

Realizado por: Sixto Rodríguez Reimúndez.

CONTENIDO

1 - Introducción.	3
1.1 – Tecnologías aplicadas.	6
2 - Preparación del Dataframe.	7
2.1 - Uniendo los csv.	7
2.2 – Eliminación de columnas.	9
2.3 - Limpieza de datos (Null/NaN).	10
2.4 – Conversión del tipo de los campos de nuestro dataframe.	14
2.5 – Creación de nuevas columnas.	19
3 – Visualización gráfica de los datos. Tableau.	21
3.1 – Análisis de datos.	23
3.1.1 – Accidentes con positivo en drogas.	23
3.1.2 - Accidentes con positivo en alcohol.	24
3.1.3 - Estado meteorológico en el que ocurrieron los accidentes.	25
3.1.4 - Accidentes por franja horaria, rango de edad, día de la semana y tipo de persona.	26
3.1.5 - Accidentes por mes, tipo de vehículo y tipo de accidente.	26
3.1.6 - Densidad de los accidentes.	28
3.1.7 - Accidentes por distrito.	29
3.1.8 - Tipo de gravedad de accidentes y muertos.	29
3.1.9 – Mapa con accidentes con muertos en Madrid.	30
3.1.10 - Zonas con más accidentados en Madrid.	31
3.2 – Conclusiones del análisis gráfico.	31
4 – MODELO DE PREDICCIÓN.	32
4.1 – Conclusiones sobre el modelo de predicción.	35
5 - BIBLIOGRAFIA.	39
6 – ENLACES GITHUB.	39

1 - Introducción.

Lo que se busca en este proyecto es sacar conclusiones para solucionar posibles problemas a las que nos podamos enfrentar a través de los datos. También buscaremos predecir el “futuro” a través de esos datos.

Los programas que utilizaremos para realizar este trabajo son Python y Tableau. Con Python estamos utilizando la última versión 3.12, y además de instalarlo, también instalaremos el Jupyter Lab para poder trabajar desde un cuaderno donde quede registrado todo nuestro trabajo.

Concretamente, en este proyecto vamos a analizar los datos de accidentalidad en la ciudad de Madrid entre 2019 y 2023, y ver si podemos predecir que índice de accidentalidad tendremos en el año 2024. Para ello utilizaremos los datos abiertos que podemos recoger en la web del ayuntamiento de Madrid:

- [Datos abiertos de accidentalidad de Madrid](#)

Cabría comentar que tener el dato del volumen de tráfico podría haber dado una mejor dimensión estadística de los accidentes ocurridos, y nos permitiría tener un mejor entendimiento de lo que podría estar causando los accidentes en la ciudad de Madrid.

Lo primero que haremos es entender los datos que podemos ver en los archivos csv que nos bajaremos desde la web y para eso veremos la estructura del conjunto del fichero que nos facilita también la web. Podremos observar los siguientes datos:

N.º de columna	Nombre de columna	Descripción	Valores Esperados
1	num_expediente	AAAASNNNNNN, donde: - AAAA es el año del accidente. - S cuando se trata de un expediente con accidente.	texto
2	fecha	Fecha en formato dd/mm/aaaa	fecha
3	hora	La hora se establece en rangos horarios de 1 hora	hora
4	localizacion	Calle 1 - Calle 2 (cruce) o una calle	Texto
5	numero	Número de la calle, cuando tiene sentido	Número

N.º de columna	Nombre de columna	Descripción	Valores Esperados
6	cod_distrito	Código del distrito	Número
7	distrito	Nombre del Distrito	Texto
8	tipo_accidente*	Tipo de accidente: tipificado mas abajo	Texto
9	estado_meteorologico	Descripción climatología	Texto
10	tipo_vehiculo	Tipo vehículo implicado	Texto
11	tipo_persona	Tipo persona implicada	Texto
12	rango_edad	Tramo edad persona afectada	Texto
13	sexo	Puede ser: hombre, mujer o no asignado	Texto
14	cod_lesividad*	Viene tipificado mas abajo	Número
15	lesividad	Descripción lesividad	Texto
16	coordenada_x_utm	Ubicación coordenada x	Número
17	coordenada_y_utm	Ubicación coordenada y	Número
18	positiva_alcohol	Puede ser: N o S	N o S
19	positiva_droga	Puede ser: NULL o 1	NULL o 1

También nos dan una breve descripción sobre como interpretar como han sido los accidentes de tráfico:

- **Colisión doble:** haría referencia a un accidente de tráfico ocurrido entre 2 vehículos en movimiento, ya sea una colisión frontal, fronto lateral o lateral.
- **Colisión múltiple:** haría referencia a un accidente de tráfico ocurrido entre más de dos vehículos en movimiento.

- **Alcance:** haría referencia a un 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:** haría referencia a un 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:** ocurrido ente 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:** haría referencia a un 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:** aquí se agrupan todas las caídas relacionadas con el desarrollo y las circunstancias del tráfico, (motocicleta, ciclomotor, bicicleta, viajero bus, etc.,)
- **Otras causas:** donde tendríamos los accidentes por atropello a un animal, despeñamiento, salida de la vía, y otros.

Y ya por último nos describe los códigos de lesividad, donde entre los códigos 01 al 07 son todos LEVES excepto el 03 que es GRAVE y el 04 que es FALLECIDO, mientras que el 14 y el 77 la lesividad es inexistente:

- **01**, atención en urgencias sin posterior ingreso.
- **02**, ingreso inferior o igual a 24 horas.
- **03**, ingreso superior a 24 horas.
- **04**, fallecido 24 horas.
- **05**, asistencia sanitaria ambulatoria con posterioridad.
- **06**, asistencia sanitaria inmediata en centro de salud o mutua.
- **07**, asistencia sanitaria sólo en el lugar del accidente.
- **14**, sin asistencia sanitaria.
- **77**, se desconoce.

Lo primero, es que deberemos unificar en un único archivo csv todos los años de datos que vamos recibiendo del ayuntamiento de Madrid, para posteriormente tratar las variables y entradas.

Después de observar la tipología de la tabla ofrecida por los archivos csv, borraremos la variable del número de expediente ya que no influye de ninguna manera en el análisis estadístico que pretendemos realizar o a nivel informativo. Hay ciertas variables que son solo descriptivas como el distrito que completa la información del código de distrito, y ocurre lo mismo con el código de lesividad y lesividad, que nos da una pequeña descripción que hemos detallado antes; por tanto, las dejaremos en nuestro dataframe.

También añadiremos columnas y cambiaremos algunas de ellas dadas el formato que nos ofrecen. Observamos que en la columna de hora nos da unas entradas o registros del tipo: 30/12/1899 1:15:00, y convertiremos en 01:15 y a partir de ella

crearemos una nueva variable de franjas horarias para poder analizar en que franja se producen más accidentes.

De la variable fecha extraeremos dos nuevas variables: día de la semana y mes, que creo que nos pueden dar datos relevantes de cuando se producen los accidentes. Podemos también observar en la forma de expresar las coordenadas UTM (Universal Transverse Mercator) que están mal expresadas en alguno de los archivos csv: 444.578.15 o 445.094,90 y deberían ser todas unificadas como 444578.15 o 445094.90. Y posteriormente las transformaremos en nuevas columnas de latitud y longitud para que en Tableau podamos tener una representación gráfica de los accidentes en un mapa.

Una vez hecho todo esto con Python empezaremos a sacar conclusiones que nos ofrecen los datos recogidos por la policía municipal de Madrid, y trataremos de representar gráficamente nuestras conclusiones. A partir de las conclusiones, sacaremos un modelo predictivo que nos permita predecir datos estadísticos de accidentes de una zona, mes, día, etc.

1.1 – Tecnologías aplicadas.

En este proyecto usaremos varias tecnologías para poder hacer un estudio de los accidentes en la ciudad de Madrid:

- **Python 3.12**, para poder tratar los datos proporcionados por el ayuntamiento de Madrid. Utilizaremos las siguientes librerías:
 - **Panda**, para poder tratar dataframes y su posterior tratamiento.
 - **Glob**, para poder unir todos los archivos csv que tenemos en uno solo, utilizando la ruta que tenemos como patrón.
 - **Calendar**, para poder tratar con fechas.
 - **Pyproj**, en concreto **transform**, para poder transformar coordenadas geográficas.
 - **Sklearn (scikit-learn)**, para poder hacer un modelo predictivo.
- **Jupyter lab**, para poder tener un cuaderno donde trabajar con Python sin la pérdida de todos los comandos y trabajo realizado.
- **Voila** para poder representar en html local el cuaderno creado en jupyter lab si fuera necesario.
- **Tableau Desktop versión 24.1.1**, para tener una representación gráfica de los datos.

2 - Preparación del Dataframe.

2.1 - Uniendo los csv.

Como hemos comentado antes nos bajamos los archivos de accidentes en Madrid por año en formato csv, así que tendremos que unir en un único archivo para luego tratar los datos. Lo primero es importar las librerías que usaremos para tratar y crear nuestro dataframe: pandas, que nos permitirá crear el dataframe y tratarlo, y glob, que nos permitirá seguir un patrón de búsqueda para unir los archivos.

El proceso que seguiremos para unir los archivos es crear un dataframe (df_list) donde iremos insertando nuestros csv. También crearemos una ruta patrón donde está nuestros csv y utilizamos *Accidentalidad.csv para que nos una todos ellos (el asterisco actúa como comodín). Posteriormente, creamos un bucle donde recorremos todos los archivos que cumplen con nuestro patrón y lo vamos añadiendo a nuestro dataframe. Cabe recalcar que normalmente por defecto los campos de los csv están separados por comas, pero que los nuestros están separados por punto y coma, así que cada vez que leemos un archivo (file) especificamos su codificación (UTF-8) y como están delimitados sus campos.

```
# Importamos las librerías

import pandas as pd
import glob

# Creamos el dataframe donde guardaremos todos los csv
df_list = []

# Creamos una variable con la ruta y patrón de los archivos usando asterico para recoger lo que varía en los diferentes archivos
pattern = 'C:/CSV/*Accidentalidad.csv'

# Bucle para recoger todos los archivos y los vamos añadiendo a nuestro dataframe
for file in glob.glob(pattern):
    try:
        df = pd.read_csv(file, encoding='utf-8', delimiter = ';')
        df_list.append(df)
    except Exception as e:
        print("Error al leer el archivo:", file)
        print("Mensaje de error:", str(e))

# Tenemos que concatenar los csv almacenados en uno solo, ignorando los índices que pudieran existir
df_Madrid = pd.concat(df_list, ignore_index = True)

# Comprobamos las primeras 5 columnas
df_Madrid.head()
```

Siendo las cinco primeras filas de nuestro nuevo dataframe los siguientes:

num_expediente	fecha	hora	localizacion	numero	cod_distrito	distrito	tipo_accidente	estado_meteorologico	tipo_vehiculo	...	rango_edad	sexo	cod_lesividad	lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga	Unamed: 19	Unamed: 20	
0	20180117842	04/02/2019	9:10:00	CALL ALBERTO AGUILERA 1	1	1.0	CENTRO	Colisión lateral	Despejado	Motociclستا > 125cc	...	De 45 a 49 años	Hombre	7.0	Asistencia sanitaria sólo en el lugar del acci...	440098.049	4479679.17	N	NaN	NaN	NaN
1	20180117842	04/02/2019	9:10:00	CALL ALBERTO AGUILERA 1	1	1.0	CENTRO	Colisión lateral	Despejado	Turismo	...	De 30 a 34 años	Mujer	7.0	Asistencia sanitaria sólo en el lugar del acci...	440098.049	4479679.17	N	NaN	NaN	NaN
2	20190000001	01/01/2019	3:45:00	PASEO, SANTA MARIA DE LA CABEZA / PLAZA ELIPTICA	168	11.0	CARABANCHEL	Alcanse	NaN	Furgoneta	...	De 40 a 44 años	Hombre	NaN	NaN	439139.603	4470836.854	S	NaN	NaN	NaN
3	20190000001	01/01/2019	3:45:00	PASEO, SANTA MARIA DE LA CABEZA / PLAZA ELIPTICA	168	11.0	CARABANCHEL	Alcanse	NaN	Turismo	...	De 40 a 44 años	Mujer	NaN	NaN	439139.603	4470836.854	N	NaN	NaN	NaN
4	20190000001	01/01/2019	3:45:00	PASEO, SANTA MARIA DE LA CABEZA / PLAZA ELIPTICA	168	11.0	CARABANCHEL	Alcanse	NaN	Turismo	...	De 45 a 49 años	Mujer	NaN	NaN	439139.603	4470836.854	N	NaN	NaN	NaN

Y las últimas cinco:

```
# Comprobamos las últimas 5 columnas
df_Madrid.tail()
```

	num_expediente	fecha	hora	localizacion	numero	cod_distrito	distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	rango_edad	sexo	cod_lesividad	lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga	Unnamed: 19	Unnamed: 20
221905	20235040267	31/12/2023	21:15:00	AVDA. GRAN VIA DE VILLAVEDE / AVDA. ANDALUCCIA	10	17.0	VILLAVEDE	Colisión fronto-lateral	Despejado	Turismo	De 45 a 49 años	Mujer	7.0	Asistencia sanitaria sólo en el lugar del acci...	441152.627	4466350.125	N	NaN	NaN	NaN
221906	20235040267	31/12/2023	21:15:00	AVDA. GRAN VIA DE VILLAVEDE / AVDA. ANDALUCCIA	10	17.0	VILLAVEDE	Colisión fronto-lateral	Despejado	Turismo	De 6 a 9 años	Hombre	7.0	Asistencia sanitaria sólo en el lugar del acci...	441152.627	4466350.125	N	NaN	NaN	NaN
221907	20235040277	29/12/2023	9:35:00	PTA. TOLEDO, 0	0	1.0	CENTRO	Alcance	Despejado	Motocicleta hasta 125cc	De 45 a 49 años	Hombre	NaN	NaN	439594.878	4473163.747	N	NaN	NaN	NaN
221908	20235040277	29/12/2023	9:35:00	PTA. TOLEDO, 0	0	1.0	CENTRO	Alcance	Despejado	Turismo	De 21 a 24 años	Hombre	NaN	NaN	439594.878	4473163.747	N	NaN	NaN	NaN
221909	20235040290	29/12/2023	5:10:00	CALL. HERNANDEZ DE TEJADA / CALL. AGASTIA	2	15.0	CIUDAD LINEAL	Solo salida de la vía	Despejado	Turismo	De 21 a 24 años	Hombre	NaN	NaN	444544.675	4477557.087	S	NaN	NaN	NaN

5 rows x 21 columns

Y con el método info() observaremos como es nuestro dataframe:

```
# También podemos inspeccionar un poco la tabla con el método .info()
df_Madrid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 221910 entries, 0 to 221909
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   num_expediente         221910 non-null object
 1   fecha                  221910 non-null object
 2   hora                   221910 non-null object
 3   localizacion           221910 non-null object
 4   numero                 221902 non-null object
 5   cod_distrito           221902 non-null float64
 6   distrito               221902 non-null object
 7   tipo_accidente         221906 non-null object
 8   estado_meteorológico  198163 non-null object
 9   tipo_vehiculo          220966 non-null object
10   tipo_persona           221907 non-null object
11   rango_edad             221910 non-null object
12   sexo                   221910 non-null object
13   cod_lesividad          121342 non-null float64
14   lesividad              121342 non-null object
15   coordenada_x_utm       221902 non-null object
16   coordenada_y_utm       221902 non-null object
17   positiva_alcohol       221128 non-null object
18   positiva_droga         688 non-null    float64
19   Unnamed: 19            0 non-null      float64
20   Unnamed: 20            0 non-null      float64
dtypes: float64(5), object(16)
memory usage: 35.6+ MB
```


2.2 – Eliminación de columnas.

Como hemos podido observar tenemos bastantes columnas, y alguna de ellas no las utilizaremos ya que no aportan nada a las posibles conclusiones que podamos sacar o son redundantes. Las columnas que borraremos son las siguientes:

- **Num_expediente**, a efectos prácticos es un dato con el que no vamos a trabajar y no es relevante.
- **Unnamed 19 y 20**, son columnas que se han creado automáticamente y vacías, quizás por que al final de cada fila hubiera puntos y comas extras.
- **Cod_distrito**, es un dato redundante ya que también tenemos los nombres de los distritos, y como creo que son más representativos los nombres: 17 vs Villaverde, nos quedaremos con los nombres de distrito.
- **Sexo**, no creo que sea relevante.
- **Número**, el numero de la calle por si solo no es representativo y ya que tendremos las coordenadas de los accidentes, serán irrelevantes.
- **Lesividad**, nos pasa lo mismo que con los códigos de distrito y sus nombres. Es redundante, pero en este caso nos quedamos con los códigos, porque los hemos explicado previamente y no son tantos.

```
# Procedemos a borrar Las columnas
df_Madrid.drop(columns=['num_expediente', 'Unnamed: 19', 'Unnamed: 20', 'cod_distrito', 'sexo', 'numero', 'lesividad'], inplace=True)
```

```
# Comprobamos
df_Madrid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 221910 entries, 0 to 221909
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   fecha                 221910 non-null object
1   hora                 221910 non-null object
2   localizacion         221910 non-null object
3   distrito             221902 non-null object
4   tipo_accidente       221906 non-null object
5   estado_meteorológico 198163 non-null object
6   tipo_vehiculo        220966 non-null object
7   tipo_persona        221907 non-null object
8   rango_edad          221910 non-null object
9   cod_lesividad        121342 non-null float64
10  coordenada_x_utm     221902 non-null object
11  coordenada_y_utm     221902 non-null object
12  positiva_alcohol     221128 non-null object
13  positiva_droga       688 non-null   float64
dtypes: float64(2), object(12)
memory usage: 23.7+ MB
```

2.3 - Limpieza de datos (Null/NaN).

La siguiente parte que nos toca hacer ahora es limpiar nuestro dataframe de entradas Null o NaN, ya que no se podrán tratar. Primero tendremos que hallar en que columnas o campos tenemos estos valores, para lo cual utilizaremos los métodos `isnull()` que nos encontrará los campos con dicho valor y con el método `sum()` los sumaremos para saber cuantos valores Null/NaN tenemos en cada columna.

```
# Observamos también que hay columnas donde hay columnas con nulls o NaN, todas las que no tienen 221.910
# Comprobamos que columnas tienen null o NaN y su cuenta
df_Madrid.isnull().sum()
```

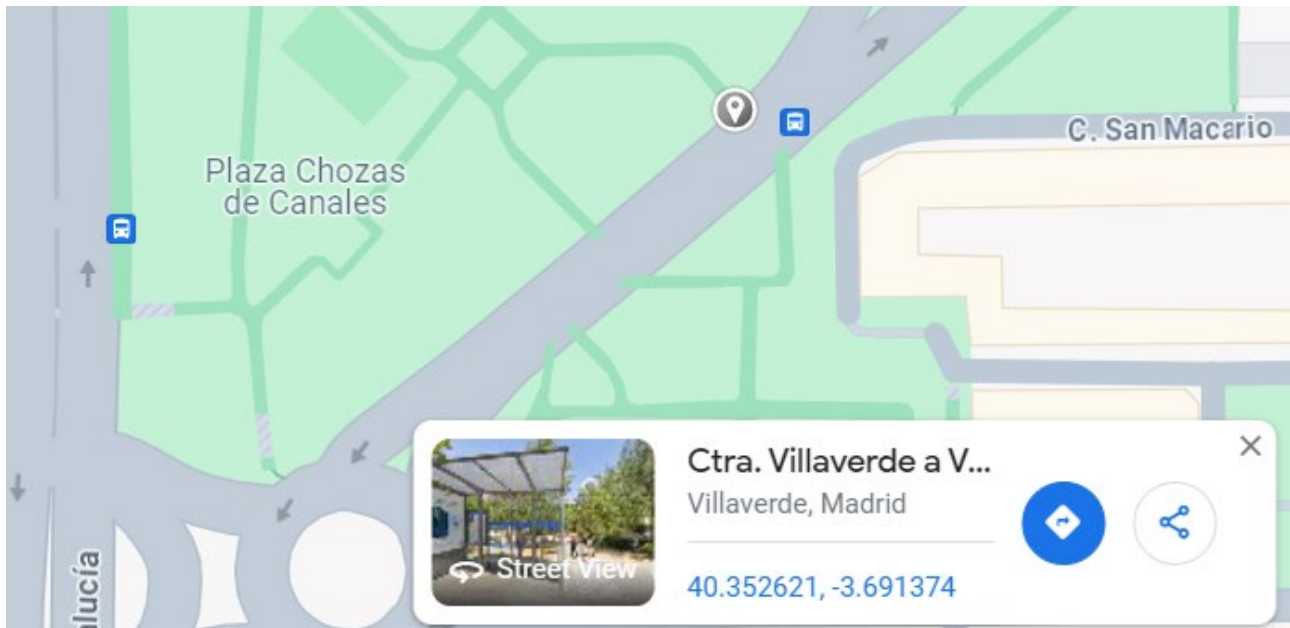
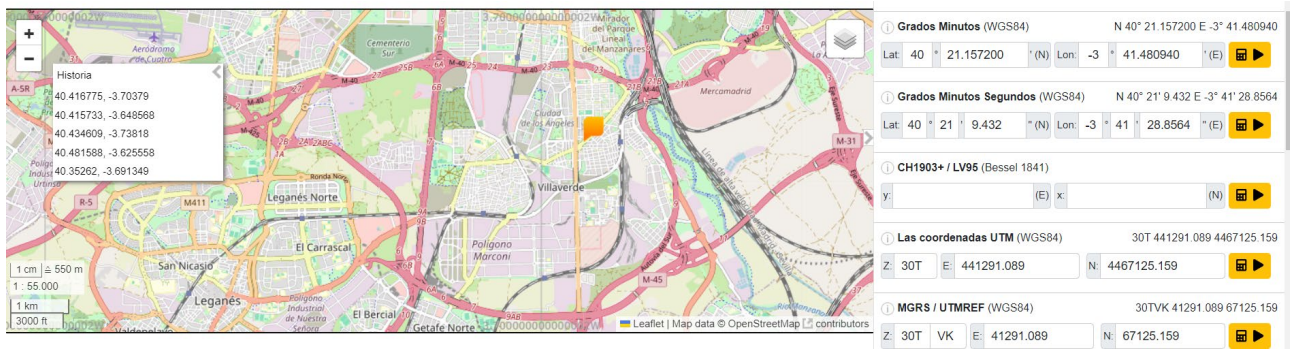
```
fecha          0
hora           0
localizacion   0
distrito       8
tipo_accidente 4
estado_meteorológico 23747
tipo_vehículo  944
tipo_persona   3
rango_edad     0
cod_lesividad 100568
coordenada_x_utm 8
coordenada_y_utm 8
positiva_alcohol 782
positiva_droga 221222
dtype: int64
```

Ahora, procederemos a sustituir esos campos por otros valores con los que si podamos trabajar:

- En los distritos en los que no tenemos valores procederemos a obtenerlos a través de la localización UTM, para lo cual visitaremos la página [Convertidor en línea a todos los sistemas de coordenadas](#) , introduciremos las coordenadas UTM y en el mapa podremos ver el lugar del accidente. Si queremos podemos ir Google Map, ver la localización y tendremos un cuadro de información donde podremos ver el nombre del distrito.

```
df_Madrid[df_Madrid['distrito'].isna()]
```

	fecha	hora	localizacion	distrito	tipo_accidente	estado_meteorológico	tipo_vehículo	tipo_persona	rango_edad	cod_lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga
79446	23/11/2020	7:45:00	AUTOV. M-23, 0 (0.8 ENTRADA)	NaN	Colisión lateral	Despejado	Turismo	Conductor	De 50 a 54 años	7.0	444975,42	4474103,079	N	0.0
79447	23/11/2020	7:45:00	AUTOV. M-23, 0 (0.8 ENTRADA)	NaN	Colisión lateral	Despejado	Turismo	Conductor	De 50 a 54 años	14.0	444975,42	4474103,079	N	0.0
109988	11/09/2021	19:40:00	AUTOV. M-500 / AUTOV. M-30	NaN	Colisión fronto-lateral	Despejado	Turismo	Conductor	De 25 a 29 años	77.0	437390,155	4476258,031	N	0.0
109989	11/09/2021	19:40:00	AUTOV. M-500 / AUTOV. M-30	NaN	Colisión fronto-lateral	Despejado	Turismo	Conductor	De 55 a 59 años	77.0	437390,155	4476258,031	N	0.0
109990	11/09/2021	19:40:00	AUTOV. M-500 / AUTOV. M-30	NaN	Colisión fronto-lateral	Despejado	Turismo	Pasajero	De 55 a 59 años	14.0	437390,155	4476258,031	N	0.0
183190	16/03/2023	22:40:00	GTA. ISIDRO GONZALEZ VELAZQUEZ / CALL. FRANCIS...	NaN	Choque contra obstáculo fijo	Despejado	Turismo	Conductor	De 45 a 49 años	77.0	446979,419	4481398,946	N	0.0
211319	21/10/2023	0:40:00	VILLAVERDE A VALLECAS (FAROLA 11)	NaN	Alcance	Despejado	Turismo	Conductor	De 25 a 29 años	14.0	441291,089	4467125,159	N	0.0
211320	21/10/2023	0:40:00	VILLAVERDE A VALLECAS (FAROLA 11)	NaN	Alcance	Despejado	VMU eléctrico	Conductor	De 18 a 20 años	3.0	441291,089	4467125,159	N	0.0



- Para el estado_meteorológico, sustituiremos por Desconocido.
- En tipo_vehículo, haremos lo mismo.
- En tipo_persona, cambiaremos los NaN por Desconocido.
- En cod_lesividad, cambiaremos los NaN por 77 que equivale a desconocido.
- En tipo_accidente, cambiaremos los NaN por Desconocido.
- En positiva_alcohol, cambiaremos los NaN por Desconocido.
- En positiva_droga, cambiaremos los NaN por N.

```
# Cambiamos los NaN de la columna 'estado_meteorológico' por 'Desconocido'
df_Madrid['estado_meteorológico'].fillna('Desconocido', inplace=True)

# Cambiamos los NaN de la columna 'tipo_vehículo' por 'Desconocido'
df_Madrid['tipo_vehículo'].fillna('Desconocido', inplace=True)

# Cambiamos los NaN de la columna 'tipo_persona' por 'Desconocido'
df_Madrid['tipo_persona'].fillna('Desconocido', inplace=True)

# Cambiamos los NaN de la columna 'cod_lesividad' por 77 (desconocido)
df_Madrid['cod_lesividad'].fillna(77, inplace=True)

# Cambiamos los NaN de la columna 'tipo_accidente' por 'Desconocido'
df_Madrid['tipo_accidente'].fillna('Desconocido', inplace=True)

# Cambiamos los NaN de la columna 'positiva_alcohol' por 'N'
df_Madrid['positiva_alcohol'].fillna('N', inplace=True)

# Cambiamos los NaN de la columna 'positiva_droga' por 0
df_Madrid['positiva_droga'].fillna(0.0, inplace=True)
```

- Para `coordenada_x_utm` y `coordenada_y_utm`, haremos algo parecido a lo realizado para los distritos, pero de manera contraria. A través de la localización obtendremos el punto del accidente y tendremos las coordenadas UTM.

```
# Vamos a través de la localización obtener las coordenadas UTM en la página web https://coordinates-converter.com/es/decimal/40.416775,-3.703790?karte=OpenStreetMap&zoom=8
df_Madrid[df_Madrid['coordenada_x_utm'].isna()]
```

	fecha	hora	localizacion	distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	tipo_persona	rango_edad	cod_lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga
84241	18/11/2020	10:57:00	CALL. LOPE DE HARO, 8	TETUÁN	Colisión fronto-lateral	Despejado	Camión rígido	Conductor	De 40 a 44 años	77.0	NaN	NaN	N	0.0
84242	18/11/2020	10:57:00	CALL. LOPE DE HARO, 8	TETUÁN	Colisión fronto-lateral	Despejado	Furgoneta	Conductor	De 45 a 49 años	77.0	NaN	NaN	N	0.0
219220	14/12/2023	9:45:00	CALL. ATOCHA, 112	CENTRO	Choque contra obstáculo fijo	Despejado	Camión rígido	Conductor	De 50 a 54 años	77.0	NaN	NaN	N	0.0
219791	17/12/2023	2:45:00	PASARELA DE LA PRINCESA	ARGANZUELA	Atropello a persona	Despejado	VMU eléctrico	Conductor	De 18 a 20 años	14.0	NaN	NaN	N	0.0
219792	17/12/2023	2:45:00	PASARELA DE LA PRINCESA	ARGANZUELA	Atropello a persona	Despejado	VMU eléctrico	Peatón	De 50 a 54 años	2.0	NaN	NaN	N	0.0
219807	17/12/2023	7:35:00	AVDA. PALOMERAS / CALL. PUERTO DEL PICO	PUENTE DE VALLECAS	Colisión fronto-lateral	Despejado	Desconocido	Conductor	De 55 a 59 años	7.0	NaN	NaN	N	0.0
219992	18/12/2023	13:40:00	AUTOV. M-30, 07NC40	SALAMANCA	Alcance	Despejado	Motocicleta > 125cc	Conductor	De 35 a 39 años	5.0	NaN	NaN	N	0.0
219993	18/12/2023	13:40:00	AUTOV. M-30, 07NC40	SALAMANCA	Alcance	Despejado	Todo terreno	Conductor	De 35 a 39 años	14.0	NaN	NaN	N	1.0

```
# - Calle Lope de Haro 8 - X-UTM = 440301.600 , Y-UTM = 4478557.900
# - Calle de Atocha 112 - X-UTM = 441156.796 , Y-UTM = 4473444.959
# - Pasarela de La Princesa - X-UTM = 440839.556 , Y-UTM = 4471102.673
# - M-30, 07NC40 (como no sabemos que punto km al que se refiere pondremos unas coord por defecto de La M-30) - X-UTM = 440839.556 , Y-UTM = 4471102.673
# - Avenida Palomeras 150 - X-UTM = 444900.672 , Y-UTM = 4470784.810
# Utilizaremos el método dataframe.at[índice, campo] = entrada (podríamos utilizar también loc en vez de at, tiene el mismo funcionamiento)
```

```
df_Madrid.at[84241, 'coordenada_x_utm'] = 440301.600
df_Madrid.at[84241, 'coordenada_y_utm'] = 4478557.900
df_Madrid.at[84242, 'coordenada_x_utm'] = 440301.600
df_Madrid.at[84242, 'coordenada_y_utm'] = 4478557.900
df_Madrid.at[219220, 'coordenada_x_utm'] = 441156.796
df_Madrid.at[219220, 'coordenada_y_utm'] = 4473444.959
df_Madrid.at[219791, 'coordenada_x_utm'] = 440839.556
df_Madrid.at[219791, 'coordenada_y_utm'] = 4471102.673
df_Madrid.at[219792, 'coordenada_x_utm'] = 440839.556
df_Madrid.at[219792, 'coordenada_y_utm'] = 4471102.673
df_Madrid.at[219807, 'coordenada_x_utm'] = 444900.672
df_Madrid.at[219807, 'coordenada_y_utm'] = 4470784.810
df_Madrid.at[219992, 'coordenada_x_utm'] = 440839.556
df_Madrid.at[219992, 'coordenada_y_utm'] = 4471102.673
df_Madrid.at[219993, 'coordenada_x_utm'] = 440839.556
df_Madrid.at[219993, 'coordenada_y_utm'] = 4471102.673
```

```
# Comprobamos los cambios en alguna línea
df_Madrid.iloc[84241:84243]
```

	fecha	hora	localizacion	distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	tipo_persona	rango_edad	cod_lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga
84241	18/11/2020	10:57:00	CALL. LOPE DE HARO, 8	TETUÁN	Colisión fronto-lateral	Despejado	Camión rígido	Conductor	De 40 a 44 años	77.0	440301.6	4478557.9	N	0.0
84242	18/11/2020	10:57:00	CALL. LOPE DE HARO, 8	TETUÁN	Colisión fronto-lateral	Despejado	Furgoneta	Conductor	De 45 a 49 años	77.0	440301.6	4478557.9	N	0.0

Y comprobamos si ha quedado algún NaN tras nuestros cambios:

```
df_Madrid.isnull().sum()
```

fecha	0
hora	0
localizacion	0
distrito	0
tipo_accidente	0
estado_meteorológico	0
tipo_vehículo	0
tipo_persona	0
rango_edad	0
cod_lesividad	0
coordenada_x_utm	0
coordenada_y_utm	0
positiva_alcohol	0
positiva_droga	0
dtype: int64	

2.4 – Conversión del tipo de los campos de nuestro dataframe.

Ahora tendremos que convertir algunos tipos de las columnas, de cadena a tiempo, de cadena a fecha y datos numéricos que no se van a tratar como dato numérico a cadena. También unificaremos criterios para el `positiva_droga` y `positiva_alcohol`, cambiaremos a este último de 0/1 a N/S. Observamos otra vez que columnas tenemos:

```
df_Madrid.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 221910 entries, 0 to 221909
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   fecha                 221910 non-null object
1   hora                 221910 non-null object
2   localizacion         221910 non-null object
3   distrito             221910 non-null object
4   tipo_accidente       221910 non-null object
5   estado_meteorológico 221910 non-null object
6   tipo_vehiculo        221910 non-null object
7   tipo_persona        221910 non-null object
8   rango_edad           221910 non-null object
9   cod_lesividad        221910 non-null float64
10  coordenada_x_utm     221910 non-null object
11  coordenada_y_utm     221910 non-null object
12  positiva_alcohol     221910 non-null object
13  positiva_droga       221910 non-null float64
dtypes: float64(2), object(12)
memory usage: 23.7+ MB
```

Procedemos hacer nuestro primer cambio, en este caso en la columna de `positivo_droga`:

```
# Conversión de positivo_droga a string sin decimales -> integer -> string
df_Madrid['positiva_droga'] = df_Madrid['positiva_droga'].astype(int)
df_Madrid['positiva_droga'] = df_Madrid['positiva_droga'].astype(str)
df_Madrid.tail(3)

# Cambiamos los valores 0 por N y 1 por S
df_Madrid['positiva_droga'] = df_Madrid['positiva_droga'].replace({'0': 'N', '1': 'S'})
df_Madrid.tail(3)
```

	fecha	hora	localizacion	distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	tipo_persona	rango_edad	cod_lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga
221907	29/12/2023	9:35:00	PTA. TOLEDO, 0	CENTRO	Alcance	Despejado	Motocicleta hasta 125cc	Conductor	De 45 a 49 años	77.0	439594.878	4473163.747	N	N
221908	29/12/2023	9:35:00	PTA. TOLEDO, 0	CENTRO	Alcance	Despejado	Turismo	Conductor	De 21 a 24 años	77.0	439594.878	4473163.747	N	N
221909	29/12/2023	5:10:00	CALL. HERNANDEZ DE TEJADA / CALL. AGASTIA	CIUDAD LINEAL	Solo salida de la vía	Despejado	Turismo	Conductor	De 21 a 24 años	77.0	444544.675	4477557.087	S	N

Las coordenadas UTM están como cadenas de texto, pero necesitamos que sean numéricas para poder trabajar con ellas. En alguno de los ficheros csv las coordenadas en vez de utilizar puntos para separar decimales se han usado comas, así que antes de poder transformar las coordenadas en datos numéricos, tendremos que usar este asunto.

Cuando convirtamos en numéricos añadiremos un campo donde si tenemos un error nos convierta el campo en nulo (NaN).

```
# Reemplazar la coma por un punto solo en los valores que tienen una coma
df_Madrid['coordenada_x_utm'] = df_Madrid['coordenada_x_utm'].apply(lambda x: x.replace(',', '.') if isinstance(x, str) else x)
df_Madrid['coordenada_y_utm'] = df_Madrid['coordenada_y_utm'].apply(lambda x: x.replace(',', '.') if isinstance(x, str) else x)

# Convertimos a float los campos de coordenada_x_utm y coordenada_y_utm
# Nos generará un campo NaN si hay errores
df_Madrid['coordenada_x_utm'] = pd.to_numeric(df_Madrid['coordenada_x_utm'], errors='coerce')
df_Madrid['coordenada_y_utm'] = pd.to_numeric(df_Madrid['coordenada_y_utm'], errors='coerce')
df_Madrid.info()
```

Vemos que hemos tenido algunos errores porque no tenemos 221910 no nulos en los campos de coordenadas UTM:

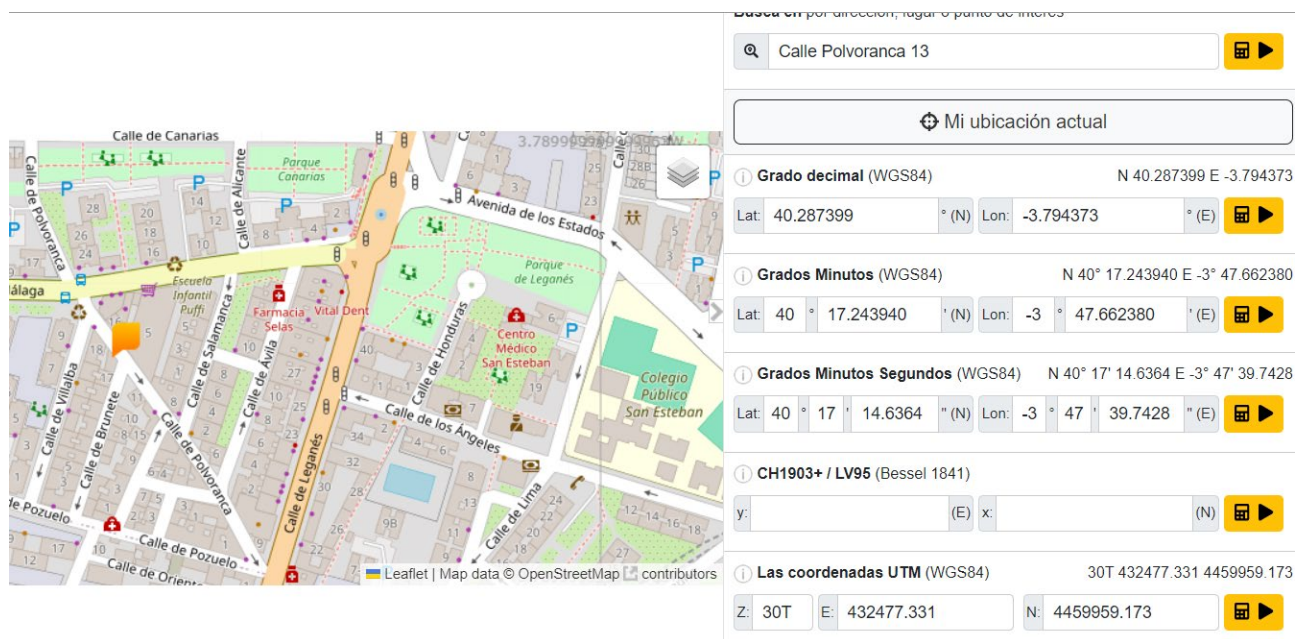
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 221910 entries, 0 to 221909
Data columns (total 14 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   fecha                                221910 non-null  object
1   hora                                221910 non-null  object
2   localizacion                         221910 non-null  object
3   distrito                            221910 non-null  object
4   tipo_accidente                      221910 non-null  object
5   estado_meteorológico               221910 non-null  object
6   tipo_vehiculo                      221910 non-null  object
7   tipo_persona                       221910 non-null  object
8   rango_edad                         221910 non-null  object
9   cod_lesividad                      221910 non-null  float64
10  coordenada_x_utm                   221880 non-null  float64
11  coordenada_y_utm                   221880 non-null  float64
12  positiva_alcohol                   221910 non-null  object
13  positiva_droga                     221910 non-null  object
dtypes: float64(3), object(11)
memory usage: 23.7+ MB
```

Tendremos que localizar los campos de coordenadas UTM con nulos, y cambiarlos por sus coordenadas correspondientes, para lo cual volveremos a ir al [Convertidor en línea a todos los sistemas de coordenadas](#) e introduciremos la calle y obtendremos las coordenadas.

```
df_Madrid[df_Madrid['coordenada_x_utm'].isna()]
```

	fecha	hora	localizacion	distrito	tipo_accidente	estado_meteorológico	tipo_vehiculo	tipo_persona	rango_edad	cod_lesividad	coordenada_x_utm	coordenada_y_utm	positiva_alcohol	positiva_droga
29308	23/07/2019	8:45:00	AUTOV, M-30, +03200E	FUENCARRAL-EL PARDO	Alcance	Despejado	Motocicleta > 125cc	Conductor	De 30 a 34 años	7.0	NaN	NaN	N	0
29309	23/07/2019	8:45:00	AUTOV, M-30, +03200E	FUENCARRAL-EL PARDO	Alcance	Despejado	Turismo	Conductor	De 50 a 54 años	14.0	NaN	NaN	N	0
69764	09/08/2020	18:10:00	CALL. POLVORANCA, 13	CARABANCHEL	Choque contra obstáculo fijo	Despejado	Desconocido	Conductor	Desconocido	77.0	NaN	NaN	N	0
69765	09/08/2020	18:10:00	CALL. POLVORANCA, 13	CARABANCHEL	Choque contra obstáculo fijo	Despejado	Furgoneta	Conductor	Desconocido	77.0	NaN	NaN	N	0
69766	09/08/2020	18:10:00	CALL. POLVORANCA, 13	CARABANCHEL	Choque contra obstáculo fijo	Despejado	Turismo	Conductor	Desconocido	77.0	NaN	NaN	N	0

Por ejemplo, para la calle Polvoranca 13, vemos el punto marcado en amarillo y nuestras coordenadas en las casillas de coordenadas UTM:



The image shows a web application interface. On the left is a map of Madrid, Spain, with a yellow dot marking a location on Calle Polvoranca. On the right is a sidebar with the following sections:

- Search:** A search bar containing 'Calle Polvoranca 13' and a button to search.
- Location:** A button labeled 'Mi ubicación actual'.
- Grado decimal (WGS84):** Displays 'N 40.287399 E -3.794373'. Below are input fields for Latitude (40.287399) and Longitude (-3.794373) with unit selectors (° (N), ° (E)) and a button.
- Grados Minutos (WGS84):** Displays 'N 40° 17.243940 E -3° 47.662380'. Below are input fields for Latitude (40) and Longitude (-3) with unit selectors (° (N), ° (E)) and a button.
- Grados Minutos Segundos (WGS84):** Displays 'N 40° 17' 14.6364 E -3° 47' 39.7428'. Below are input fields for Latitude (40, 17, 14.6364) and Longitude (-3, 47, 39.7428) with unit selectors (° (N), ° (E)) and a button.
- CH1903+ / LV95 (Bessel 1841):** Input fields for Y and X coordinates with unit selectors (E, N) and a button.
- Las coordenadas UTM (WGS84):** Displays '30T 432477.331 4459959.173'. Below are input fields for Zone (30T), Easting (432477.331), and Northing (4459959.173) with a button.

Procedemos a introducir las coordenadas en cada una de las entradas:

```
df_Madrid.at[29308, 'coordenada_x_utm'] = 436432.168
df_Madrid.at[29308, 'coordenada_y_utm'] = 4480599.260
df_Madrid.at[29309, 'coordenada_x_utm'] = 436432.168
df_Madrid.at[29309, 'coordenada_y_utm'] = 4480599.260
df_Madrid.at[69764, 'coordenada_x_utm'] = 432477.331
df_Madrid.at[69764, 'coordenada_y_utm'] = 4459959.173
df_Madrid.at[69765, 'coordenada_x_utm'] = 432477.331
df_Madrid.at[69765, 'coordenada_y_utm'] = 4459959.173
df_Madrid.at[69766, 'coordenada_x_utm'] = 432477.331
df_Madrid.at[69766, 'coordenada_y_utm'] = 4459959.173
df_Madrid.at[72304, 'coordenada_x_utm'] = 445302.242
df_Madrid.at[72304, 'coordenada_y_utm'] = 4471156.069
df_Madrid.at[72305, 'coordenada_x_utm'] = 445302.242
df_Madrid.at[72305, 'coordenada_y_utm'] = 4471156.069
df_Madrid.at[72306, 'coordenada_x_utm'] = 445302.242
df_Madrid.at[72306, 'coordenada_y_utm'] = 4471156.069
df_Madrid.at[115061, 'coordenada_x_utm'] = 443894.339
```

df_Madrid.at[115061, 'coordenada_y_utm'] = 4469676.887
df_Madrid.at[115062, 'coordenada_x_utm'] = 443894.339
df_Madrid.at[115062, 'coordenada_y_utm'] = 4469676.887
df_Madrid.at[122183, 'coordenada_x_utm'] = 439092.309
df_Madrid.at[122183, 'coordenada_y_utm'] = 4475315.926
df_Madrid.at[122184, 'coordenada_x_utm'] = 439092.309
df_Madrid.at[122184, 'coordenada_y_utm'] = 4475315.926
df_Madrid.at[123687, 'coordenada_x_utm'] = 444731.062
df_Madrid.at[123687, 'coordenada_y_utm'] = 4474213.984
df_Madrid.at[123688, 'coordenada_x_utm'] = 444731.062
df_Madrid.at[123688, 'coordenada_y_utm'] = 4474213.984
df_Madrid.at[123689, 'coordenada_x_utm'] = 444731.062
df_Madrid.at[123689, 'coordenada_y_utm'] = 4474213.984
df_Madrid.at[125475, 'coordenada_x_utm'] = 445063.833
df_Madrid.at[125475, 'coordenada_y_utm'] = 4475362.909
df_Madrid.at[125669, 'coordenada_x_utm'] = 441111.754
df_Madrid.at[125669, 'coordenada_y_utm'] = 4477732.611
df_Madrid.at[125768, 'coordenada_x_utm'] = 438010.148
df_Madrid.at[125768, 'coordenada_y_utm'] = 4471237.295
df_Madrid.at[125858, 'coordenada_x_utm'] = 444790.650
df_Madrid.at[125858, 'coordenada_y_utm'] = 4473965.359
df_Madrid.at[125859, 'coordenada_x_utm'] = 444790.650
df_Madrid.at[125859, 'coordenada_y_utm'] = 4473965.359
df_Madrid.at[126829, 'coordenada_x_utm'] = 439110.247
df_Madrid.at[126829, 'coordenada_y_utm'] = 4472050.069
df_Madrid.at[168597, 'coordenada_x_utm'] = 442817.511
df_Madrid.at[168597, 'coordenada_y_utm'] = 4475153.037
df_Madrid.at[168598, 'coordenada_x_utm'] = 442817.511
df_Madrid.at[168598, 'coordenada_y_utm'] = 4475153.037
df_Madrid.at[169146, 'coordenada_x_utm'] = 444094.054

```
df_Madrid.at[169146, 'coordenada_y_utm'] = 4471642.517
df_Madrid.at[170224, 'coordenada_x_utm'] = 439539.588
df_Madrid.at[170224, 'coordenada_y_utm'] = 4471257.897
df_Madrid.at[170225, 'coordenada_x_utm'] = 439539.588
df_Madrid.at[170225, 'coordenada_y_utm'] = 4471257.897
df_Madrid.at[171337, 'coordenada_x_utm'] = 443229.003
df_Madrid.at[171337, 'coordenada_y_utm'] = 4472166.963
df_Madrid.at[173064, 'coordenada_x_utm'] = 440901.496
df_Madrid.at[173064, 'coordenada_y_utm'] = 4477557.433
df_Madrid.at[173065, 'coordenada_x_utm'] = 440901.496
df_Madrid.at[173065, 'coordenada_y_utm'] = 4477557.433
df_Madrid.at[173066, 'coordenada_x_utm'] = 440901.496
df_Madrid.at[173066, 'coordenada_y_utm'] = 4477557.433
```

Haremos un cambio de tipo en la columna de `cod_lesividad` y los pasaremos de decimal a entero y de entero a cadena, ya que no trataremos en ningún momento ese campo como numérico.

```
# Pasamos a entero para eliminar el decimal y luego a cadena
df_Madrid['cod_lesividad'] = df_Madrid['cod_lesividad'].astype(int)
df_Madrid['cod_lesividad'] = df_Madrid['cod_lesividad'].astype(str)
```

Y por último cambiamos el tipo de la columna `fecha` de tipo cadena a tipo fecha, para poder trabajar con ella en Tableau.

```
# Cambiamos el tipo de fecha de cadena a tipo fecha
df_Madrid['fecha'] = pd.to_datetime(df_Madrid['fecha'], format='%d/%m/%Y')
```

2.5 – Creación de nuevas columnas.

Crearemos algunas columnas con las que trabajaremos en Tableau, en vez de tener una hora, organizaremos los accidentes por franjas horarias por hora para que sea más fácil agrupar los accidentes. También, crearemos una columna de día de la semana, otra de mes, y por último, obtendremos la longitud y latitud y borraremos las coordenadas UTM ya que no trabajaremos con estas coordenadas en Tableau porque no las reconoce.

Para crear la franja horaria, crearemos una función que obtenga la primera parte de nuestro campo hora (antes de los 2 puntos ':'), y compondremos la hora inicial añadiendo a los dos puntos '00' como hora inicial y como hora final añadimos '59'.

```
# Crea una nueva columna para las franjas horarias
def asignar_franja_horaria(hora):
    hora_inicio = hora.split(':')[0] + ':00'
    hora_fin = hora.split(':')[0] + ':59'
    return f"{hora_inicio} - {hora_fin}"

df_Madrid['franja_horaria'] = df_Madrid['hora'].apply(asignar_franja_horaria)
```

Para la columna día de la semana 'dia_semana' tendremos que importar la librería calendar y utilizaremos el método dayofweek que nos devolverá el día de la semana de 0 a 6, siendo el 0 el lunes y el 6 el domingo. Para transformar los números a las cadenas correspondientes utilizaremos un mapa para transformarlos:

```
# Creamos nueva columna de día de la semana, primero transformamos la clase de la columna a datetime
import calendar

dias_semana_espanol = {0: 'lunes', 1: 'martes', 2: 'miércoles', 3: 'jueves', 4: 'viernes', 5: 'sábado', 6: 'domingo'}
df_Madrid['dia_semana'] = df_Madrid['fecha'].dt.dayofweek.map(dias_semana_espanol)
```

Para obtener el mes haremos lo mismo, creando un mapa de 1 a 12 para cada mes, y utilizaremos el método month de datetime (dt).

```
# Nueva columna del mes en español
mes_espanol = {1: 'enero', 2: 'febrero', 3: 'marzo',
               4: 'abril', 5: 'mayo', 6: 'junio',
               7: 'julio', 8: 'agosto', 9: 'septiembre',
               10: 'octubre', 11: 'noviembre', 12: 'diciembre'}
df_Madrid['mes'] = df_Madrid['fecha'].dt.month.map(mes_espanol)
```

Para transformar las coordenadas UTM a coordenadas de longitud y latitud utilizaremos la librería pyproj, y en concreto su método Transformer, donde haremos referencia a las coordenadas UTM con "EPSG:25830" y "EPSG:4326" para las coordenadas de longitud y latitud. Crearemos una función a la que le daremos las coordenadas UTM y nos las transformará en longitud y latitud, a la vez que creamos las columnas 'longitud' y 'latitud'.

```

from pyproj import Transformer

# Definir el transformer para la conversión de UTM "EPSG:25830" a Latitud/Longitud "EPSG:4326"
transformer = Transformer.from_crs("EPSG:25830", "EPSG:4326")

# Función para convertir coordenadas UTM a Longitud y Latitud
def utm_to_lat_lon(x, y):
    lon, lat = transformer.transform(x, y)
    return lon, lat

# Aplicar la función a las columnas 'coordenada_x_utm' y 'coordenada_y_utm'
df_Madrid['latitud'], df_Madrid['longitud'] = utm_to_lat_lon(df_Madrid['coordenada_x_utm'].values,
                                                            df_Madrid['coordenada_y_utm'].values)

```

Una vez obtenidos los campos de longitud y latitud, borraremos los campos de coordenadas UTM como hemos explicado anteriormente.

```

# Eliminaremos las columnas de coordenadas UTM, ya que solo trabajaremos con latitud y longitud
df_Madrid.drop(columns=['coordenada_x_utm', 'coordenada_y_utm'], inplace=True)

```

Ya tenemos nuestro dataframe preparado para poder utilizarlo y estudiar los datos que tenemos a través de cualquier programa que nos permita visualizar los datos gráficamente y analizarlos. En nuestro caso, utilizaremos el Tableau. Procedemos a crear nuestro archivo csv con el que trabajaremos en Tableau. Cabe destacar que guardaremos los decimales con un separador de coma y no de punto para evitarnos problemas en Tableau.

```

# Guardamos el dataframe en un archivo para ir explorando los resultados
df_Madrid.to_csv('C:/CSV/' + 'Madrid_Final.csv', index=False, decimal=',')

```


3 – Visualización gráfica de los datos. Tableau.

Llego el momento de interpretar y analizar los datos de nuestro archivo csv. En nuestro caso, utilizaremos Tableau que es una aplicación de pago, pero que consta de una versión gratuita de 14 días o una licencia de estudiantes de un año también gratuita. Su proceso de instalación es convencional:

- Nos bajamos la última versión de Tableau Desktop (2024.1).
- Decidimos la ruta donde lo queremos instalar.
- Y presionamos siguiente hasta que el botón cambie al de finalizar.

Una vez instalado llega el paso de conectar con el archivo csv, Madrid_Final.csv. Escogeremos la opción de conectar con un archivo de texto, ya que los csv se consideran como tales.



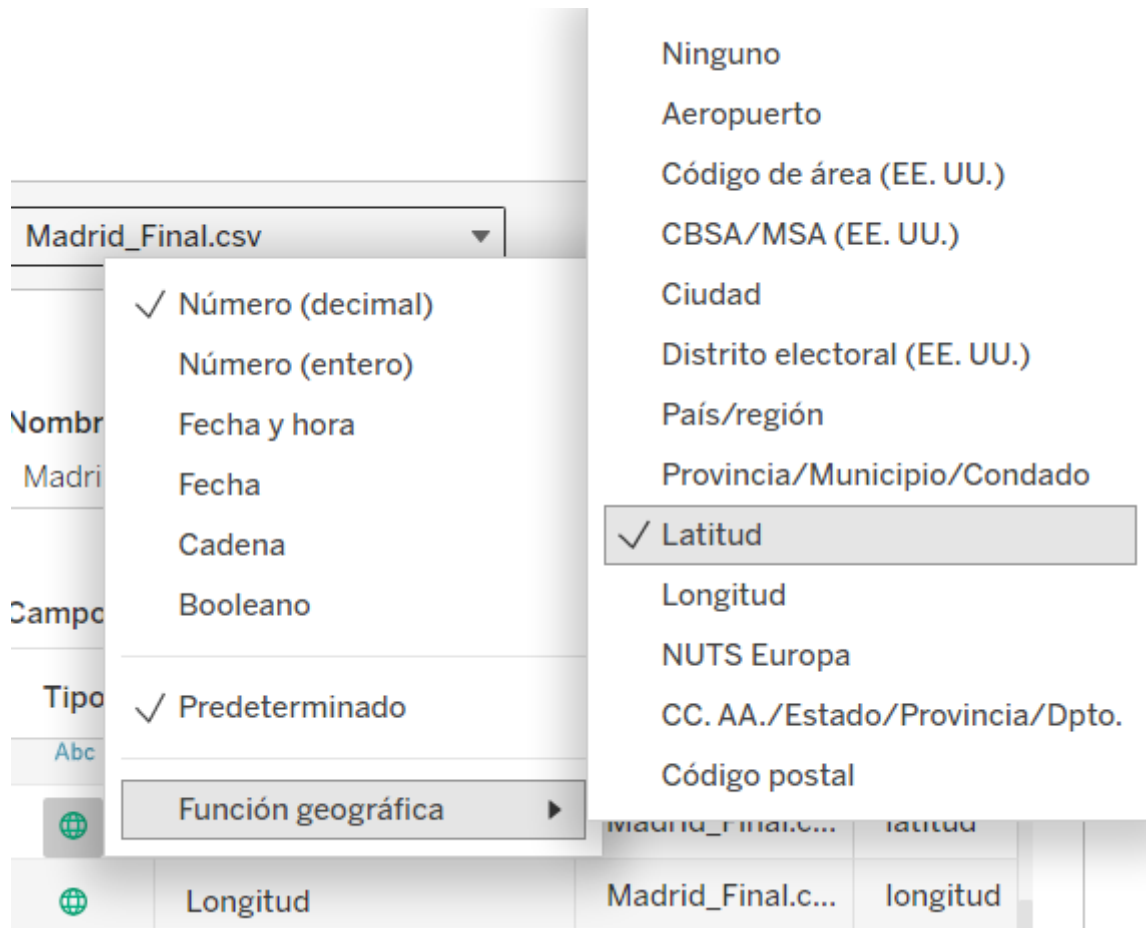
La pantalla nos mostrará al conectarnos una pantalla donde nos ofrecerán algo de información de nuestra tabla como todas las columnas, sus tipos, la posibilidad de conectar nuestra tabla con otras tablas, etc... En la esquina inferior izquierda, nos aparecerá unos iconos que nos permitirán movernos entre zonas de trabajo o crear nuevas.



- La primera de ellas y donde nos encontramos es la **Fuente de datos**, que como hemos explicado previamente es donde obtenemos información de nuestra tabla y nos permitirá realizar cambios como el de tipo de los campos o su nombre.
- La siguiente sería la **Hoja 1**, donde podremos crear nuestra primera gráfica. Podemos cambiar su nombre simplemente haciendo doble click en Hoja 1.
- El siguiente icono nos ofrece la posibilidad de crear una **nueva hoja de trabajo** y, por tanto, una nueva gráfica.
- A continuación, podemos seleccionar crear un **nuevo dashboard**, que es un entorno donde podremos combinar varias de nuestras gráficas ya creadas.

- Y, por último, **nueva historia** donde podremos poner todas nuestras gráficas o dashboard de manera secuencial para una presentación.

Antes de empezar en la fuente de datos, aprovecharemos para cambiar el tipo de datos de longitud y latitud, actualmente en formato numérico-decimal, a un tipo geográfico de longitud y latitud respectivamente para poder tener una imagen gráfica en el mapa de Madrid.



3.1 – Análisis de datos.

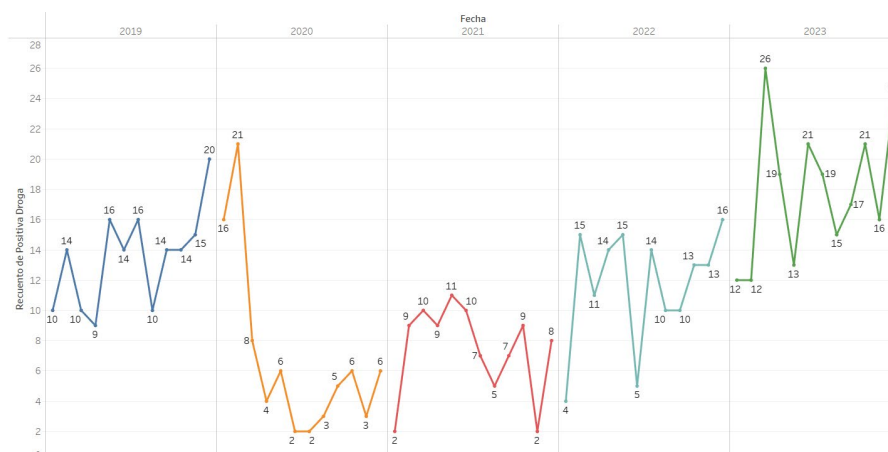
Vamos a analizar los datos, para ver cuales tienen más incidencia (más alta de probabilidad) en los accidentes ocurridos durante estos años, para ello haremos gráficas sobre los siguientes parámetros:

- Accidentes con positivo en drogas.
- Accidentes con positivo en alcohol.
- Estado meteorológico en el que ocurrieron los accidentes.
- Accidentes por franja horaria.
- Accidentes por rangos de edad.
- Accidentes por día de la semana.
- Accidentes por tipo de persona.
- Accidentes por mes.
- Accidentes por tipo de vehículo.
- Accidentes por tipo de accidente.
- Densidad de los accidentes.
- Accidentes por distrito.
- Tipo de gravedad de accidentes.
- Muertos.
- Zonas con más accidentados en Madrid.

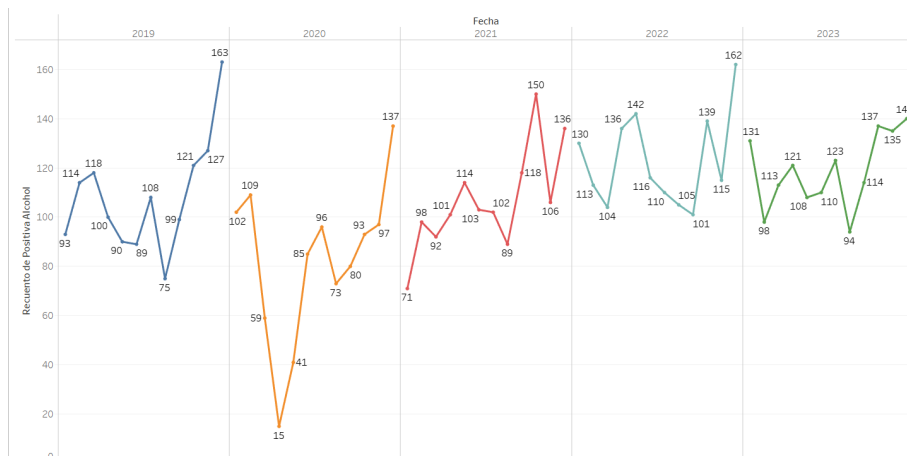
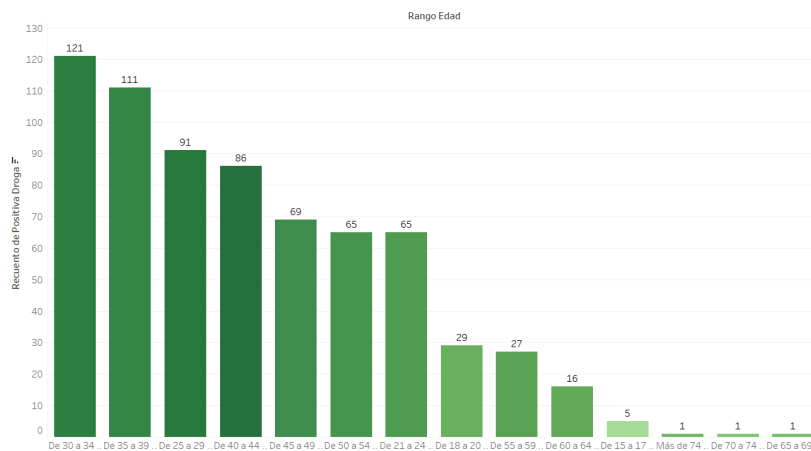
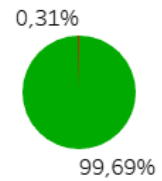
3.1.1 – Accidentes con positivo en drogas.

Para poder analizar la incidencia de las drogas en el total de accidentes en Madrid haremos 4 gráficas.

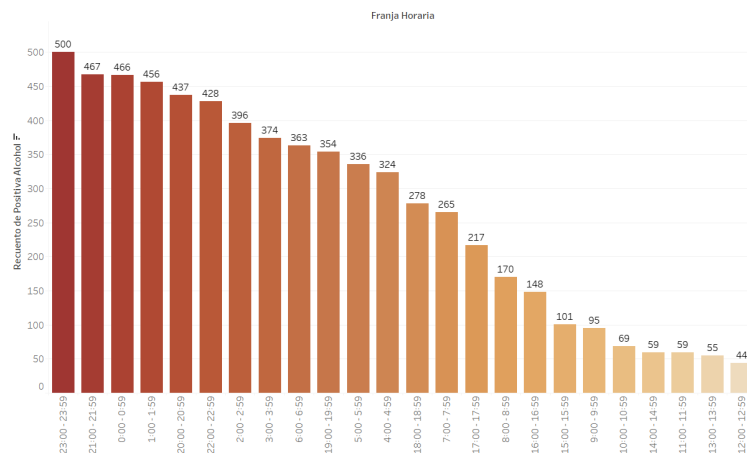
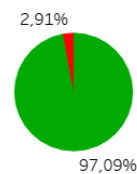
En la primera de ellas representaremos la cantidad de accidentes con positivo en droga en cada mes y año, para lo cual filtraremos el campo de positivos en droga y solo tomaremos los positivos (S). También, identificaremos cada año con un color para que nos sea más fácil identificarlo. Y, por último, añadiremos en la gráfica la cantidad de accidentes para poder tener una idea exacta.



	Fecha					
Positiva ..	2019	2020	2021	2022	2023	Total ge..
N	51.649	32.351	41.694	46.913	48.615	221.222
S	162	82	89	140	215	688
Total ge..	51.811	32.433	41.783	47.053	48.830	221.910



	Fecha					
Positiva ..	2019	2020	2021	2022	2023	Total ge..
N	50.514	31.446	40.503	45.580	47.406	215.449
S	1.297	987	1.280	1.473	1.424	6.461
Total ge..	51.811	32.433	41.783	47.053	48.830	221.910



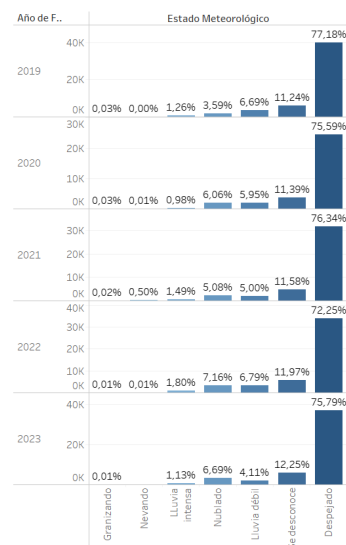
3.1.3 - Estado meteorológico en el que ocurrieron los accidentes.

Aquí analizaremos que parte porcentual de los accidentes se producen en que estado meteorológico. Para ellos analizaremos el porcentaje en cada año, además de sus cifras totales anuales. Las desviaciones que hay en cada accidente debido al estado meteorológico y la franja horaria.

Para esta última gráfica, deberemos crear varios campos calculados para poder hallar varios campos: recuento de accidentes, total por estado meteorológico, total por franja horaria y estado, porcentaje total por estado meteorológico, porcentaje por franja horaria y estado y diferencia de porcentaje. Las diferencias negativas serán representadas por tonalidades verdes y querrá decir que hay menos accidentes que el promedio habitual con ese estado meteorológico y rojo si ha habido más que el promedio porcentual en ese estado meteorológico.

Franja Horaria	Estado Meteorológico						Diferencia de %
	Despejado	Se desconoce	Lluvia débil	Nublado	Lluvia inten..	Nevando	
0:00 - 0:59	74,60%	12,03%	6,97%	4,35%	1,96%	0,09%	
1:00 - 1:59	76,75%	11,66%	6,77%	3,31%	1,51%		
2:00 - 2:59	74,04%	14,75%	6,38%	3,41%	1,39%	0,03%	
3:00 - 3:59	73,72%	15,21%	5,81%	3,31%	1,83%		0,12%
4:00 - 4:59	73,30%	12,80%	7,53%	4,49%	1,74%	0,14%	
5:00 - 5:59	75,33%	14,80%	6,39%	3,52%	0,91%	0,04%	
6:00 - 6:59	72,76%	13,31%	7,62%	4,81%	1,50%		
7:00 - 7:59	74,02%	9,62%	7,47%	6,89%	1,83%	0,05%	0,12%
8:00 - 8:59	75,52%	8,17%	6,59%	8,51%	1,15%	0,02%	0,04%
9:00 - 9:59	76,94%	9,18%	5,74%	7,36%	0,78%		
10:00 - 10:59	77,65%	9,16%	4,82%	7,56%	0,79%	0,01%	
11:00 - 11:59	79,63%	8,87%	4,36%	6,84%	0,78%	0,12%	
12:00 - 12:59	79,49%	8,90%	4,46%	6,34%	0,68%	0,12%	
13:00 - 13:59	81,06%	8,16%	3,74%	5,85%	0,98%	0,21%	
14:00 - 14:59	79,74%	9,96%	4,14%	5,05%	0,97%	0,13%	
15:00 - 15:59	76,86%	11,75%	4,76%	5,46%	1,03%	0,13%	0,01%
16:00 - 16:59	74,38%	13,88%	4,51%	5,89%	1,13%	0,20%	0,01%
17:00 - 17:59	74,42%	13,72%	5,00%	5,26%	1,38%	0,17%	0,06%
18:00 - 18:59	72,58%	14,37%	5,66%	5,65%	1,52%	0,15%	0,08%
19:00 - 19:59	72,67%	13,95%	6,55%	5,25%	1,54%	0,04%	
20:00 - 20:59	71,69%	13,65%	6,92%	5,81%	1,82%	0,08%	0,03%
21:00 - 21:59	70,94%	14,15%	7,15%	5,53%	2,09%	0,12%	0,02%
22:00 - 22:59	74,48%	11,57%	7,52%	4,25%	2,11%	0,07%	
23:00 - 23:59	75,02%	11,18%	7,84%	3,76%	2,18%	0,03%	

Estado Meteorológico	Fecha					Total
	2019	2020	2021	2022	2023	
Granizando	14	10	5	6	7	45
Nevando	2	2	208	6		218
Lluvia Intensa	653	317	622	849	552	2.993
Nublado	1.861	1.965	2.122	3.370	3.269	12.587
Lluvia débil	3.468	1.930	2.089	3.193	2.009	12.689
Se desconoce	5.826	3.694	4.838	5.632	5.984	25.974
Despejado	39.987	24.515	31.896	33.997	37.009	167.404
Total	51.811	32.433	41.783	47.053	48.830	221.910

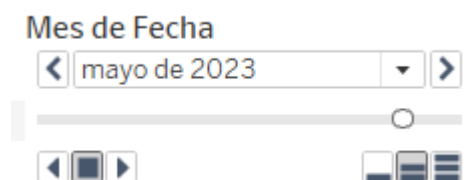


3.1.4 - Accidentes por franja horaria, rango de edad, día de la semana y tipo de persona.

Para el análisis de estos cuatro campos utilizaremos los mismos tipos de gráficas, una gráfica convencional de barras donde veremos el recuento de accidentes por cada campo, restringido a mes y año. Además, tendremos un pequeño gráfico circular donde tendremos las secciones de cada valor de cada campo y el gráfico circular será más grande o pequeño en referencia a la cantidad de accidentes ese año y mes.



Reseñar que tendremos un pequeño control, para bien, seleccionar el mes y el año que queremos ver, o podremos verlo automáticamente dándole al botón de siguiente o anterior y podremos seleccionar la velocidad de reproducción.



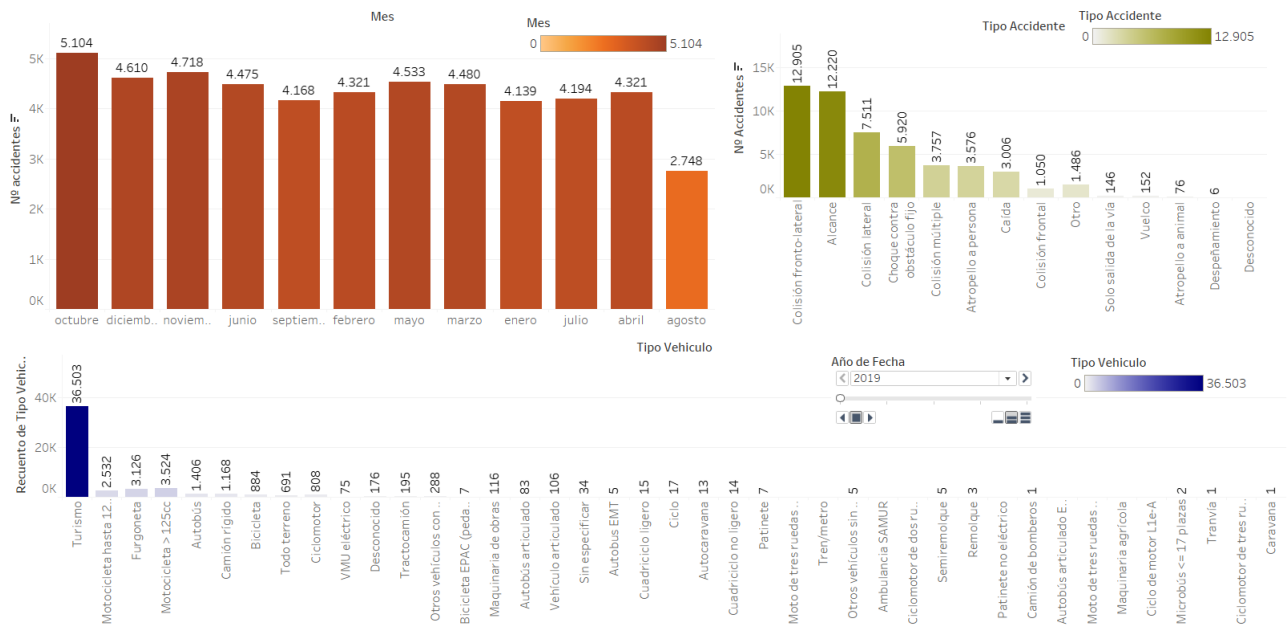
3.1.5 - Accidentes por mes, tipo de vehículo y tipo de accidente.

En este dashboard, controlaremos los accidentes por mes para ver si la incidencia que tiene estar en un determinado mes en el total de los accidentes. Utilizaremos una gráfica convencional de barras y con una representación por tonalidades de color, siendo el color más vivo/fuerte el que tiene más cantidad de accidentes.

Destacar que los datos están ordenados por los totales de los accidentes en los cinco años que estamos controlando, pero, los datos que vemos en pantalla son de cada año y tendremos como anteriormente un control para poder movernos entre ellos.

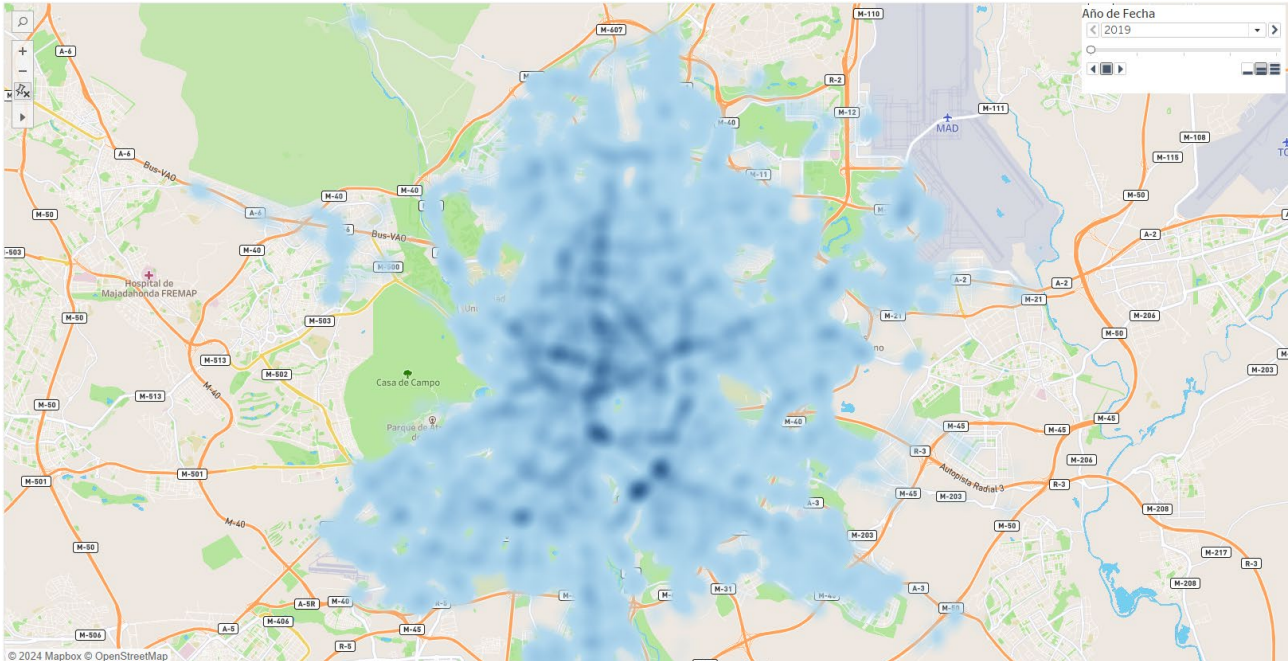
También controlaremos de la misma forma que por mes, la incidencia en los accidentes del tipo de vehículo involucrado y como es el tipo de accidente ocurrido. Utilizaremos el mismo tipo gráfica para ambos, gráfica de barras.

Algo que no hemos comentado hasta ahora, es que todas las gráficas son interactivas, es decir puedes ver datos específicos, en este caso podríamos ver solo los datos de tipo de vehículo y del tipo de accidente en agosto de un año al pinchar en la gráfica el mes de agosto. Todo esto funciona en los otros dashboard anteriores.



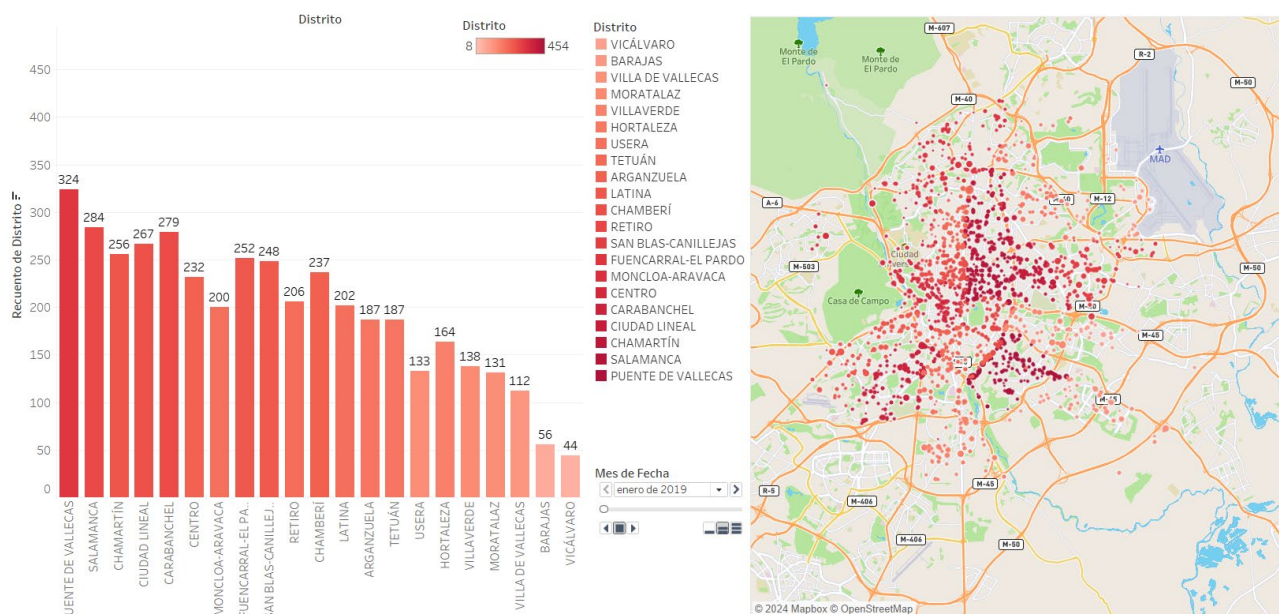
3.1.6 - Densidad de los accidentes.

Esta sería nuestra primera gráfica en la que utilizaremos un mapa, para ello utilizaremos las coordenadas de latitud y longitud, y escogeremos tenerlo para que podamos tener una idea de donde ha habido más accidentes, ya que tendremos zonas más oscuras. Representamos los datos por año teniendo como siempre un controlador para movernos entre los años.



3.1.7 - Accidentes por distrito.

Ahora veremos si hay más accidentes en algunos distritos que en otros, tendremos un par de gráficas: una de barras donde veremos el recuento en barras de la cantidad de accidentes en cada distrito, ordenados por el total de accidentes desde 2019 a 2023, pero que veremos por año y mes, y en el otro veremos una representación en un mapa donde se han producido estos accidentes. Representaremos la totalidad de accidentes en tonalidades más oscuras donde más accidentes ha habido.



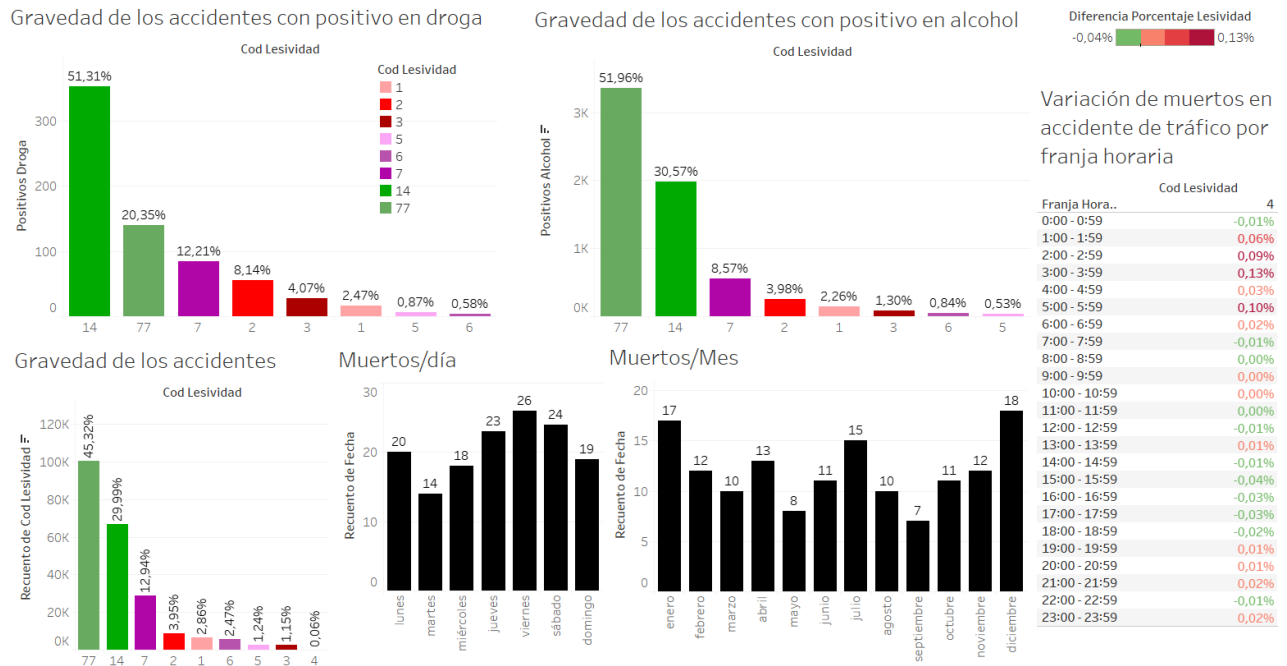
3.1.8 - Tipo de gravedad de accidentes y muertos.

Ahora, vamos a analizar la gravedad de los accidentes, y en especial, de los accidentes con muertos (04). Si recordamos los códigos de lesividad son los siguientes:

- **01**, atención en urgencias sin posterior ingreso.
- **02**, ingreso inferior o igual a 24 horas.
- **03**, ingreso superior a 24 horas.
- **04**, fallecido 24 horas.
- **05**, asistencia sanitaria ambulatoria con posterioridad.
- **06**, asistencia sanitaria inmediata en centro de salud o mutua.
- **07**, asistencia sanitaria sólo en el lugar del accidente.
- **14**, sin asistencia sanitaria.
- **77**, se desconoce.

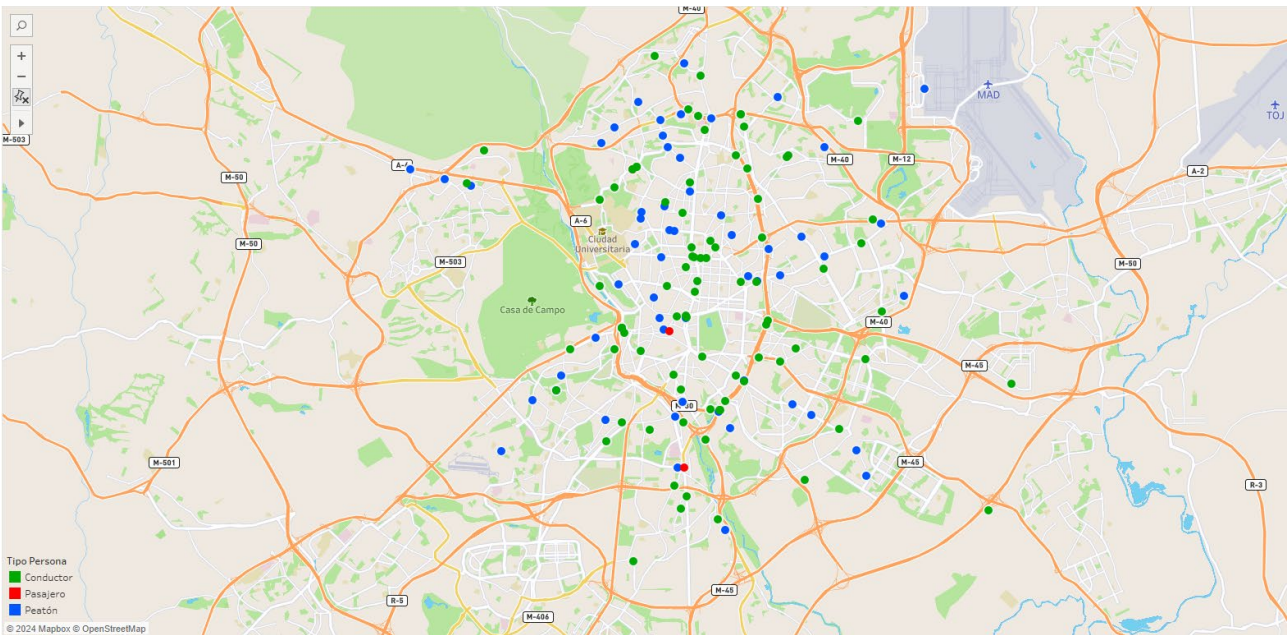
Analizaremos si ha habido muertos tanto con positivos en droga tanto con positivos con alcohol, y, vemos que, no habido muertos, para sorpresa al menos mía. También, analizamos los muertos totales en cada día de la semana y los muertos totales en cada mes.

Por último, analizaremos si la hora del día tiene incidencia en las muertes ocurridas en los accidentes en Madrid, y aunque hay muertos en todas las franjas horarias vemos que entre la 01:00 y las 07:00 hay una mayor tasa de mortalidad.



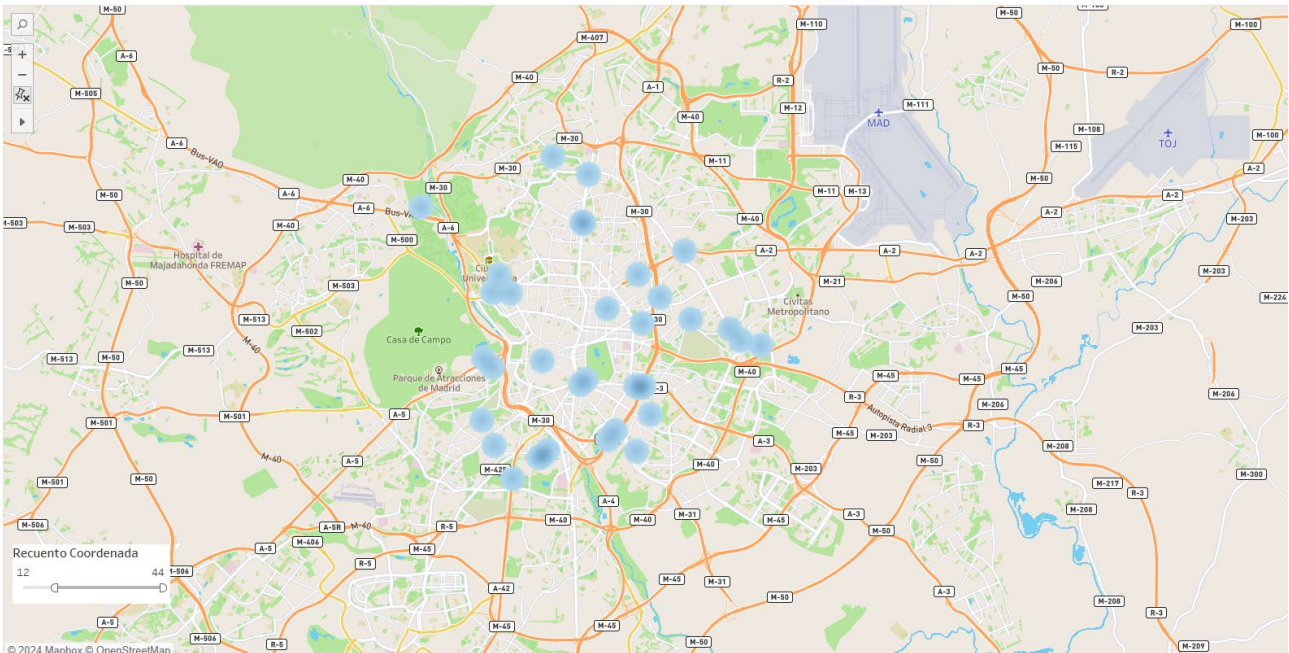
3.1.9 – Mapa con accidentes con muertos en Madrid.

Vamos a mostrar en el mapa de la ciudad de Madrid donde han ocurrido los accidentes con muertos y, también, que tipo de persona (conductor, pasajero o peatón). Tenemos una descripción emergente que nos mostrará entre otras cosas la fecha del accidente, la franja horaria y su localización.



3.1.10 - Zonas con más accidentados en Madrid.

Ya, por último, hacemos un gráfico con un mapa geográfico, pero con un filtro para ser más concretos en los puntos con más accidentados. El filtro es interactivo y podemos manipular tanto el mínimo de accidentes o el máximo. Haciendo zoom, podemos ver exactamente el punto de los accidentes y la posible causa de los accidentes, ya sea, una incorporación a la M-30, una glorieta, etc...



3.2 – Conclusiones del análisis gráfico.

Dados los datos analizados podemos concluir que no hay muchos accidentes de gente con positivo en drogas o alcohol, ya bien, porque son “responsables” y no conducen bajo esos estados o bien porque esos estados no inciden en la forma de conducir. También podemos deducir que, a más información, mejor análisis estadístico se puede hacer de una situación.

Los demás campos que hemos analizado al contrario que, las drogas y el alcohol, vemos que, si pueden tener mayor influencia en el total de los accidentes ocurridos, aunque al desconocer el volumen de automóviles que no han tenido accidentes y toda su información adicional, no podemos conocer la verdadera incidencia de cualquiera de los campos.

4 – MODELO DE PREDICCION.

Como hemos visto al trabajar en nuestra tabla no tenemos casi datos numéricos, así que para el modelo de predicción deberemos manejar datos categóricos. Intentaremos utilizar los datos que tenemos para determinar la lesividad que podríamos tener un accidente. Para ello utilizaremos un modelo de árbol de decisión que nos permitirá tratar con este tipo de datos.

Podríamos definir los árboles de decisión como un algoritmo de aprendizaje supervisado. Se denomina árbol de decisión su estructura, partiremos de un nodo raíz a un nodo interno (por ejemplo, franja horaria) que a su vez se dividirá entre otros nodos hasta llegar a los nodos hoja que serán los que representen el resultado final del conjunto de datos.

En concreto, aquí vamos a utilizar un Random Forest que es un conjunto de árboles de decisión que tienen las siguientes características:

- **Bagging (Bootstrap Aggregating):** Cada árbol se entrena con una muestra aleatoria (con reemplazo) del conjunto de datos.
- **Selección Aleatoria de Características:** En cada nodo de cada árbol, un subconjunto aleatorio de características es considerado para la división.
- **Agregación de Resultados:** La predicción final se realiza mediante votación mayoritaria para clasificación o promedio para regresión.

Voy a explicar brevemente que librerías vamos a implementar para poder hacer el modelo de predicción:

1. **Train_test_split**, lo utilizaremos para dividir nuestros datos en dos conjuntos: entrenamiento y testeo.
2. **Cross_val_score**, que nos permitirá el conjunto de datos de entrenamiento en 'k' partes y entrenando/evaluando el modelo 'k' veces.
3. **OneHotEncoder**, para convertir variables categóricas en numéricas, en concreto, en una columna binaria (0 o 1).
4. **StandardScaler**, para estandarizar las características numéricas.
5. **ColumnTransformer**, que nos permite aplicar diferentes transformaciones a diferentes columnas del conjunto de datos. Es útil para preprocesar características numéricas y categóricas de manera diferente en una sola línea de código.
6. **Pipeline**, encapsula un flujo de trabajo completo de machine learning en una sola estructura. Permite encadenar múltiples pasos, como preprocesamiento y modelado, en una secuencia ordenada.
7. **RandomForestClassifier**, es un clasificador basado en un conjunto de árboles de decisión (bosque aleatorio). Utiliza múltiples árboles de decisión y agrega sus predicciones para mejorar la precisión y reducir el riesgo de sobreajuste.
8. **Accuracy_score**, calcula la precisión del modelo, que es la proporción de predicciones correctas sobre el total de predicciones.

9. **Classification_report**, genera un informe detallado de las métricas de clasificación, como precisión, recall y f1-score, para cada clase en el conjunto de datos.
10. **KFold**, implementa la técnica de validación cruzada "K-Fold", dividiendo el conjunto de datos en "k" partes y entrenando/evaluando el modelo "k" veces para asegurar una evaluación robusta.
11. **Numpy**, para cálculos científicos en Python.
12. **Time**, lo utilizaremos para medir el tiempo de ejecución del código.

Primero establecemos cual va a ser nuestra variable objetivo, en nuestro caso el código de lesividad. Posteriormente, haremos lo siguiente:

1. Dividiremos el conjunto de datos en dos partes: el conjunto de entrenamiento (X_train y y_train) y el conjunto de prueba (X_test y y_test). Utilizaremos el 80% para entrenamiento y el 20% para testeo, son unas cifras estándar (test_size=0.2). Y finalmente usamos el random state 42 que nos asegura que la división sea reproducible. El número 42 no tiene un significado técnico especial, proviene del libro "Guía del autoestopista galáctico" de Douglas Adams, y es el número que representa "la respuesta a la vida, el universo y el todo".
2. Creamos y configuramos el preprocesador que transforme nuestras las variables numéricas (StandardScaler) y categóricas (OneHotEncoder).
3. Encapsulamos el preprocesador y el modelo (RandomForestClassifier) en una sola 'tubería' para facilitar el flujo de trabajo.
4. Evaluamos el modelo con validación cruzada, dividiendo el conjunto de entrenamiento en 5 pliegues, donde entrenaremos con 4 de ellos y evaluaremos en el restante. Este proceso se repetirá cinco veces. Reflejaremos los resultados con 'np.mean' para obtener nuestra tasa de acierto y 'np.std' para ver la desviación estándar de la tasa de acierto entre los cinco pliegues.
5. Entrenamos el modelo completo y evaluamos en el conjunto de prueba. Obtendremos nuestro porcentaje de acierto con 'accuracy_score(y_test, y_pred)' y obtendremos métricas detalladas de los resultados con 'classification_report(y_test, y_pred)'.
6. Por último, veremos cuantos segundos ha durado la realización del modelo de aprendizaje.

Por último, explicaremos que datos obtenemos con classification_report para que los podamos entender mejor:

1. **Precision**, nos dará qué proporción de las predicciones positivas fueron correctas.

$$\text{Precisión} = \frac{\text{Verdaderos Positivos (VP)}}{\text{Verdaderos Positivos (VP)} + \text{Falsos Positivos (FP)}}$$

2. **Recall** (cobertura/sensibilidad), medirá la proporción de los verdaderos positivos fueron correctamente identificados por el modelo.

$$\text{Cobertura} = \frac{\text{Verdaderos Positivos (VP)}}{\text{Verdaderos Positivos (VP)} + \text{Falsos Negativos (FN)}}$$

3. **F1-Score**, sería la media armónica de la precisión y la cobertura. Este valor solo será alto cuando la precisión y la cobertura sean altas.

$$\text{F1-score} = 2 \cdot \frac{\text{Precisión} \cdot \text{Cobertura}}{\text{Precisión} + \text{Cobertura}}$$

4. **Support**, es el número de ocurrencias reales de cada clase en el conjunto de datos.
5. **Macro avg**, calcula la métrica (precisión, cobertura, F1-score) para cada clase por separado y luego promedia estos valores, sin importar cuántos ejemplos haya en esa clase.

$$\text{Precisión Macro} = \frac{\sum_{i=1}^n \text{Precisión}_i}{n}$$

6. **Weighted avg**, calcula la métrica para cada clase, pero pondera cada métrica por el número de ejemplos en esa clase (su soporte). Esto significa que las clases con más ejemplos tienen un mayor impacto en el promedio.

$$\text{Precisión Ponderada} = \frac{\sum_{i=1}^n (\text{Precisión}_i \times \text{Soporte}_i)}{\sum_{i=1}^n \text{Soporte}_i}$$

4.1 – Conclusiones sobre el modelo de predicción.

Vamos a ver los resultados obtenidos en los diferentes modelos de predicción, para ello representaremos los datos obtenidos en tablas que nos permitan una rápida visualización de los resultados:

- Primero evaluamos todos los campos con las coordenadas, nuestras únicas variables numéricas, y sin las coordenadas.

Columnas numéricas: ['latitud', 'longitud'] Columnas categóricas: ['franja_horaria', 'estado_meteorológico', 'rango_edad', 'distrito', 'tipo_accidente', 'tipo_vehículo', 'tipo_persona', 'positiva_alcohol', 'positiva_droga', 'dia_semana', 'mes']				
Acierto en validación cruzada: 61,41%				
Desviación estándar en validación cruzada: 0,01%				
Acierto en el banco de test: 62,72%				
Cod. lesividad	Precision	Recall	F1-Score	Support
1	22%	5%	8%	1.289
14	59%	64%	61%	13.469
2	25%	9%	13%	1.741
3	24%	6%	10%	489
4	0%	0%	0%	26
5	30%	3%	.6%	548
6	24%	4%	7%	1.111
7	37%	46%	41%	5.634
77	76%	82%	79%	20.075
Accuracy			63%	44.382
Macro Avg	33%	24%	25%	44.382
Weighted Avg	60%	63%	60%	44.382

Columnas categóricas: ['franja_horaria', 'estado_meteorológico', 'rango_edad', 'distrito', 'tipo_accidente', 'tipo_vehículo', 'tipo_persona', 'positiva_alcohol', 'positiva_droga', 'dia_semana', 'mes']				
Acierto en validación cruzada: 59,31%				
Desviación estándar en validación cruzada: 0,2%				
Acierto en el banco de test: 60,28%				
Cod. lesividad	Precision	Recall	F1-Score	Support
1	21%	5%	8%	1.289
14	57%	61%	59%	13.469
2	20%	9%	12%	1.741
3	16%	5%	18%	489
4	0%	0%	0%	26
5	24%	3%	.5%	548
6	20%	4%	6%	1.111
7	36%	44%	40%	5.634
77	73%	79%	76%	20.075
Accuracy			60%	44.382
Macro Avg	30%	23%	24%	44.382
Weighted Avg	57%	60%	58%	44.382

- Ahora, solo utilizaremos 6 variables categóricas, y otra vez, compararemos los resultados añadiendo longitud y latitud.

Columnas categóricas: ['franja_horaria', 'estado_meteorológico', 'rango_edad', 'distrito', 'dia_semana', 'mes']				
Acierto en validación cruzada: 49,32%				
Desviación estándar en validación cruzada: 0,16%				
Acierto en el banco de test: 50,13%				
Cod. lesividad	Precision	Recall	F1-Score	Support
1	15%	4%	7%	1.289
14	41%	50%	45%	13.469
2	15%	6%	8%	1.741
3	6%	2%	3%	489
4	0%	0%	0%	26
5	11%	3%	.5%	548
6	7%	2%	3%	1.111
7	23%	14%	17%	5.634
77	64%	72%	68%	20.075
Accuracy			50%	44.382
Macro Avg	20%	17%	17%	44.382
Weighted Avg	46%	50%	47%	44.382

Columnas numéricas: ['latitud', 'longitud']				
Columnas categóricas: ['franja_horaria', 'estado_meteorológico', 'rango_edad', 'distrito', 'dia_semana', 'mes']				
Acierto en validación cruzada: 53,32%				
Desviación estándar en validación cruzada: 0,22%				
Acierto en el banco de test: 54,43%				
Cod. lesividad	Precision	Recall	F1-Score	Support
1	23%	4%	8%	1.289
14	43%	56%	49%	13.469
2	23%	6%	9%	1.741
3	7%	1%	2%	489
4	0%	0%	0%	26
5	19%	4%	.7%	548
6	12%	2%	4%	1.111
7	27%	13%	18%	5.634
77	68%	78%	73%	20.075
Accuracy			54%	44.382
Macro Avg	25%	18%	19%	44.382
Weighted Avg	49%	54%	51%	44.382

- Y, por último, analizaremos un modelo de predicción con solo dos campos categóricos: franja horaria y mes, que consideramos que pueden tener más incidencia en la lesividad. Y como en las anteriores, lo haremos con las coordenadas y sin ellas. Al sacar el modelo de predicción con solo dos campos Python nos advierte de la debilidad de este modelo, en cambio, con las coordenadas no nos advierte de esta circunstancia.

Columnas categóricas: ['franja_horaria', 'mes']				
Acierto en validación cruzada: 45,23%				
Desviación estándar en validación cruzada: 0,06%				
Acierto en el banco de test: 45,29%				
Cod. lesividad	Precision	Recall	F1-Score	Support
1	0%	0%	0%	1.289
14	41%	1%	2%	13.469
2	0%	0%	0%	1.741
3	0%	0%	0%	489
4	0%	0%	0%	26
5	0%	0%	0%	548
6	0%	0%	0%	1.111
7	0%	0%	0%	5.634
77	45%	99%	62%	20.075
Accuracy			45%	44.382
Macro Avg	10%	11%	7%	44.382
Weighted Avg	33%	45%	29%	44.382

Columnas numéricas: ['latitud', 'longitud']				
Columnas categóricas: ['franja_horaria', 'mes']				
Acierto en validación cruzada: 54,83%				
Desviación estándar en validación cruzada: 0,04%				
Acierto en el banco de test: 57,59%				
Cod. lesividad	Precision	Recall	F1-Score	Support
1	13%	9%	11%	1.289
14	44%	48%	46%	13.469
2	16%	11%	13%	1.741
3	7%	5%	6%	489
4	0%	0%	0%	26
5	12%	9%	10%	548
6	11%	7%	9%	1.111
7	24%	19%	21%	5.634
77	82%	88%	85%	20.075
Accuracy			58%	44.382
Macro Avg	23%	22%	22%	44.382
Weighted Avg	55%	58%	56%	44.382

A la vista de los resultados, podemos decir que tenemos unos datos muy dispersos o que no son muy preponderantes o influyentes en la lesividad de los accidentes y por eso, nuestro mejor modelo de predicción nos da un 62,72%, usando todos los datos y campos que tenemos a nuestra disposición. También, comparando la tabla 4 y la tabla 6, con menos datos categóricos, obtiene mejores tasas de aciertos (54,53% por 57,59%), así que podemos pensar que hay datos que no ayudan a obtener mejores resultados.

A posteriori, podemos pensar que a lo mejor acotando un poco más los campos podríamos tener unos resultados mejores. Por ejemplo: la franja horaria, la hemos dividido en franjas de 1 hora, a lo mejor haciendo franjas mayores de 3 horas.

Hay un desbalance de clases (códigos de lesividad) lo que hace más difícil predecir los casos con menos casos: 4 solo tiene 26 y el 5 tiene 548, y debido a eso obtenemos una baja precisión y recall. Por tanto, no somos capaces de obtener un patrón consistente.

Sin duda, con el paso del tiempo y con la recolección de más datos podría ayudar a fortalecer nuestros modelos de predicción. También, podríamos haber cruzado datos que tienen cierta relevancia, que no dominancia, en la lesividad para hacer más importante su influencia en la relación que tendría más importancia, y, por tanto, patrones de aprendizaje más fuertes. Creo que tener más datos categóricos nos podría también ayudar algo más a obtener datos más sólidos.

5 - BIBLIOGRAFIA.

- [Validación cruzada](#)
- [Árboles de decisión](#)
- [Curso gráfico de Tableau de OpenWebinars](#)
- [Curso de Tableau Online de OpenWebinars](#)
- [Análisis de datos con Python \(pandas\)](#)
- [Ciencia de datos con Python](#)
- [Análisis y visualización de datos usando Python](#)
- [Accidentes de tráfico de la Ciudad de Madrid](#)
- [Convertidor en línea a todos los sistemas de coordenadas](#)

6 – ENLACES GITHUB.

- [Enlace al repositorio del proyecto](#)
- [CSV utilizados y final](#)
- [Funcionalidad en Tableau de las gráficas](#)