

INTELIGENCIA ARTIFICIAL

PROYECTO FINAL

JAVIER RODRÍGUEZ GUILLÉN



JAVIER RODRÍGUEZ GUILLÉN

RESPONSABLE DEL DESARROLLO DE LA APLICACIÓN

Fecha: 20-04-2025



1. DEFINICIÓN Y OBJETIVOS DEL PROYECTO.	4
2. DEFINICIÓN DE DATOS.	5
2.1 Lectura y visualización de la estructura del fichero.	7
2.2 Clasificación de características.	8
3. ANÁLISIS Y LIMPIEZA DE DATOS.	11
3.1 Análisis de nulos.	11
3.2 Análisis de características numéricas.	12
3.3 Análisis de características categóricas.	13
3.4 Limpieza de datos.	14
3.4.1 Eliminación de características.	14
3.4.2 Tratamiento de nulos.	15
3.4.3 Modificación de características.	16
3.4.4 Creación de características.	19
4. EXPLORACIÓN DE DATOS.	21
4.1 Análisis descriptivo de características.	21
4.2 Balance de datos.	43
5. TRATAMIENTO DE VARIABLES CATEGÓRICAS Y NUMÉRICAS.	47
5.1 Mapeado de características.	47
5.2 Eliminación de características.	50
5.3 Normalización.	50
5.4 Estructura final de nuestro dataframe.	51
5.5 Matriz de correlación.	51
6. MODELO DE ENTRENAMIENTO.	54
6.1 Elección del modelo.	54
6.2 Librerías utilizadas para el modelado y predicción.	55
7. ANÁLISIS DE RESULTADOS Y CONCLUSIONES.	57
7.1 Resultados de nuestros modelos.	57
7.2 Resultados de nuestros modelos previo balanceado de la clase objetivo.	63
7.3 Resultados de nuestros modelos previa simplificación de la clase objetivo.	65
7.4 Conclusiones.	68



1. DEFINICIÓN Y OBJETIVOS DEL PROYECTO.

El presente proyecto tiene como objetivo principal el análisis de la lesividad en un conjunto de datos correspondiente a víctimas de accidentes de tráfico ocurridos en la ciudad de Madrid en el periodo de 2019 a 2024.

Cuando hablamos de lesividad, nos referimos al grado de daño sufrido por las personas implicadas en dichos accidentes. Como veremos más adelante, aunque nuestro dataset cuenta con múltiples tipos de lesividad, se ha clasificado de forma simplificada en tres categorías: ileso, herido y fallecido.

Este estudio nace del objetivo o la idea de comprender mejor los factores que inciden en la gravedad de las consecuencias de los accidentes de tráfico, con el fin de aportar información útil para la prevención, la planificación urbana y las políticas de seguridad vial. Para ello, se ha trabajado con un conjunto de datos que incluye múltiples variables como son la fecha, el tipo de accidente, las condiciones meteorológicas, la edad y el sexo de las víctimas, el distrito donde ocurrió el suceso, entre otros aspectos.

A partir de estos datos, se han aplicado técnicas de análisis exploratorio, visualización de datos y modelos de aprendizaje automático supervisado con el fin de:

- Identificar patrones y relaciones entre las variables del accidente y el grado de lesividad resultante.
- Visualizar la distribución geográfica y temporal de los casos con diferentes niveles de gravedad.
- Construir modelos predictivos capaces de estimar el nivel de lesividad a partir de las características del accidente.
- Evaluar el impacto del desbalanceo de clases y aplicar técnicas como el sobremuestreo.
- Comparar distintos algoritmos de clasificación para determinar cuál ofrece mejores resultados.

El enfoque del proyecto es tanto analítico como predictivo, con el propósito de generar conocimientos útiles en el campo de los accidentes de tráfico.



2. DEFINICIÓN DE DATOS.

Los datos utilizados para el presente proyecto recoge información detallada sobre víctimas de accidentes de tráfico ocurridos en la ciudad de Madrid y han sido extraídos de la web municipal de datos abiertos del Ayuntamiento: <https://datos.madrid.es/>.

El dataset incluye registros individuales por víctima, con variables que describen tanto las circunstancias del accidente como características personales de las víctimas. A continuación se detallan la información contenida en los archivos csv facilitados por el Ayuntamiento:

- Columna 1: **num_expediente** (número de expediente) – texto
- Columna 2: **fecha** (formato dd/mm/aaaa) - fecha
- Columna 3: **hora** (formato hh:mm:ss) - hora
- Columna 4: **localizacion** (dirección del accidente) - texto
- Columna 5: **numero** (número de la calle) - número
- Columna 6: **cod_distrito** (código del distrito) - número
- Columna 7: **distrito** (nombre del distrito) - texto
- Columna 8: **tipo_accidente** (tipo de accidente) - texto
- Columna 9: **estado_meteorológico** (descripción climatología) - texto
- Columna 10: **tipo_vehiculo** (tipo de vehículo implicado) - texto
- Columna 11: **tipo_persona** (tipo persona implicada) - texto
- Columna 12: **rango_edad** (tramo edad persona afectada) - texto
- Columna 13: **sexo** (puede ser: hombre, mujer o no asignado) - texto
- Columna 14: **cod_lesividad** (código de lesividad) - número
- Columna 15: **lesividad** (descripción lesividad) - texto
- Columna 16: **coordenada_x_utm** (ubicación coordenada x) - número
- Columna 17: **coordenada_y_utm** (ubicación coordenada y) - número
- Columna 18: **positivo_alcohol** (puede ser N o S) - texto



- Columna 19: **positivo_droga** (puede ser 1 o Null) - texto

En la información asociada al dataset se explican los tipos de accidentes y los códigos de lesividad. Empezaremos explicando los tipos de accidente:

- Colisión doble: Accidente de tráfico ocurrido entre dos vehículos en movimiento, (colisión frontal, fronto lateral, lateral)
- Colisión múltiple: Accidente de tráfico ocurrido entre más de dos vehículos en movimiento.
- Alcance: Accidente que se produce cuando un vehículo circulando o detenido por las circunstancias del tráfico es golpeado en su parte posterior por otro vehículo.
- Choque contra obstáculo o elemento de la vía: Accidente ocurrido entre un vehículo en movimiento con conductor y un objeto inmóvil que ocupa la vía o zona apartada de la misma, ya sea vehículo estacionado, árbol, farola, etc.
- Atropello a persona: Accidente ocurrido entre un vehículo y un peatón que ocupa la calzada o que transita por aceras, refugios, paseos o zonas de la vía pública no destinada a la circulación de vehículos.
- Vuelco: Accidente sufrido por un vehículo con más de dos ruedas y que por alguna circunstancia sus neumáticos pierden el contacto con la calzada quedando apoyado sobre un costado o sobre el techo.
- Caída: Se agrupan todas las caídas relacionadas con el desarrollo y las circunstancias del tráfico, (motocicleta, ciclomotor, bicicleta, viajero bus, etc.,)
- Otras causas: Recoge los accidentes por atropello a animal, despeñamiento, salida de la vía, y otros.

En cuanto a los códigos de lesividad tenemos los siguientes:

- 01: Atención en urgencias sin posterior ingreso. - LEVE.
- 02: Ingreso inferior o igual a 24 horas – LEVE.
- 03: Ingreso superior a 24 horas. - GRAVE.



- 04: Fallecido 24 horas – FALLECIDO.
- 05: Asistencia sanitaria ambulatoria con posterioridad – LEVE.
- 06: Asistencia sanitaria inmediata en centro de salud o mutua – LEVE.
- 07: Asistencia sanitaria sólo en el lugar del accidente – LEVE.
- 14: Sin asistencia sanitaria.
- 77: Se desconoce.
- En blanco: Sin asistencia sanitaria.

2.1 Lectura y visualización de la estructura del fichero.

Los archivos csv facilitados por el Ayuntamiento están divididos por años. Utilizando la librería Pandas hemos realizado la lectura y unificación de todos los archivos en un único dataframe como se muestra en la siguiente imagen:

```
import pandas as pd
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

df_2019 = pd.read_csv('2019_Accidentalidad.csv', sep=';')
df_2020 = pd.read_csv('2020_Accidentalidad.csv', sep=';')
df_2021 = pd.read_csv('2021_Accidentalidad.csv', sep=';')
df_2022 = pd.read_csv('2022_Accidentalidad.csv', sep=';')
df_2023 = pd.read_csv('2023_Accidentalidad.csv', sep=';')
df_2024 = pd.read_csv('2024_Accidentalidad.csv', sep=';')

# unificamos todos los dataframes en uno reindexando todo el dataframe
df = pd.concat([df_2019, df_2020, df_2021, df_2022, df_2023, df_2024], ignore_index=True)
```

Podemos observar que nuestro archivo tiene un total de 271250 registros y 19 columnas.

```
df.shape # nuestro archivo tiene un total de 271250 registros y 19 variables o columnas
(271250, 19)
```

Si mostramos las primeras filas de nuestro dataframe podemos ver los primeros registros:



df.head()												
	num_expediente	fecha	hora	localizacion	numero	cod_distrito	distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	tipo_persona	
0	2018S017842	04/02/2019	9:10:00	CALL. ALBERTO AGUILERA, 1	1	1.0	CENTRO	Colisión lateral	Despejado	Motocicleta > 125cc	Conductor	
1	2018S017842	04/02/2019	9:10:00	CALL. ALBERTO AGUILERA, 1	1	1.0	CENTRO	Colisión lateral	Despejado	Turismo	Conductor	
2	2019S000001	01/01/2019	3:45:00	PASEO. SANTA MARÍA DE LA CABEZA / PLAZA. FUENTES	168	11.0	CARABANCHEL	Alcance	Nan	Furgoneta	Conductor	

rango_edad	sexo	cod_lesividad	lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga
De 45 a 49 años	Hombre	7.0	Asistencia sanitaria sólo en el lugar del acci...	440068.0	4475679.0	N	Nan
De 30 a 34 años	Mujer	7.0	Asistencia sanitaria sólo en el lugar del acci...	440068.0	4475679.0	N	Nan
De 40 a 44 años	Hombre	Nan	Nan	439139.0	4470836.0	S	Nan

2.2 Clasificación de características.

Utilizando el método de pandas info podemos ver las características de nuestro dataframe:

df.info() # los tipos de datos de nuestro dataframe son los siguientes			
<class 'pandas.core.frame.DataFrame'>			
RangeIndex: 271250 entries, 0 to 271249			
Data columns (total 19 columns):			
#	Column	Non-Null Count	Dtype
0	num_expediente	271250 non-null	object
1	fecha	271250 non-null	object
2	hora	271250 non-null	object
3	localizacion	271250 non-null	object
4	numero	271242 non-null	object
5	cod_distrito	271242 non-null	float64
6	distrito	271242 non-null	object
7	tipo_accidente	271241 non-null	object
8	estado_meteorológico	241587 non-null	object
9	tipo_vehiculo	269912 non-null	object
10	tipo_persona	271247 non-null	object
11	rango_edad	271250 non-null	object
12	sexo	271250 non-null	object
13	cod_lesividad	148686 non-null	float64
14	lesividad	148686 non-null	object
15	coordenada_x_utm	271209 non-null	object
16	coordenada_y_utm	271209 non-null	object
17	positiva_alcohol	270301 non-null	object
18	positiva_droga	891 non-null	float64
dtypes: float64(3), object(16)			



Analizando una por una, podemos dividirlas en categóricas ordinales y nominales, y en numéricas discretas o continuas:

- Características Categóricas Ordinales:
 - Columna 12: rango_edad (tramo edad persona afectada) - texto
 - Columna 18: positivo_alcohol (puede ser N o S) - texto
- Características Categóricas Nominales:
 - Columna 4: localizacion (dirección del accidente) - texto
 - Columna 5: numero (número de la calle) - texto
 - Columna 7: distrito (nombre del distrito) - texto
 - Columna 8: tipo_accidente (tipo de accidente) - texto
 - Columna 9: estado_meteorologico (descripción climatología) - texto
 - Columna 10: tipo_vehiculo (tipo de vehículo implicado) - texto
 - Columna 11: tipo_persona (tipo persona implicada) - texto
 - Columna 13: sexo (puede ser: hombre, mujer o no asignado) - texto
 - Columna 15: lesividad (descripción lesividad) - texto
 - Datos temporales:
 - Columna 2: fecha (formato dd/mm/aaaa) - texto
 - Columna 3: hora (formato hh:mm:ss) - texto
- Características Numéricas Discretas:
 - Columna 6: cod_distrito (código del distrito) - entero
- Características Numéricas Continuas:
 - Columna 14: cod_lesividad (código de lesividad) - float
 - Columna 16: coordenada_x_utm (ubicación coordenada x) - float
 - Columna 17: coordenada_y_utm (ubicación coordenada y) - float
 - Columna 19: positivo_droga (puede ser 1 o Null) - float



- Características de Datos Mixtos:
 - Columna 1: num_expediente (número de expediente) - texto



3. ANÁLISIS Y LIMPIEZA DE DATOS.

3.1 Análisis de nulos.

Para un total de 271250 registros tenemos nulos en las siguientes variables:

```
df.isna().sum() # análisis de nulos

num_expediente      0
fecha               0
hora                0
localizacion        0
numero              8
cod_distrito        8
distrito             8
tipo_accidente      9
estado_meteorológico 29663
tipo_vehiculo        1338
tipo_persona          3
rango_edad            0
sexo                 0
cod_lesividad         122564
lesividad             122564
coordenada_x_utm      41
coordenada_y_utm      41
positiva_alcohol       949
positiva_droga        270359
dtype: int64
```

- **numero**: 8 nulos (número de la dirección de la calle del accidente).
- **cod_distrito**: tenemos un total de 8 nulos.
- **tipo_accidente**: 9 nulos.
- **estado_meteorológico**: 29663 nulos (el 10,94% de los datos).
- **tipo_vehiculo**: 1338 nulos (el 0,49% de los datos).
- **tipo_persona**: tan solo 3 nulos.
- **cod_lesividad** y **lesividad**: 122564 nulos. Según la información asociada al dataframe, cuando se deja en blanco este campo significa que no ha recibido atención sanitaria, lo que es equivalente al código de lesividad 14.
- **coordenada_x_utm** y **coordenada_y_utm**: 41 nulos. Esta cantidad es totalmente despreciable.



- **positivo_alcohol:** 949 nulos (el 0,35% de los datos). Hay que tener en cuenta que para esta característica sólo registran los valores 0 o 1. El problema está en no se sabe realmente a cuántas víctimas se le ha realizado el test de alcohol, porque un 0 puede significar un resultado negativo, o también puede significar que no se le ha realizado la prueba, por lo que la información de esta columna no es fiable para extraer conclusiones o correlaciones respecto a ella.
- **positivo_drogas:** 270359 nulos. Hay que tener en cuenta que para esta característica sólo registran los valores 1 (positivo) o 'null'. Presenta el mismo problema que la variable positivo_alcohol, por lo que la información de esta columna tampoco es fiable para extraer conclusiones o correlaciones respecto a ella.

3.2 Análisis de características numéricas.

df.describe() # Mostramos las variables numéricas		
	cod_distrito	cod_lesividad
count	271242.000000	148686.000000
mean	10.000000	14.000000
std	1.000000	1.000000
min	1.000000	1.000000
25%	9.000000	13.000000
50%	10.000000	14.000000
75%	11.000000	15.000000
max	12.000000	16.000000

- **cod_distrito:** tenemos prácticamente todas las muestras documentadas.
- **cod_lesividad:** tenemos un total de 30042 muestras. Como veremos más adelante, 21769 son nulos. Según la información asociada al dataframe, cuando se deja en blanco este campo significa que no ha recibido atención sanitaria, lo que equivale al código 14.
- **positiva_droga:** puede tener valor 1 o Null. Tan sólo 162 positivos.



3.3 Análisis de características categóricas.

df.describe(include=['O']) # Mostramos las variables categóricas									
	num_expediente	fecha	hora	localizacion	numero	distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo
count	271250	271250	271250	271250	271242	271242	271241	241587	269912
unique	115640	2192	1434	57447	2940	21	13	7	40
top	2023S039284	05/04/2019	19:00:00	AUTOV. A-2, +00500E	1	PUENTE DE VALLECAS	Colisión fronto- lateral	Despejado	Turismo
freq	44	272	2397	545	25710	21537	68697	204139	187483

tipo_persona	rango_edad	sexo	lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol
271247	271250	271250	148686	271209.0	271209.0	270301
3	18	3	9	25406.0	26540.0	2
Conductor	Desconocido	Hombre	Sin asistencia sanitaria	441507.0	4476593.0	N
218752	29725	163972	82673	121.0	112.0	262405

- **num_expediente:** tenemos un total de 115640, es decir, este es el número total de accidentes, respecto a 271250 registros.
- **distrito:** tenemos un total de 21 en la ciudad de Madrid. El distrito de Puente de Vallecas representa el 7,93% de las víctimas de accidentes.
- **tipo_accidente:** tenemos un total de 13 tipos (alcance, atropello a animal, atropello a persona, caída, choque contra obstáculo, colisión frontal, colisión fronto-lateral, colisión lateral, colisión múltiple, despeñamiento, otro, solo salida de la vía, vuelco). Sería conveniente reducir y agrupar en menos categorías. El tipo de accidente más común es la colisión fronto-lateral (25,32% del total).
- **estado_meteorológico:** tenemos un total de 7 estados (despejado, granizado, lluvia débil, lluvia intensa, nevando, nublado, null, se desconoce). Tenemos que analizar el total de nulos y total de 'se desconoce' para valorar correctamente la muestra. Descartando nulos, el tiempo predominante cuando hay accidentes es despejado, con un total del 75,26%.
- **tipo_vehiculo:** tenemos un total de 40. Necesitamos reducir el número de categorías creando agrupaciones para simplificar los datos. El tipo de vehículo más común es el turismo (69,11%).
- **tipo_persona:** existen 3 tipos (conductor, pasajero y peatón).



- **rango_edad:** tenemos un total de 18 rangos. Quizás se pueda agrupar los datos en rangos de 10 años para simplificar. El rango más común es 'Desconocido' (10,96%).
- **sexo:** tenemos 3 tipos (hombre, mujer o desconocido). Haremos una proporción para distribuir los de categoría 'desconocido' a hombres y mujeres. El sexo más común de la víctima es hombre, con un 60,45%.
- **lesividad:** tenemos 9 categorías de lesividad. La más frecuente es 'sin asistencia sanitaria'
- **positivo_alcohol:** dos valores, 0 o 1. El valor negativo es el más común, con 96,73%. De todas formas, no se realizan test de alcohol a todas las víctimas de un accidente, por lo que este dato no es fiable.

3.4 Limpieza de datos.

3.4.1 Eliminación de características.

- **positiva_alcohol:** no se realizan test de alcohol a todas las víctimas de un accidente, por lo que este dato no es fiable.
- **positiva_droga:** no se realizan test de drogas a todas las víctimas de un accidente, por lo que este dato no es fiable.
- **localizacion y numero:** existen más de 57447 registros de localizaciones diferentes, aproximadamente el 21% de las muestras. No sigue un estándar en su registro, lo que requiere dedicarle demasiado tiempo a su corrección.
- **num_expediente:** se trata de una cadena de letras y números. Únicamente nos puede servir para saber el número de accidentes y la pertenencia de las víctimas a un mismo accidente.

```
# Eliminación de las columnas no deseadas
df_edited = df_edited.drop(columns=['localizacion','numero','positiva_alcohol','positiva_droga', 'num_expediente'])
df_edited.shape
```



3.4.2 Tratamiento de nulos.

- **estado_meteorológico:** gestión de los 29663 nulos. Asignar a la categoría ya existente de 'Se desconoce'.
- **tipo_vehiculo:** gestión de los 1338 nulos. Asignar a la categoría ya existente de 'Sin especificar'. Además, esta variable presenta hasta 40 tipos de vehículos, por lo que puede ser conveniente reducirlo para simplificar dicha característica.
- **cod_lesividad y lesividad:** el valor en blanco o nulo equivale a asistencia no sanitaria (código 14).
- **tipo_accidente:** al ser un valor despreciable (9 nulos) los asignamos a la categoría ya existente 'Otro'.
- **tipo_persona:** sólo 3 nulos. Los sustituiremos por la categoría ya existente 'Conductor', que es la más común.

Tratamiento de nulos de las siguientes características:

- estado meteorológico
- tipo_vehículo
- cod_lesividad
- lesividad
- tipo_accidente
- tipo_persona

```
# Tratamiento de nulos
# estado_meteorológico: los nulos los sustituimos por la categoría ya existente 'Se desconoce'
df_edited['estado_meteorológico'] = df_edited['estado_meteorológico'].fillna('Se desconoce')

# tipo_vehiculo: los nulos los sustituimos por la categoría ya existente 'Sin especificar'
df_edited['tipo_vehiculo'] = df_edited['tipo_vehiculo'].fillna('Sin especificar')

# cod_lesividad: siguiendo la info asociada al dataframe, los nulos los sustituimos por el código 14, que equivale
df_edited['cod_lesividad'] = df_edited['cod_lesividad'].fillna(14)

# lesividad: siguiendo la info asociada al dataframe, los nulos los sustituimos por 'Sin asistencia sanitaria'
df_edited['lesividad'] = df_edited['lesividad'].fillna('Sin asistencia sanitaria')

# tipo_accidente: los nulos (9 en total) los sustituimos por la categoría ya existente 'Otro'
df_edited['tipo_accidente'] = df_edited['tipo_accidente'].fillna('Otro')

# tipo_persona: los 3 nulos los sustituimos por la categoría ya existente 'Conductor', que es la más común
df_edited['tipo_persona'] = df_edited['tipo_persona'].fillna('Conductor')
```

- **cod_distrito y distrito:** sólo tenemos 8 nulos (valor despreciable). Los asignaremos a un distrito aleatorio.



```
# vamos a reemplazar los valores nulos en 'cod_distrito' por un número de distrito aleatorio entre 1 y 21
# extraemos los indices de los registros con valor nulo
indices = df_edited[df_edited['cod_distrito'].isnull()].index

# generamos los valores aleatorios
nuevos_valores = np.random.randint(1, 22, size=len(indices))

# asignamos los valores generados a los registros 'cod_distrito' es igual a nulo
df_edited.loc[indices, 'cod_distrito'] = nuevos_valores

# vamos a reemplazar los nulos en 'distrito'
# creamos un mapeo a partir de los datos existentes en nuestro Dataframe que relacionan 'cod_distrito' y 'distrito'
# extraemos la relación entre 'cod_distrito' y 'distrito'
mapeo_distritos = df_edited.dropna().set_index('cod_distrito')[['distrito']].to_dict()
print(mapeo_distritos)

{1.0: 'CENTRO', 11.0: 'CARABANCHEL', 10.0: 'LATINA', 12.0: 'USERA', 9.0: 'MONCLOA-ARAVACA', 14.0: 'MORATALAZ', 4.0: 'SALAMANCA', 18.0: 'VILLA DE VALLECAS', 17.0: 'VILLVERDE', 7.0: 'CHAMBERÍ', 5.0: 'CHAMARTÍN', 16.0: 'HORTALEZA', 15.0: 'CIUDAD LINEAL', 3.0: 'RETIRO', 8.0: 'FUENCARRA L-EL PARDO', 19.0: 'VICÁLVARO', 13.0: 'PUENTE DE VALLECAS', 21.0: 'BARAJAS', 2.0: 'ARGANZUELA', 6.0: 'TETUÁN', 20.0: 'SAN BLAS-CANILLEJAS'}

# asignamos los nombres de distrito correspondientes a cada distrito usando nuestro mapeo
df_edited.loc[indices, 'distrito'] = df_edited.loc[indices, 'cod_distrito'].map(mapeo_distritos)
```

- **coordenada_x_utm** y **coordenada_y_utm**: 41 nulos. Realizaremos la media de las coordenadas por distrito para asignar unos valores lógicos a la ubicación aproximada del accidente.

```
# convertimos las coordenadas a tipo float (errors='coerce' -> en caso de error nos crea un NaN (nulo)
df_edited['coordenada_x_utm'] = pd.to_numeric(df_edited['coordenada_x_utm'], errors='coerce')
df_edited['coordenada_y_utm'] = pd.to_numeric(df_edited['coordenada_y_utm'], errors='coerce')
df_edited.isna().sum()

fecha          0
hora           0
cod_distrito   0
distrito       0
tipo_accidente 0
estado_meteorológico 0
tipo_vehiculo  0
tipo_persona   0
rango_edad     0
sexo           0
cod_lesividad  0
lesividad      0
coordenada_x_utm    47
coordenada_y_utm    47
dtype: int64

# hemos pasado de 41 a 47 nulos (ha fallado la conversión de registros) en ambas coordenadas
# calculamos la media de las coordenadas por distrito
media_x_por_distrito = df_edited.groupby('cod_distrito')['coordenada_x_utm'].transform('mean')
media_y_por_distrito = df_edited.groupby('cod_distrito')['coordenada_y_utm'].transform('mean')

# sustituimos los valores nulos con la media correspondiente de su distrito
df_edited['coordenada_x_utm'] = df_edited['coordenada_x_utm'].fillna(media_x_por_distrito)
df_edited['coordenada_y_utm'] = df_edited['coordenada_y_utm'].fillna(media_y_por_distrito)
df_edited.isna().sum()
```

3.4.3 Modificación de características.

- **fecha** y **hora**: convertir a formato datetime para facilitar su manejo y extracción de información.



- **sexo:** asignar un valor de hombre o mujer a los valores 'no asignado' según proporción de muestra.

Modificación de las siguientes características:

```

• fecha
• hora
• sexo

# convertimos la variable fecha a tipo datetime, para poder realizar operaciones con fechas
df_edited['fecha'] = pd.to_datetime(df_edited['fecha'], format='%d/%m/%Y')
print(df_edited['fecha'].dtype) # Salida: datetime64[ns]
datetime64[ns]

df_edited['hora'] = pd.to_timedelta(df_edited['hora'])
print(df_edited['hora'].dtype)
timedelta64[ns]

# analizamos la proporción de los valores de sexo: Hombre, Mujer y Desconocido
proporciones_sexo = df_edited['sexo'].value_counts(normalize=True)
print(proporciones_sexo)

sex
Hombre      0.604505
Mujer       0.288988
Desconocido  0.106507
Name: proportion, dtype: float64

# realizamos una distribución con la proporción ajustada a hombres y mujeres, eliminando la proporción Desconocido
proporcion_hombre = proporciones_sexo.get('Hombre')
proporcion_mujer = proporciones_sexo.get('Mujer')
nueva_prop_hombe = round((proporcion_hombre / (proporcion_hombre + proporcion_mujer)),2)
print('Cálculo nueva proporción de hombres: ', nueva_prop_hombe)
nueva_prop_mujer = round((proporcion_mujer / (proporcion_hombre + proporcion_mujer)), 2)
print('Cálculo nueva proporción de mujeres: ', nueva_prop_mujer)

# vamos a reemplazar los valores Desconocido por valores Hombre y Mujer según proporción
# extraemos los índices de los registros con valor 'Desconocido'
indices = df_edited[df_edited['sexo'] == 'Desconocido'].index

# utilizando random.choice() de numpy, generamos valores aleatorios Hombre y Mujer aplicando la proporción
nuevos_valores = np.random.choice(['Hombre','Mujer'], size=len(indices), p=[nueva_prop_hombe, nueva_prop_mujer])

# asignamos los valores generados a los registros donde sexo es igual 'Desconocido'
df_edited.loc[indices, 'sexo'] = nuevos_valores

# volvemos a contar para ver que el resultado es correcto
df_edited['sexo'].value_counts(normalize=True)

Cálculo nueva proporción de hombres:  0.68
Cálculo nueva proporción de mujeres:  0.32
sex
Hombre      0.676431
Mujer       0.323569

```

- **rango_edad:** modificación de los rangos de edad en rangos de 10 años (0 a 9, 10 a 19, 20 a 29, etc).
- **tipo_accidente:** simplificación de la clasificación de los tipos de accidente a las siguientes categorías:
 - colision
 - choque_obstaculo



- atropello
- vuelco
- caida
- otros

Modificación de las siguientes características:

- rango_edad
- tipo_accidente

```
# creamos un diccionario para el mapeo de edad para pasar de los 18 rangos actuales a los 10 nuevos rangos
mapeo_edad = {
    'Menor de 5 años': '0-9', 'De 6 a 9 años': '0-9',
    'De 10 a 14 años': '10-20', 'De 15 a 17 años': '10-20', 'De 18 a 20 años': '10-20',
    'De 21 a 24 años': '21-29', 'De 25 a 29 años': '21-29',
    'De 30 a 34 años': '30-39', 'De 35 a 39 años': '30-39',
    'De 40 a 44 años': '40-49', 'De 45 a 49 años': '40-49',
    'De 50 a 54 años': '50-59', 'De 55 a 59 años': '50-59',
    'De 60 a 64 años': '60-69', 'De 65 a 69 años': '60-69',
    'De 70 a 74 años': '70-74', 'Más de 74 años': '>74',
    'Desconocido': 'desconocido'
}

# reemplazamos los valores en la columna 'rango_edad'
df_edited['rango_edad'] = df_edited['rango_edad'].replace(mapeo_edad)
df_edited['rango_edad'].value_counts() # comprobamos que los cambios se han realizado correctamente

rango_edad
40-49      54339
30-39      54084
21-29      46137
50-59      41455
desconocido 29725
60-69      17909
10-20      13257
>74        6023
0-9        4432
70-74      3889
```



```
df_edited['tipo_accidente'].value_counts()

tipo_accidente
Colisión fronto-lateral      68697
Alcance                         64049
Colisión lateral                40100
Choque contra obstáculo fijo   32638
Colisión múltiple               18020
Atropello a persona              17830
Caída                            16029
Colisión frontal                 6569
Otro                             4627
Solo salida de la vía            1144
Vuelco                            1025
Atropello a animal                504
Despeñamiento                     18
Name: count, dtype: int64

# vamos a simplificar los tipos de accidentes existentes
mapeo_tipo_accidente = {
    'Colisión fronto-lateral': 'colision',
    'Alcance': 'colision',
    'Colisión lateral': 'colision',
    'Choque contra obstáculo fijo': 'obstaculo fijo',
    'Colisión múltiple': 'colision',
    'Atropello a persona': 'atropello',
    'Caída': 'caida',
    'Colisión frontal': 'colision',
    'Otro': 'otros',
    'Solo salida de la vía': 'otros',
    'Vuelco': 'vuelco',
    'Atropello a animal': 'otros',
    'Despeñamiento': 'vuelco',
}

# reemplazamos los valores en la columna 'tipo_accidente'
df_edited['tipo_accidente'] = df_edited['tipo_accidente'].replace(mapeo_tipo_accidente)
df_edited['tipo_accidente'].value_counts() # comprobamos que los cambios se ha realizado correctamente

tipo_accidente
colision          197435
obstaculo fijo    32638
atropello          17830
caida              16029
otros              6275
vuelco              1043
Name: count, dtype: int64
```

3.4.4 Creación de características.

Crearemos una nueva columna llamada **dia_semana** a partir de nuestra columna 'fecha' tipo datetime, ya que puede ser un dato interesante para nuestro análisis.



- 0 corresponde a lunes.
- 1 corresponde a martes.
- 2 corresponde a miércoles.
- 3 corresponde a jueves.
- 4 corresponde a viernes.
- 5 corresponde a sábado.
- 6 corresponde a domingo.

```
# creamos una nueva columna con el día de la semana usando la función de pandas dt.weekday
df_edited['dia_semana'] = df_edited['fecha'].dt.weekday
df_edited['dia_semana']
```

También crearemos una nueva columna llamada **fallecido** donde el valor 1 representará 'fallecido' y el valor 0 'no fallecido'.

```
# vamos a convertir los código de lesividad de float a enteros
df_edited['cod_lesividad'] = df_edited['cod_lesividad'].astype(int)
# el código de lesividad 4 significa fallecido. La comparación nos devolverá True o False (lo convertimos a 0 o 1 con astype)
df_edited['fallecido'] = (df_edited['cod_lesividad'] == 4).astype(int)
df_edited['fallecido']
```

Por último, crearemos una característica llamada **lesividad_simplificada** con las categorías 'ilesos', 'herido' y 'fallecido'.

```
df_edited['lesividad'].value_counts()

lesiividad
Sin asistencia sanitaria                205237
Asistencia sanitaria sólo en el lugar del accidente    34497
Ingreso inferior o igual a 24 horas        10633
Atención en urgencias sin posterior ingreso      7694
Asistencia sanitaria inmediata en centro de salud o mutua  6594
Asistencia sanitaria ambulatoria con posterioridad   3279
Ingreso superior a 24 horas                  3130
Fallecido 24 horas                         173
Se desconoce                                13
Name: count, dtype: int64

# realizamos un mapeo de la lesividad para crear nuestra nueva variable 'lesiividad_simplificada'
mapeo_lesiividad = {
    'Sin asistencia sanitaria': 'ilesos',
    'Asistencia sanitaria sólo en el lugar del accidente': 'herido',
    'Ingreso inferior o igual a 24 horas': 'herido',
    'Atención en urgencias sin posterior ingreso': 'herido',
    'Asistencia sanitaria inmediata en centro de salud o mutua': 'herido',
    'Asistencia sanitaria ambulatoria con posterioridad': 'herido',
    'Ingreso superior a 24 horas': 'herido',
    'Fallecido 24 horas': 'fallecido',
    'Se desconoce': 'ilesos',
}
df_edited['lesiividad_simplificada'] = df_edited['lesiividad'].replace(mapeo_lesiividad)
df_edited['lesiividad_simplificada'].value_counts()

lesiividad_simplificada
ilesos      205250
herido       65827
fallecido     173
```



4. EXPLORACIÓN DE DATOS.

Una vez eliminado algunas características, creado otras y tratado los nulos, vamos a proceder a la exploración y visualización mediante gráficos de los datos, de manera que nos puedan dar información de qué características pueden ser útiles para nuestro modelo y puedan afectar a nuestra variable objetivo lesividad.

4.1 Análisis descriptivo de características.

- **dia_semana:** utilizamos agrupaciones y funciones pivotantes para ver el número total y la proporción de víctimas de accidentes según el día de la semana y lo visualizamos usando la librería matplotlib. Analizamos también la variable 'fallecido' en particular.

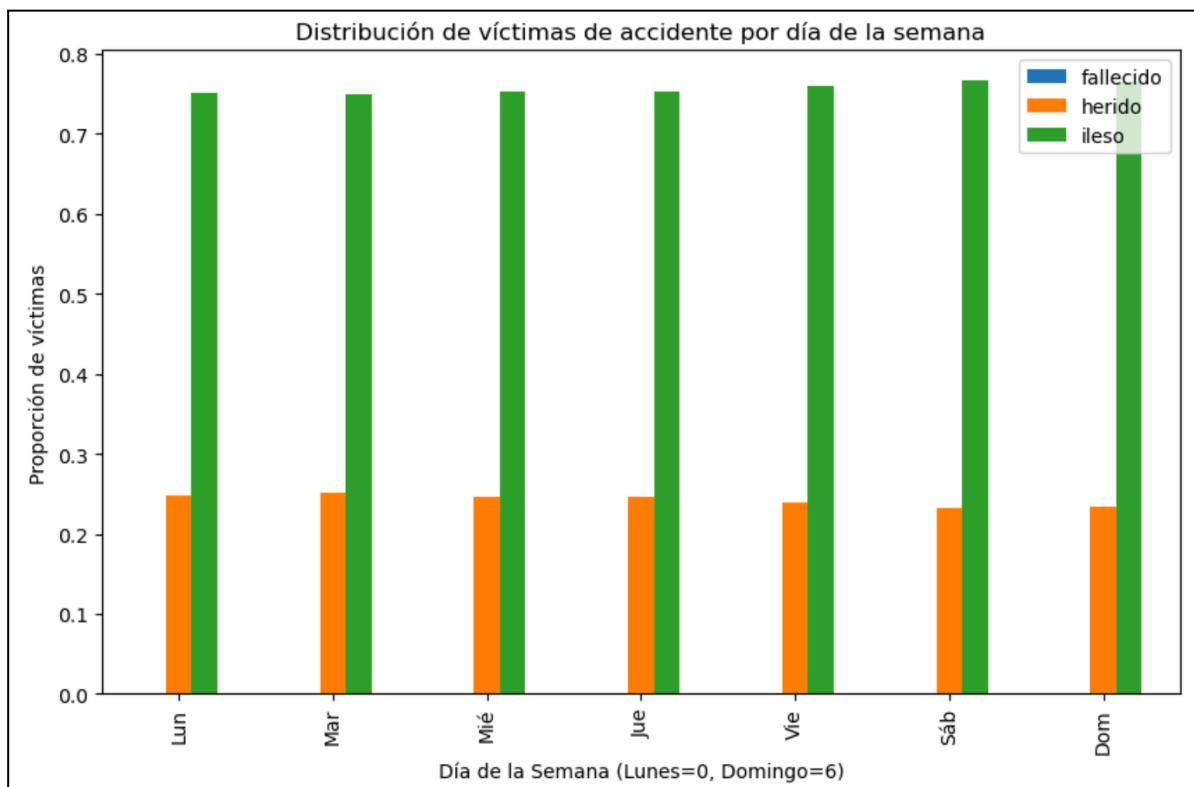
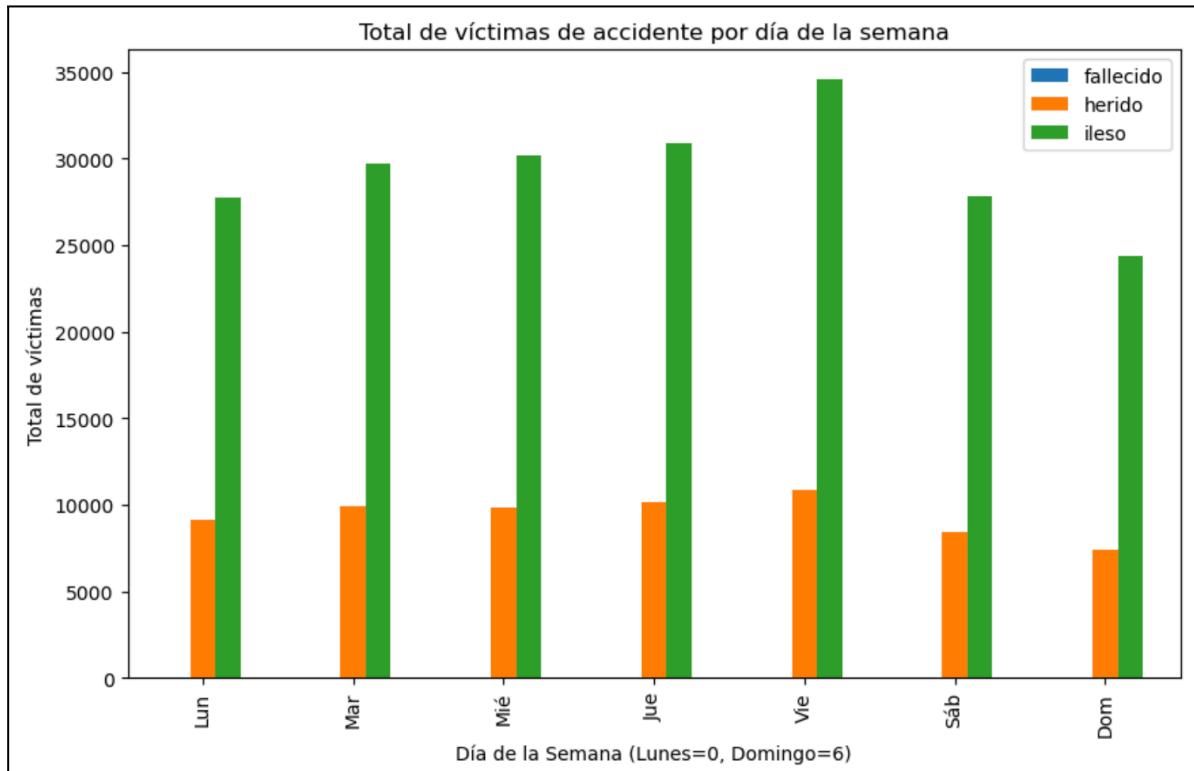
```
# Calculamos el total de accidentados para cada tipo de lesividad a lo largo de los días
total_por_dia = total_victimas_dia.pivot(index='dia_semana', columns='lesividad_simplificada', values='total')
total_por_dia
```

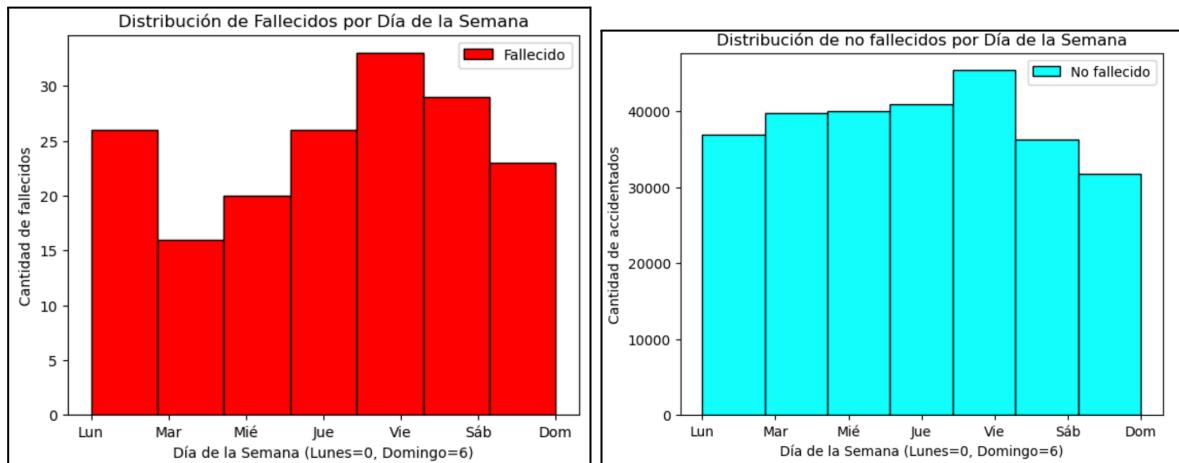
```
# Calculamos el total de accidentados para cada tipo de lesividad a lo largo de los días
total_por_dia = total_victimas_dia.pivot(index='dia_semana', columns='lesividad_simplificada', values='total')
total_por_dia
```

lesividad_simplificada	fallecido	herido	ilesos
dia_semana			
0	26	9131	27747
1	16	9963	29741
2	20	9842	30184
3	26	10127	30843
4	33	10869	34532
5	29	8462	27828
6	23	7433	24375

```
# Calculamos la proporción por día
# Usamos div() con axis=1 para dividir cada valor entre la suma de su fila (total de víctimas ese día)
proporcion_por_dia = total_por_dia.div(total_por_dia.sum(axis=1), axis=0)
proporcion_por_dia
```

lesividad_simplificada	fallecido	herido	ilesos
dia_semana			
0	0.000705	0.247426	0.751870
1	0.000403	0.250831	0.748766
2	0.000499	0.245767	0.753733
3	0.000634	0.247024	0.752342
4	0.000726	0.239226	0.760048
5	0.000798	0.232991	0.766211
6	0.000723	0.233514	0.765763





Conclusión: podemos deducir que la variable `dia_semana` puede tener importancia en nuestro modelo, ya que vemos que tiene influencia en la variable `lesividad` dependiendo del día de la semana, aunque aparentemente no es muy destacada. En los gráficos se puede apreciar que el viernes es el día de mayor incidencia de tanto de accidentados como de fallecidos.

- **sexo:** utilizamos agrupaciones y funciones pivotantes para ver el número total y la proporción de víctimas de accidentes según sexo de la víctima y lo visualizamos usando la librería matplotlib. Analizamos también la variable 'fallecido' en particular.

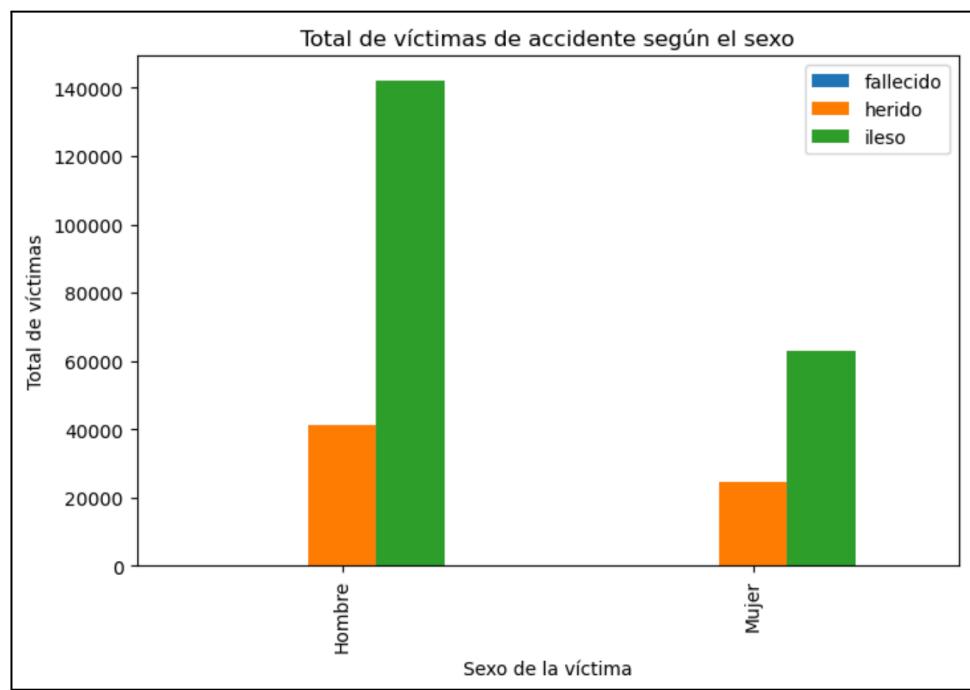
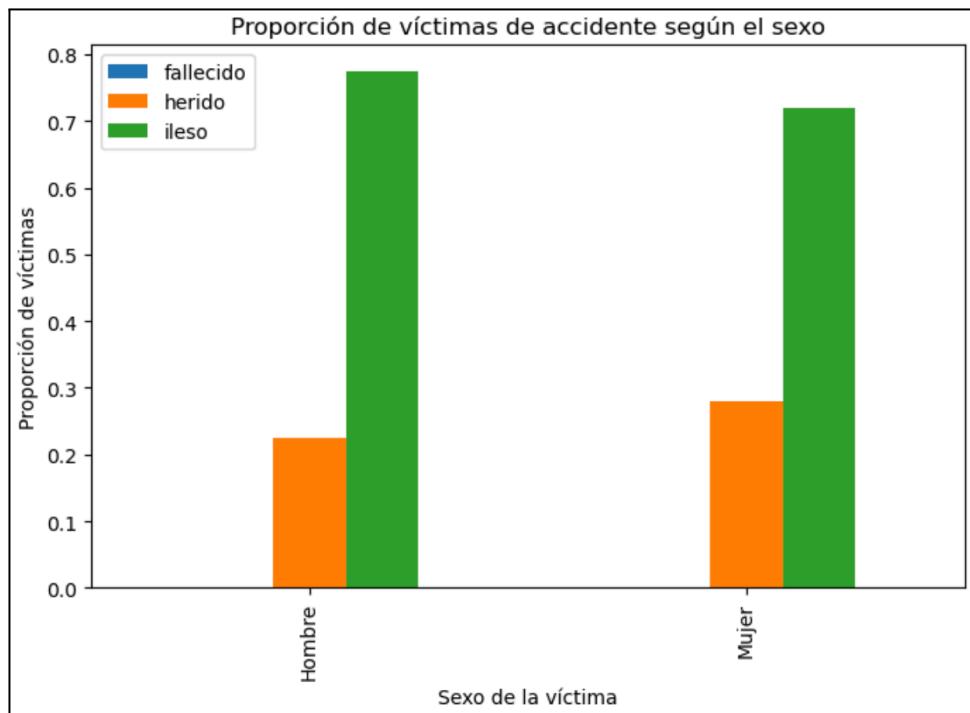
```
# Agrupamos y contamos el número de registros por sexo según el tipo de lesividad
total_victimas_sexo = df_edited.groupby(['sexo', 'lesividad_simplificada']).size().reset_index(name='total')
```

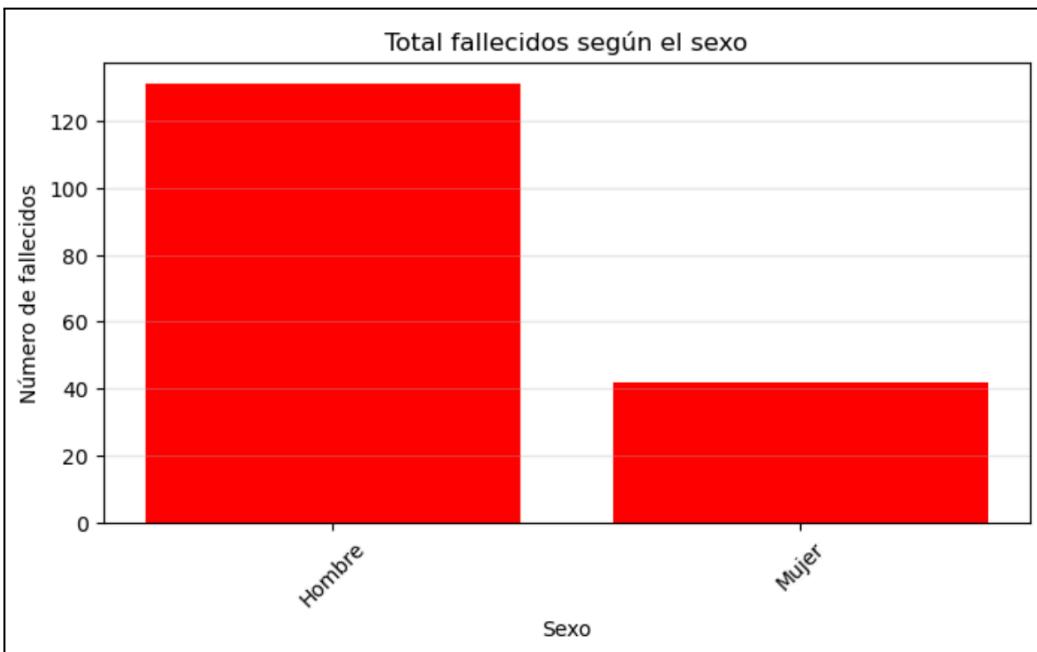
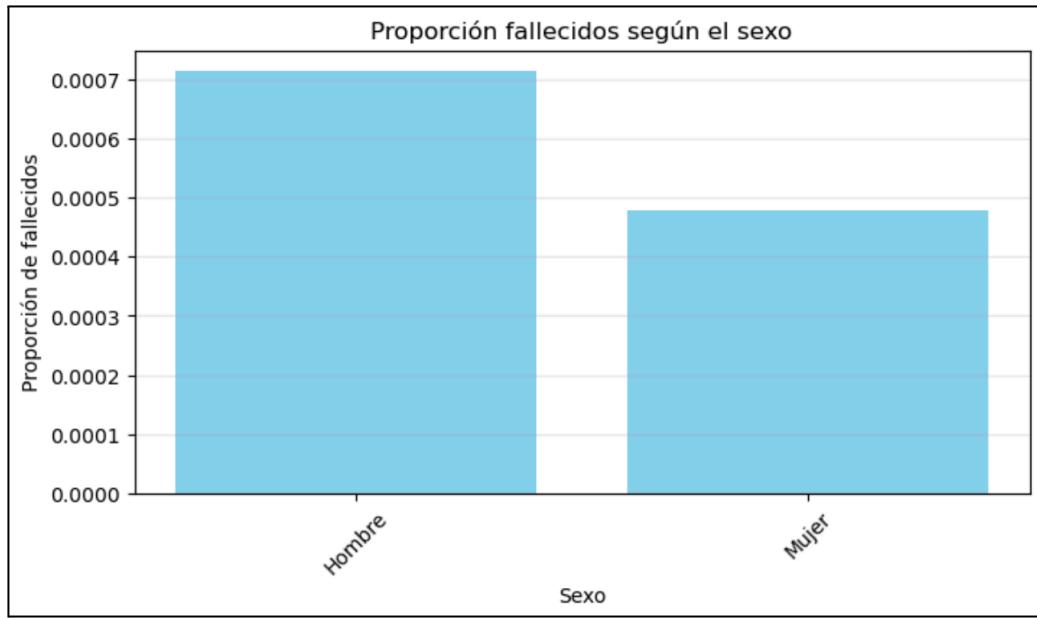
```
# Calculamos el total de accidentados para cada tipo de lesividad según el sexo
total_por_sexo = total_victimas_sexo.pivot(index='sexo', columns='lesividad_simplificada', values='total')
total_por_sexo
```

lesividad_simplificada	fallecido	herido	ilesos
sexo			
Hombre	131	41192	142159
Mujer	42	24635	63091

```
# Calculamos la proporción por sexo
proporcion_por_sexo = total_por_sexo.div(total_por_sexo.sum(axis=1), axis=0)
proporcion_por_sexo
```

lesividad_simplificada	fallecido	herido	ilesos
sexo			
Hombre	0.000714	0.224502	0.774784
Mujer	0.000479	0.280683	0.718838





Conclusión: vemos que tanto la proporción como el número de víctimas de accidentes, así como de fallecidos, es mayor en hombres que en mujeres, por lo que a priori, el sexo es otra variable a tener en cuenta.



- **rango_edad**: al igual que con características anteriores, valoramos el rango de edad respecto a la lesividad.

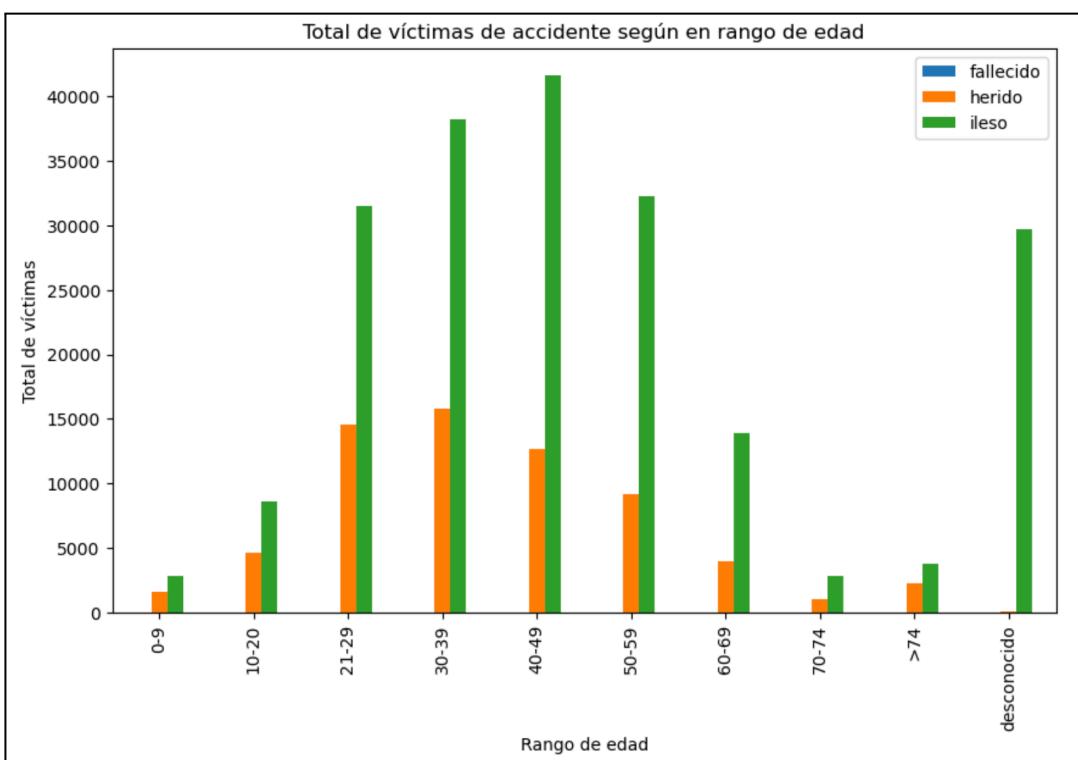
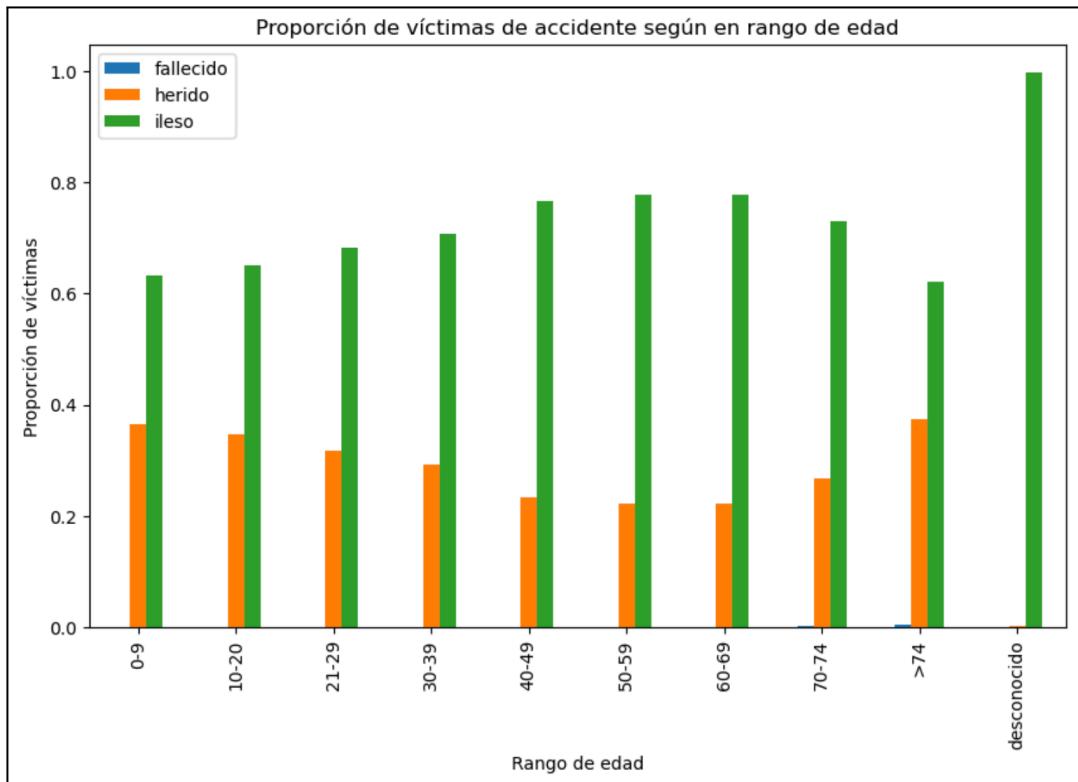
```
# Agrupamos y contamos el número de registros por rango de edad según el tipo de lesividad
total_victimas_edad = df_edited.groupby(['rango_edad', 'lesividad_simplificada']).size().reset_index(name='total')
```

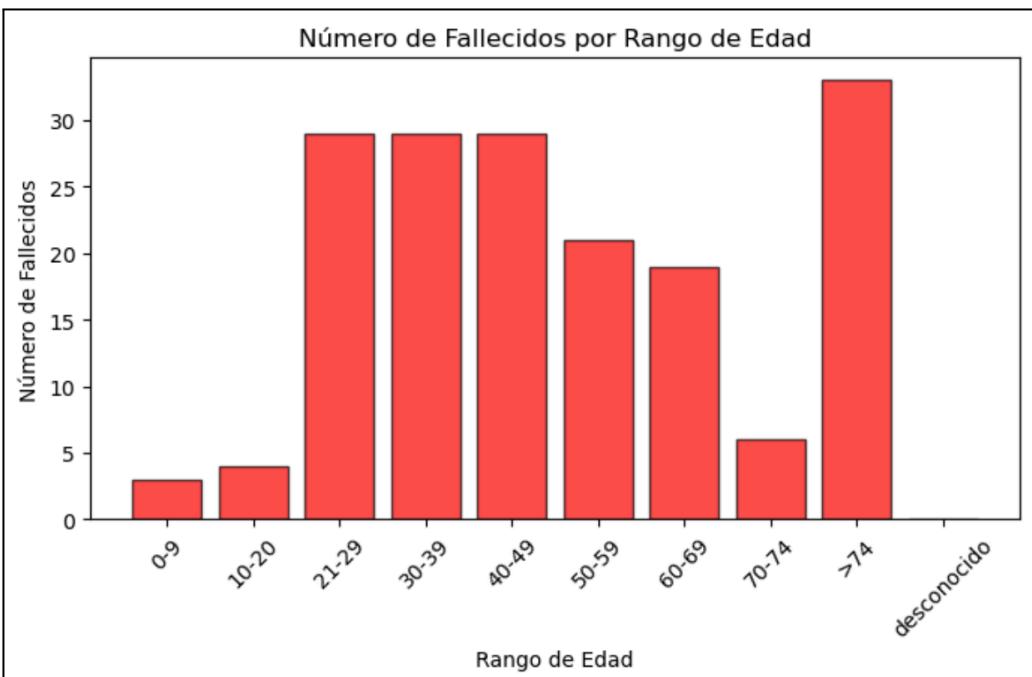
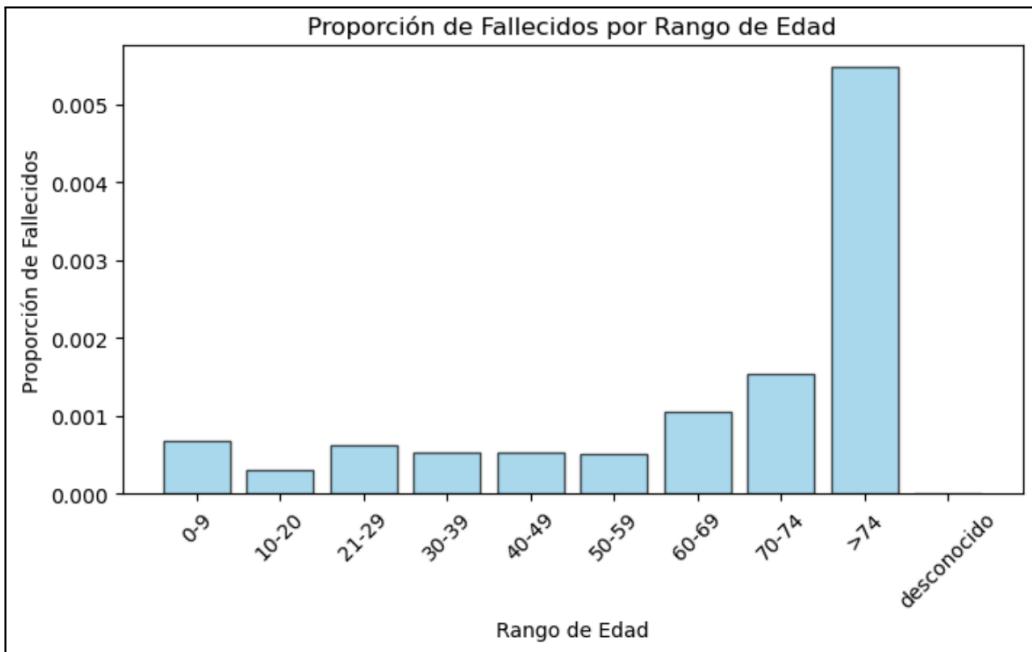
```
# Calculamos el total de accidentados para cada tipo de lesividad según el rango de edad
total_por_edad = total_victimas_edad.pivot(index='rango_edad', columns='lesividad_simplificada', values='total')
total_por_edad
```

lesividad_simplificada	fallecido	herido	ilesos
rango_edad			
0-9	3.0	1622.0	2807.0
10-20	4.0	4613.0	8640.0
21-29	29.0	14595.0	31513.0
30-39	29.0	15810.0	38245.0
40-49	29.0	12682.0	41628.0
50-59	21.0	9180.0	32254.0
60-69	19.0	3964.0	13926.0
70-74	6.0	1044.0	2839.0
>74	33.0	2251.0	3739.0
desconocido	NaN	66.0	29659.0

```
# Calculamos la proporción por rango de edad
proporcion_por_edad = total_por_edad.div(total_por_edad.sum(axis=1), axis=0)
proporcion_por_edad
```

lesividad_simplificada	fallecido	herido	ilesos
rango_edad			
0-9	0.000677	0.365975	0.633348
10-20	0.000302	0.347967	0.651731
21-29	0.000629	0.316340	0.683031
30-39	0.000536	0.292323	0.707141
40-49	0.000534	0.233387	0.766080
50-59	0.000507	0.221445	0.778048
60-69	0.001061	0.221341	0.777598
70-74	0.001543	0.268449	0.730008
>74	0.005479	0.373734	0.620787
desconocido	NaN	0.002220	0.997780





- conclusión: de los gráficos anteriores podemos deducir que nuestra característica 'rango_edad' es importante para nuestro modelo, ya que vemos que el rango de mortalidad es muy superior para rangos de edad superiores a 70 años respecto al resto de rangos. También, como es lógico, existen muy pocos fallecidos menores de edad, ya que estos no pueden conducir y sólo podrán ser víctimas mortales en caso de ser acompañantes o peatón.



- **tipo_persona:** analizamos la característica tipo_persona respecto a la lesividad.

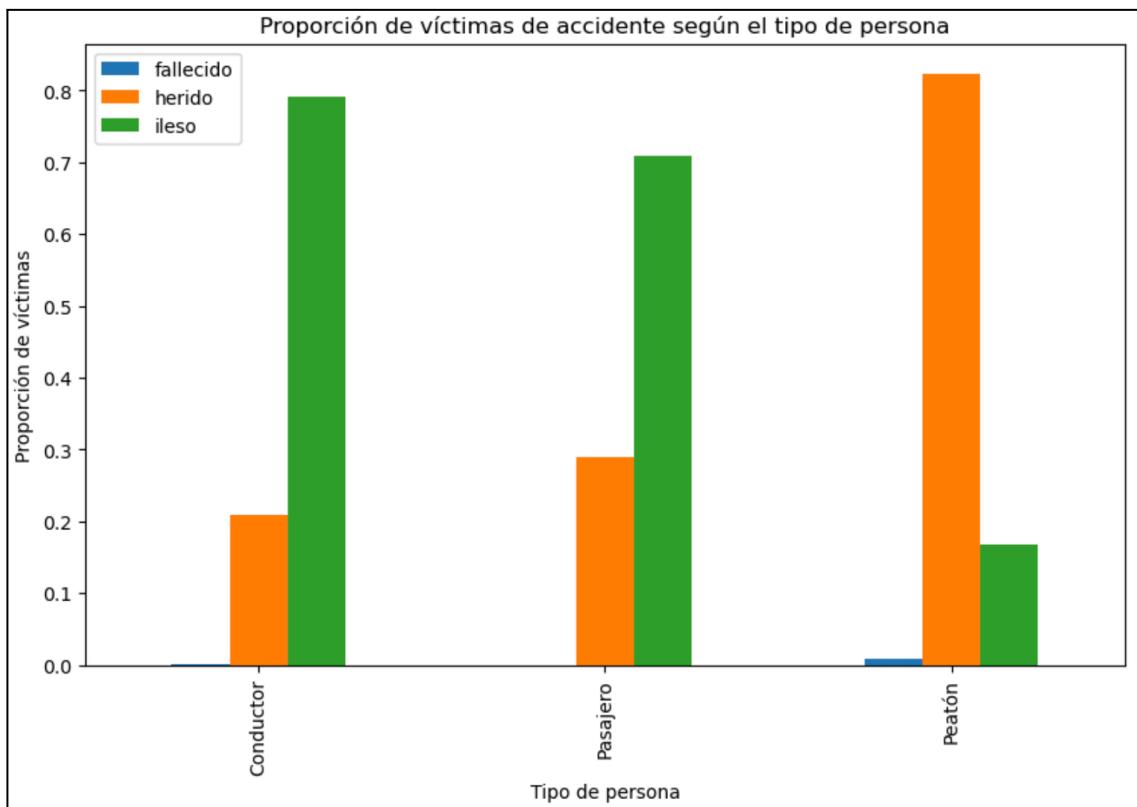
```
# Agrupamos y contamos el número de registros por tipo de persona según el tipo de lesividad
total_victimas_tpersona = df_edited.groupby(['tipo_persona', 'lesividad_simplificada']).size().reset_index(name='total')
```

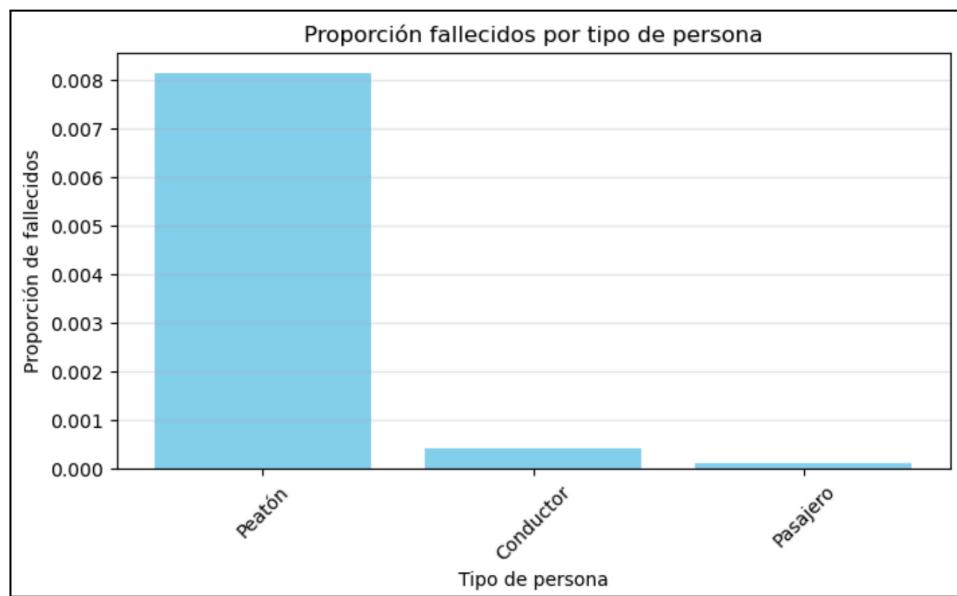
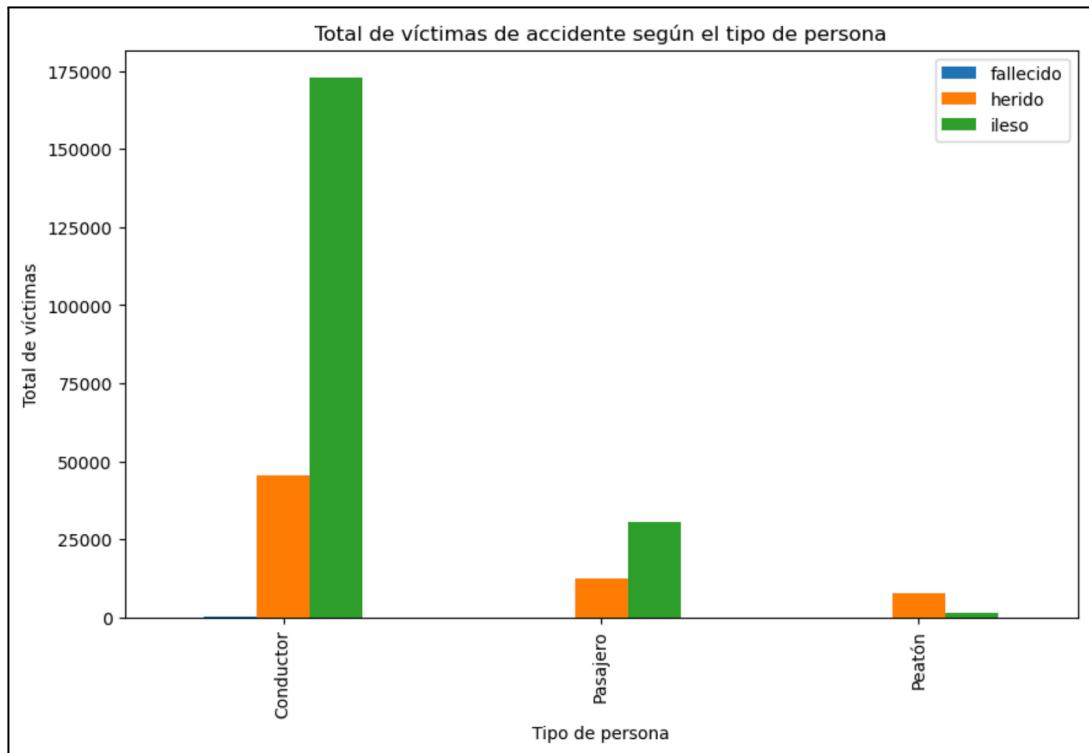
```
# Calculamos el total de accidentados para cada tipo de lesividad según el tipo de persona
total_por_tpersona = total_victimas_tpersona.pivot(index='tipo_persona', columns='lesividad_simplificada', values='total')
total_por_tpersona
```

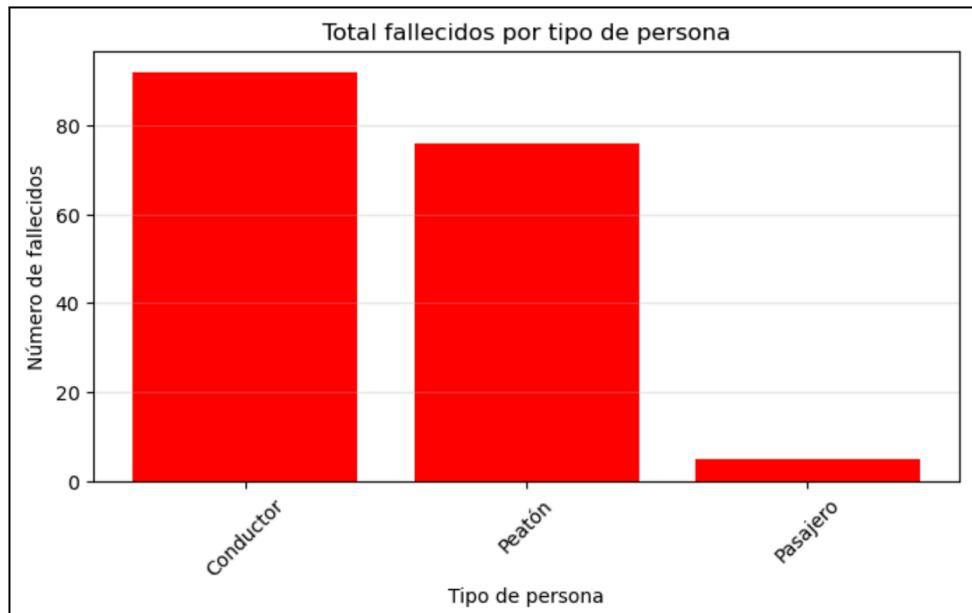
lesividad_simplificada	fallecido	herido	ilesos
tipo_persona			
Conductor	92	45620	173043
Pasajero	5	12538	30642
Peatón	76	7669	1565


```
# Calculamos la proporción por tipo de persona
proporcion_por_tpersona = total_por_tpersona.div(total_por_tpersona.sum(axis=1), axis=0)
proporcion_por_tpersona
```

lesividad_simplificada	fallecido	herido	ilesos
tipo_persona			
Conductor	0.000421	0.208544	0.791036
Pasajero	0.000116	0.290332	0.709552
Peatón	0.008163	0.823738	0.168099







Conclusión: vemos que la característica 'tipo_persona' es importante para nuestro modelo. De los gráficos podemos deducir que la víctima más común es un conductor, mientras que cuando existen accidentados que son peatones, la posibilidad de fallecimiento es muy superior.



- **tipo_vehiculo:** en el análisis inicial, pudimos ver que contamos con una variedad de tipos de vehículos bastante elevada, hasta 40 en total. Existen muchas categorías que representan menos de un 0,5% del total de registros del dataframe, por lo que se ha optado por agruparlas en un tipo de vehículo llamado 'Otros', además de renombrar algunas de ellas para simplificar la lectura de datos.

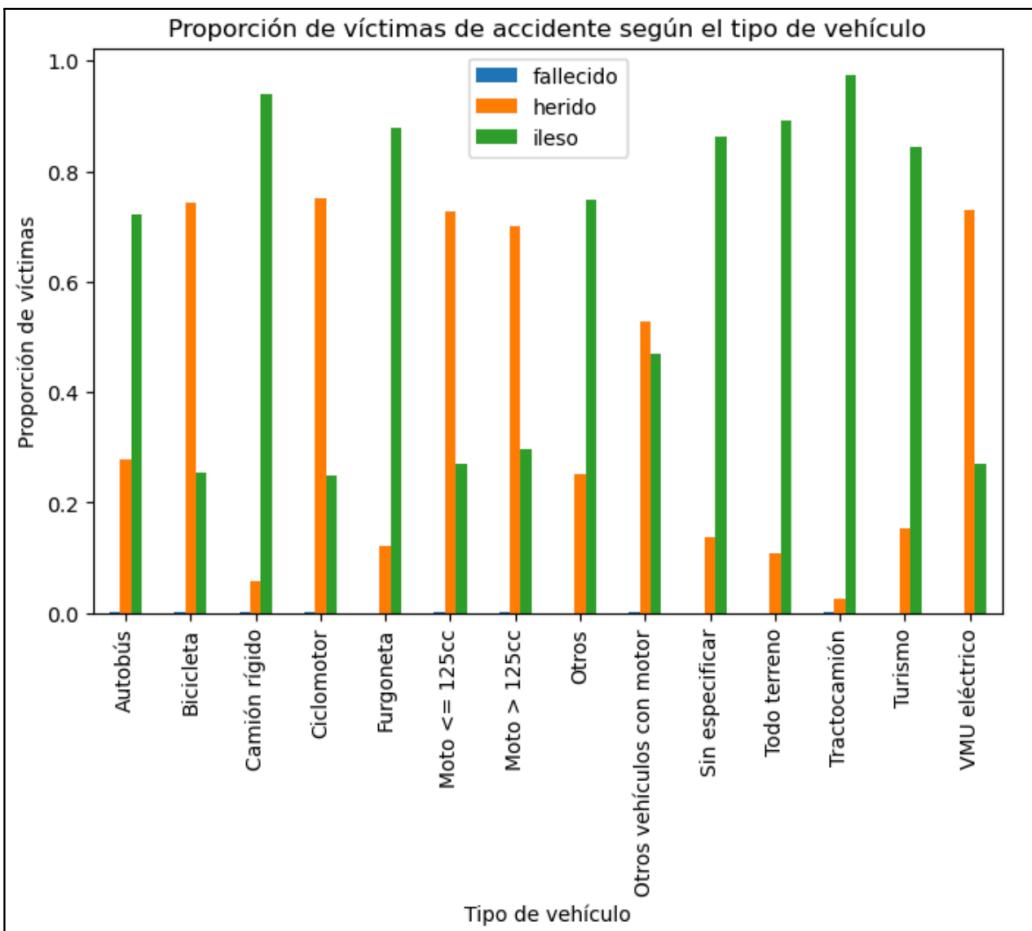
En cuanto al análisis de esta variable respecto a la lesividad el resultado ha sido el siguiente:

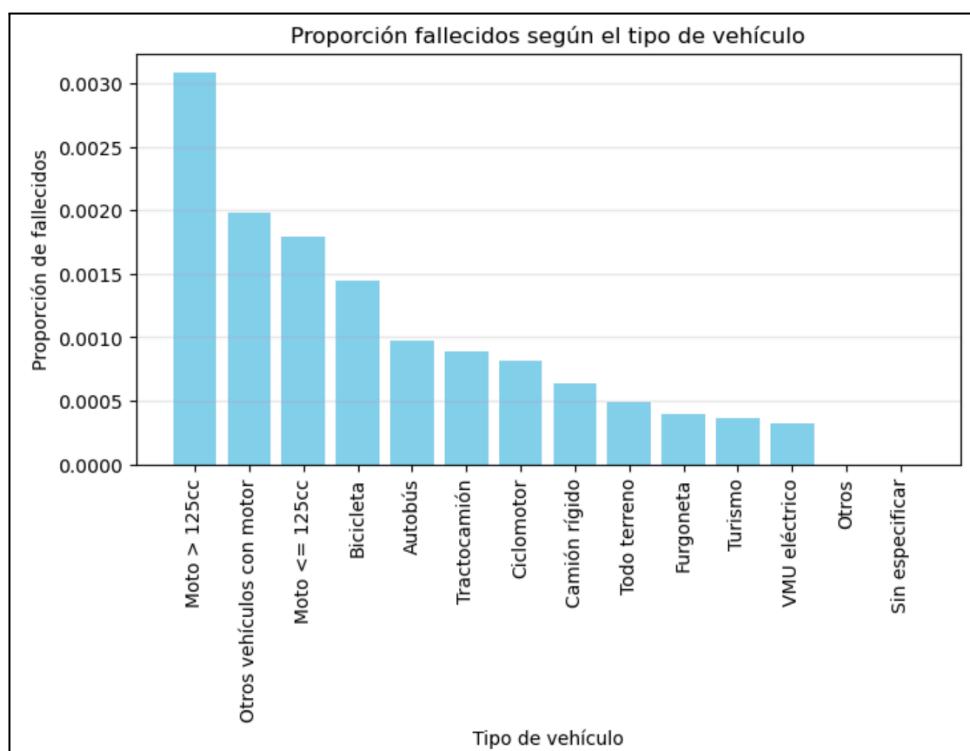
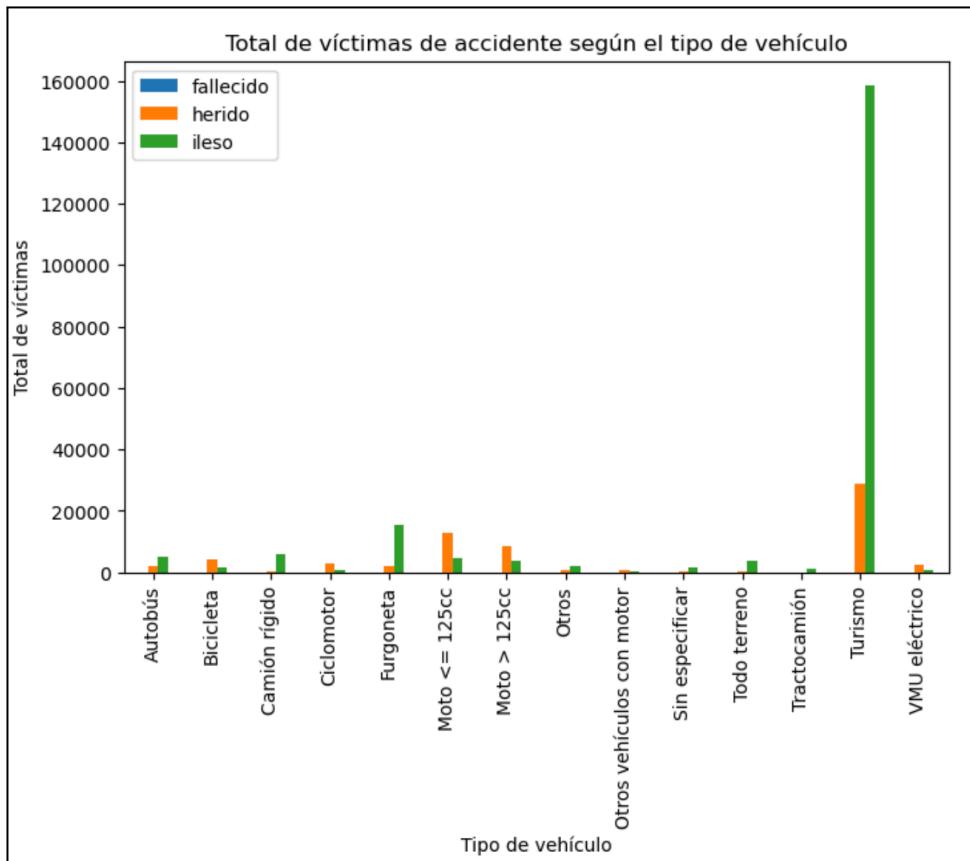
# Agrupamos y contamos el número de registros por tipo de vehículo según el tipo de lesividad																																																																
total_victimas_tvehiculo = df_edited.groupby(['tipo_vehiculo', 'lesividad_simplificada']).size().reset_index(name='total')																																																																
# Calculamos la proporción por tipo de vehículo																																																																
proporcion_por_tvehiculo = total_por_tvehiculo.div(total_por_tvehiculo.sum(axis=1), axis=0)																																																																
proporcion_por_tvehiculo																																																																
<table border="1"><thead><tr><th>lesividad_simplificada</th><th>fallecido</th><th>herido</th><th>ilesos</th></tr><tr><th>tipo_vehiculo</th><th></th><th></th><th></th></tr></thead><tbody><tr><td>Autobús</td><td>0.000975</td><td>0.277043</td><td>0.721982</td></tr><tr><td>Bicicleta</td><td>0.001448</td><td>0.744026</td><td>0.254526</td></tr><tr><td>Camión rígido</td><td>0.000636</td><td>0.058319</td><td>0.941046</td></tr><tr><td>Ciclomotor</td><td>0.000819</td><td>0.751024</td><td>0.248157</td></tr><tr><td>Furgoneta</td><td>0.000399</td><td>0.120629</td><td>0.878973</td></tr><tr><td>Moto <= 125cc</td><td>0.001792</td><td>0.726631</td><td>0.271577</td></tr><tr><td>Moto > 125cc</td><td>0.003079</td><td>0.699592</td><td>0.297329</td></tr><tr><td>Otros</td><td>NaN</td><td>0.252083</td><td>0.747917</td></tr><tr><td>Otros vehículos con motor</td><td>0.001984</td><td>0.528770</td><td>0.469246</td></tr><tr><td>Sin especificar</td><td>NaN</td><td>0.137291</td><td>0.862709</td></tr><tr><td>Todo terreno</td><td>0.000495</td><td>0.107125</td><td>0.892380</td></tr><tr><td>Tractocamión</td><td>0.000890</td><td>0.024911</td><td>0.974199</td></tr><tr><td>Turismo</td><td>0.000368</td><td>0.154238</td><td>0.845394</td></tr><tr><td>VMU eléctrico</td><td>0.000320</td><td>0.730621</td><td>0.269058</td></tr></tbody></table>	lesividad_simplificada	fallecido	herido	ilesos	tipo_vehiculo				Autobús	0.000975	0.277043	0.721982	Bicicleta	0.001448	0.744026	0.254526	Camión rígido	0.000636	0.058319	0.941046	Ciclomotor	0.000819	0.751024	0.248157	Furgoneta	0.000399	0.120629	0.878973	Moto <= 125cc	0.001792	0.726631	0.271577	Moto > 125cc	0.003079	0.699592	0.297329	Otros	NaN	0.252083	0.747917	Otros vehículos con motor	0.001984	0.528770	0.469246	Sin especificar	NaN	0.137291	0.862709	Todo terreno	0.000495	0.107125	0.892380	Tractocamión	0.000890	0.024911	0.974199	Turismo	0.000368	0.154238	0.845394	VMU eléctrico	0.000320	0.730621	0.269058
lesividad_simplificada	fallecido	herido	ilesos																																																													
tipo_vehiculo																																																																
Autobús	0.000975	0.277043	0.721982																																																													
Bicicleta	0.001448	0.744026	0.254526																																																													
Camión rígido	0.000636	0.058319	0.941046																																																													
Ciclomotor	0.000819	0.751024	0.248157																																																													
Furgoneta	0.000399	0.120629	0.878973																																																													
Moto <= 125cc	0.001792	0.726631	0.271577																																																													
Moto > 125cc	0.003079	0.699592	0.297329																																																													
Otros	NaN	0.252083	0.747917																																																													
Otros vehículos con motor	0.001984	0.528770	0.469246																																																													
Sin especificar	NaN	0.137291	0.862709																																																													
Todo terreno	0.000495	0.107125	0.892380																																																													
Tractocamión	0.000890	0.024911	0.974199																																																													
Turismo	0.000368	0.154238	0.845394																																																													
VMU eléctrico	0.000320	0.730621	0.269058																																																													

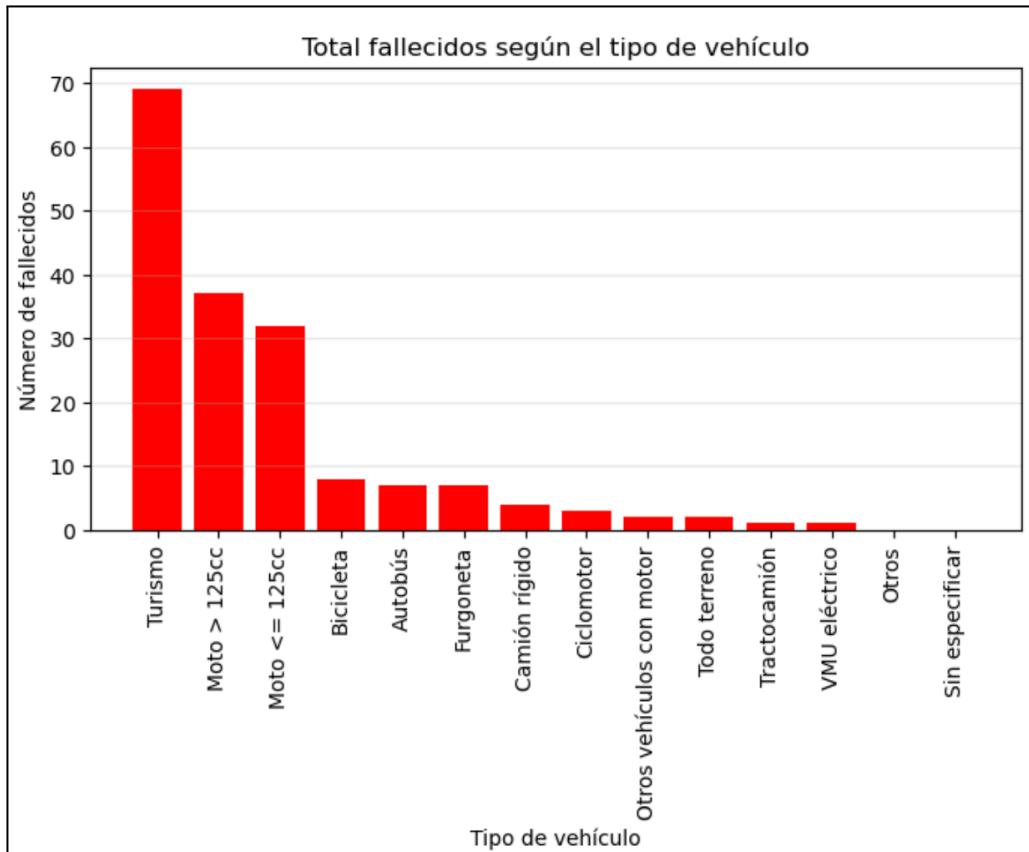


```
# Calculamos el total de accidentados para cada tipo de lesividad según el tipo de vehículo
total_por_tvehiculo = total_victimas_tvehiculo.pivot(index='tipo_vehiculo', columns='lesividad_simplificada', values='total')
total_por_tvehiculo
```

lesividad_simplificada	fallecido	herido	ilesos
tipo_vehiculo			
Autobús	7.0	1990.0	5186.0
Bicicleta	8.0	4110.0	1406.0
Camión rígido	4.0	367.0	5922.0
Ciclomotor	3.0	2751.0	909.0
Furgoneta	7.0	2118.0	15433.0
Moto <= 125cc	32.0	12974.0	4849.0
Moto > 125cc	37.0	8407.0	3573.0
Otros	NaN	696.0	2065.0
Otros vehículos con motor	2.0	533.0	473.0
Sin especificar	NaN	222.0	1395.0
Todo terreno	2.0	433.0	3607.0
Tractocamión	1.0	28.0	1095.0
Turismo	69.0	28917.0	158497.0
VMU eléctrico	1.0	2281.0	840.0



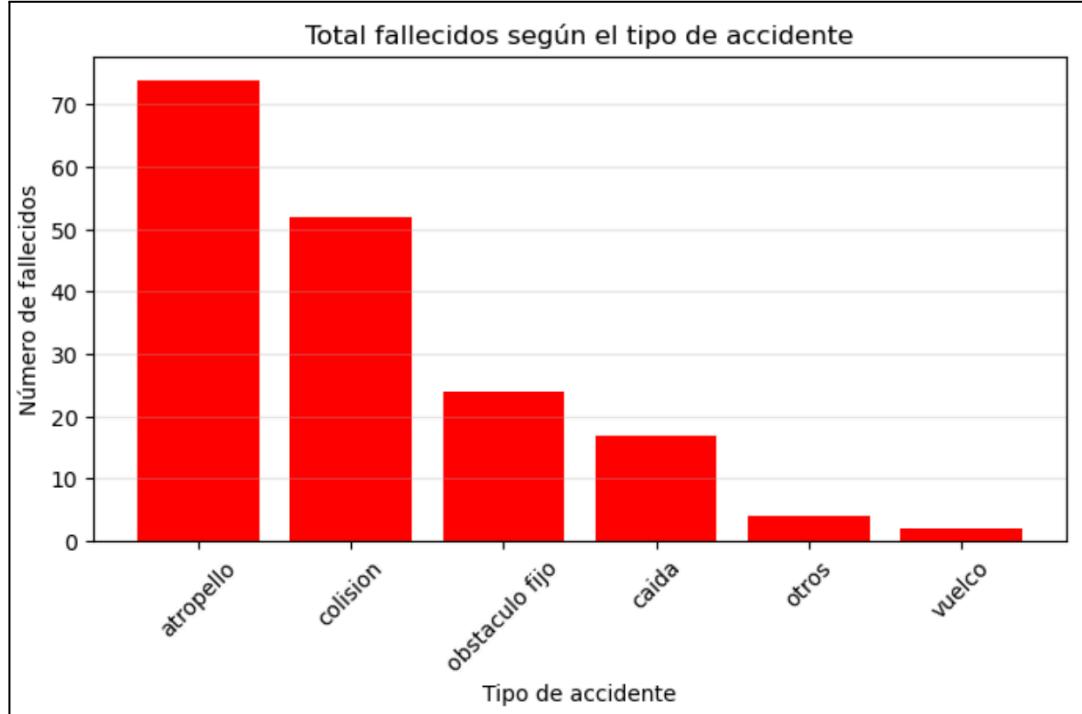
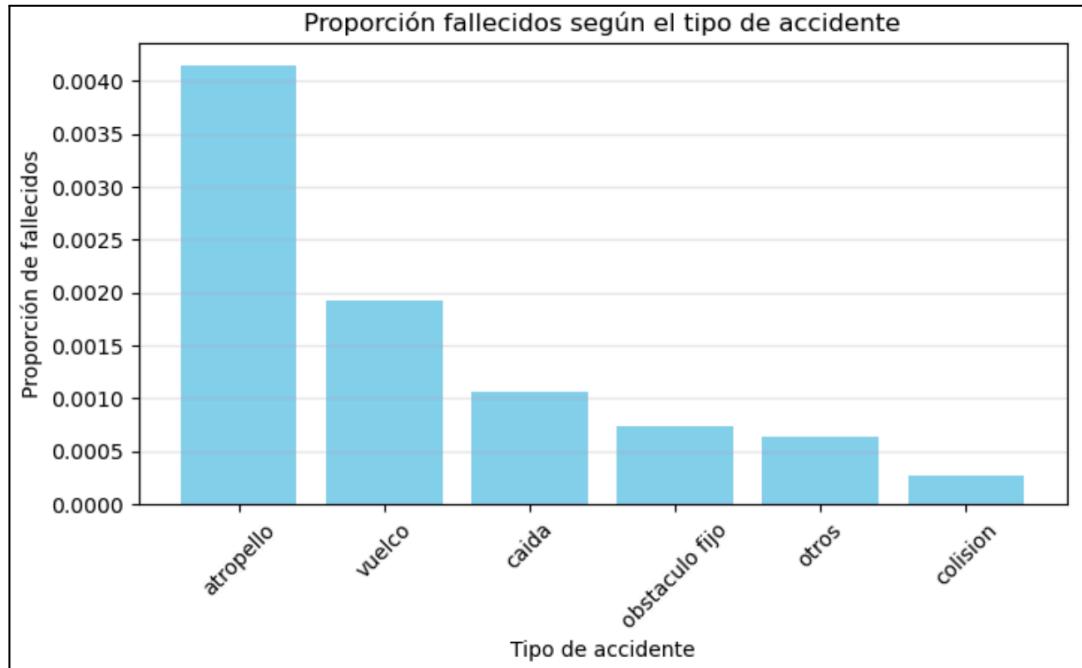




Conclusión: el tipo de vehículo más representado es claramente el turismo. Dependiendo del tipo de vehículo puede tener una mayor incidencia a la hora de resultar herido o fallecido una víctima de accidente, ya que vehículos con poca representación muestran índices de fallecidos relativamente destacados.



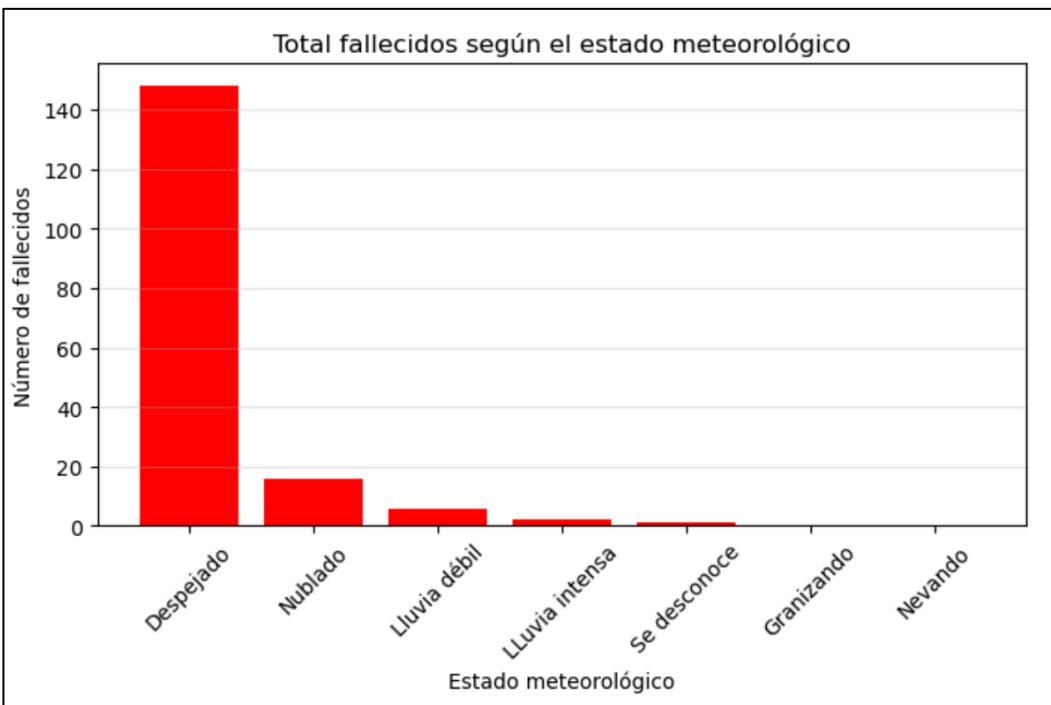
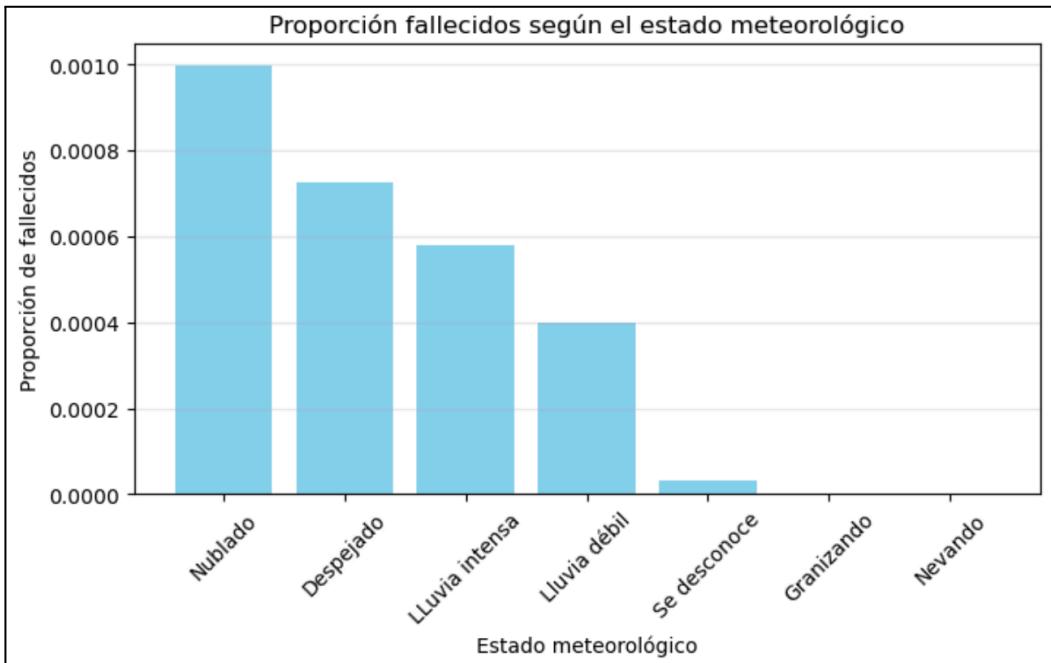
- **tipo_accidente**: para esta característica, vamos solamente a centrarnos en el caso de los fallecidos.



- Conclusión: vemos que accidentes de tipo 'despeñamiento' o 'atropello' presentan una mayor mortalidad que el resto. El mayor número de fallecidos se producen en accidentes de tipo 'atropello' o 'colisión'.



- **estado_meteorológico**: para esta característica, vamos solamente a centrarnos en el caso de los fallecidos.



- Conclusión: de estos gráficos no podemos sacar una conclusión clara de que, en caso de tiempo meteorológico adverso, este incide claramente en caso de víctimas mortales.

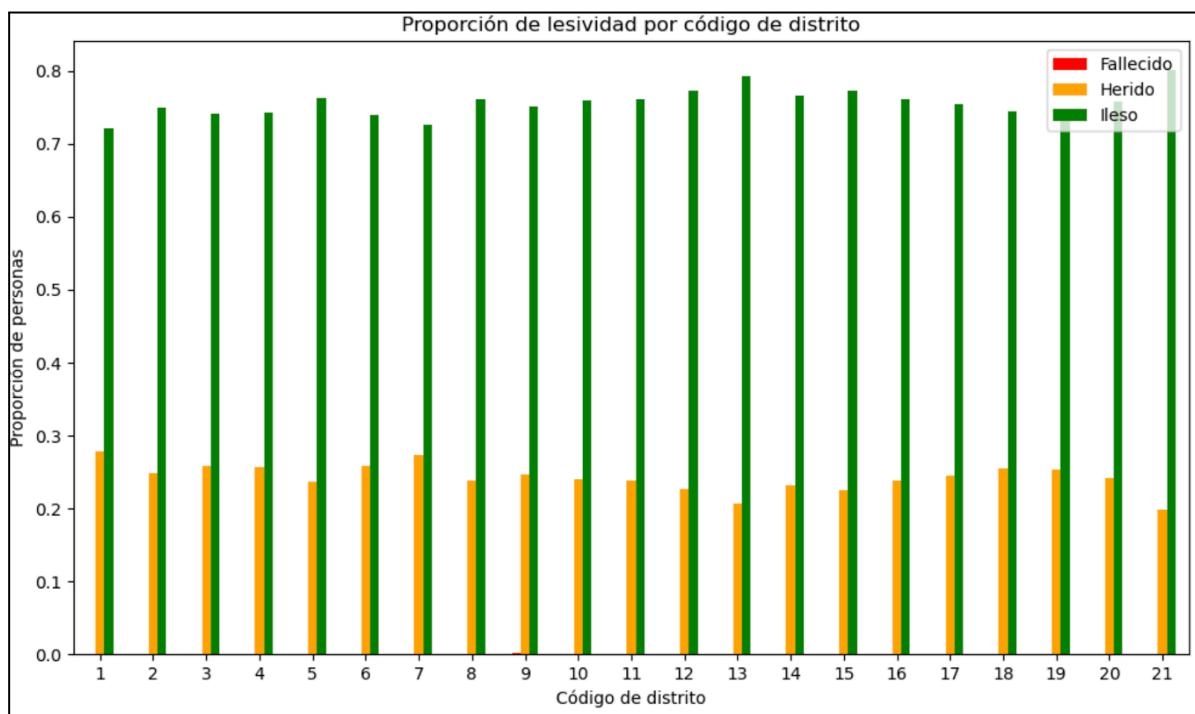
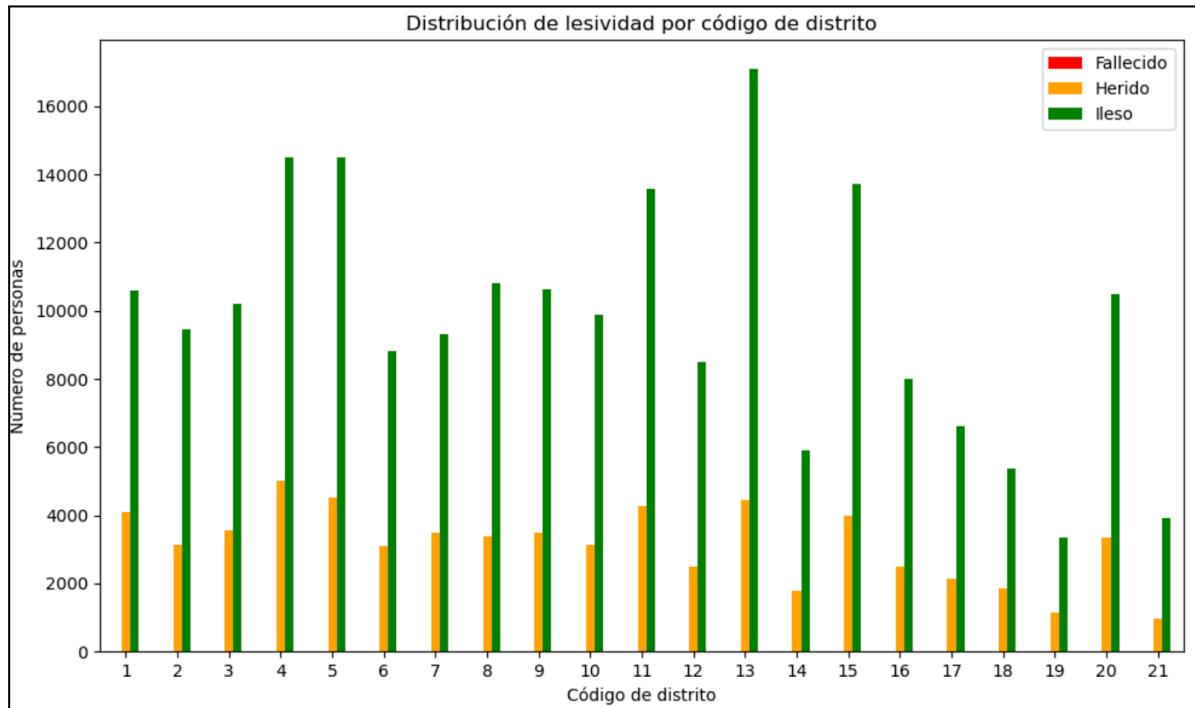


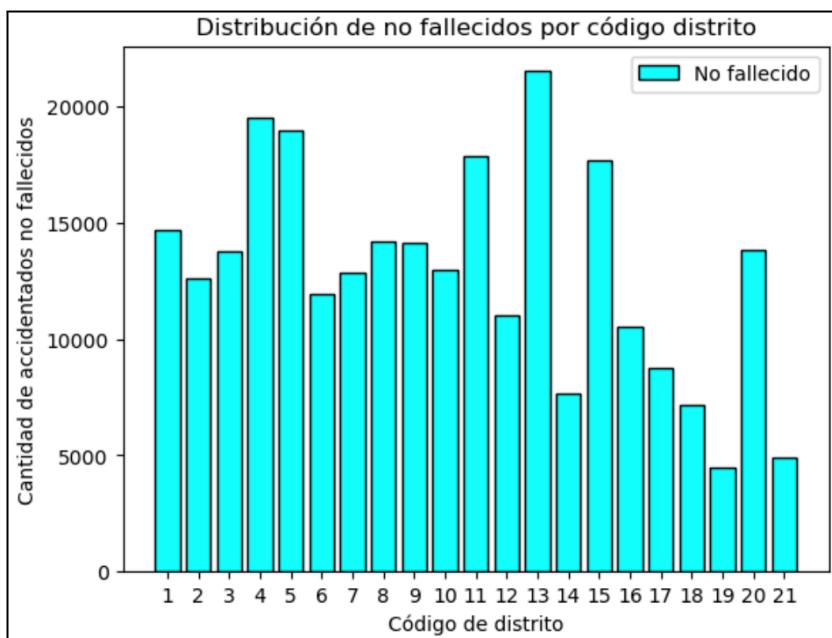
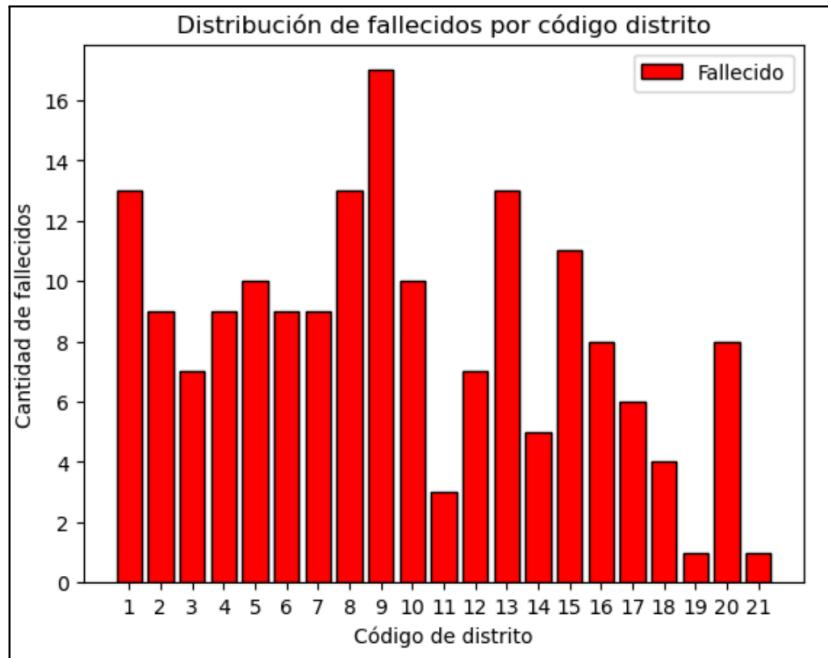
- **cod_distrito**: para esta característica volveremos agrupaciones y funciones pivotantes para su análisis respecto a la variable objetivo lesividad, así como en el caso particular de los fallecidos.

```
# Vamos a obtener un dataframe para representarlo: agrupamos por código de distrito y lesividad
lesividad_distrito = df_edited.groupby(['cod_distrito', 'lesividad_simplificada']).size()
lesividad_distrito
```

```
# pasamos de formato largo a ancho para conseguir una mejor visualización
lesividad_distrito = lesividad_distrito.unstack(fill_value=0)
lesividad_distrito
```

cod_distrito	lesividad_simplificada	fallecido	herido	ilesos
1.0		13	4087	10597
2.0		9	3139	9460
3.0		7	3562	10209
4.0		9	5012	14519
5.0		10	4504	14490
6.0		9	3089	8823
7.0		9	3504	9328
8.0		13	3390	10820
9.0		17	3489	10624
10.0		10	3120	9870
11.0		3	4268	13585
12.0		7	2491	8500
13.0		13	4441	17083
14.0		5	1788	5896
15.0		11	3997	13710
16.0		8	2502	8004
17.0		6	2145	6613
18.0		4	1838	5360
19.0		1	1140	3351
20.0		8	3349	10498
21.0		1	972	3910

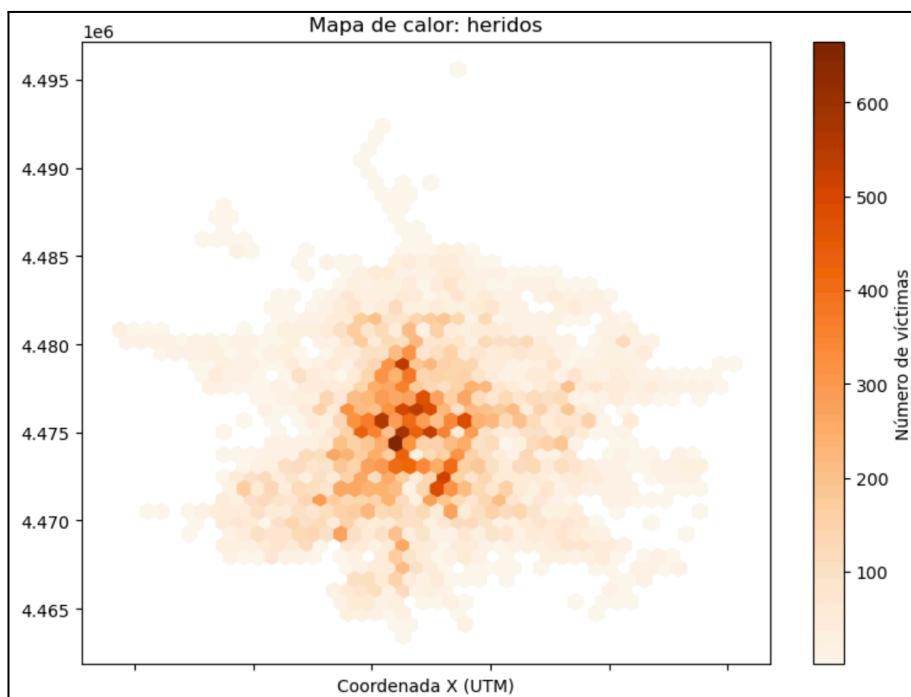
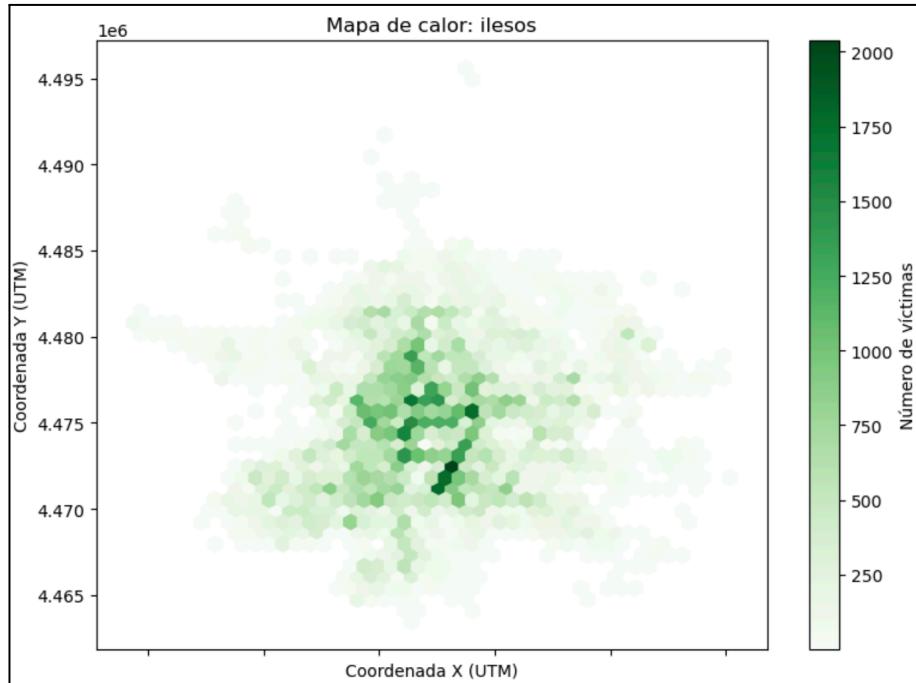


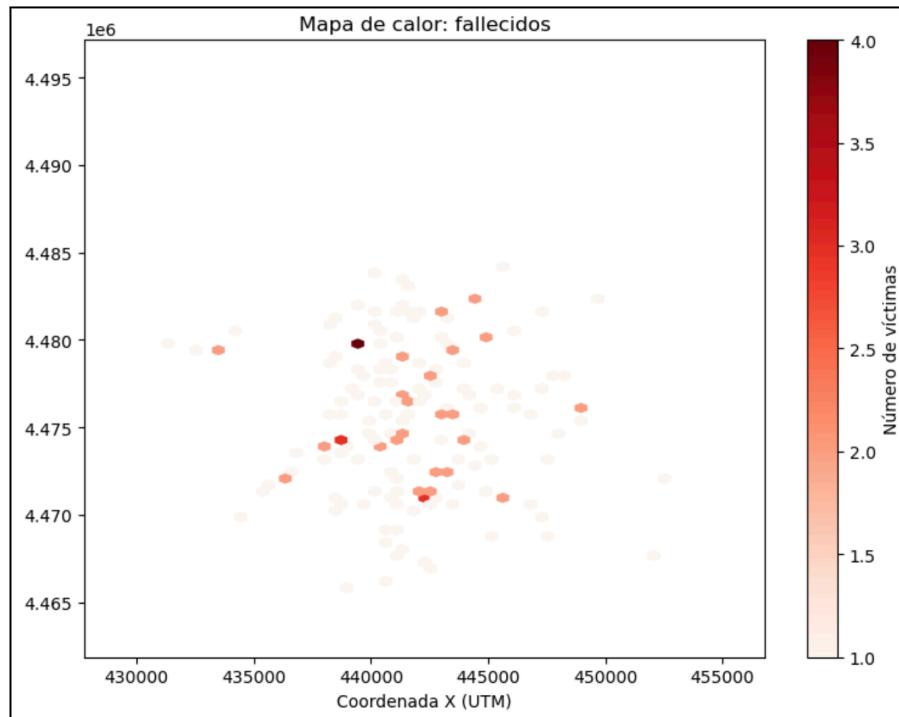


- Conclusión: podemos deducir que la variable 'cod_distrito' podría tener cierta importancia en nuestro modelo, ya que vemos influencia en la variable 'lesividad' dependiendo del distrito donde se ha producido el accidente, y podemos ver distritos donde el porcentaje de fallecidos es mayor a pesar de tener menos víctimas de accidente.



- **coordenada_x_utm** y **coordenada_y_utm**: dada que esta variable se trata de las coordenadas de los accidentes, hemos optado por utilizar un mapa de calor que represente una distribución espacial según el tipo de lesividad (ilesos, herido y fallecido).



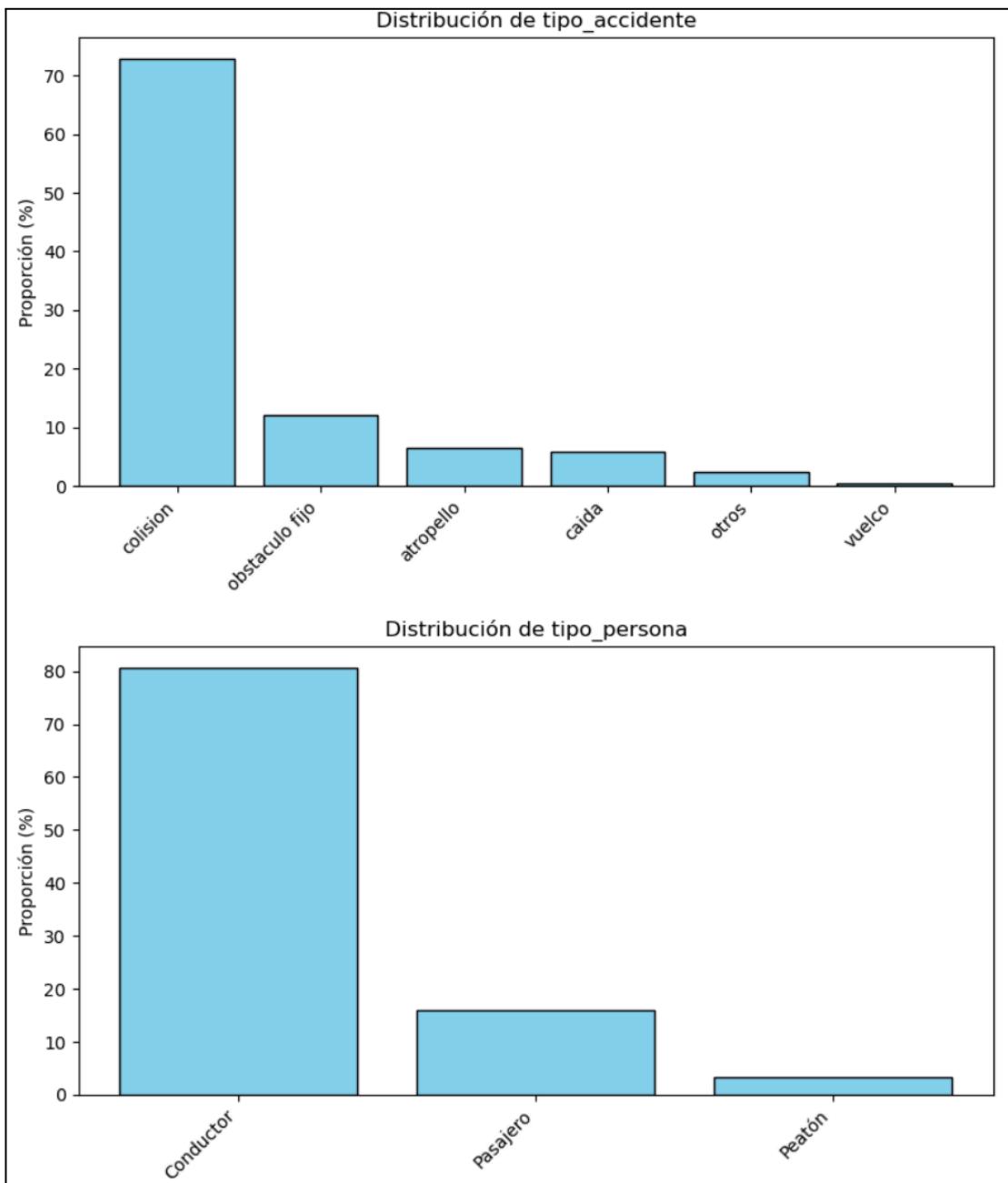


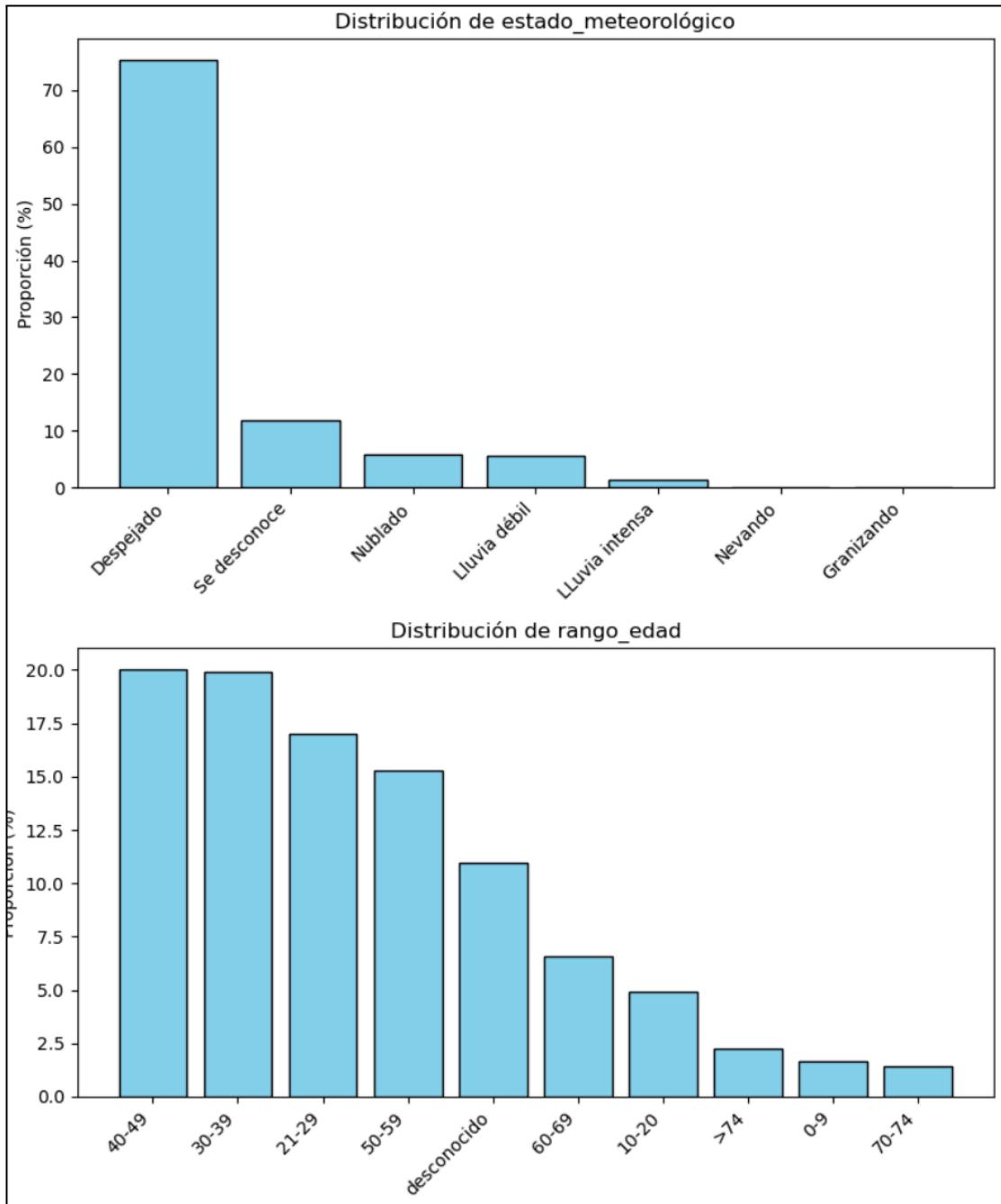
- Conclusión: los mapas de calor utilizados para el análisis de las características 'coordenada_x_utm' y 'coordenada_y_utm' nos muestran que hay puntos más propensos o con más víctimas mortales en accidentes, por lo que a prioria son características a tener en cuenta en nuestro modelo.

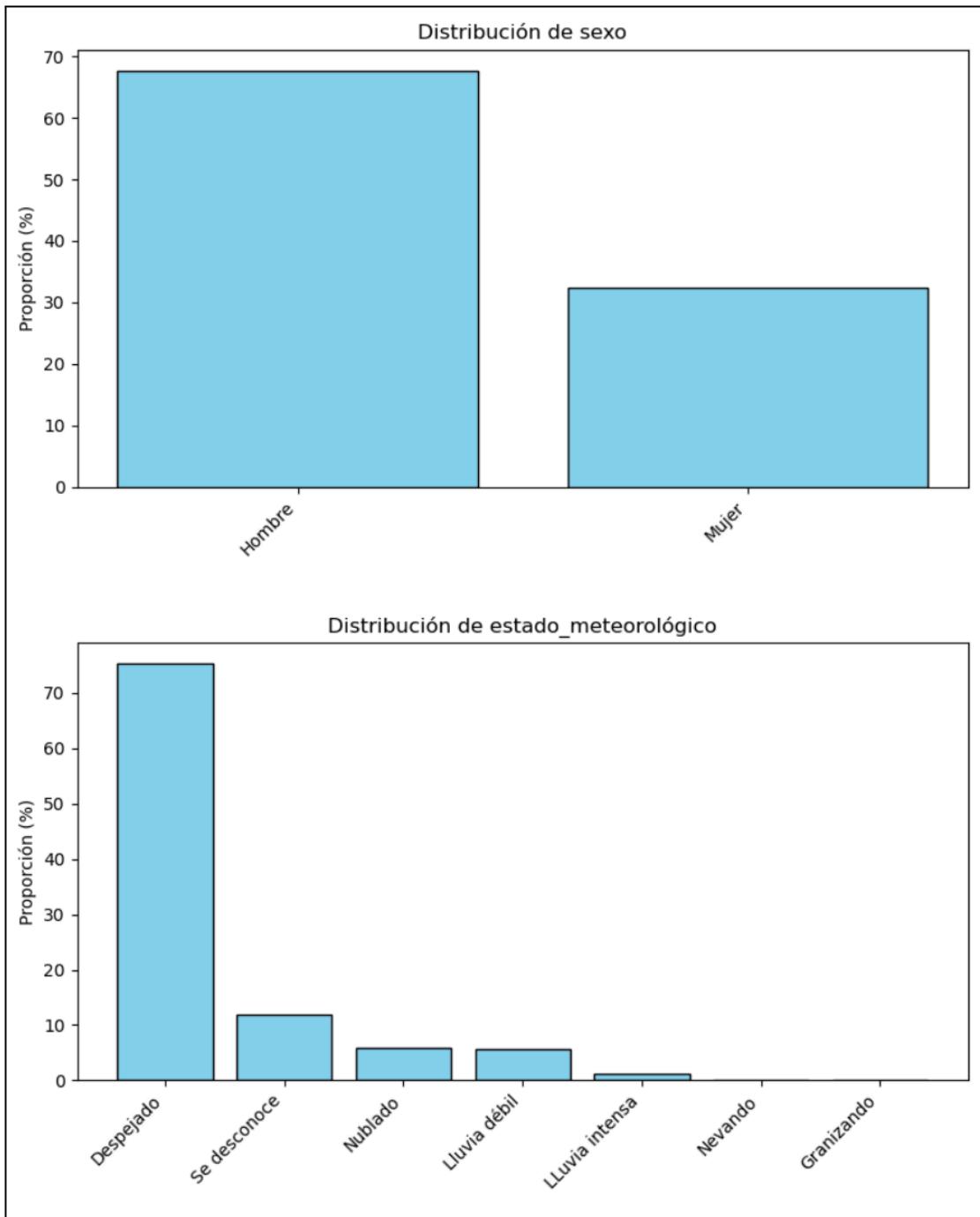


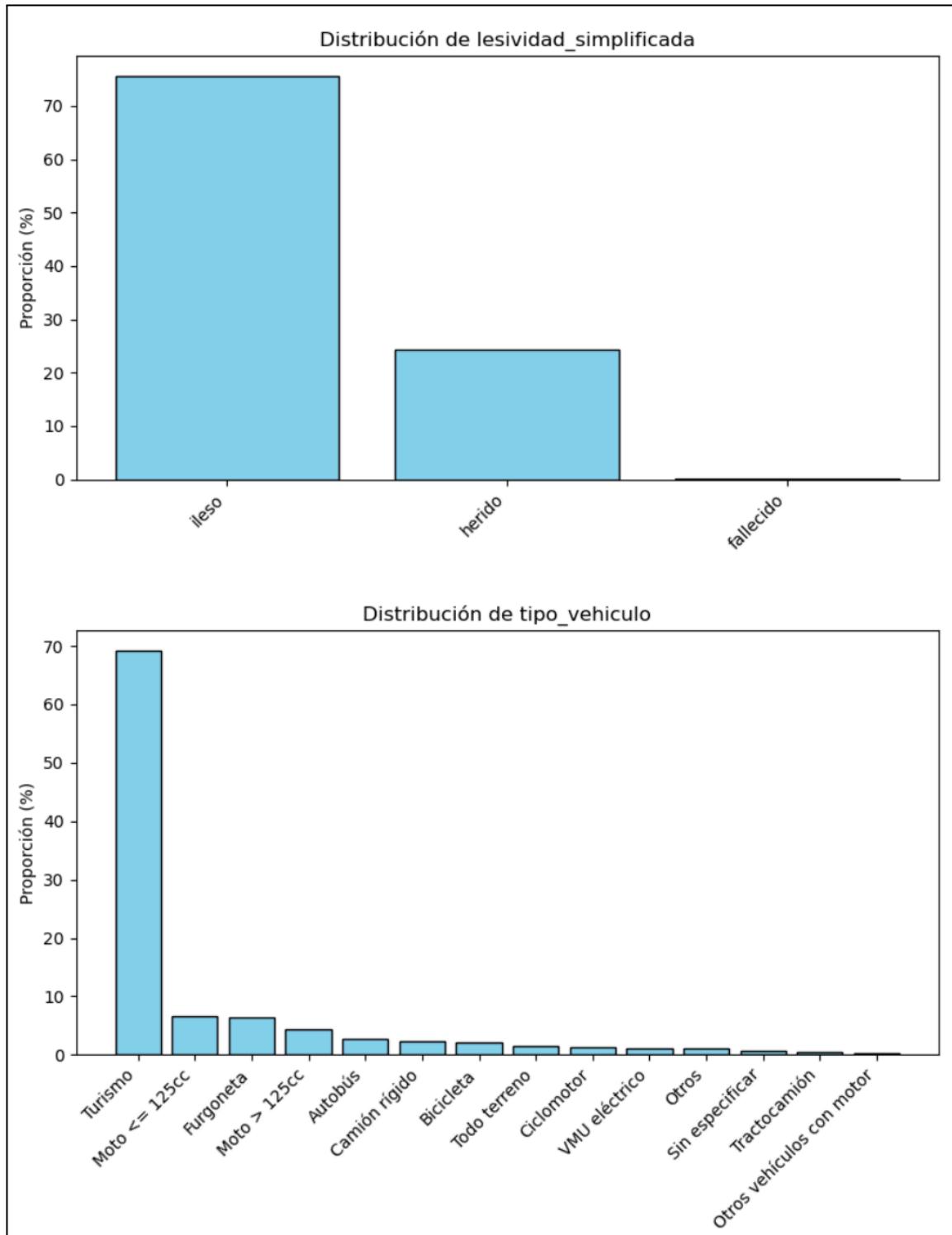
4.2 Balance de datos.

El balance de los datos es un aspecto muy importante de cara al funcionamiento de modelos de aprendizaje o Machine Learning. De manera visual y utilizando la librería Matplotlib vamos analizar las principales características categóricas para analizar su representación en nuestro dataset.









- Conclusión: partiendo de que tenemos un total de 271250 registros, las variables 'sexo' y 'rango_edad' están relativamente balanceadas. En cambio, el resto de variables tenemos una distribución muy desbalanceada, lo que puede generar problemas en nuestro modelo.



5. TRATAMIENTO DE VARIABLES CATEGÓRICAS Y NUMÉRICAS.

5.1 Mapeado de características.

Durante el proceso de preparación del conjunto de datos, ha sido necesario aplicar diversas técnicas de preprocesamiento para asegurar que los algoritmos de aprendizaje automático puedan trabajar de forma adecuada.

Se ha realizado un mapeo de las características categóricas, el cual ha consistido en asignar un valor numérico a cada categoría. Este tipo de transformación no modifica la escala de los datos, sino que simplemente permite a los modelos interpretar valores originalmente no numéricos. Las características modificadas han sido las siguientes:

- sexo
- tipo_accidente
- estado_meteorológico
- tipo_vehiculo
- tipo_persona
- rango_edad
- lesividad_simplificada
- hora

En las siguientes imágenes se muestra el proceso del mapeo de las características:

```
# convertimos sexo a 0 para hombres y 1 para mujeres
df_edited['sexo'] = df_edited['sexo'].map({'Hombre': 0, 'Mujer': 1}).astype(int)

mapeo_tipo_accidente = {
    'colisión': 0,
    'obstáculo fijo': 1,
    'atropello': 2,
    'caída': 3,
    'otros': 4,
    'vuelco': 5
}
df_edited['tipo_accidente'] = df_edited['tipo_accidente'].map(mapeo_tipo_accidente).astype(int)

mapeo_estado_meteo = {
    'Despejado': 0,
    'Se desconoce': 1,
    'Lluvia débil': 2,
    'Nublado': 3,
    'LLuvia intensa': 4,
    'Granizando': 5,
    'Nevando': 6
}
df_edited['estado_meteorológico'] = df_edited['estado_meteorológico'].map(mapeo_estado_meteo).astype(int)
```



```

mapeo_tipo_vehiculo = {
    'Turismo': 0,
    'Moto <= 125cc': 1,
    'Furgoneta': 2,
    'Moto > 125cc': 3,
    'Autobús': 4,
    'Camión rígido': 5,
    'Bicicleta': 6,
    'Todo terreno': 7,
    'Ciclomotor': 8,
    'VMU eléctrico': 9,
    'Otros': 10,
    'Sin especificar': 11,
    'Tractocamión': 12,
    'Otros vehículos con motor': 13
}
df_edited['tipo_vehiculo'] = df_edited['tipo_vehiculo'].map(mapeo_tipo_vehiculo).astype(int)

mapeo_tipo_persona = {
    'Conductor': 0,
    'Pasajero': 1,
    'Peatón': 2,
}
df_edited['tipo_persona'] = df_edited['tipo_persona'].map(mapeo_tipo_persona).astype(int)

mapeo_edad = {
    '0-9': 0,
    '10-20': 1,
    '21-29': 2,
    '30-39': 3,
    '40-49': 4,
    '50-59': 5,
    '60-69': 6,
    '70-74': 7,
    '>74': 8,
    'desconocido': 9
}
df_edited['rango_edad'] = df_edited['rango_edad'].map(mapeo_edad).astype(int)

mapeo_lesividad_simplificada = {
    'ilesos': 0,
    'herido': 1,
    'fallecido': 2
}
df_edited['lesividad_simplificada'] = df_edited['lesividad_simplificada'].map(mapeo_lesividad_simplificada).astype(int)

```

Para la característica 'hora' vamos a crear una nueva característica llamada 'franja_horaria', de manera que dividamos las horas entre 'Madrugada' (de 00:00 a 06:00), 'Mañana' (de 06:00 a 12:00), 'Tarde' (de 12:00 a 18:00) y 'Noche' (de 18:00 a 00:00).



```

def clasificar_hora(hora):
    if 0 <= hora < 6:
        return 'Madrugada'
    elif 6 <= hora < 12:
        return 'Mañana'
    elif 12 <= hora < 18:
        return 'Tarde'
    else:
        return 'Noche'

df_edited['franja_horaria'] = df_edited['hora'].dt.total_seconds() // 3600
df_edited['franja_horaria'] = df_edited['franja_horaria'].apply(clasificar_hora)

mapeo_franja_horaria = {
    'Madrugada': 0,
    'Mañana': 1,
    'Tarde': 2,
    'Noche': 3
}
df_edited['franja_horaria'] = df_edited['franja_horaria'].map(mapeo_franja_horaria).astype(int)

```

En cuanto a las características numéricas también se han modificado. El 'cod_distrito' se ha pasado de float a entero, y en el 'cod_lesividad' se han reordenado las categorías.

```

# convertimos cod_distrito de float a entero
df_edited['cod_distrito'] = df_edited['cod_distrito'].astype(int)

# convertimos cod_lesividad de float a entero
df_edited['cod_lesividad'] = df_edited['cod_lesividad'].astype(int)
# mapeamos los valores 14 y 77
mapeo_lesividad = {
    0: 0,
    1: 1,
    2: 2,
    3: 3,
    4: 4,
    5: 5,
    6: 6,
    7: 7,
    14: 8,
    77: 9
}
df_edited['cod_lesividad'] = df_edited['cod_lesividad'].map(mapeo_lesividad).astype(int)

```

En cuanto a las características 'coordenada_x_utm' y 'coordenada_y_utm', estas son valores tipo float que van aproximadamente desde 429000 a 454000 para 'x' y entre 4460000 y 4489000 para 'y'. Se ha optado por reducir la dimensionalidad de estas variables.

Para ello, se ha aplicado una discretización en regiones, agrupando las coordenadas en celdas de una cuadrícula (1000x1000 metros), convirtiendo las coordenadas continuas (valores decimales) en coordenadas discretas (valores enteros). Así reduciremos la cantidad de valores únicos y las variables serán más útiles para nuestro modelo.



```
# Definimos el tamaño de las celdas
cell_size_x = 1000 # Agrupamos en bloques de 1000 metros
cell_size_y = 1000

# Creamos nuevas columnas (grid_x y grid_y) con la celda a la que pertenece cada punto
df_edited['grid_x'] = (df_edited['coordenada_x_utm'] // cell_size_x).astype(int)
df_edited['grid_y'] = (df_edited['coordenada_y_utm'] // cell_size_y).astype(int)
```

5.2 Eliminación de características.

Tras la creación de nuevas características y la modificación de otras, las siguientes características ya no las necesitamos para nuestro modelo:

- fallecido: nos ha servido para el análisis pero no puede estar en nuestro modelo ya que es respuesta directa a nuestro objetivo 'lesividad'.
- fecha
- hora
- distrito
- cod_lesividad
- coordenada_x_utm
- coordenada_y_utm

En cuanto a la 'lesividad_simplificada', ésta pasará a ser 'lesividad'.

```
# Actualización de eliminación de las columnas no deseadas
df_edited = df_edited.drop(columns=['fallecido','fecha','hora','distrito','lesividad','cod_lesividad','coordenada_x_utm','coordenada_y_utm'])

# modificación del nombre la variable objetivo a 'lesividad'
df_edited = df_edited.rename(columns={'lesividad_simplificada': 'lesividad'})
```

5.3 Normalización.

La normalización se refiere al proceso de escalado de variables numéricas para que todas tengan un rango comparable. Esto es especialmente útil para algoritmos sensibles a la escala, como redes neuronales o regresión logística.

Sin embargo, como se verá en el siguiente apartado, en este proyecto se han utilizado modelos basados en árboles de decisión, como Decision Tree, Random Forest y Gradient Boosting. Estos algoritmos no son sensibles a la escala de los datos, por lo que no requieren normalización previa para funcionar correctamente. Por tanto, en este caso concreto, no se ha aplicado normalización a las variables numéricas, centrándonos en el correcto tratamiento de las variables categóricas.



5.4 Estructura final de nuestro dataframe.

La estructura final de nuestro dataframe preparado para el modelado y predicción es la siguiente:

df_edited # estado actual de nuestro dataframe tras las modificaciones realizadas												
	cod_distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	tipo_persona	rango_edad	sexo	dia_semana	lesividad	franja_horaria	grid_x	grid_y
0	1	0	0	3	0	4	0	0	1	1	440	447€
1	1	0	0	0	0	3	1	0	1	1	440	447€
2	11	0	1	2	0	4	0	1	0	0	439	447€
3	11	0	1	0	0	4	1	1	0	0	439	447€
4	11	0	1	0	0	4	1	1	0	0	439	447€
...
271245	20	4	0	5	0	2	0	0	0	1	448	447€
271246	9	4	0	4	0	4	0	1	0	1	438	447€
271247	9	4	0	0	0	5	0	1	0	1	438	447€
271248	3	0	0	0	0	3	0	6	0	3	441	447€
271249	3	0	0	0	0	4	0	6	0	3	441	447€

271250 rows × 12 columns

```
df_edited.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271250 entries, 0 to 271249
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   cod_distrito     271250 non-null   int64  
 1   tipo_accidente   271250 non-null   int64  
 2   estado_meteorológico  271250 non-null   int64  
 3   tipo_vehiculo    271250 non-null   int64  
 4   tipo_persona     271250 non-null   int64  
 5   rango_edad       271250 non-null   int64  
 6   sexo             271250 non-null   int64  
 7   dia_semana      271250 non-null   int32  
 8   lesividad        271250 non-null   int64  
 9   franja_horaria   271250 non-null   int64  
 10  grid_x           271250 non-null   int64  
 11  grid_y           271250 non-null   int64  
dtypes: int32(1), int64(11)
memory usage: 23.8 MB
```

```
df_edited.columns
```

```
Index(['cod_distrito', 'tipo_accidente', 'estado_meteorológico',
       'tipo_vehiculo', 'tipo_persona', 'rango_edad', 'sexo', 'dia_semana',
       'lesividad', 'franja_horaria', 'grid_x', 'grid_y'],
      dtype='object')
```

5.5 Matriz de correlación.

Una vez realizado el preprocessamiento de los datos, incluyendo el mapeo de variables categóricas y la preparación del conjunto para el entrenamiento, se ha generado una matriz de correlación. Esta matriz nos permite analizar la relación entre las diferentes variables numéricas



del dataset, y resulta especialmente útil para identificar patrones o relaciones fuertes entre variables.

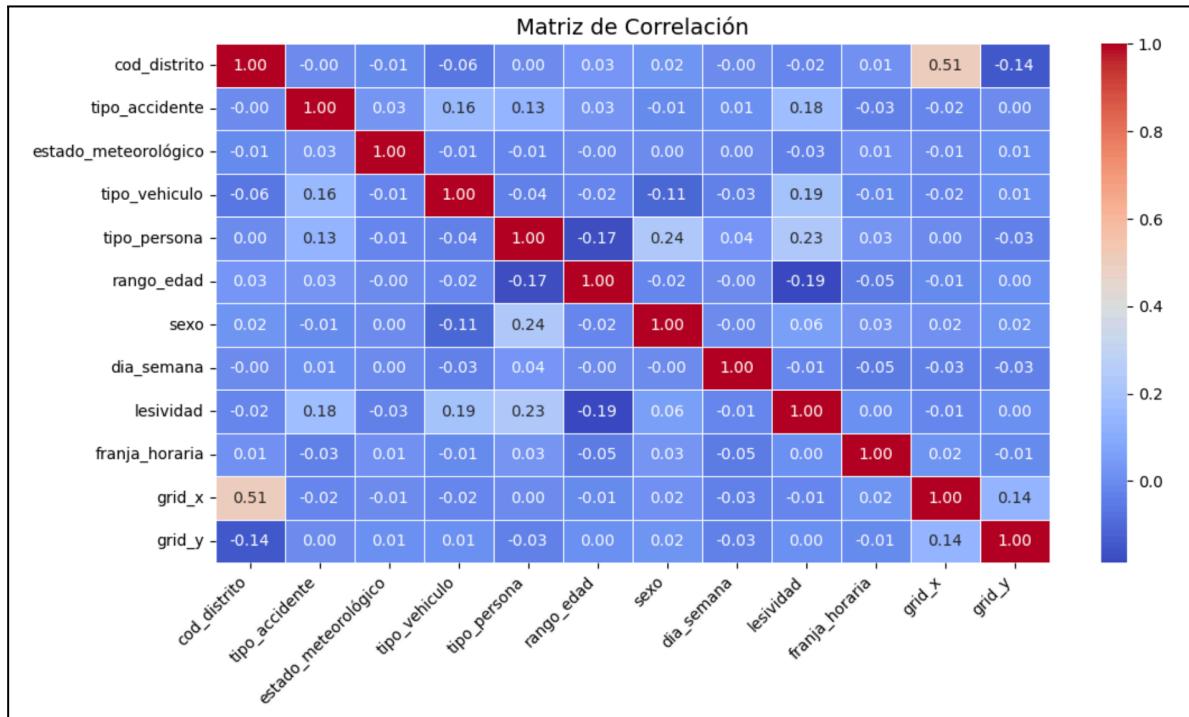
Se trata de una herramienta estadística que muestra el grado de asociación lineal entre pares de variables. Cada celda de la matriz contiene un valor que varía entre -1 y 1:

- Un valor cercano a 1 indica una correlación positiva fuerte: a medida que una variable aumenta, la otra también tiende a aumentar.
- Un valor cercano a -1 indica una correlación negativa fuerte: a medida que una variable aumenta, la otra tiende a disminuir.
- Un valor cercano a 0 sugiere que no existe una relación lineal significativa entre las variables.

En este proyecto, la matriz de correlación se ha calculado a partir del dataframe ya transformado, en el que todas las variables categóricas han sido codificadas como valores numéricos. Se ha aplicado la función 'corr()' de pandas, para calcular la correlación de Pearson entre variables, generando una matriz interpretable visualmente mediante un mapa de calor ayudándonos de la librería Seaborn, como se muestra en las siguientes imágenes.

```
fig, ax = plt.subplots(figsize=(12, 6))

sns.heatmap(df_edited.corr(numeric_only=True),
            annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5,
            ax=ax, cbar=True)
plt.xticks(rotation=45, ha='right') # Rotamos etiquetas del eje X para mejor visualización
plt.title("Matriz de Correlación", fontsize=14)
plt.show()
```



Si analizamos la matriz de correlación para nuestra variable objetivo 'lesividad', a pesar de no existir características que muestre grandes correlaciones, sí podemos apreciar una correlación débil con las siguientes variables:

- tipo_persona
- tipo_vehiculo
- tipo_accidente
- rango_edad
- sexo (muy leve).

Tener una correlación baja como es nuestro caso, no significa que la variable no sea útil. Además, con modelos tipo árbol (son los que usaremos como veremos en el siguiente apartado de esta memoria), no es tan crítico eliminar variables con baja correlación, ya que estos modelos pueden manejar bien relaciones no lineales o interacciones entre variables.



6. MODELO DE ENTRENAMIENTO.

6.1 Elección del modelo.

Uno de los principales retos detectados en el análisis del dataset ha sido el fuerte desbalance en la variable objetivo 'lesividad', donde la mayoría de los registros correspondían a personas ilesas, seguidas por heridos, y con un número muy reducido de fallecidos. Este tipo de distribución dificulta el rendimiento de muchos algoritmos de clasificación tradicionales, ya que tienden a sesgarse hacia la clase mayoritaria, minimizando la capacidad predictiva sobre las clases minoritarias.

Teniendo en cuenta este escenario, se ha optado por la utilización de modelos basados en árboles de decisión, concretamente:

- Random Forest Classifier
- Gradient Boosting Classifier (XGBoost)
- Decision Tree Classifier

Estos modelos presentan varias ventajas frente a este tipo de problemas:

- Robustez ante datos desbalanceados: aunque no eliminan completamente el sesgo hacia la clase mayoritaria, los modelos de árbol pueden manejar mejor los desequilibrios.
- Capacidad para capturar relaciones no lineales entre las variables predictoras y la variable objetivo.
- Flexibilidad para trabajar con variables categóricas.
- Interpretabilidad: los modelos de árbol permiten analizar la importancia de las variables y la lógica de las decisiones tomadas, lo cual es útil para la interpretación de resultados en un contexto aplicado.

A continuación, vamos a mostrar la división de nuestro dataframe entre la variable objetivo 'lesividad' y el resto del dataframe, y a su vez, la división de estos entre la parte de 'entrenamiento' y la parte de 'test'.



```
df_t = df_edited.copy() # trabajaremos con una copia del dataframe
df_t.columns

Index(['cod_distrito', 'tipo_accidente', 'estado_meteorológico',
       'tipo_vehiculo', 'tipo_persona', 'rango_edad', 'sexo', 'dia_semana',
       'lesividad', 'franja_horaria', 'grid_x', 'grid_y'],
      dtype='object')

X = df_t.drop(['lesividad'], axis=1) # dataframe de entrenamiento
y = df_t.lesividad # este será el target de nuestro modelo, es decir, los resultados

X # comprobamos nuestro dataframe de entrenamiento
```

	cod_distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	tipo_persona	rango_edad	sexo	dia_semana	franja_horaria	grid_x	grid_y	
0	1	0		0	3	0	4	0	0	1	440	4475
1	1	0		0	0	0	3	1	0	1	440	4475
2	11	0		1	2	0	4	0	1	0	439	4470
3	11	0		1	0	0	4	1	1	0	439	4470
4	11	0		1	0	0	4	1	1	0	439	4470
...
271245	20	4		0	5	0	2	0	0	1	448	4476
271246	9	4		0	4	0	4	0	1	1	438	4474
271247	9	4		0	0	0	5	0	1	1	438	4474
271248	3	0		0	0	0	3	0	6	3	441	4474
271249	3	0		0	0	0	4	0	6	3	441	4474

271250 rows × 11 columns

```
y # comprobamos nuestra variable objetivo
```

0	1
1	1
2	0
3	0
4	0
..	
271245	0
271246	0
271247	0
271248	0
271249	0

Name: lesividad, Length: 271250, dtype: int64

6.2 Librerías utilizadas para el modelado y predicción.

Para el desarrollo de los modelos de clasificación y la evaluación de su rendimiento, se han utilizado distintas librerías de Python especializadas en aprendizaje automático y análisis de métricas. Estas librerías se detallan a continuación:

- `sklearn.model_selection.train_test_split`: permite dividir el conjunto de datos en subconjuntos de entrenamiento y prueba, asegurando que el modelo se entrene con una parte de los datos y se evalúe con otra independiente.



- `sklearn.ensemble.RandomForestClassifier`: implementa el algoritmo de Random Forest, un modelo de ensamblado basado en múltiples árboles de decisión para mejorar la precisión y reducir el sobreajuste.
- `sklearn.tree.DecisionTreeClassifier`: proporciona un árbol de decisión simple, útil para interpretar de manera visual y comprensible las decisiones del modelo.
- `xgboost.XGBClassifier`: corresponde al modelo Extreme Gradient Boosting (XGBoost), un algoritmo de boosting optimizado para rendimiento y precisión, especialmente efectivo en problemas complejos y datos desbalanceados.
- `sklearn.metrics`: métricas para evaluar el resultado de nuestros modelos.
 - `accuracy_score`: mide la proporción de aciertos totales del modelo.
 - `precision_score`: evalúa la precisión por clase, es decir, la proporción de verdaderos positivos sobre el total de predicciones positivas.
 - `recall_score`: mide la capacidad del modelo para encontrar todos los ejemplos positivos reales.
 - `f1_score`: combina precisión y recall en una única métrica equilibrada.
 - `confusion_matrix`: representa las predicciones del modelo en una matriz para comparar con los valores reales.
 - `classification_report`: resume las métricas de clasificación (precisión, recall y F1-score) para cada clase.

Estas herramientas nos han permitido entrenar modelos de aprendizaje, realizar predicciones y validar los resultados obtenidos.



7. ANÁLISIS DE RESULTADOS Y CONCLUSIONES.

7.1 Resultados de nuestros modelos.

- Random Forest:

```
# Random Forest

random_forest = RandomForestClassifier(n_estimators=100, random_state=42) # bosque de 100 árboles
random_forest.fit(X_train, y_train) # entrenamos el modelo
y_pred = random_forest.predict(X_test) # hacemos predicciones

# Calcula la matriz de confusión
confusion_mat = confusion_matrix(y_test, y_pred)
print(" > Matriz de confusión:")
print(confusion_mat)

# Obtener el informe de clasificación
report = classification_report(y_test, y_pred, zero_division=0)
print(" > Reporte de clasificación:")
print(report)

# Calcula la precisión
accuracy = accuracy_score(y_test, y_pred)
print(" > Accuracy:", accuracy)

# Calcula la precisión por clase (solo para clasificación multiclas)
precision = precision_score(y_test, y_pred, average=None, zero_division=0)
print(" > Precisión por clase:", precision)

# Calcula la precision promedio
precision = precision_score(y_test, y_pred, average='macro', zero_division=0)
print(" > Precisión promedio:", precision)

# Calcula el recall por clase (solo para clasificación multiclas)
recall = recall_score(y_test, y_pred, average=None)
print(" > Recall por clase:", recall)
# Calcula el recall promedio
recall = recall_score(y_test, y_pred, average='macro')
print(" > Recall promedio:", recall)

# Calcula la puntuación F1 por clase (solo para clasificación multiclas)
f1 = f1_score(y_test, y_pred, average=None)
print(" > F1 Score por clase:", f1)

# Calcula la puntuación F1 promedio
f1 = f1_score(y_test, y_pred, average='macro')
print(" > F1 Score promedio:", f1)

#####
# Guardamos una de las métricas (F1 Score) como métrica global para la comparativa final
metrica_random_forest = round(f1 * 100, 2)
print(" >> Métrica final Random Forest:", metrica_random_forest)
#####
```



```
> Matriz de confusión:  
[[37731 3319 0]  
 [ 5482 7683 0]  
 [ 5 30 0]]  
> Reporte de clasificación:  
 precision recall f1-score support  
  
 0 0.87 0.92 0.90 41050  
 1 0.70 0.58 0.64 13165  
 2 0.00 0.00 0.00 35  
  
accuracy 0.84 54250  
macro avg 0.52 0.50 0.51 54250  
weighted avg 0.83 0.84 0.83 54250  
  
> Accuracy: 0.8371244239631337  
> Precisión por clase: [0.87303901 0.69642857 0.]  
> Precisión promedio: 0.5231558609838493  
> Recall por clase: [0.91914738 0.58359286 0.]  
> Recall promedio: 0.500913413699355  
> F1 Score por clase: [0.89550007 0.6350374 0.]  
> F1 Score promedio: 0.5101791575107162  
>>> Metraca final Random Forest: 51.02
```



- Gradient Boosting.

```
# Gradient Boosting (XGBoost)

gradient_boosting = XGBClassifier(objective="multi:softmax", num_class=3)
gradient_boosting.fit(X_train, y_train)
y_pred = gradient_boosting.predict(X_test)

# Calcula la matriz de confusión
confusion_mat = confusion_matrix(y_test, y_pred)
print(" > Matriz de confusión:")
print(confusion_mat)

# Obtener el informe de clasificación
report = classification_report(y_test, y_pred)
print(" > Reporte de clasificación:")
print(report)

# Calcula la precisión
accuracy = accuracy_score(y_test, y_pred)
print(" > Accuracy:", accuracy)

# Calcula la precisión por clase (solo para clasificación multiclas)
precision = precision_score(y_test, y_pred, average=None)
print(" > Precisión por clase:", precision)
# Calcula la precision promedio
precision = precision_score(y_test, y_pred, average='macro')
print(" > Precisión promedio:", precision)

# Calcula el recall por clase (solo para clasificación multiclas)
recall = recall_score(y_test, y_pred, average=None)
print(" > Recall por clase:", recall)
# Calcula el recall promedio
recall = recall_score(y_test, y_pred, average='macro')
print(" > Recall promedio:", recall)

# Calcula la puntuación F1 por clase (solo para clasificación multiclas)
f1 = f1_score(y_test, y_pred, average=None)
print(" > F1 Score por clase:", f1)
# Calcula la puntuación F1 promedio
f1 = f1_score(y_test, y_pred, average='macro')
print(" > F1 Score promedio:", f1)

#####
# Guardamos una de las métricas (F1 Score) como métrica global para la comparativa final
metrica_gradient_boost = round(f1 * 100, 2)
print(" >> Métrica final Gradient Boost:", metrica_gradient_boost)
#####
```



```
> Matriz de confusión:  
[[39021  2029      0]  
 [ 5382  7783      0]  
 [     4    30      1]]  
> Reporte de clasificación:  
          precision    recall   f1-score   support  
  
          0         0.88      0.95      0.91     41050  
          1         0.79      0.59      0.68     13165  
          2         1.00      0.03      0.06       35  
  
accuracy                      0.86     54250  
macro avg                     0.89      0.52      0.55     54250  
weighted avg                   0.86      0.86      0.86     54250  
  
> Accuracy: 0.8627649769585254  
> Precisión por clase: [0.87871282 0.79079455 1. ]  
> Precisión promedio: 0.8898357898331825  
> Recall por clase: [0.95057247 0.59118876 0.02857143]  
> Recall promedio: 0.523444219745489  
> F1 Score por clase: [0.91323122 0.67657669 0.05555556]  
> F1 Score promedio: 0.5484544884347721  
>>> Metrica final Gradient Boost: 54.85
```



- Decision Tree:

```
# Decision Tree

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
y_pred = decision_tree.predict(X_test)

# Calcula la matriz de confusión
confusion_mat = confusion_matrix(y_test, y_pred)
print(" > Matriz de confusión:")
print(confusion_mat)

# Obtener el informe de clasificación
report = classification_report(y_test, y_pred)
print(" > Reporte de clasificación:")
print(report)

# Calcula la precisión
accuracy = accuracy_score(y_test, y_pred)
print(" > Accuracy:", accuracy)

# Calcula la precisión por clase (solo para clasificación multiclas)
precision = precision_score(y_test, y_pred, average=None)
print(" > Precisión por clase:", precision)
# Calcula la precision promedio
precision = precision_score(y_test, y_pred, average='macro')
print(" > Precisión promedio:", precision)

# Calcula el recall por clase (solo para clasificación multiclas)
recall = recall_score(y_test, y_pred, average=None)
print(" > Recall por clase:", recall)
# Calcula el recall promedio
recall = recall_score(y_test, y_pred, average='macro')
print(" > Recall promedio:", recall)

# Calcula la puntuación F1 por clase (solo para clasificación multiclas)
f1 = f1_score(y_test, y_pred, average=None)
print(" > F1 Score por clase:", f1)
# Calcula la puntuacion F1 promedio
f1 = f1_score(y_test, y_pred, average='macro')
print(" > F1 Score promedio:", f1)

#####
# Guardamos una de las metricas (F1 Score) como metrica global para la comparativa final
metrica_decision_tree = round(f1 * 100, 2)
print(" >> Metrica final Decision Tree:", metrica_decision_tree)
#####
```



```
> Matriz de confusión:  
[[35959  5083     8]  
 [ 5931  7213    21]  
 [    8   26     1]]  
> Reporte de clasificación:  
      precision    recall    f1-score   support  
  
       0          0.86      0.88      0.87     41050  
       1          0.59      0.55      0.57     13165  
       2          0.03      0.03      0.03      35  
  
accuracy                      0.80     54250  
macro avg                     0.49      0.48      0.49     54250  
weighted avg                  0.79      0.80      0.79     54250  
  
> Accuracy: 0.795815668202765  
> Precisión por clase: [0.85825099 0.58537575 0.03333333]  
> Precisión promedio: 0.49232002484129883  
> Recall por clase: [0.87598051 0.54789214 0.02857143]  
> Recall promedio: 0.48414802612934354  
> F1 Score por clase: [0.86702512 0.56601405 0.03076923]  
> F1 Score promedio: 0.48793613377333206  
>>> Metrica final Decision Tree: 48.79
```



7.2 Resultados de nuestros modelos previo balanceado de la clase objetivo.

Con el objetivo de mejorar el resultado de nuestros modelos, vamos aplicar técnicas de sobremuestreo con SMOTENC, incluido en la librería 'imblearn', diseñada para abordar el problema de clases desbalanceadas en datasets de clasificación.

Está específicamente adaptada para datasets mixtos, es decir, que contienen tanto variables categóricas como numéricas. Esta técnica genera nuevas instancias sintéticas para la clase minoritaria respetando la naturaleza categórica de ciertas variables, evitando así combinaciones inválidas de valores.

Por ello, se utiliza principalmente para mejorar el rendimiento de los modelos de machine learning cuando la clase objetivo está desbalanceada, permitiendo un entrenamiento más equilibrado y justo para todas las clases.

```
from imblearn.over_sampling import SMOTENC # librería para realizar balanceo de clases categóricas
cat_features = [0, 1, 2, 3, 4, 5, 6, 7, 8] # Índices de columnas categóricas
# Definir SMOTENC indicando las columnas categóricas
smote_nc = SMOTENC(categorical_features=cat_features, random_state=42)
# Aplicar SMOTENC
X_train_resampled, y_train_resampled = smote_nc.fit_resample(X_train, y_train)

# comprobamos el balanceo de la clase objetivo 'lesividad'
from collections import Counter
print("Antes del balanceo y_train:", Counter(y_train))
print("Después del balanceo y_train_resampled:", Counter(y_train_resampled))

Antes del balanceo y_train: Counter({0: 164200, 1: 52662, 2: 138})
Después del balanceo y_train_resampled: Counter({0: 164200, 1: 164200, 2: 164200})
```



- Random Forest tras balanceado de la clase objetivo.

```
> Matriz de confusión:  
[[34975  6003   72]  
 [ 4441  8630   94]  
 [    5    29    1]]  
> Reporte de clasificación:  
      precision    recall   f1-score   support  
  
       0         0.89     0.85     0.87     41050  
       1         0.59     0.66     0.62     13165  
       2         0.01     0.03     0.01      35  
  
accuracy                      0.80     54250  
macro avg                     0.49     0.51     0.50     54250  
weighted avg                   0.81     0.80     0.81     54250  
  
> Accuracy: 0.8037972350230415  
> Precisión por clase: [0.88721747 0.58859637 0.00598802]  
> Precisión promedio: 0.4939339561484644  
> Recall por clase: [0.85200974 0.65552602 0.02857143]  
> Recall promedio: 0.5120357295790625  
> F1 Score por clase: [0.86925725 0.6202609 0.00990099]  
> F1 Score promedio: 0.49980637858032767  
>>> Metraca final Random Forest tras balanceo: 49.98
```

- Gradient Boosting tras balanceado de la clase objetivo.

```
> Matriz de confusión:  
[[34379  6462  209]  
 [ 3546  9293  326]  
 [    4    27    4]]  
> Reporte de clasificación:  
      precision    recall   f1-score   support  
  
       0         0.91     0.84     0.87     41050  
       1         0.59     0.71     0.64     13165  
       2         0.01     0.11     0.01      35  
  
accuracy                      0.81     54250  
macro avg                     0.50     0.55     0.51     54250  
weighted avg                   0.83     0.81     0.81     54250  
  
> Accuracy: 0.8050875576036867  
> Precisión por clase: [0.90640407 0.58883538 0.00742115]  
> Precisión promedio: 0.5008868677076467  
> Recall por clase: [0.83749086 0.70588682 0.11428571]  
> Recall promedio: 0.552554466733779  
> F1 Score por clase: [0.87058585 0.64206999 0.01393728]  
> F1 Score promedio: 0.5088643747156602  
>>> Metraca final Gradient Boost tras balanceo: 50.89
```



- Decision Tree tras balanceado de la clase objetivo.

```
> Matriz de confusión:  
[[33598 7350 102]  
 [ 5090 7933 142]  
 [ 7 25 3]]  
> Reporte de clasificación:  
 precision recall f1-score support  
  
 0 0.87 0.82 0.84 41050  
 1 0.52 0.60 0.56 13165  
 2 0.01 0.09 0.02 35  
  
 accuracy 0.77 54250  
 macro avg 0.47 0.50 0.47 54250  
 weighted avg 0.78 0.77 0.77 54250  
  
> Accuracy: 0.7656036866359447  
> Precisión por clase: [0.86827756 0.51822576 0.01214575]  
> Precisión promedio: 0.4662163561779315  
> Recall por clase: [0.81846529 0.60258261 0.08571429]  
> Recall promedio: 0.5022540591145569  
> F1 Score por clase: [0.8426359 0.55722966 0.0212766 ]  
> F1 Score promedio: 0.47371405128253213  
>>> Metraca final Decision Tree tras balanceo: 47.37
```

7.3 Resultados de nuestros modelos previa simplificación de la clase objetivo.

Dado que los resultados obtenidos no han sido satisfactorios, a modo de prueba, vamos a simplificar la clase objetivo 'lesividad' a tan solo dos opciones, 0 (ilesos) o 1 (herido-fallecido), y comprobaremos sus resultados.



```

df_2 = df_t.copy() # trabajamos con una copia

df_2['lesividad'].value_counts()

lesividad
0    205250
1     65827
2      173
Name: count, dtype: int64

# mapeamos a tan sólo dos opciones, 0 (ilesos) o 1 (herido-fallecido)
mapeo_lesividad = {
    0: 0,
    1: 1,
    2: 1
}
df_2['lesividad'] = df_2['lesividad'].map(mapeo_lesividad)
df_2['lesividad'].value_counts()

lesividad
0    205250
1    66000
Name: count, dtype: int64

```

- Random Forest simplificando la variable objetivo.

```

> Matriz de confusión:
[[37669 3381]
 [ 5511 7689]]
> Reporte de clasificación:
              precision    recall   f1-score  support
          0       0.87      0.92      0.89     41050
          1       0.69      0.58      0.63     13200

          accuracy           0.84     54250
          macro avg       0.78      0.75      0.76     54250
          weighted avg     0.83      0.84      0.83     54250

> Accuracy: 0.8360921658986175
> Precisión por clase: [0.87237147 0.69457995]
> Precisión promedio: 0.7834757070359032
> Recall por clase: [0.91763703 0.5825      ]
> Recall promedio: 0.7500685140073082
> F1 Score por clase: [0.89443191 0.63362176]
> F1 Score promedio: 0.7640268339368029
>>> Metrica final Random Forest simplificado: 76.4

```



- Gradient Boost simplificando la variable objetivo.

```
> Matriz de confusión:  
[[39015 2035]  
 [ 5403 7797]]  
> Reporte de clasificación:  
 precision recall f1-score support  
  
 0 0.88 0.95 0.91 41050  
 1 0.79 0.59 0.68 13200  
  
accuracy 0.86 54250  
macro avg 0.84 0.77 0.80 54250  
weighted avg 0.86 0.86 0.86 54250  
  
> Accuracy: 0.8628940092165899  
> Precisión por clase: [0.87836012 0.79302278]  
> Precisión promedio: 0.8356914535120732  
> Recall por clase: [0.95042631 0.59068182]  
> Recall promedio: 0.7705540637803123  
> F1 Score por clase: [0.91297328 0.67705801]  
> F1 Score promedio: 0.7950156414000593  
>>> Metrica final Gradient Boost simplificado: 79.5
```

- Decision Tree simplificando la variable objetivo.

```
> Matriz de confusión:  
[[35930 5120]  
 [ 5994 7206]]  
> Reporte de clasificación:  
 precision recall f1-score support  
  
 0 0.86 0.88 0.87 41050  
 1 0.58 0.55 0.56 13200  
  
accuracy 0.80 54250  
macro avg 0.72 0.71 0.72 54250  
weighted avg 0.79 0.80 0.79 54250  
  
> Accuracy: 0.7951336405529954  
> Precisión por clase: [0.857027 0.58461788]  
> Precisión promedio: 0.7208224410712489  
> Recall por clase: [0.87527406 0.54590909]  
> Recall promedio: 0.7105915734691618  
> F1 Score por clase: [0.86605443 0.5646008 ]  
> F1 Score promedio: 0.7153276129365114  
>>> Metrica final Decision Tree simplificado: 71.53
```



7.4 Conclusiones.

Tras aplicar diferentes algoritmos de clasificación al conjunto de datos sobre víctimas de accidentes, se han comparado los resultados obtenidos con los tres modelos basados en árboles: Random Forest, Gradient Boosting y Decision Tree. Todos los modelos fueron evaluados utilizando métricas estándar como accuracy, precision, recall, f1-score y la matriz de confusión.

A modo resumen, como primera referencia de los resultados obtenidos y tomando el F1 Score como valor indicativo de referencia tenemos los siguientes resultados:

```
>>> Metrica final Random Forest: 51.02
>>> Metrica final Gradient Boost: 54.85
>>> Metrica final Decision Tree: 48.79
-----
>>> Metrica final Random Forest tras balanceo: 49.98
>>> Metrica final Gradient Boost tras balanceo: 50.89
>>> Metrica final Decision Tree tras balanceo: 47.37
-----
>>> Metrica final Random Forest con 2 clases: 76.4
>>> Metrica final Gradient Boost con 2 clases: 79.5
>>> Metrica final Decision Tree con 2 clases: 71.53
```

Como primera conclusión, vemos que el resultado obtenido para los 3 modelos respecto a la variable objetivo con tres categorías (ilesos, herido y fallecido) es bastante pobre. El modelo que obtiene un mejor resultado es el Gradient Boosting con 54,85%.

Si entramos a valorar más los datos al detalle, Random Forest mostró un rendimiento general aceptable con una precisión del 84%, aunque sabemos que el dato de precisión (accuracy) no es un dato objetivo. Analizando por clases, obtuvo buenos resultados clasificando la clase mayoritaria (ilesos), con un f1-score de 0.896, pero tuvo dificultades al predecir la clase minoritaria (fallecido), con un f1-score de 0.00. Esto indica que el modelo no fue capaz de predecir correctamente ni un solo caso de esa clase, lo cual es un reflejo directo del fuerte desbalance de clases en el dataset.

Gradient Boosting fue el modelo con mayor rendimiento general, obtiene un mejor resultado en f1-score que Random Forest (0,68 respecto a 0,63). Aunque también el mismo problema con la clase minoritaria (fallecido), aunque en este caso logró al menos detectar un caso correctamente, con un f1-score de 0.06 en esta clase.



Decision Tree, siendo un modelo más simple, obtuvo el peor rendimiento de los tres con el menor f1-score global. Aunque presentó una razonable capacidad para clasificar las clases más frecuentes, al igual que los modelos anteriores obtuvo un bajo rendimiento en la clase minoritaria con un f1-score del 0,03.

En general, los resultados reflejan que el desbalance de clases sigue siendo un gran problema, especialmente a la hora de predecir correctamente las clases con muy pocos ejemplos, como los fallecidos.

En segundo lugar, utilizando técnicas de sobremuestreo no hemos conseguido mejorar nuestro resultado, incluso empeora ligeramente.

Por último, donde sí se ve una mejora considerable es reduciendo la clase objetivo a dos opciones (ilesos o herido-fallecido), dado simplificamos nuestro problema a 0 o 1, y eliminamos el problema de la categoría fallecido, la cual representaba menos del 1% de los datos, lo que dificultaba enormemente la detección de estos casos.

Con el fin de mejorar la capacidad predictiva de los modelos y mitigar los efectos del fuerte desbalance en la variable objetivo y reducir la complejidad del problema, se optó por simplificar la clasificación de la lesividad en dos categorías: "ilesos" (0) y "herido" (1), unificando los casos de heridos leves, graves y fallecidos.

Random Forest mejoró su rendimiento, con un f1-score de 0,764, teniendo un f1-score de 0.89 para la clase mayoritaria (ilesos) y presentando algo más de dificultad para la clase 'herido-fallecido', con un 0.63 para. Aunque sigue mostrando dificultades, el modelo mejora notablemente respecto a la clasificación multiclas.

Gradient Boosting continuó siendo el modelo más eficaz, con un f1-score de 0.68 para la clase 'herido-fallecido'. Esto lo convierte en el modelo más equilibrado. Como media obtiene un 0,795 de media de f1-score.

Decision Tree, volvió a mostrar peores resultados, con un f1-score de 0.56 en la clase 'herido'. Aun así, su comportamiento mejoró respecto al escenario multiclas.

En conjunto, la simplificación de la variable objetivo ha permitido mejorar los resultados, especialmente si el objetivo del proyecto es predecir de forma fiable si un individuo resulta herido



o no. Esta reducción de clases ha permitido a los modelos ser más precisos. En particular, Gradient Boosting se confirma como la mejor opción, seguido por Random Forest, quedando Decision Tree como una alternativa más sencilla pero menos potente.

Para finalizar, indicar que también se han hecho pruebas eliminando algunas características que a priori no presentan una clara correlación con la variable objetivo, como pueden ser 'estado_meteorológico' o 'grid_x' y 'grid_y' sin obtener mejoras en los resultados.