C_SEV']=y_res
```

```
car_oversampled['C_SEV'].value_counts().plot.bar(figsize=(10,5))
plt.show()
```



Análisis gráfico y Encoding

En esta sección del trabajo contestaremos a las siguientes preguntas clave a través del análisis gráfico y de la codificación de variables, para así obtener información de la relación entre ellas.

¿Qué tipos de vehículos y conductores son más propensos a tener accidentes?

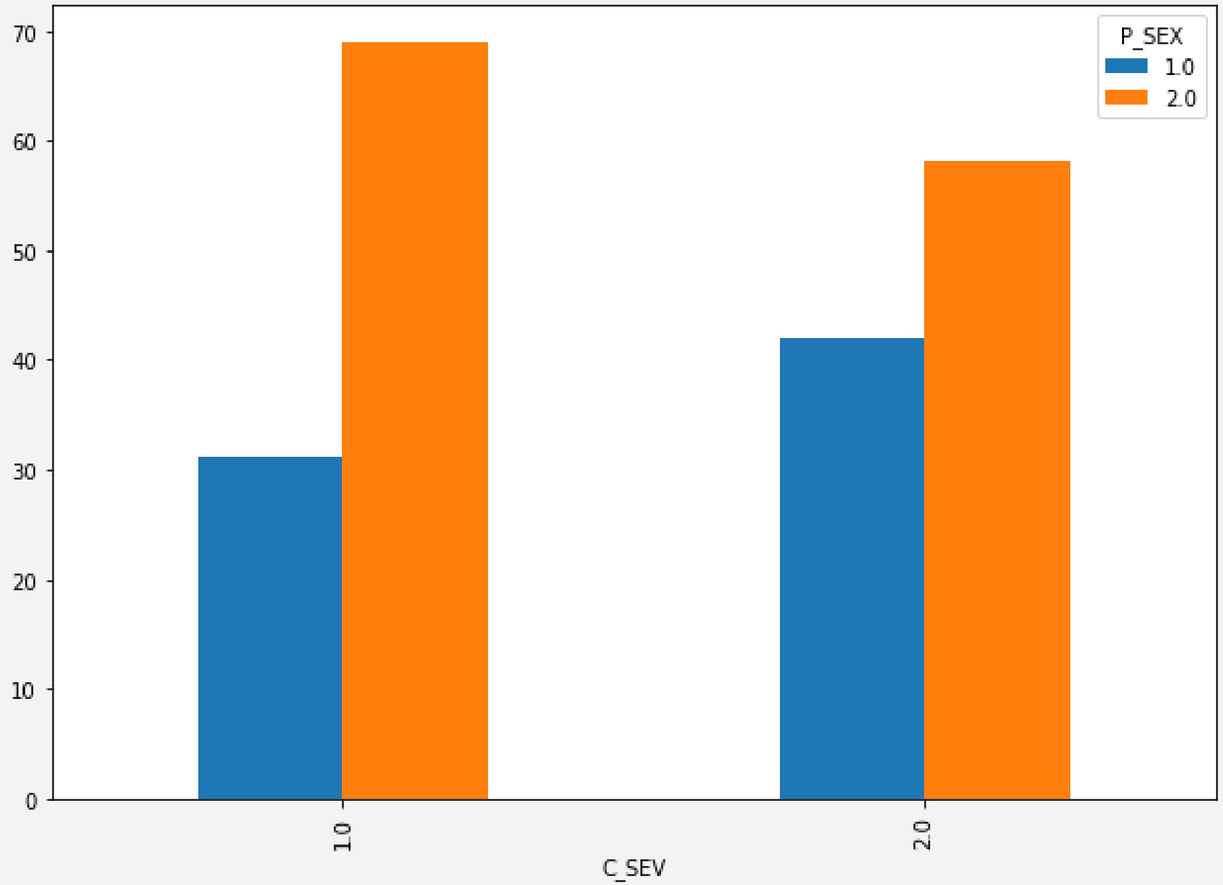
¿Qué es lo que más contribuye a que existan fallecimientos en un accidente?

¿Hay algún tipo de relación con temperaturas medias, precipitación media del día/mes, nieve...?
¿a más días festivos o de vacaciones, más accidentes?
etc.

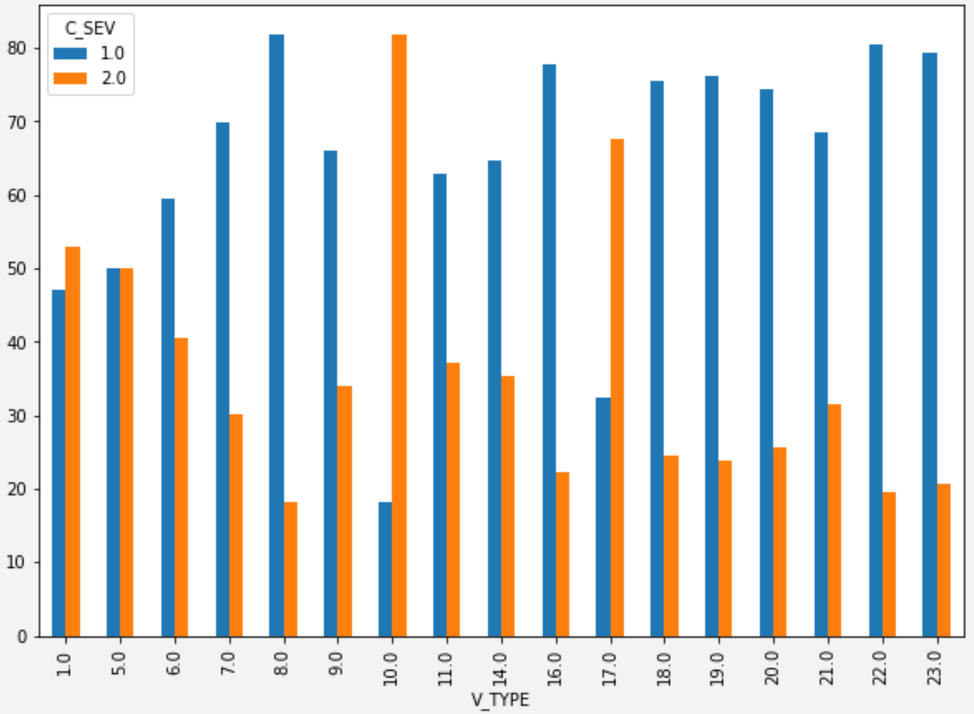
Para contestar a la pregunta:

¿Qué tipos de vehículos y conductores son más propensos a tener accidentes?

Exploramos los siguientes gráficos:



Observamos como los hombres son más propensos a tener accidentes, ya sean graves o leves, mientras que las mujeres suelen sufrir menos y si los sufren suelen ser leves .



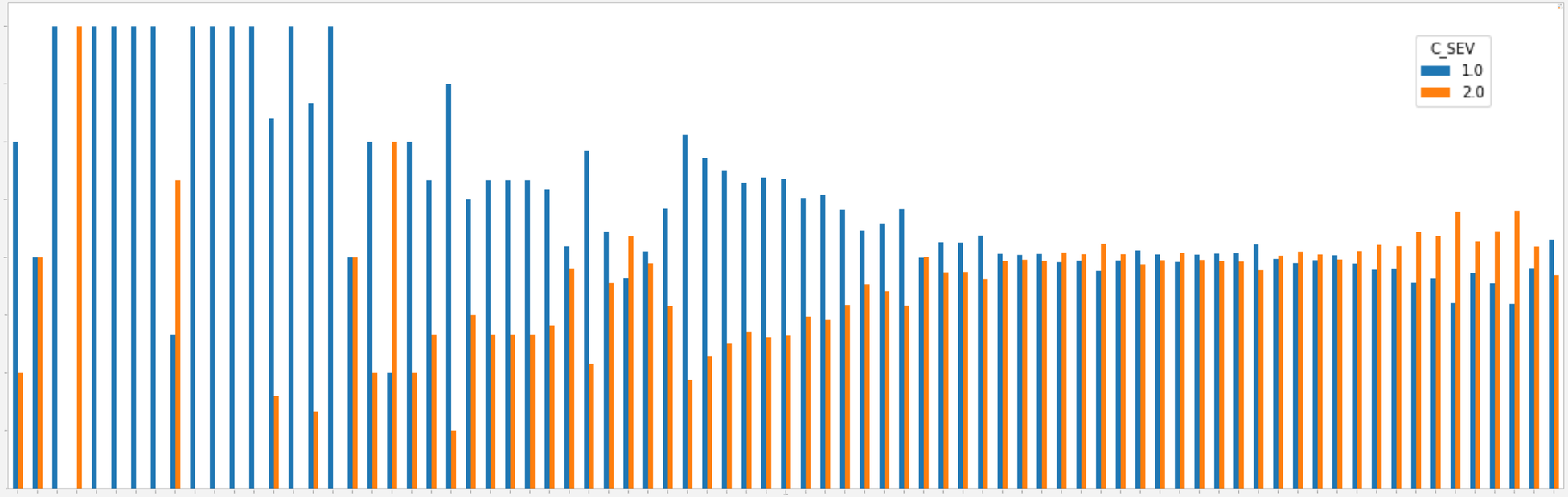
- 1: Light Duty Vehicle
- 5: Panel/cargo van
- 6: Other trucks and vans
- 7: Unit trucks
- 8: Road tractor
- 9: School bus
- 10: Smaller school bus
- 11: Urban and Intercity Bus
- 14: Motorcycle and moped
- 16: Off road vehicles
- 17: Bicycle
- 18: Purpose-built motorhome
- 19: Farm equipment
- 20: Construction equipment
- 21: Fire engine
- 22: Snowmobile
- 23: Street car

Los vehículos con los que se tienen más accidentes:

- Graves, son Road tractor (8), Snowmobile (22) y Street car (23).
- Leves, son Smaller school bus (10) o Bicycle (17).

Mientras que los vehículos con los que se tienen menos accidentes:

- Graves, es Smaller school bus (10), con el que apenas hay accidentes de este tipo.
- Leves, son Road tractor (8), Snowmobile (22) y Street car (23)



Vemos como están distribuidos los accidentes según la gravedad a lo largo de todos los años:

- Hasta 1955 la gran mayoría de accidentes eran de gravedad, esto se puede deber a que los coches de la época no estaban equipado con muchas medidas de seguridad .
- Entre 1955 y 1989 podemos ver como cada vez las cifras de accidentes leves van creciendo hasta igualarse con las de los accidentes graves.
- Desde 1989 y hasta el 2015 observamos como cada vez hay más accidentes leves y en los últimos años de estudio como las cifras de accidentes graves son mucho menores que las de los leves, debido a la cantidad de medidas de seguridad.

Hacemos un Encoding para observar qué variables influyen sobre la categoría de C_SEV de accidentes con muertes

```
from sklearn.preprocessing import OneHotEncoder
```

```
#ONE HOT ENCODING PARA 'C_SEV'
ohe = OneHotEncoder()
oh_array = ohe.fit_transform(car_undersampled['C_SEV'].values.reshape(-1, 1)).toarray()
oh_df = pd.DataFrame(oh_array,columns=['C_SEV_1','C_SEV_2'])
oh_df
```

	C_SEV_1	C_SEV_2
0	1.0	0.0
1	1.0	0.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0
...	...	...
197221	0.0	1.0
197222	0.0	1.0
197223	0.0	1.0
197224	0.0	1.0
197225	0.0	1.0

197226 rows × 2 columns

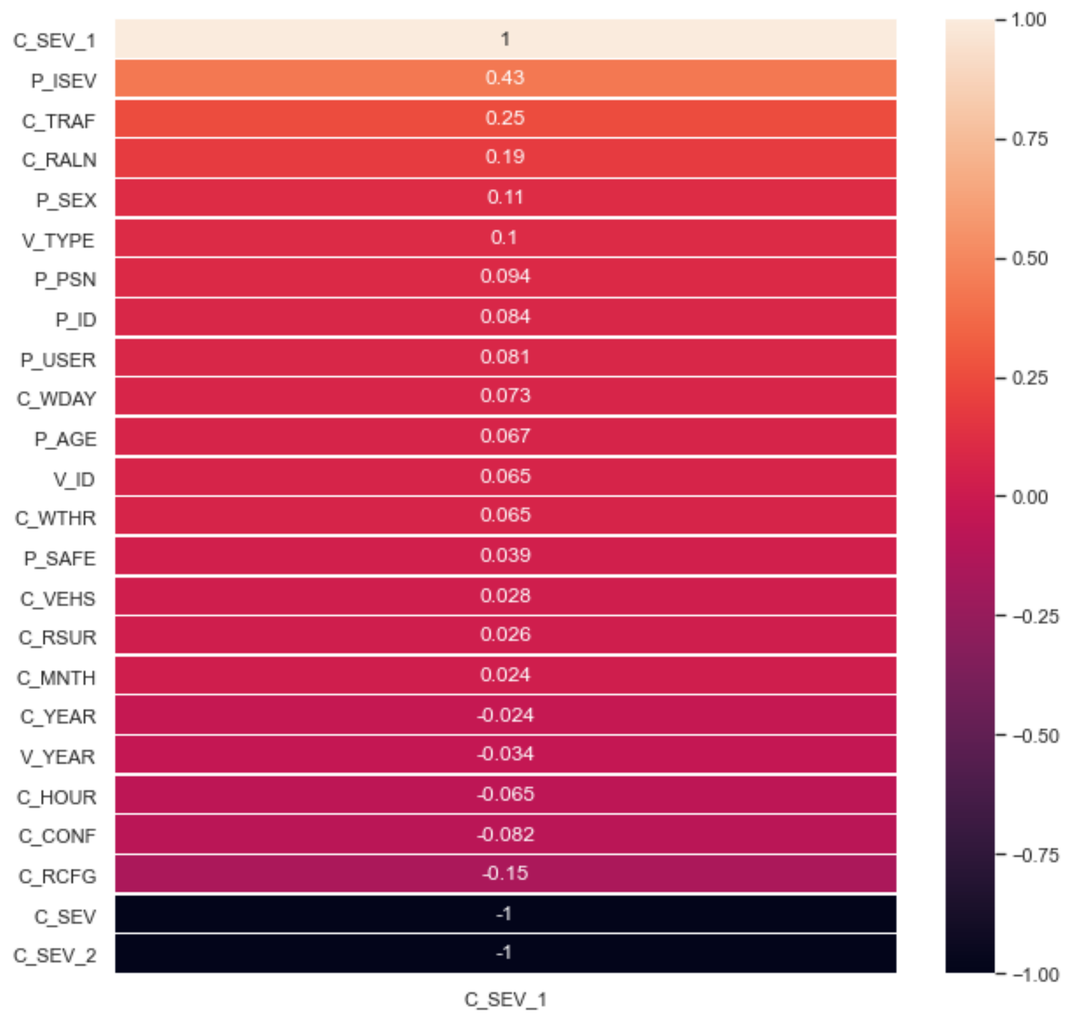
Para contestar a la pregunta:

¿Qué es lo que más contribuye a que existan fallecimientos en un accidente?

Exploramos los siguientes gráficos:

```
car_undersampled=car_undersampled.merge(oh_df,how='left',left_index=True,right_index=True)
```

```
corr = car_undersampled.corr()[['C_SEV_1']].sort_values(by='C_SEV_1', ascending=False)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr, annot=True,linewidths=.5, ax=ax)
plt.show()
```



La variable que más correlación tiene es P_ISEV esto es debido a que ambas explican lo mismo, estudian la gravedad de los accidentes.

La siguiente con más correlación es C_TRAF, que nos indica las colisiones debido a diversos controles de tráfico, por lo que podemos decir que esta es la mayor causa por la que existen fallecimientos en los accidentes;

A continuación estudiamos más a fondo qué variable dentro de C_TRAF es la que más correlación tiene, para ello realizamos otro Encoding.

```
#ONE HOT ENCODING PARA 'C_TRAF'
ohe = OneHotEncoder()
oh_array = ohe.fit_transform(car_undersampled['C_TRAF'].values.reshape(-1, 1)).toarray()
oh_df = pd.DataFrame(oh_array,columns=['C_TRAF_1','C_TRAF_2','C_TRAF_3','C_TRAF_4','C_TRAF_5','C_TRAF_6','C_TRAF_7',
oh_df
```

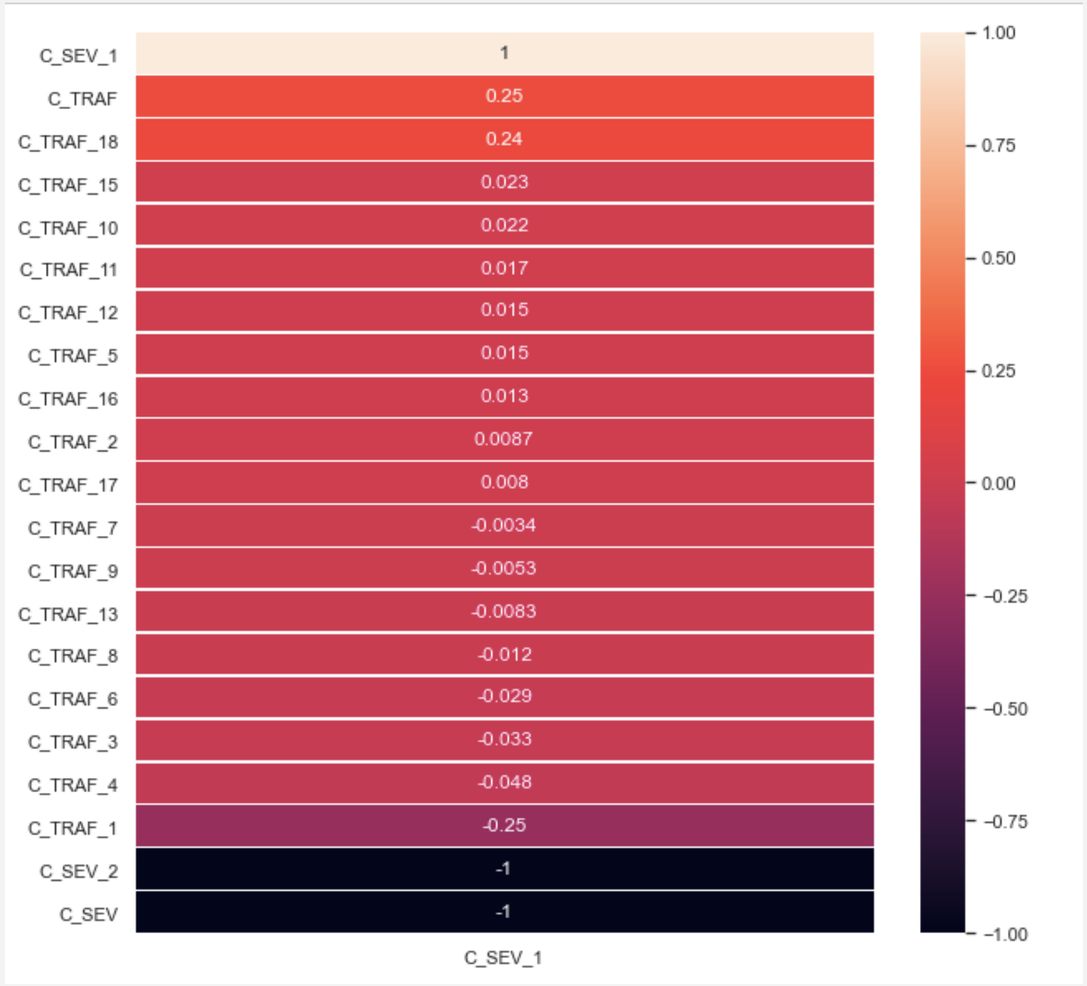
	C_TRAF_1	C_TRAF_2	C_TRAF_3	C_TRAF_4	C_TRAF_5	C_TRAF_6	C_TRAF_7	C_TRAF_8	C_TRAF_9	C_TRAF_10	C_TRAF_11	C_TRAF_12	C_TRAF
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
197221	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
197222	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
197223	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
197224	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
197225	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

197226 rows × 17 columns

```
car_undersampled2 =car_undersampled.merge(oh_df,how='left',left_index=True,right_index=True)
```

```
car_undersampled2.drop(['C_MNTH','C_HOUR','C_YEAR','C_WDAY','C_VEHS','C_CONF','C_RCFG','C_WTHR','C_RSUR','C_RALN',
                        'V_ID','V_TYPE','V_YEAR','P_ID','P_SEX','P_AGE','P_PSN','P_ISEV','P_SAFE','P_USER'],axis=1,i
```

```
corr = car_undersampled2.corr()[['C_SEV_1']].sort_values(by='C_SEV_1', ascending=False)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr, annot=True,linewidths=.5, ax=ax)
plt.show()
```



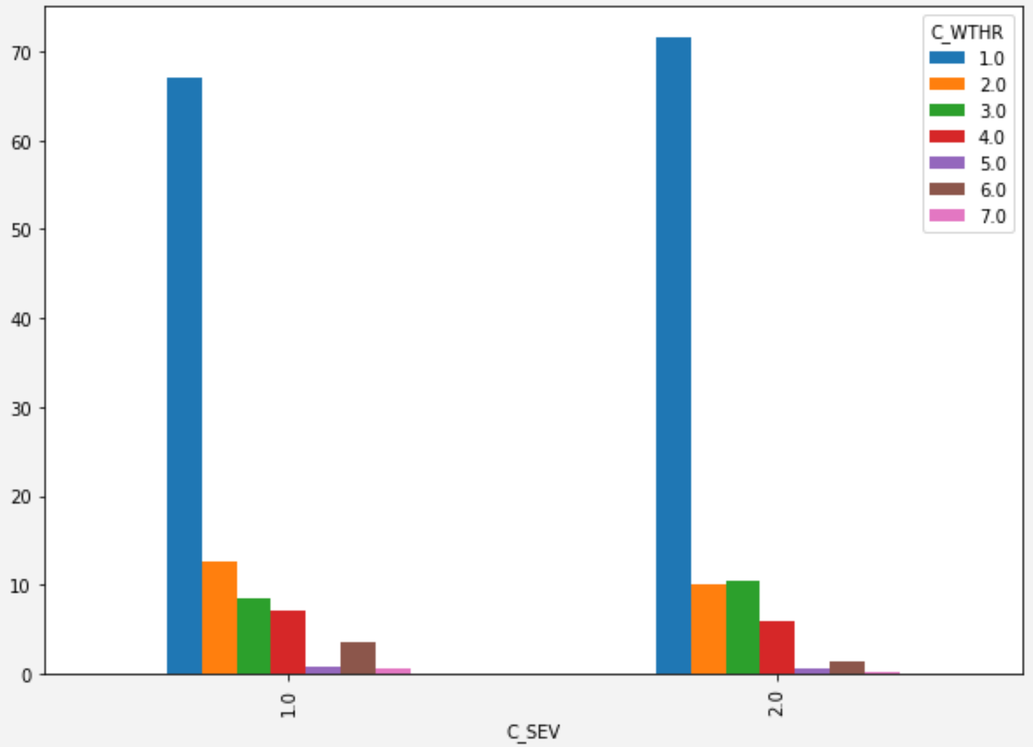
Podemos ver que C_SEV_1 tiene una mayor correlación con C_TRAF_18, No control present, algo bastante coherente ya que al no haber ningún tipo de control la gente conduce con menos cuidado y a mayor velocidad y esto solo hace que las probabilidades de sufrir accidentes sean mayores.

Por lo que lo que más contribuye a que existan fallecimientos en un accidente es la ausencia de controles de tráfico.

Para contestar a la pregunta:

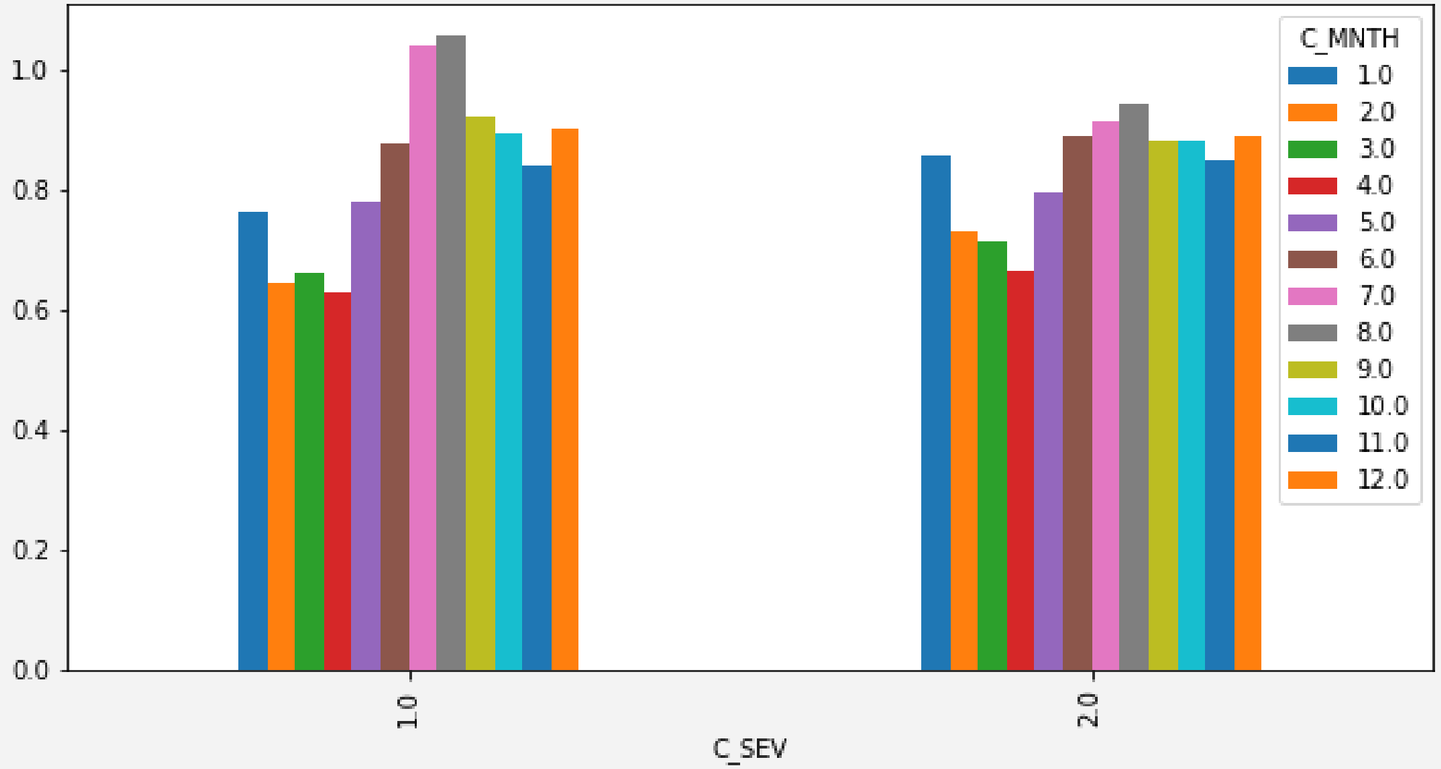
¿Hay algún tipo de relación con temperaturas medias, precipitación media del día/mes, nieve...?
¿a más días festivos o de vacaciones, más accidentes?
etc.

Exploramos los siguientes gráficos:



Observamos que tanto los accidentes graves como los leves ocurren mayoritariamente cuando el día esta despejado y con sol, esto puede ocurrir ya que durante estos días

- Hay un mayor flujo de vehículos en las carreteras debido a las vacaciones y los viajes.
- Un exceso de confianza ya que cuando el clima es perfecto, algunos conductores pueden volverse demasiado confiados.
- O debido a la fatiga o somnolencia, ya que los días soleados pueden provocar una sensación de relajación.



Vemos como están distribuidos los accidentes según su gravedad en función del mes en el que nos encontremos.

- Podemos ver claramente como los accidentes tanto graves como leves ocurren más durante aquellos meses con vacaciones como son, julio, agosto, septiembre y diciembre.
- Mientras que en los meses que no hay vacaciones como son febrero, marzo, abril y mayo el número de accidentes de ambos tipos es considerablemente menor.

Hacemos un Encoding para observar qué variables influyen más sobre la categoría de C_SEV de accidentes con muertes

```
ONE HOT ENCODING PARA 'C_WTHR'
he = OneHotEncoder()
h_array = ohe.fit_transform(car_undersampled['C_WTHR'].values.reshape(-1, 1)).toarray()
h_df = pd.DataFrame(oh_array,columns=['C_WTHR_1','C_WTHR_2','C_WTHR_3','C_WTHR_4','C_WTHR_5','C_WTHR_6','C_WTHR_7'])
h_df
```

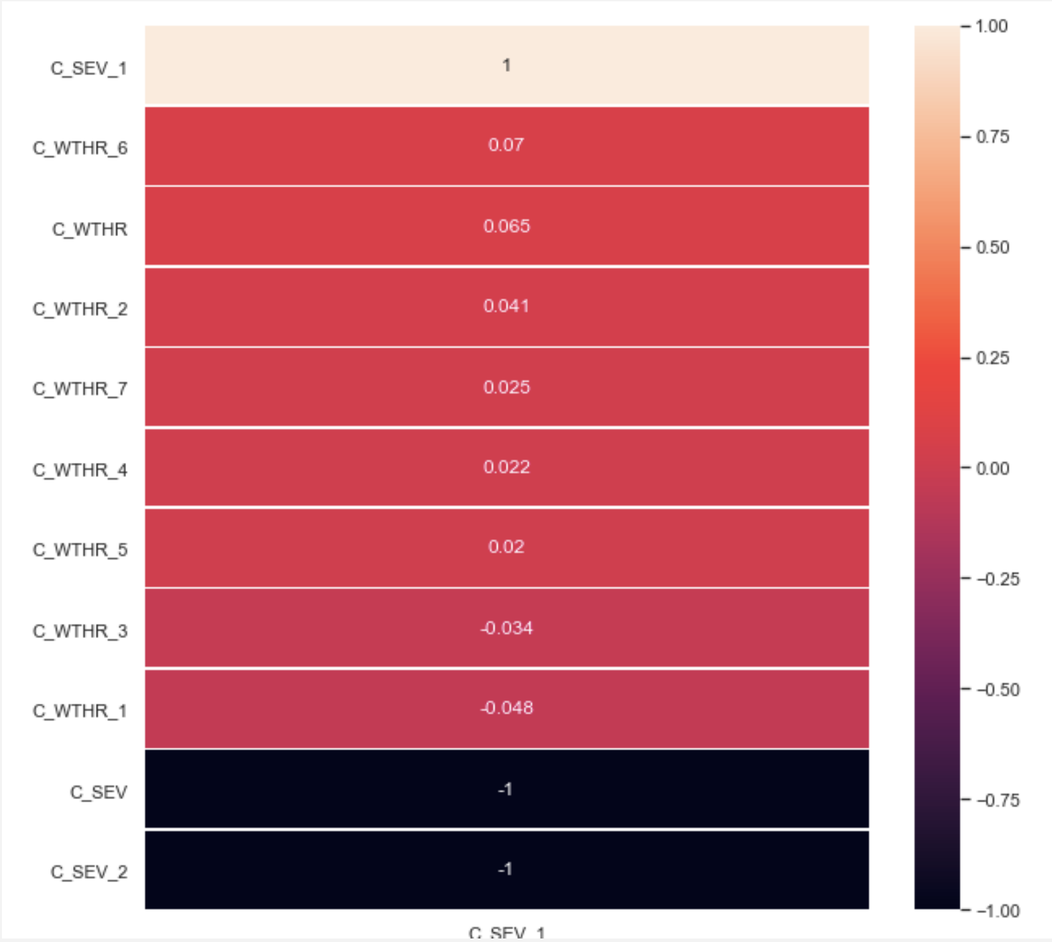
	C_WTHR_1	C_WTHR_2	C_WTHR_3	C_WTHR_4	C_WTHR_5	C_WTHR_6	C_WTHR_7
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
197221	1.0	0.0	0.0	0.0	0.0	0.0	0.0
197222	1.0	0.0	0.0	0.0	0.0	0.0	0.0
197223	1.0	0.0	0.0	0.0	0.0	0.0	0.0
197224	1.0	0.0	0.0	0.0	0.0	0.0	0.0
197225	0.0	1.0	0.0	0.0	0.0	0.0	0.0

197226 rows × 7 columns

```
car_undersampled3 =car_undersampled.merge(oh_df,how='left',left_index=True,right_index=True)
```

```
car_undersampled3.drop(['C_MNTH','C_HOUR','C_YEAR','C_WDAY','C_VEHS','C_CONF','C_RCFG','C_TRAF','C_RSUR','C_RALN',
            'V_ID','V_TYPE','V_YEAR','P_ID','P_SEX','P_AGE','P_PSN','P_ISEV','P_SAFE','P_USER'],axis=1,inplace=True)
```

```
corr = car_undersampled3.corr()[['C_SEV_1']].sort_values(by='C_SEV_1', ascending=False)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr, annot=True,linewidths=.5, ax=ax)
plt.show()
```



Podemos ver que las correlaciones que tiene C_SEV_1 con todas las variables son practimcanente insignificantes, tanto directa como inversamente, esto nos quiere decir que ningun tipo de condición climática condiciona verdaderamente el hecho de tener o no accidentes mortales en la carretera.

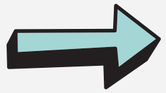
Ajuste de los modelos

Haremos 7 modelos distintos para compararlos entre sí y escoger finalmente el que mejores resultados aporte.

- KNN
- Árbol de decisión
- Bagging
- Random Forest
- Gradient Boost
- XGBoost
- Naïve Bayes

¿Qué métricas utilizaremos en cada modelo?

- Precision
- Recall
- Matriz de Confusión
- F-Score
- Curva ROC
- Área bajo la curva (AUC)

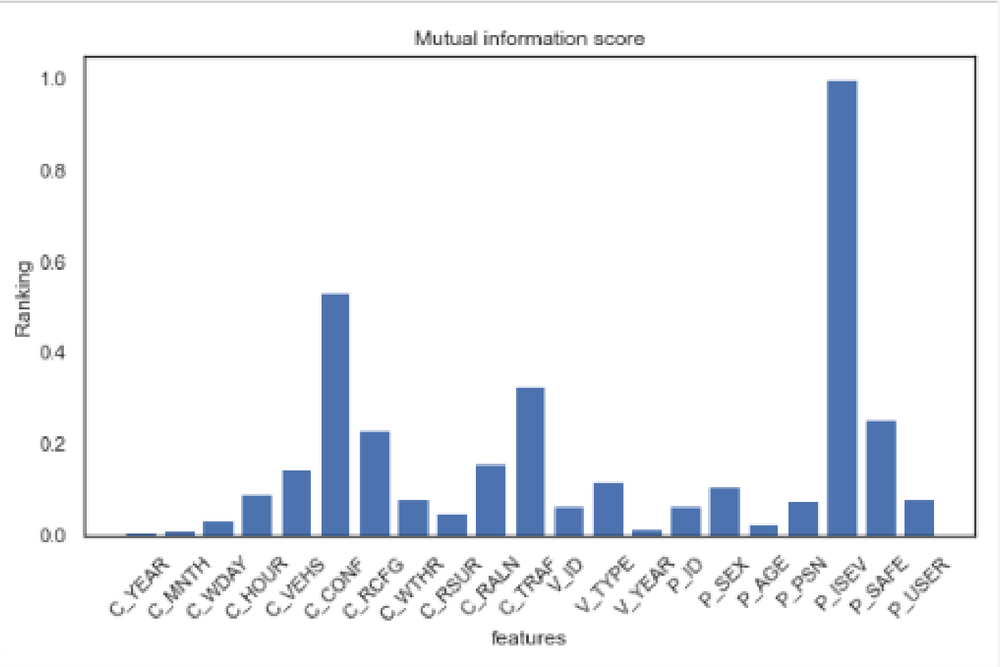


Métricas para modelos de clasificación

Lo primero que haremos será utilizar la técnica de filtrado para la selección de variables más relevantes para el análisis.

```
X1 = car_undersampled.iloc[:, [0,1,2,3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]].values
y1 = car_undersampled['C_SEV'].values
Feature_names1 = car_undersampled.iloc[:, [0,1,2,3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]].columns.values
```

```
from sklearn.feature_selection import f_classif, mutual_info_classif

f_test, _ = f_classif(X1, y1)
f_test /= np.max(f_test)

mi = mutual_info_classif(X1, y1)
mi /= np.max(mi)

plt.figure(figsize=(20, 5))

plt.subplot(1,2,1)
plt.bar(range(X1.shape[1]),f_test,  align="center")
plt.xticks(range(X1.shape[1]),Feature_names1, rotation = 45)
plt.xlabel('features')
plt.ylabel('Ranking')
plt.title('$F-Test$ score')

plt.subplot(1,2,2)
plt.bar(range(X1.shape[1]),mi,  align="center")
plt.xticks(range(X1.shape[1]),Feature_names1, rotation = 45)
plt.xlabel('features')
plt.ylabel('Ranking')
plt.title('Mutual information score')

plt.show()
```



Escogeremos las variables que tengan un mayor peso sobre la variable objetivo y eliminamos las sobrantes, para obtener una tabla final con la que trabajaremos en los distintos modelos

Ahora creamos los subconjuntos de train y test para poder entrenar el modelo: para la X tomamos las columnas asociadas a la variables que nos hemos quedado e Y será la variable a predecir: C_SEV

```
y = car_undersampled['C_SEV'].values
X = car_undersampled.iloc[:,1:].values
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
X_train.shape, X_test.shape
```

```
((157780, 7), (39446, 7))
```

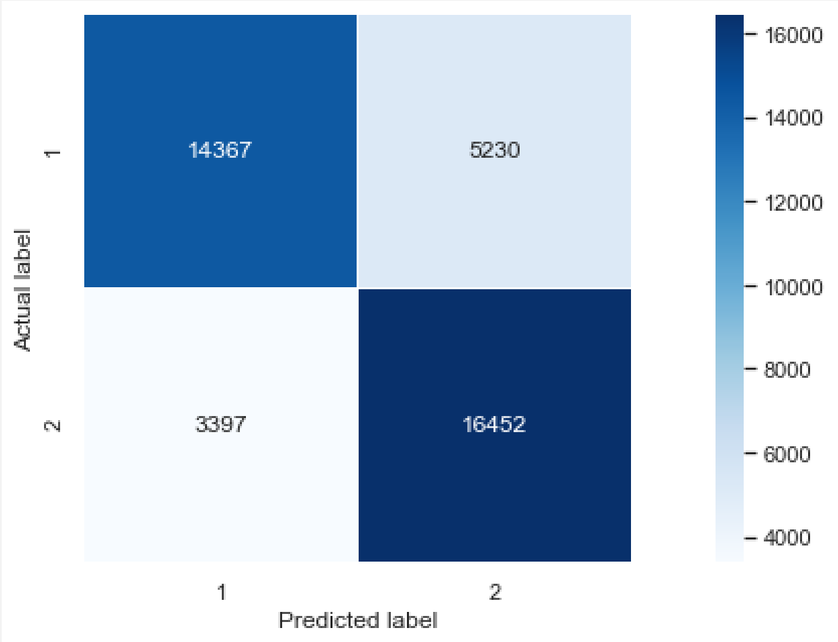
Estandarizamos las variables para que todas ellas estén en la misma medida y no existan problemas en ciertos modelos en los que la no estandarización podría ser perjudicial

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

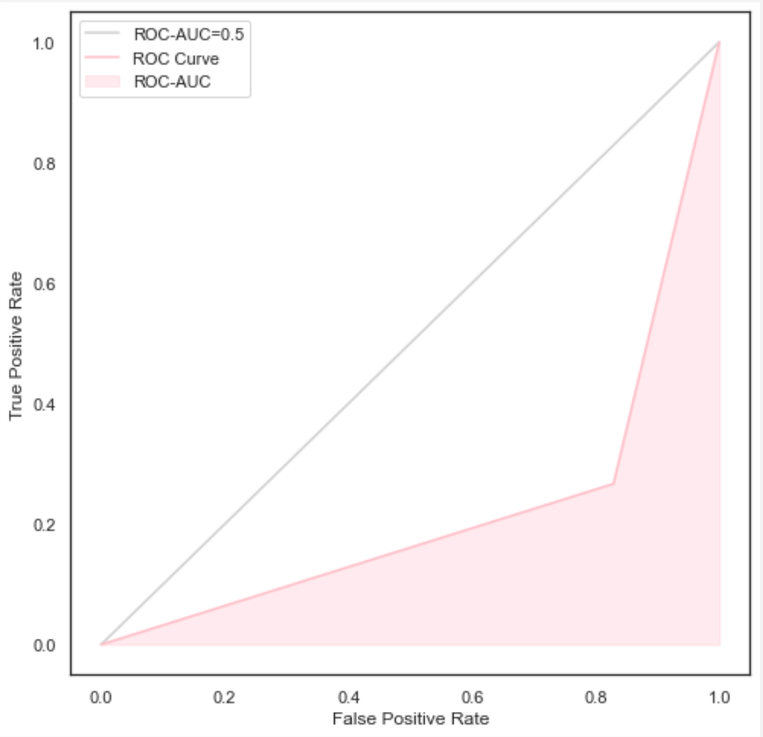
A continuación vamos a llevar a cabo los algoritmos de clasificación con la base de datos ya limpia y balanceada a través del método undersampling

KNN

Matriz de confusión:



Curva Roc:



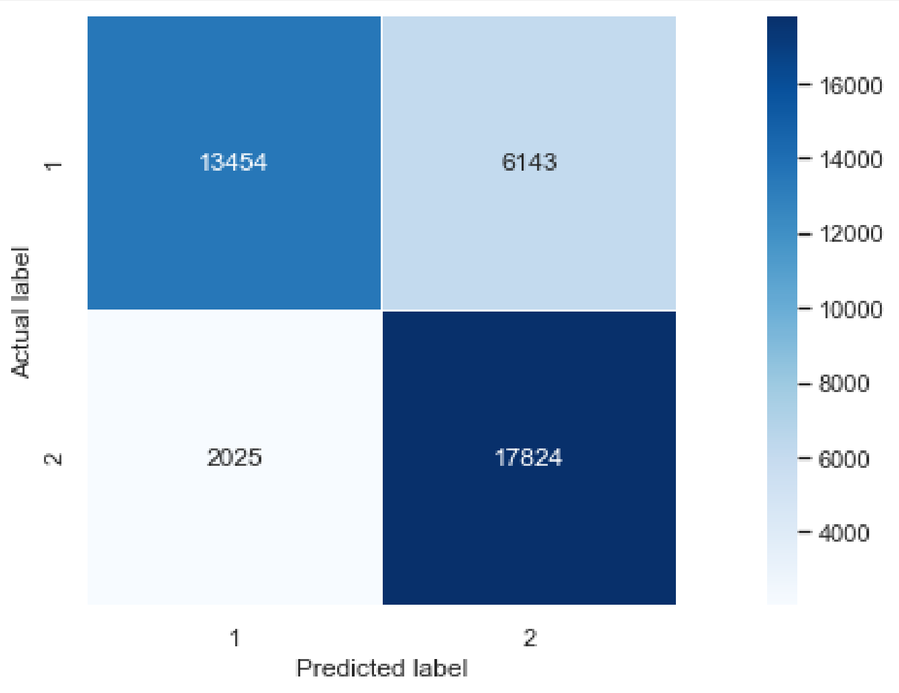
Precision, Recall y F1-Score:

	precision	recall	f1-score	support
1	0.81	0.73	0.77	19597
2	0.76	0.83	0.79	19849
accuracy			0.78	39446
macro avg	0.78	0.78	0.78	39446
weighted avg	0.78	0.78	0.78	39446

AUC: 0.7922567001003518

Árbol de decisión

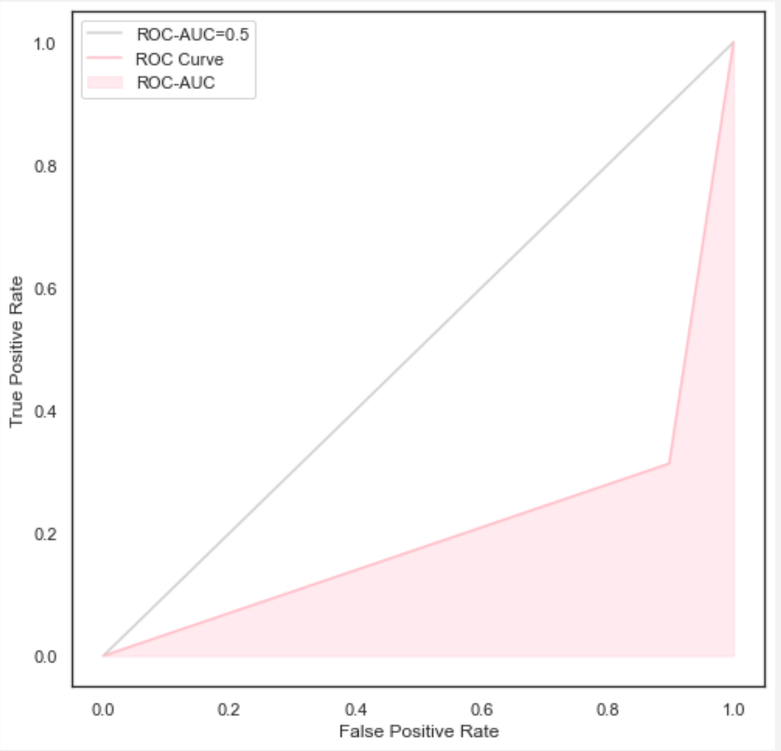
Matriz de confusión:



Precision, Recall y F1-Score:

	precision	recall	f1-score	support
1	0.87	0.69	0.77	19597
2	0.74	0.90	0.81	19849
accuracy			0.79	39446
macro avg	0.81	0.79	0.79	39446
weighted avg	0.81	0.79	0.79	39446

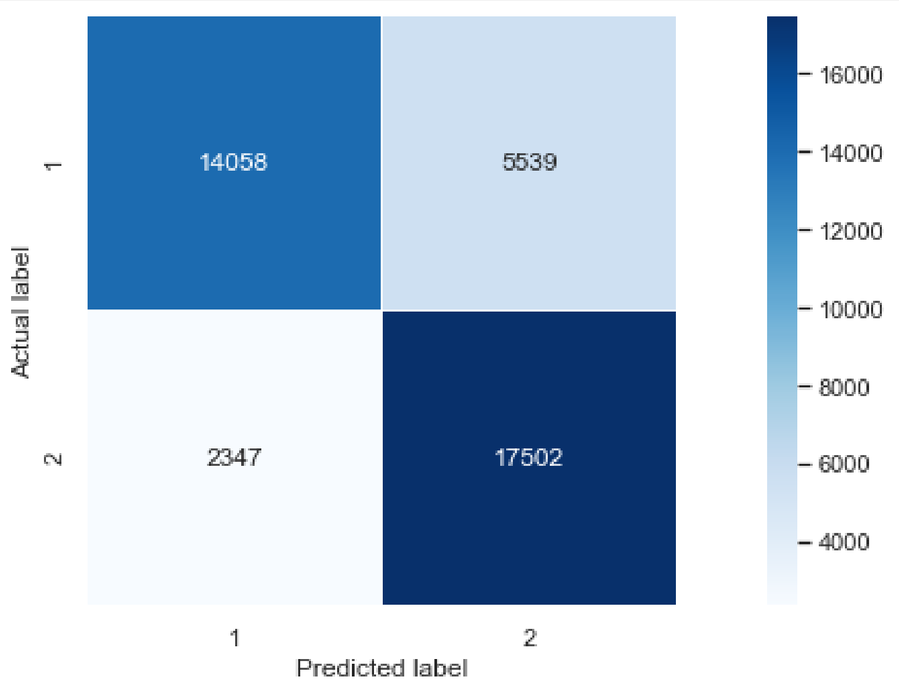
Curva Roc:



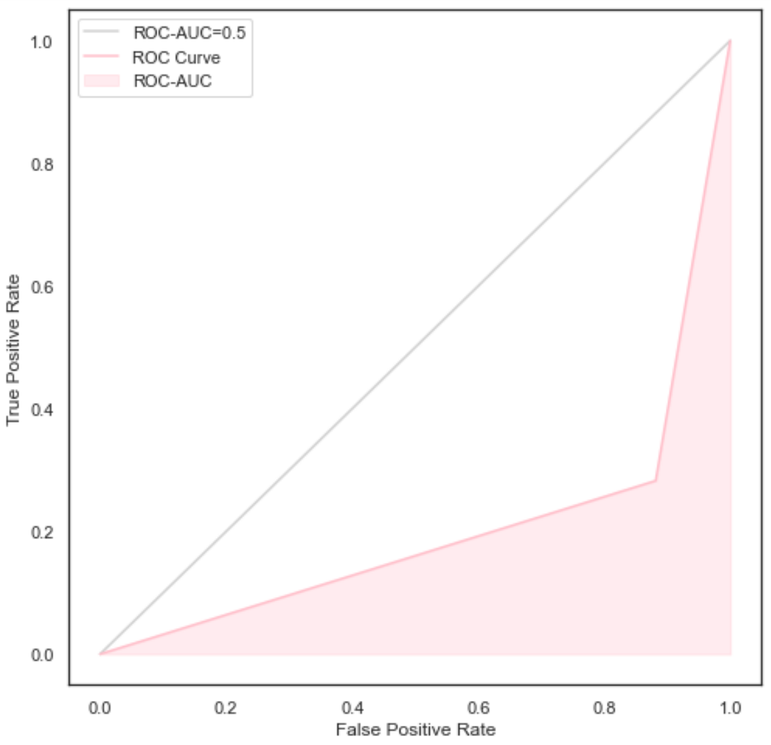
AUC: `0.7922567001003518`

Bagging

Matriz de confusión:



Curva Roc:



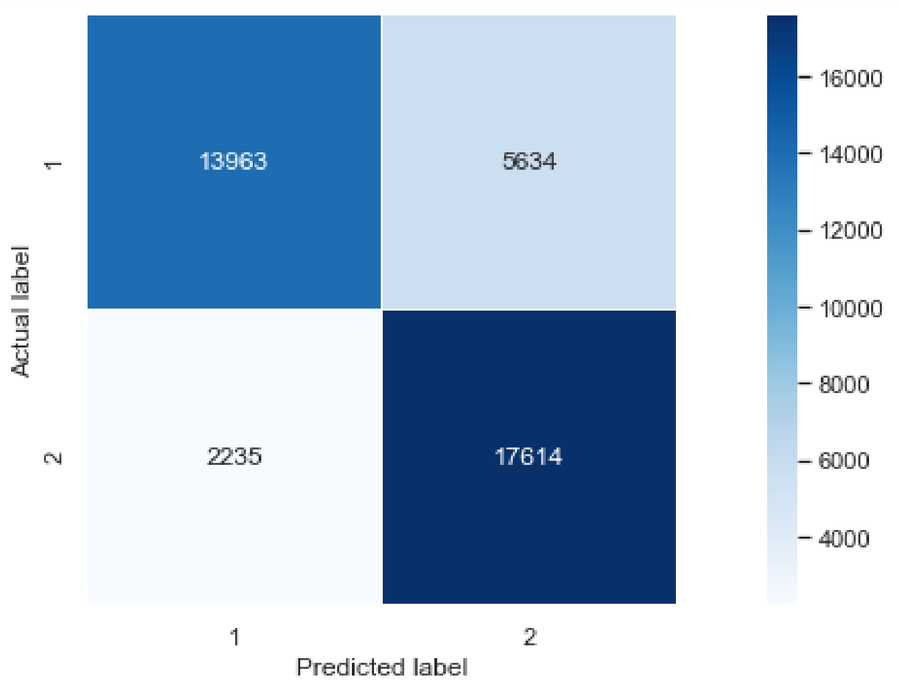
Precision, Recall y F1-Score:

	precision	recall	f1-score	support
1	0.86	0.72	0.78	19597
2	0.76	0.88	0.82	19849
accuracy			0.80	39446
macro avg	0.81	0.80	0.80	39446
weighted avg	0.81	0.80	0.80	39446

AUC: 0.799555982258078

Random Forest

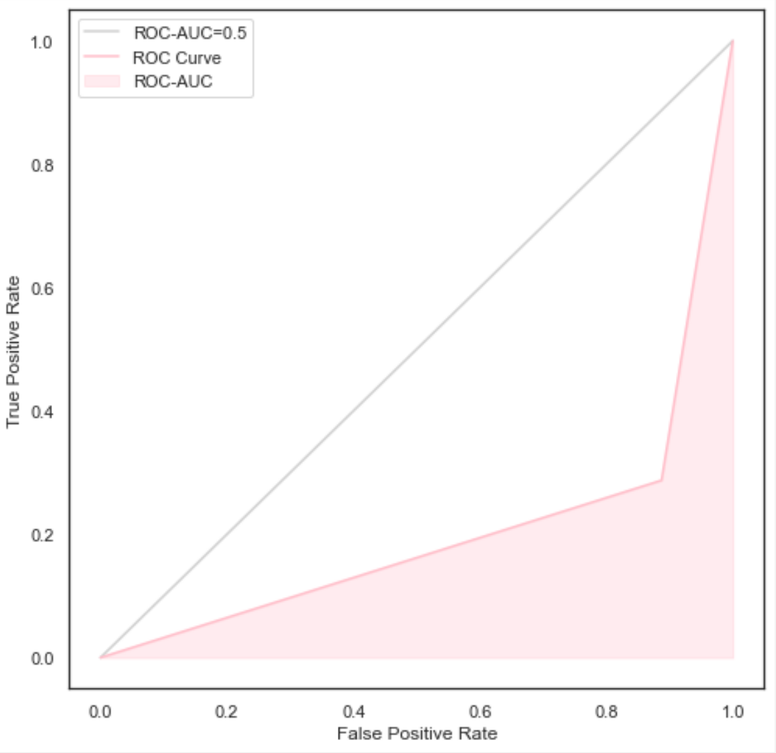
Matriz de confusión:



Precision, Recall y F1-Score:

	precision	recall	f1-score	support
1	0.86	0.71	0.78	19597
2	0.76	0.89	0.82	19849
accuracy			0.80	39446
macro avg	0.81	0.80	0.80	39446
weighted avg	0.81	0.80	0.80	39446

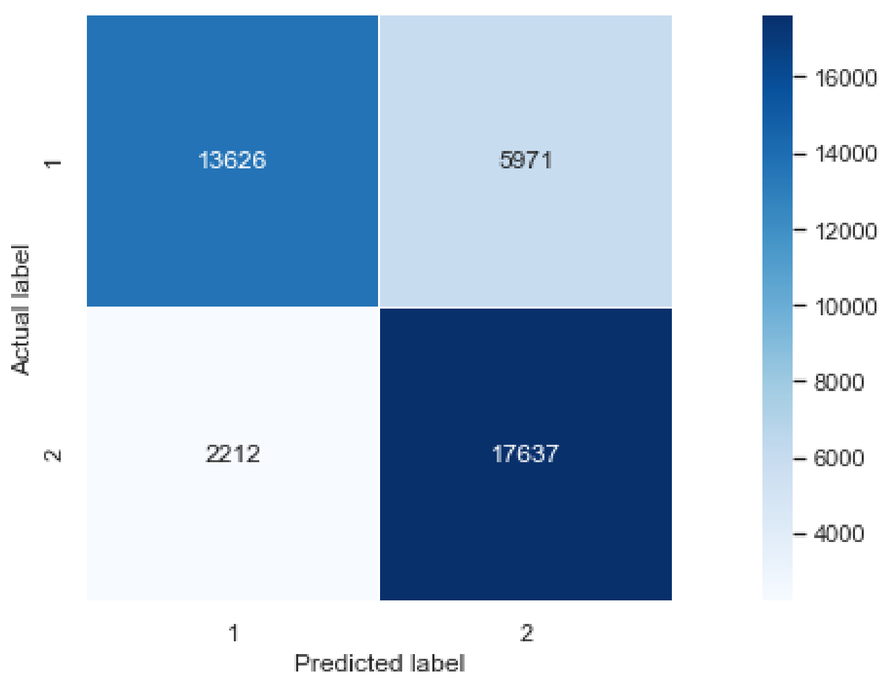
Curva Roc:



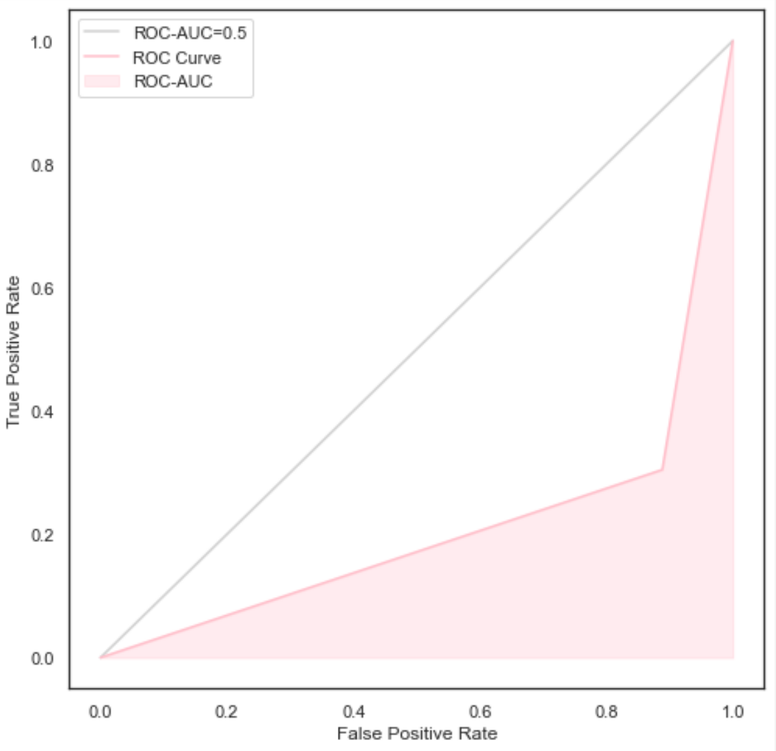
AUC: 0.7999534426955458

Gradient Boost

Matriz de confusión:



Curva Roc:



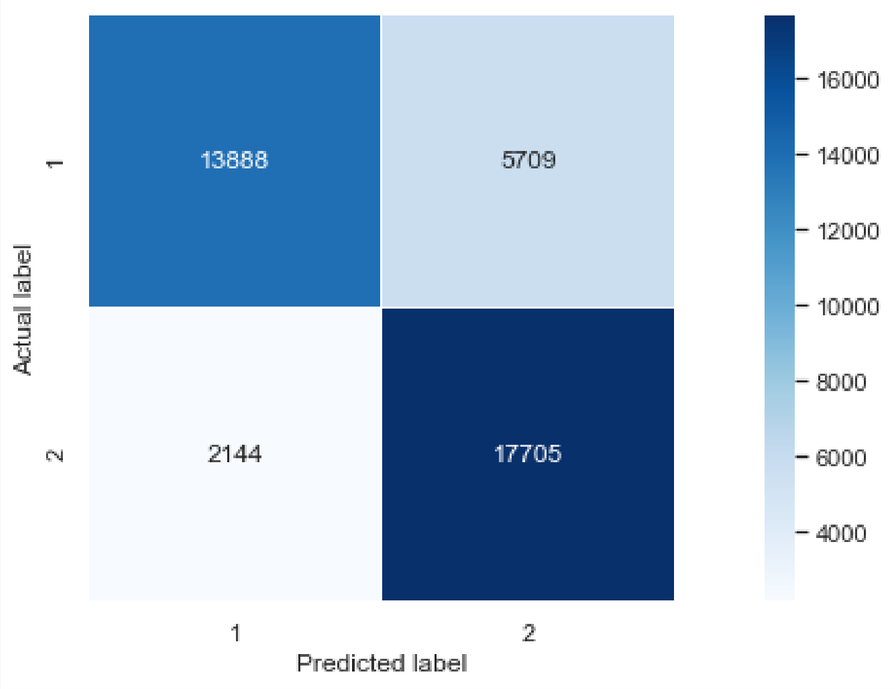
Precision, Recall y F1-Score:

	precision	recall	f1-score	support
1	0.86	0.70	0.77	19597
2	0.75	0.89	0.81	19849
accuracy			0.79	39446
macro avg	0.80	0.79	0.79	39446
weighted avg	0.80	0.79	0.79	39446

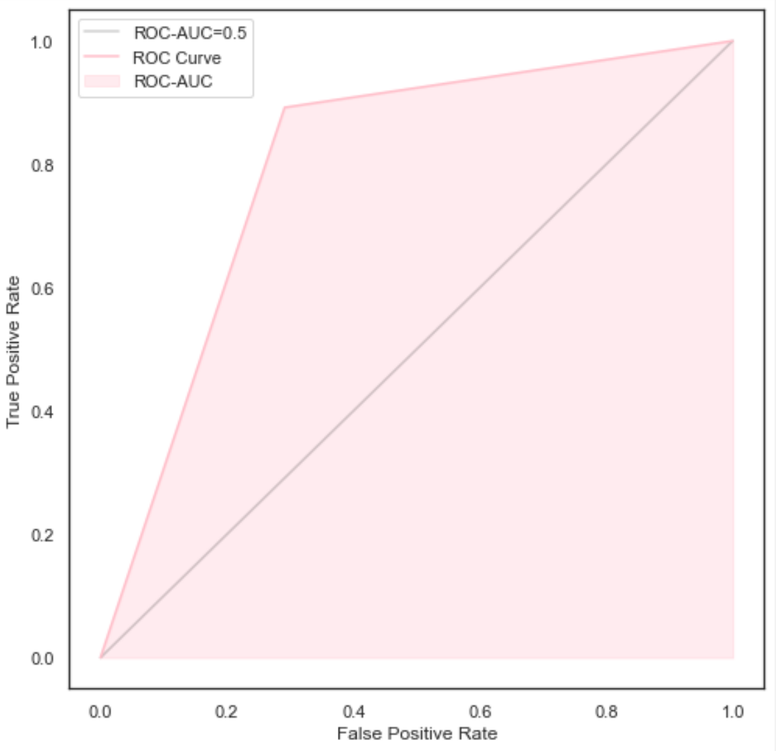
AUC: 0.7919345621364042

XGBoost

Matriz de confusión:



Curva Roc:



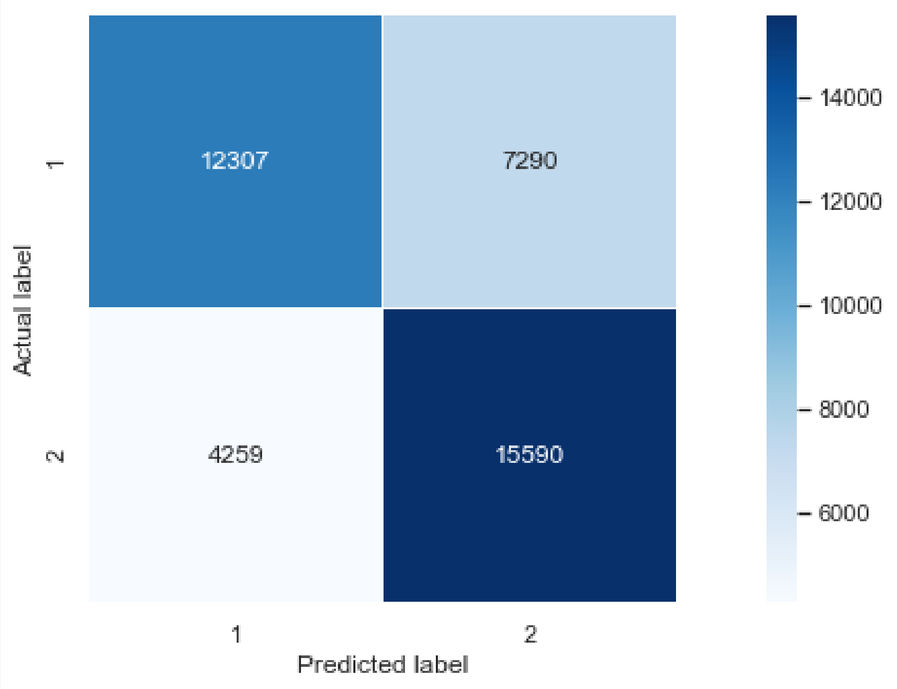
Precision, Recall y F1-Score:

	precision	recall	f1-score	support
1	0.87	0.71	0.78	19597
2	0.76	0.89	0.82	19849
accuracy			0.80	39446
macro avg	0.81	0.80	0.80	39446
weighted avg	0.81	0.80	0.80	39446

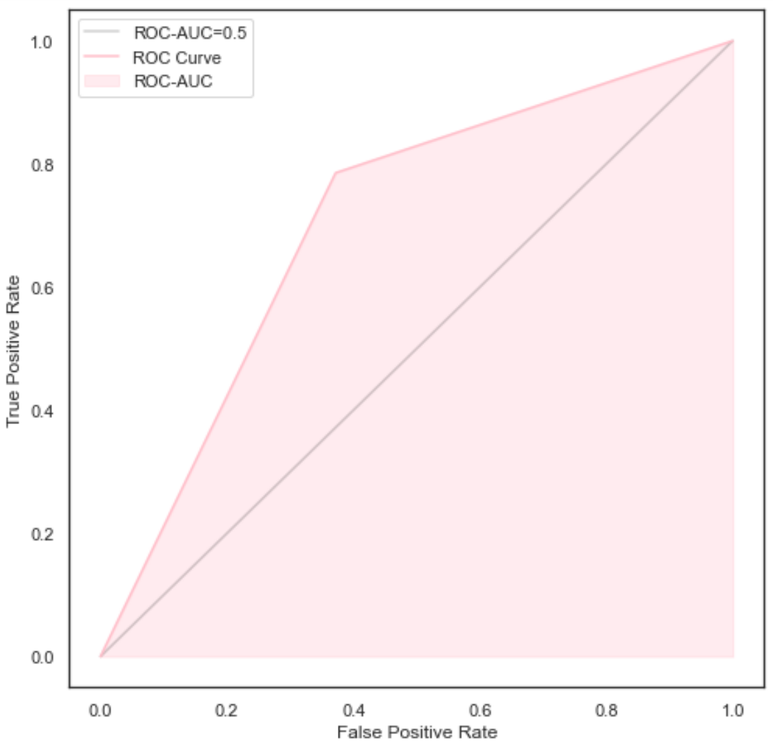
AUC: 0.8003321914150874

Naive Bayes

Matriz de confusión:



Curva Roc:



Precision, Recall y F1-Score:

	precision	recall	f1-score	support
1	0.74	0.63	0.68	19597
2	0.68	0.79	0.73	19849
accuracy			0.71	39446
macro avg	0.71	0.71	0.71	39446
weighted avg	0.71	0.71	0.71	39446

AUC: 0.7067171414218683

Elección de la métrica

La variable objetivo ya hemos visto que se divide en dos, siendo:

- 1: Accidentes mortales
-2: Accidentes leves

Equivocarse en la clasificación de ambas situaciones puede tener consecuencias para la empresa.

Creemos que clasificar un '1' como un '2' es una situación peor para la aseguradora, ya que los accidentes mortales conllevan mayor cantidad de pagos y estaríamos determinando falsamente que un cliente pertenece al grupo de accidentes leves cuando realmente no es así. Es decir, la aseguradora le cobraría una cuota menor de la que le corresponde y si finalmente tuviese un accidente de gravedad podría llegar a provocar grandes pérdidas.

Aun así la clasificación errónea de los '2' podría llevar a una pérdida de clientes ya que estarían pagando una cuota mayor de la que les corresponde y podrían buscar otras opciones en la competencia. Por tanto intentaremos también que el acierto en este caso sea lo mayor posible.

Para la clasificación de los '1' utilizaremos la métrica de **PRECISION**.

Como tampoco nos interesa que los '2' estén mal clasificados, para ello no usaremos precision porque da resultados peores que **RECALL**, que será la métrica elegida.

Elección del modelo

Los modelos que nos dan mejores resultados son XGBoost y Árboles de decisión donde en ambos los '1' están clasificados con un **87%** de acierto (un resultado bastante bueno).

Pero si tenemos en cuenta la clasificación del '2' vemos que en XGBoost: recall: **0.89** y en Árbol de decisión: recall: **0.90**.

Pero fijándonos también en la accuray:
XGBoost: **0.80** y en Árbol de decisión: **0.79**.

Decidimos que es mejor tener una predicción aunque sea un 1% mejor en el total que en la clasificación de los '2'.

Por tanto el mejor modelo será **XGBoost**, aunque en realidad podemos ver que muchos de los modelos nos dan resultados muy parecidos variando únicamente en un 1%.

Importancia de las variables

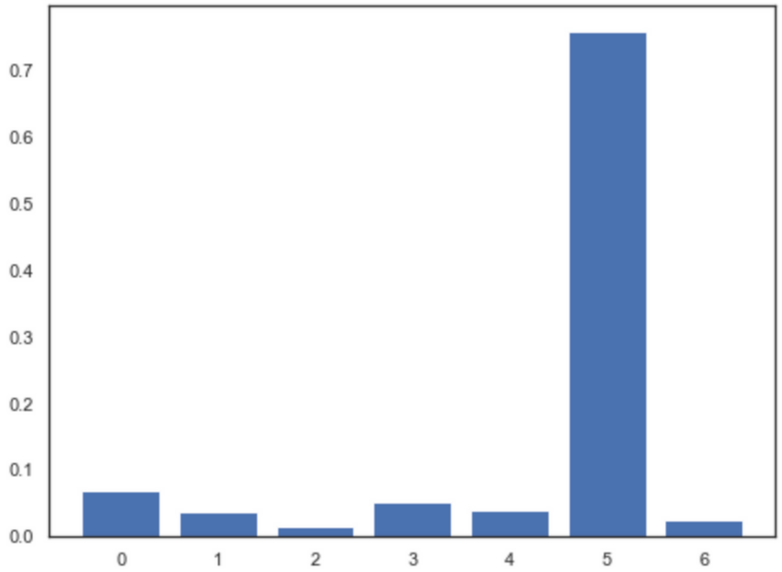
Hacemos uso del modelo XGBoost ya que ha sido elegido cómo el modelo que mejor se ajusta a nuestro conjunto de datos.

```
xgb.feature_importances_
```

```
array([0.07034078, 0.03718036, 0.01604533, 0.0517585 , 0.03975544,
       0.7598124 , 0.02510714], dtype=float32)
```

```
car_undersampled.head()
```

	C_SEV	C_CONF	C_RCFG	C_RALN	C_TRAF	V_TYPE	P_ISEV	P_SAFE
0	1.0	4.0	2.0	4.0	18.0	6.0	3.0	2.0
1	1.0	1.0	2.0	1.0	6.0	1.0	1.0	2.0
2	1.0	1.0	2.0	1.0	6.0	1.0	3.0	2.0
3	1.0	1.0	2.0	1.0	1.0	1.0	2.0	2.0
4	1.0	1.0	2.0	1.0	1.0	1.0	3.0	2.0



La variable más importante con diferencia para este modelo ha sido C_TRAF.

Conclusiones

Si planteásemos este problema teniendo en cuenta las situaciones más beneficiosas para una aseguradora deberíamos valorar sobre todo:

- Que no se produzcan pérdidas por errores en la clasificación en el caso de accidentes graves, ya que esto causaría grandes pérdidas a la empresa.
- Por otro lado que se de la situación de fuga de clientes por errores en la clasificación en los accidentes leves.

Para cumplir todas estos objetivos haremos uso de las métricas comentadas previamente. Estas métricas obtienen sus valores máximos en el modelo **XGBoost** donde

- Precision: **0.87**
- Recall: **0.89**

Por tanto recomendaríamos a la aseguradora utilizar este modelo para clasificar a sus nuevos potenciales clientes.